This section is about fitting linear models, so the possibility of an exponential fit is put to one side for the time being; building non-linear models, including better fitting non-linear models to this data, is discussed later; see section 11.5.

The following call to `glm` fits an equation that is quadratic in the variable `Number_days`; the righthand side of the formula contains `Number_days+I(Number_days^2)`. The `I` (sometimes known as *as-is*) causes its argument to remain unevaluated and is treated as a distinct explanatory variable (the `^` operator has a distinct meaning within `glm`'s formula notation, see table 11.2, and use of `I` prevents the expected binary operator usage being overridden). An alternative way of including a squared explanatory variable in the model, is to assign the value `Number_days^2` to a new variable and include this new variable's name on the righthand side of the formula. Figure 11.7, lower plot, shows the result of fitting this equation.

```
linux_mod=glm(sloc ~ Number_days+I(Number_days^2), data=linux_info)
```

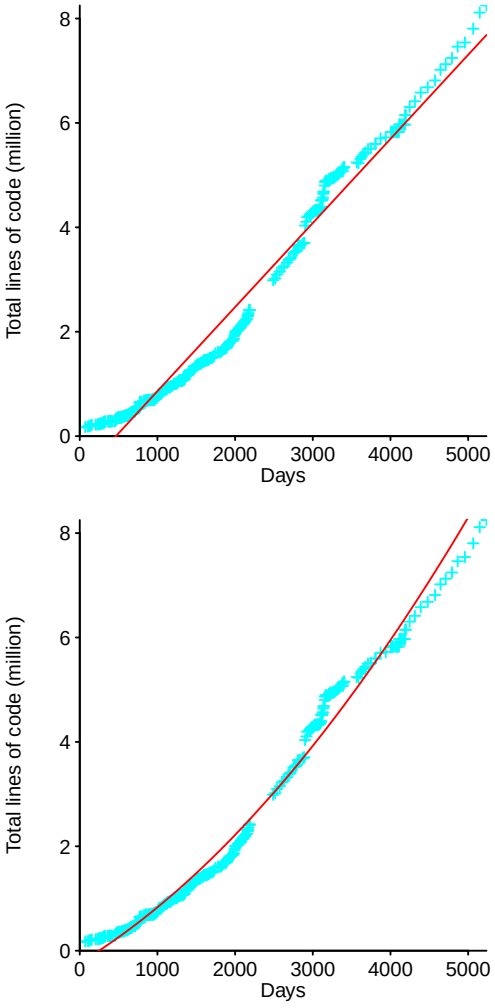


Figure 11.7: Lines of code in every initial release (i.e., excluding bug-fix versions of a release) of the Linux kernel since version 1.0, along with fitted straight line (upper) and quadratic (lower) regression models. Data from Israeli et al.[902] Github–Local

The quadratic fit looks like it is better than linear, but perhaps a cubic, quartic or higher degree polynomial would be even better fits. The higher the order of the polynomial used, the smaller the error between the fitted model and the data. The error decreases because the additional terms make it possible to do a better job of following the random fluctuations in the data. A method of applying Occam's razor is needed, to select the number of terms that produces the simplest model consistent with the data, and having an acceptable error.

The Akaiki Information Criterion, AIC, is a commonly used metric for comparing two or more models (available in the `AIC` function). It takes into account both how well a model fits the data, and the number of free coefficients in the model (i.e., constants selected by the model building process, such as polynomial coefficients); free coefficients have to pay their way by providing an appropriate improvement in a model's fit to the data.[xii] AIC can also be viewed as the information loss when the true model is not among those being considered.[281]

One set of selection criteria[281] are that models whose AIC differs by less than 2 are more or less equivalent, those that differ by between 4 and 7 are clearly distinguishable, while those differing by more than 10 are definitely different.

How much better does a quadratic equation fit Linux SLOC growth, compared to a straight line and how much better do higher degree polynomials fit? The following list gives the AIC for models fitted using polynomials of degree 1 to 4 (lower values of AIC are better). After initially decreasing the AIC starts to increase, once a fourth degree polynomial is reached; the third degree polynomial is thus the chosen linear polynomial, of those tested (other forms of equation could provide better fits using fewer free coefficients.) Github–Local

```
[1] Degree 1, AIC= 13998.0004739753
[1] Degree 2, AIC= 13674.6883243397
[1] Degree 3, AIC= 13220.8542892188
[1] Degree 4, AIC= 13221.7072389496
```

The following is the `summary` output for the fitted cubic model: Github–Local

```
Call:
glm(formula = LOC ~ Number_days + I(Number_days^2) + I(Number_days^3),
    data = latest_version)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-428217   -80061     6503    64889   620500

Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
(Intercept)       3.432e+05  2.876e+04  11.935  < 2e-16 ***
Number_days      -3.664e+02  5.144e+01  -7.123 3.79e-12 ***
I(Number_days^2)  8.167e-01  2.456e-02  33.258  < 2e-16 ***
I(Number_days^3) -9.184e-05  3.371e-06 -27.242  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

[xii] A negative AIC value may be the result of a nominal explanatory variable having many values, e.g., dates that are represented as strings, which are could be converted using `as.Date`.

```
(Dispersion parameter for gaussian family taken to be 23001633959)

    Null deviance: 1.8867e+15  on 494  degrees of freedom
Residual deviance: 1.1294e+13  on 491  degrees of freedom
AIC: 13221

Number of Fisher Scoring iterations: 2
```

Regression modeling produces the best fit for an equation over the range of data values used, and AIC helps prevent overfitting. No claims are made about how well the model is likely to fit data outside the range values used to fit it. Using a model, optimized to fit the available data, to make predictions outside the interval of the data values used can produce unexpected results.

What is the behavior of the above cubic model outside its fitted range, and in particular, what predictions does it make about future growth? Figure 11.8 shows that the model predicts a future decrease in the number of lines of code. A decreasing number of lines is the opposite of previous behavior, this prediction does not appear believable (if this decreasing behavior were predicted by a more detailed model of code growth, that closely mimicked real-world development by using information on the number of developers actively involved, and a list of functionality likely to be implemented, it would be more believable).

A quadratic equation might not fit the data as well as a cubic equation, but the form of its predictions (increasing growth) is consistent with expectations.

If the purpose of modeling is to gain understanding, then the quadratic model maps more closely to anticipated behavior; if the purpose is prediction within the interval of the fitted data, then the cubic model is likely to have a smaller error.

What about fitting other kinds of equations to the data? Equations such as $Y = \alpha e^{\beta X} + \varepsilon$ and $Y = \alpha X^{\beta} + \varepsilon$ are nonlinear in β; non-linear model building is discussed in section 11.5.

For a software system to grow, more code has to be added to it than is deleted. A constant rate of growth suggests either a constant amount of developer effort, or a bottleneck holding things up; an increasing rate of growth (i.e., quadratic) suggests an increasing rate of effort. The different code growth pattern seen in the Linux kernel, compared to NetBSD/FreeBSD and various other applications, has been tracked down[689] to device driver development; new hardware devices often share many interface similarities with existing devices; for Linux developers tend to copy an existing driver, modifying it to handle the hardware differences. It is this reuse of existing code that is the source of what appears to be a non-linear growth in developer effort. This method of creating a new device driver, performed by many developers working independently, can continue for as long as the new devices coming to market have common interfaces.

A linear regression model is not restricted to combining explanatory variables using polynomials, any function can be used as long as the coefficients of the model occur in linear form. For instance, the FreeBSD model plotted in figure 11.2 might include a seasonal term that varies with time of year; while a model containing the term $A\sin(2\pi ft+\phi)$[xiii] is nonlinear (because of ϕ, the phase shift), it can be written in linear form the follows:

$$A\sin(2\pi ft+\phi) = \alpha_s \sin(2\pi ft) + \alpha_c \cos(2\pi ft)$$

where: $\alpha_s = A\cos\phi$ and $\alpha_c = A\sin\phi$; $A = \sqrt{\alpha_s^2+\alpha_c^2}$ and $\phi = \arctan\frac{\alpha_s}{\alpha_c}$.

The call to `glm` is now (the argument to the trig functions is in radians):

```
rad_per_day=(2*pi)/365
freebsd$rad_Number_days=rad_per_day*freebsd$Number_days
season_mod=glm(sloc ~ Number_days+sin(rad_Number_days)+cos(rad_Number_days),
                      data=freebsd)
```

The `summary` output, from the fitted model, shows that while a seasonal component exists, its overall contribution is small; see Github–regression/Herraiz-BSD-season.R.

While fitting a model using all available measurements points is a reasonable first step, subsequent analysis may suggest that the data might best be treated as two or more disjoint

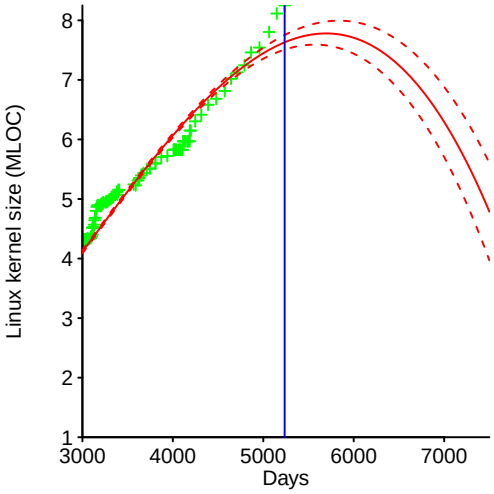


Figure 11.8: Actual (left of vertical line), and predicted (right of vertical line) total lines of code in Linux at a given number of days since the release of version 1.0, derived from a regression model built from fitting a cubic polynomial to the data (dashed lines are 95% confidence bounds). Data from Israeli et al.[902] Github–Local

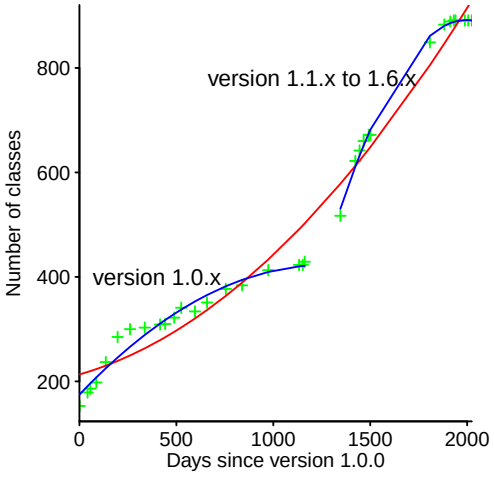


Figure 11.9: Number of classes in the Groovy compiler at each release, in days since version 1.0. Data From Vasa.[1883] Github–Local

[xiii] Some books use $A\cos(2\pi ft+\phi)$, which changes the phase by 90°and flips some signs.

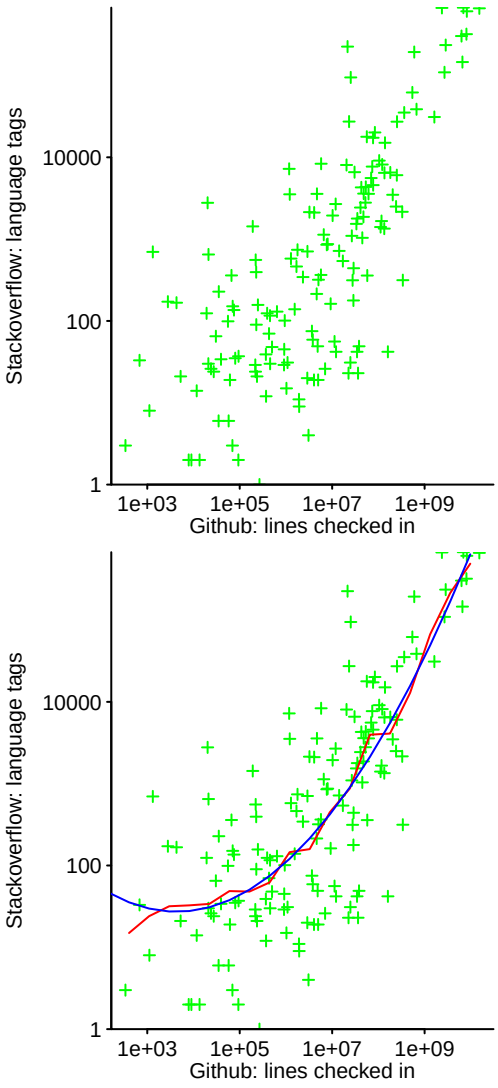


Figure 11.10: For each distinct language, the number of lines committed on Github, and the number of questions tagged with that language. Data from Kunst.[1059] Github–Local

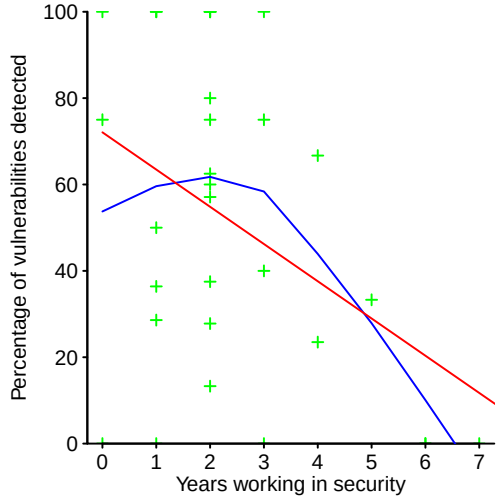


Figure 11.11: Percentage of vulnerabilities detected by developers who have worked a given number of years in security. Data extracted from Edmundson et al.[531] Github–Local

samples. There may be time dependent factors that have a strong influence on growth patterns.

Figure 11.9 shows the number of classes in the Groovy compiler at each release, in days since version 1.0. There are noticeable kinks in the growth rate at around 1,300 and 1,500 days. Fitting a model to the complete sample, shows upward trending quadratic growth in the number of classes over time, but fitting separate models to two halves of the sample, shows quadratic growth that flattens out.

An investigation the Groovy compiler developer, finds that the kink occurs at a transition between version numbers. It is possible to invent a variety of explanations for the pattern of behavior seen, but treating the measurements as-if they came from a single continuously developed code base, is probably not one of them; further investigation of the circumstances behind the development of the Groovy compiler is needed, to obtain the desired level of confidence in one of these, or other, models.

11.2.5 Visualizing the general trend

Even when the measurement points are scattered in what appears to be a general direction, it is little work to confirm this general trend.

A general technique for highlighting the trend followed by data is to fit a regression model to a consecutive sequence of small intervals of the data, joining this sequence of fits together to form a continuous line. Two methods based on this idea (both fitting such that the lines smoothly run together), are LOWESS (LOcally WEighted Scatterplot Smoothing) and LOESS (LOcal regrESSion); `lowess` and `loess` are the respective functions, with `loess` being used in this book.

A study by Kunst[1059] counted, for 148 languages, the number of lines committed to Github (between February 2013 and July 2014), and the number of questions tagged with that language name on Stackoverflow.

Figure 11.10, upper plot, shows lots of points that look as-if they trend along a straight line. The `loess` fit, red line in lower plot, shows the trend having a distinct curve. Experimenting with a quadratic equation in `log(lines_committed)` shows (blue line in lower plot) that this more closely follows the loess fit, than a straight line (a quadratic fit also has a lower AIC than a linear one; see Github–regression/langpop-corger-nl.R).

A call to `loess` has the same pattern as a call to `glm`, with the possible addition of an extra argument; `span` is used to control the degree of smoothing:

```
loess_mod=loess(log(stackoverflow) ~ log_github, data=langpop, span=0.3)
x_points=1:max(langpop$log_github)
loess_pred=predict(loess_mod, newdata=data.frame(log_github=x_points))
lines(exp(x_points), exp(loess_pred), col=pal_col[1])
```

A study by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner[531] investigated the effectiveness of web security code reviews, asking professional developers to locate vulnerabilities in code.

The `lowess` fit, blue line in figure 11.11, suggests that the percentage of vulnerabilities found increases as the number of years working in security increases, but then rapidly decreases. This performance profile seems unrealistic. A fitted straight line, in red, shows a decreasing percentage with years of work in the security field (its p-value is 0.02).

Perhaps the correct interpretation of this data, is that average performance does increase with years worked in the field, but that the subjects with many years working in security, who took part in the study, were more managerial and customer oriented people (who had time available to take part in the experiment), i.e., this data contains sampling bias. At the time of the study, software security work was rapidly expanding, so the experience profile is likely to be skewed with more subjects being less experienced.

When it is not necessary to transform either argument, the value returned by the `loess.smooth` function can be passed directly to `lines`.

```
lines(loess.smooth(dev$experience, dev$written, span=0.5), col=pal_col[2])
```

A `loess` visualization can also helpful when the number of data points is so large, they coalesce into formless blobs. The Ultimate Debian Database is an example; see fig 11.23.

The `loess` function divides the range of x-axis values into fixed intervals, which means that when the range of x-values varies by orders of magnitude, the fitted curve can appear over stretched at the low values and compressed at high values.

One solution is to reduce the range of x-values by, for instance, taking the log, smoothing, and then expanding (see Github–regression/java-api-size.R); the following code is used in figure 11.32:

```
t=loess.smooth(log(API$Size), API$APIs, span=0.3)
lines(exp(t$x), t$y, col=loess_col)
```

11.2.6 Influential observations and Outliers

Influential observations are observations that have a disproportionate impact on the values of a model's fitted coefficients, e.g., a single observation significantly changes the slope of a fitted straight line. The terms *leverage* (or, based on the mathematical symbol used, *hat-value*) refer to the amount of influence a data point has on a fitted model; the `hatvalues` function takes the model returned by `glm` and returns the leverage of each point.

Influential observations might be removed or modified, or a regression technique used that reduces the weight given to what are otherwise overly influential points, e.g., the `glmrob` function in the `robustbase` package (which is not always as robust as desired and manual help may be required; see Github–regression/a174454-reg.R).

Outliers are discussed as a general issue in section 14, this subsection discusses outliers in the context of regression modeling. In this context an outlier might be defined as a data point having a disproportionately large standardized residual (here Studentized residuals are used). To repeat an important point made in that chapter: excluding any influential observations or outliers from the analysis is an important decision that needs to be documented in the results.

Cook's distance (also known as *Cook's D*) is a commonly used metric, which combines leverage and outlierness into a single number.

A study by Fenton, Neil, Marsh, Hearty, Radliński and Krause[593] involved data from 31 software systems for embedded consumer products. Figure 11.12 shows development effort against the number of lines of code, along with a fitted straight line and standard error bounds. At the right edge of the plot are two projects that consumed over 50,000 hours of effort, and the number of lines of code for these projects looks very small in comparison with other projects. Is the fitted model overly influenced by these two projects and should they be ignored or adjusted in some way?

As the number of points in a sample grows, there is an increasing probability that one or more of them will be some distance away from the fitted line; in any large sample a few apparent outliers are to be expected as a natural consequence of the distribution of the error. The following analysis illustrates the dangers of not taking sample size into account, when making judgements about the outlier status of a measurement point.

Figure 11.13 shows the result of building a model, after removing measurements having both a high Cook's distance and Studentized residuals, and repeating the process until points stop being removed. At the end of the process, most measurement points have been removed.

Removing overly influential points until everything looks respectable is seductive, it is an easy-to-follow process that does not require much thought about the story that the data might have to tell. For those who don't want to think about their data, the `outlierTest` function, in the `car` package, can be used to automate outlier detection and removal (it takes a model returned by `glm` and returns the Studentized residuals of points whose Bonferroni corrected p-value is below a cutoff threshold; default `cutoff=0.05`).

A method of visualizing the important influential observation and outlier information is required. The `influenceIndexPlot` function, in the `car` package, takes the model returned by `glm` and plots the Cook's distance, Studentized residual, Bonferroni corrected p-value and hat-value for each data-point; figure 11.14 is for the Fenton et al data.

```
all_mod=glm(KLoC ~ I(Hours^0.5), data=loc_hour)
influenceIndexPlot(all_mod, main="", col=point_col, cex.axis=0.9, cex.lab=1.0)
```

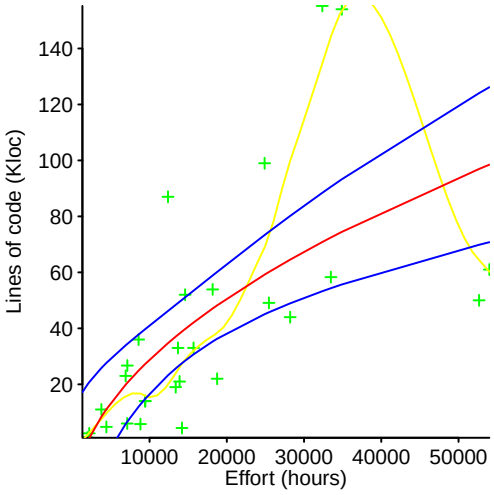


Figure 11.12: Hours to develop software for 29 embedded consumer products, and the amount of code they contain, with fitted regression model and loess fit (yellow). Data from Fenton el al.[593] Github–Local

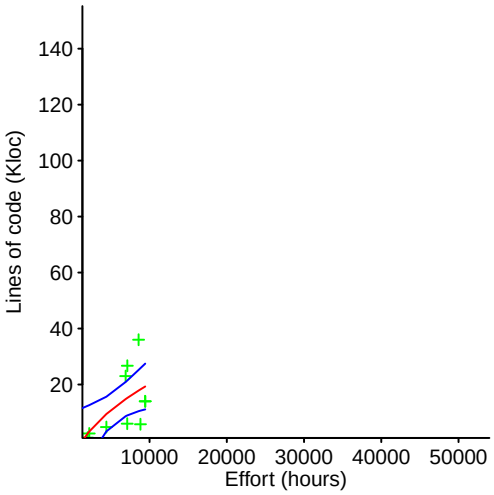


Figure 11.13: Points remaining after removal of overly influential observations, repeatedly applying Cook's distance and Studentized residuals. Data from Fenton el al.[593] Github–Local

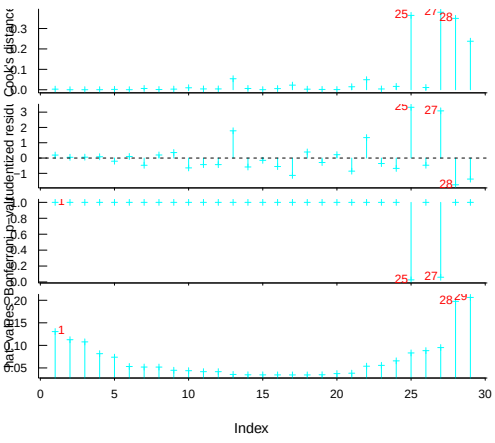


Figure 11.14: `influenceIndexPlot` for the model having the fitted line shown in figure 11.12; top three data points highlighted. Data from Fenton el al.[593] Github–Local

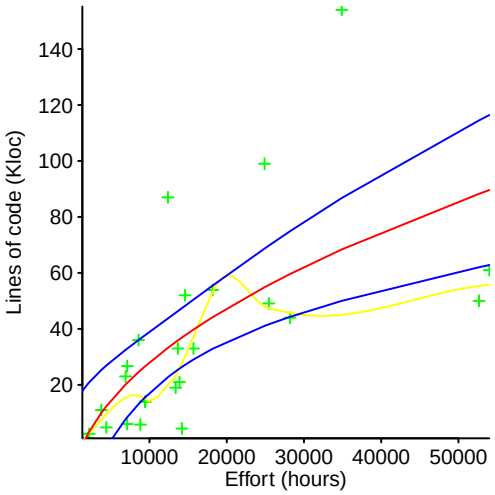


Figure 11.15: Points remaining after removal of overly influential observations, also taking into account the Bonferroni p-value of the Studentized residuals; the line shows the fitted model and 95% confidence interval (loess fit in yellow). Data from Fenton el al.[593] Github–Local

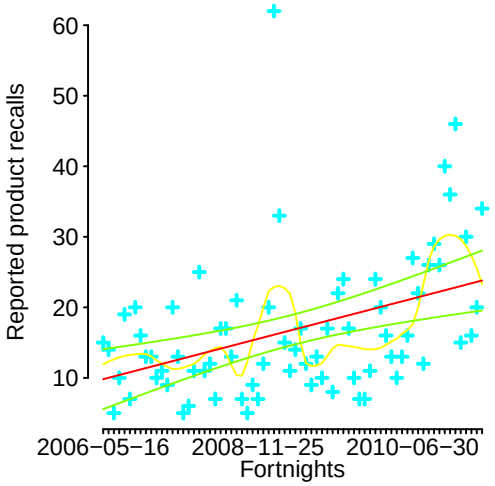


Figure 11.16: Number of medical devices reported recalled by the US Food and Drug Administration, in two week bins; fitted straight line and confidence bounds, with loess fit (yellow). Data from Alemzadeh et al.[30] Github–Local

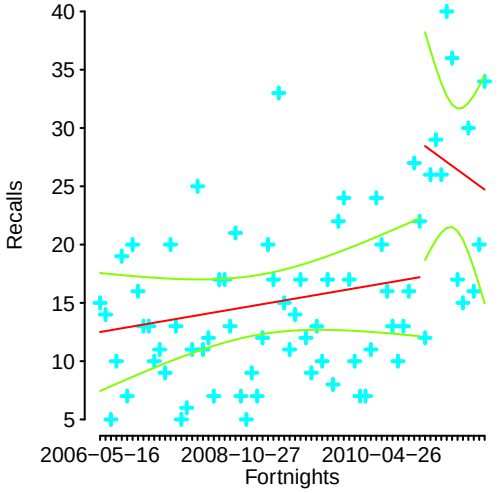


Figure 11.17: Two fitted straight lines and confidence intervals, one up to the end of 2010 and one after 2010. Data from Alemzadeh et al.[30] Github–Local

The upper plot shows four data points having a large Cook's distance, but only two of them have a significant corrected p-value (second plot from the bottom). These two data points were removed, and the process of building a model and calling `influenceIndexPlot` repeated; on this iteration one point is removed and iterating again finds no other data points as worthwhile candidates for removal.

Figure 11.15 shows the results of removing data points having both a high Cook's distance, and Studentized residuals whose corrected p-value is below the specified limit.

Outliers are loners, appearing randomly scattered within a plot. When multiple points appear to be following a different pattern than the rest of the data, the reason for this may be a different process driving behavior, or a change of behavior in what went before.

A study by Alemzadeh, Iyer, Kalbarczyk and Raman[30] investigated safety-critical computer failures in medical devices between 2006 and 2011 (as reported by the US Food and Drug Administration). Figure 11.16 shows the number of devices recalled for computer related problems (20-30% of all recalls), binned by two-week intervals.

Data points that stand out in figure 11.16 are the two large recall rates in the middle of the measurement interval, and recall rates at later dates appearing to increase faster than earlier; adding a loess fit (yellow) shows peaks around the two suspicious periods. The fitted straight line shows a distinct upward trend. Is this fitted line being overly influenced by the two middle period points or end of measurement period recall rates?

The measurement points appear in regular time slots, and deleting one of these time slots does not make sense; replacing an outlier with the mean of all measurements is one solution for handling this situation. Doing this (see Github–regression/Alemzadeh-Recalls.R) finds there is little change in the fitted regression model, i.e., these two outliers had little influence. Did a substantive change in the processes driving recalls, or recording of recalls, occur around the start of 2011? Further investigation, or domain knowledge, is needed to answer this question.

Figure 11.17 shows two fitted models, one using data up until the end of 2010 and the other using the data after 2010. This illustrates that blindly fitting a straight line to a sample can produce a misleading model. A change in reporting appears to have occurred around the end of 2010, which had a significant impact on reported recalls (work is needed to uncover the reason for this change) and fitting data up to the end of 2010 shows a much smaller increase, and perhaps even no increase, in recall rates, compared to when measurements after this date are included.

A change-point analysis of this data is discussed in section 11.2.9.

When combining results from multiple studies, it is possible for an entire study to be an outlier, relative to the other related studies.

A study by Amiri and Padmanabhuni[52] analysed the methods used by eleven other studies to convert between two common methods of counting function points.[xiv] Many studies included in the analysis have small sample sizes, include both student and commercial projects, and the function points are sometimes counted by academics rather than industrial developers.

Figure 11.18 shows function points counted using the COSMIC and FPA algorithms (counts made by students have been excluded). Both lines are loess fits, with red used for industry points and blue for academic researchers; the academic line overlays the industry line if one sample (i.e., Cuadtado_2007) is excluded.

The impact of influential observations on a fitted model can vary enormously, depending on the form on the equation being fitted. Figure 11.19 shows the lines of five separate equations fitted to the Embedded subset of the COCOMO 81[215] data, with the upper plot using the original data, and the lower plot the data after three influential observations have been removed.

In some cases outlier removal has had little impact on the model fitted, while in other cases there has been dramatic changes in the coefficients of the fitted model.

11.2.7 Diagnosing problems in a regression model

The commonly used regression modeling functions are capable of fitting a model to almost any sample, without reporting an error (some functions are so user-friendly they

[xiv] Function point counting is a technique for estimating development effort by counting the functionality contained in the software requirements specification.

gracefully handle data that produces a singular matrix, an error that is traditionally flagged, because it suggests that something somewhere is wrong). It is the analysts' responsibility to diagnose any problems in the model returned.

Looking at figure 11.20, it is visually obvious that at least two of the fitted regression lines completely fail to capture the pattern present in the data. The data set is famous, it is known as the Anscombe quartet.[64] The four samples each contain two variables, with each sample having the same mean, standard deviation, Pearson correlation coefficient and are fitted using linear regression to produce a straight line having the same slope and intercept.

Problems with a regression model are not always as obvious as the Anscombe quartet case, and diagnosing the cause of the problem can be difficult. As always, domain knowledge is very useful for suggesting alternative models or possible changes to a fitted model.

The difference between the measured value of the response variable, and the value predicted by a fitted model is known as the *residual*. While many model diagnosis techniques are based on the use of the residual, they often require more knowledge of the mathematics of regression modeling than is covered in this book.[xv]

The suggested model diagnostic techniques, for casual users of statistics, are visualization based.

Figure 11.21, upper plot, shows the residual of the straight line fitted to the Linux kernel growth data analysed in figure 11.7. Ideally the residual is randomly scattered around zero, and the V-shape seen in this plot is typical of a straight line fitted to values that curve around it (the smallest residual is in the center, where the model fits best, and is greatest at the edges; the smaller peak is a localised change of behavior, and may explain why a cubic produces a slightly better fit). This plot is one of the four diagnostic visualizations produced by `plot`, when it is passed a regression model, as follows:

```
m1=glm(LOC ~ Number_days, data=latest_version)
plot(m1, which=1, caption="", col=point_col)
```

Figure 11.21, lower plot, shows the original data, straight line fit (red) and loess fit (blue). Both the residual plot and loess fit express the same pattern of curvature around about the straight line fit. Both visualizations have their advantage, the loess line can be drawn before any model is fitted, while details are easier to extract, from a residual plot, e.g., values for the size of the difference.

The mathematics behind linear regression requires that each measurement be independent of all the other measurements in a sample. A common form of dependence between measurements is serial correlation, i.e., correlation between successive measurements. A fitted regression model can be tested for serial correlation using the Durbin Watson test; supported by the `durbinWatsonTest` function in the `car` package.

A study by Flater and Guthrie[614] measured the time taken to assign a value to an array element in C and C++, using twelve different techniques, some of which checked that the assignment was within the defined bounds of the array (two array sizes were used, large and small); the programs benchmarked were compiled using seven different compiler optimization options.

Figure 11.22 shows the timings from 2,000 executions of one technique for assigning to an array element, compiled using gcc with the 00 option (upper) and 03 option (lower). The results for 00 show a clustering of execution times for groups of successive measurements.

A Durbin Watson test confirms that the 00 measurements are serially correlated; see Github–benchmark/array-durbanwatson.R.

Some regression modeling functions can adjust for the presence of serial correlation (information about the correlation is passed in an optional argument). The `gls` function, in the `nlme` package, supports a `correlation` option; the `dynlm` package supports the use of time series operators (e.g., diff and lag) in the specification of model formula; the `tscount` package supports the fitting of generalized linear models to time series of count data.

When measurements contain a significant amount of serial correlation, time-series analysis techniques may provide useful information; see section 11.10. The `tsglm` function, in the `tscount` package, supports regression modeling of count time series.

[xv] It is not obvious that the cost/benefit of learning the necessary mathematics is worthwhile (but it is a good source of homework exercises for students).

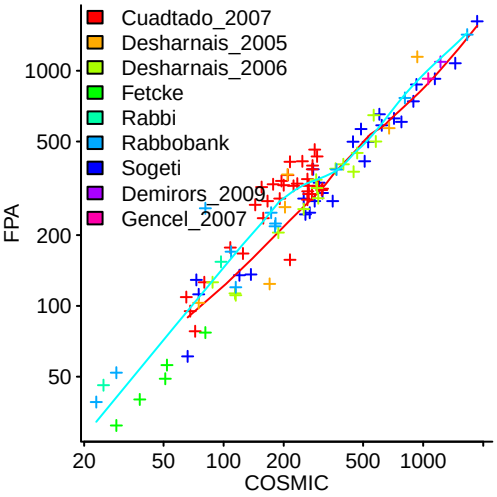
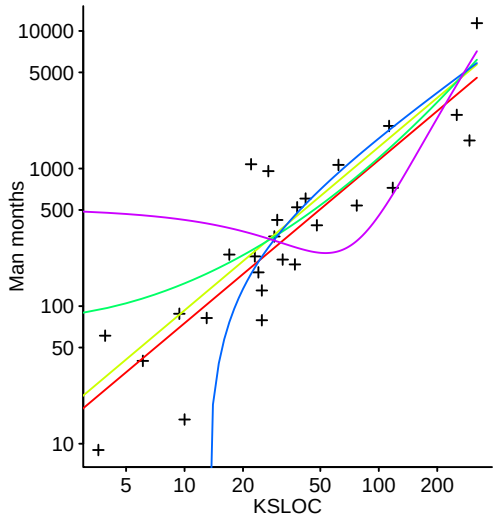


Figure 11.18: Results from various studies of software requirements function points counted using COSMIC and FPA; lines are loess fits to studies based on industry and academic counters. Data from Amiri et al.[52] Github–Local



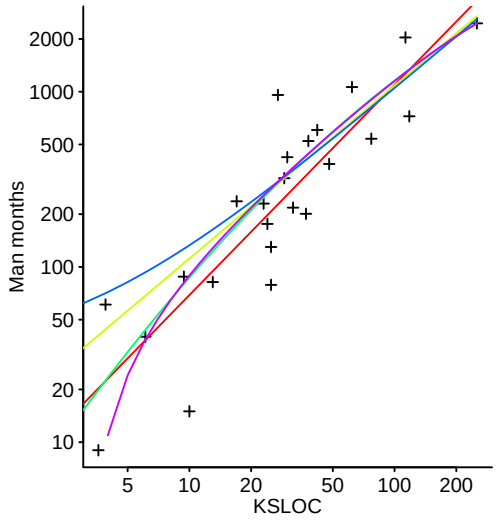


Figure 11.19: Five different equations fitted to the Embedded subset of the COCOMO 81 data before influential observation removal (upper) and after influential observation removal (lower). Data from Boehm.[215] Github–Local

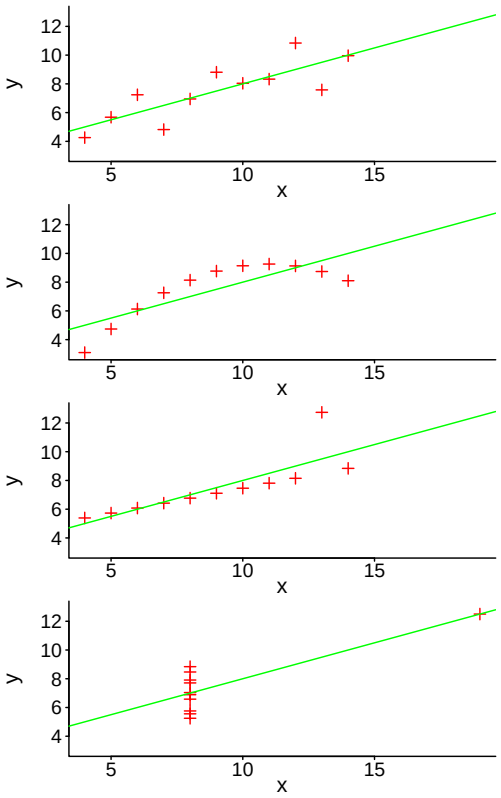


Figure 11.20: Anscombe data sets with Pearson correlation coefficient, mean, standard deviation, and line fitted using linear regression. Data from Anscombe.[64] Github–Local

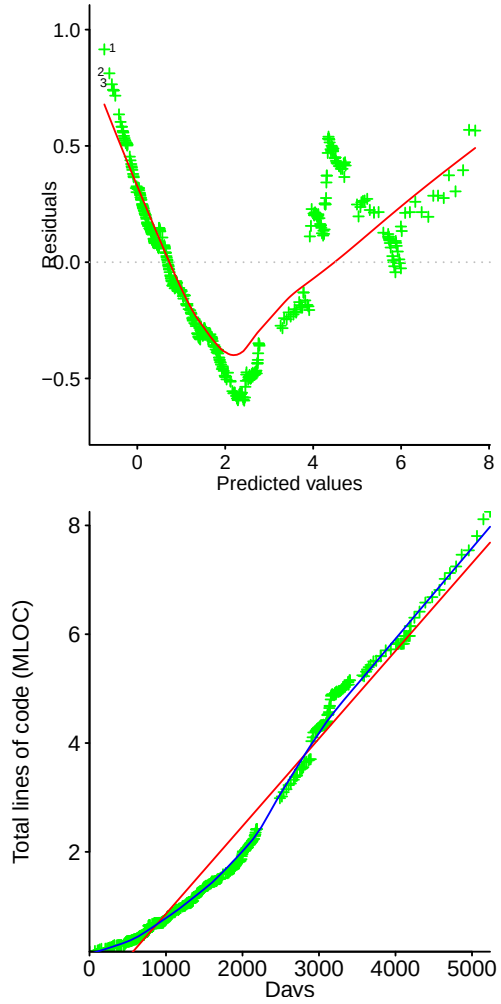


Figure 11.21: Residual of the straight line fit to the Linux growth data analysed in figure 11.7. Data from Israeli et al.[902] Github–Local

11.2.8 A model's goodness of fit

How well does a model fit the data? The term *goodness of fit* is often used to describe this quantity. Various formula for calculating a goodness of fit have been proposed, and often involve the difference between the value measured and the corresponding value predicted by the model.

For the end-user of results of the analysis, meeting expectations of behavior is an important model characteristic.

When fitting equations to gain understanding, the structure of the processes suggested by the terms of the equation are an important characteristic.

When making predictions, the primary quantity of interest is the accuracy of new predictions, i.e., the amount of expected error in predictions for values that are not in the sample used to build the model. The error structure is also a consideration; is the priority to minimise total error, worse case error, to prefer over-estimates to under-estimates (or vice versa) or does some complicated weighting (over the range of values that explanatory variable(s) might take) have to be taken into account?

When dealing with one explanatory variable, it is possible to get a good idea of how well a model fits the data through visualization, e.g., by plotting them both. Does the fitted line look correct and how wide are the confidence intervals? However, for data containing more than one explanatory variable, accurate visualizations are problematic.

To create a model by fitting it to data, is to create a just so story. The predictions made by a model, outside the range of the data used to build it, are just something to discuss when considering expectations of behavior (which might be derived from a theory of the processes involved in generating the data used to fit the model).

Confidence intervals, see fig 11.3, provide information about the goodness of fit at every point. The following discussion looks at some ways of producing a single numeric value, to represent goodness of fit.

The leftover variation in a sample that is not accounted for by the fitted model, the residual, is invariably a component in any equation in the calculation of a single value to summarise how well the model performs. Some of the metrics used include:

- `null deviance` is a measure of the difference between the data and the mean of the data, `deviance` is a measure of the difference between the data and a fitted model (both values are listed in the `summary` output of a model fitted by `glm`). The percentage difference between the deviance and null deviance is a measure of the variance in the data that is not explained by the mode,
- R-squared (also known as the *coefficient of determination* and commonly written R^2) can be interpreted as the amount of variance in the data (as measured by the residuals) that is explained by a model. It takes values between zero and one (which has the advantage of being scale invariant) and is a measure of correlation, not accuracy.

  Sometimes the adjusted R^2, written $\bar{R}^2$, is used, which takes into account the number of explanatory variables, p, and sample size, n: $\bar{R}^2 = R^2 - (1 - R^2)\frac{p-1}{n-p}$
- mean squared error (MSE): the mean squared error is the mean value of the square of the residuals and as such has no upper bound (and will be heavily influenced by outliers); root mean squared error (RMSE) is the square-root of MSE.

  The following equation shows how MSE and R^2 are related: $R^2 = 1 - \frac{MSE}{\sigma^2}$
- mean absolute error (MAE): the mean absolute error is the mean value of the absolute value of the residuals. This measure is more robust in the presence of outliers than MSE.

Apart from the R^2 metric, the metrics listed (plus AIC) are scale dependent, e.g., mapping measurements from centimeters to inches changes their value; transforming the scale (e.g., taking logs) will also change metric values.

The choice of a metric is driven by what information is available and what model characteristics are considered important, e.g., how important is being able to handle outlier. In a competitive situation, people might not be willing to reveal details about their model and so any public metric has to be based on predictive accuracy, e.g., model builders provide the predictions made by their model to a test data set.

R^2 is the only scale invariant metric, and it provides an indication of how much improvement might be possible over an existing model.

It is possible for the coefficients of a fitted model to be known with a high degree of accuracy, and yet for this model to explain very little of the variance present in the data, and for there to appear to be little chance of improving on the model given the available data.

The Ultimate Debian Database project[1862] collects information about packages included in the Debian Linux distribution. Figure 11.23 shows the age of a packaged application plotted against the number of systems on which that application is installed, for 14,565 applications in the "wheezy" version of Debian; also, see fig 6.4 and fig 8.18.

The fitted linear model (red line, hidden by the 95% confidence interval in green overwriting it; loess fit in blue) has a very low p-value, a consequence of the large number of, and uniform distribution of, data points. The predictive accuracy of this model is almost non-existent, the only information it contains is that older packages are a little more likely to be installed that younger ones.

A study by Jørgensen and Sjøberg[958] investigated developers' ability to predict whether any major unexpected problems would occur during a software maintenance task. Building a regression model, using the available measured attributes, finds that lines of code is the only explanatory variable having a p-value less than 0.05. However, only 3.3% of the variance in the response variable is explained by the number of lines of code; while the explanatory variable was statistically significant, its practical significance was negligible; see Github–maintenance/10.1.1.37.38.R.

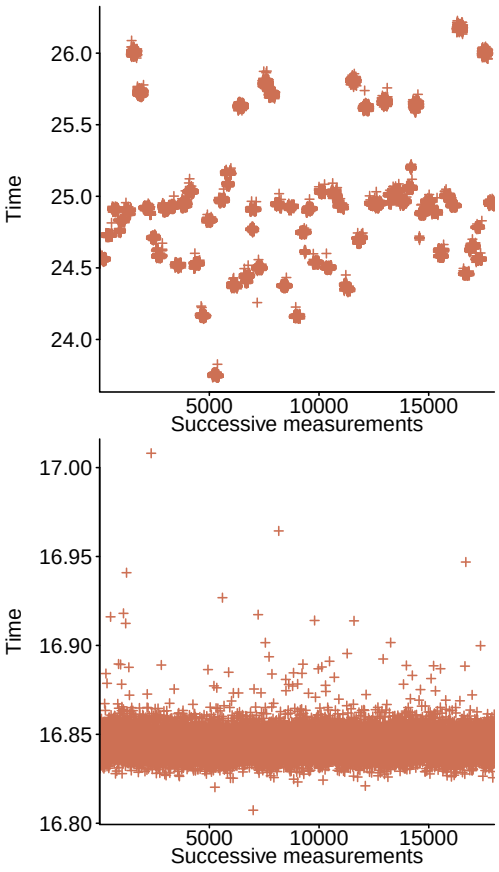


Figure 11.22: Array element assignment benchmark compiled with gcc using the 00 (upper) and 03 (lower) options (measurements were grouped into runs of 2,000 executions). Data from Flater et al.[614] Github–Local

11.2.9 Abrupt changes in a sequence of values

When the processes generating the measured values change, the statistical properties of the post-change sequence of values may abruptly change. The point where the statistical properties of a sequence of values significantly changes is known as a *change-point*.

The `changepoint` package supports basic change-point analysis of the mean and variance of a sequence of values. The `cpt.mean` function checks for significant shifts in the mean value; the `method="AMOC"` (At Most One Change) option searches for what its name implies; other values support searching for a specified maximum number of changes, with `method="PELT"` selecting what is considered to be the optimum number of changes.

An earlier analysis of electronic device recalls (see fig 11.17) suggested that a significant shift in the processes driving reported recalls occurred at the end of 2010. Figure 11.24 shows the output from the following calls to `cpt.man`:

```
library("changepoint")

change_at=cpt.mean(as.vector(t2))
plot(change_at, col=point_col,
        xlab="", ylab="Reported product recalls\n")

change_at=cpt.mean(as.vector(t2), method="PELT")
plot(change_at, col=point_col,
        xlab="Fortnights", ylab="Reported product recalls\n")
```

For an example of detecting changes in variance and changes in both mean and variance, see Github–regression/hpc-read-write.R.

The `segmented` function, in the `segmented` package, adjusts a fitted regression model to take account of change-points; at the time of writing this function only fits connected line segments, i.e., no disjoint line boundaries, such as that present in figure 11.17. A model is fitted using `glm` is passed to `segmented`, which attempts to estimate the appropriate change points and fit a series of line segments between each change-point (the number of change-points can be explicitly specified). Figure 11.25 shows the output from the following code (also see fig 10.20):

```
library("segmented")

plot(t2$fortnight, t2$freq, type="l", col=pal_col[2], xaxs="i",
        xlab="Fortnights", ylab="Recalls\n")

al_mod=glm(freq ~ fortnight, data=t2) # fit model as usual
```

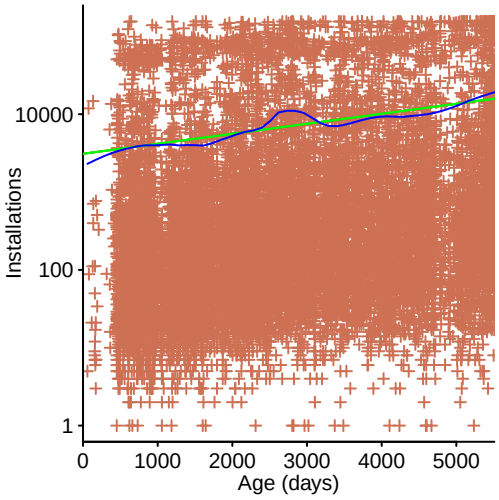


Figure 11.23: Number of installations of Debian packages against the age of the package, plus fitted model and loess fit. Data from the "wheezy" version of the Ultimate Debian Database project.[1862] Github–Local

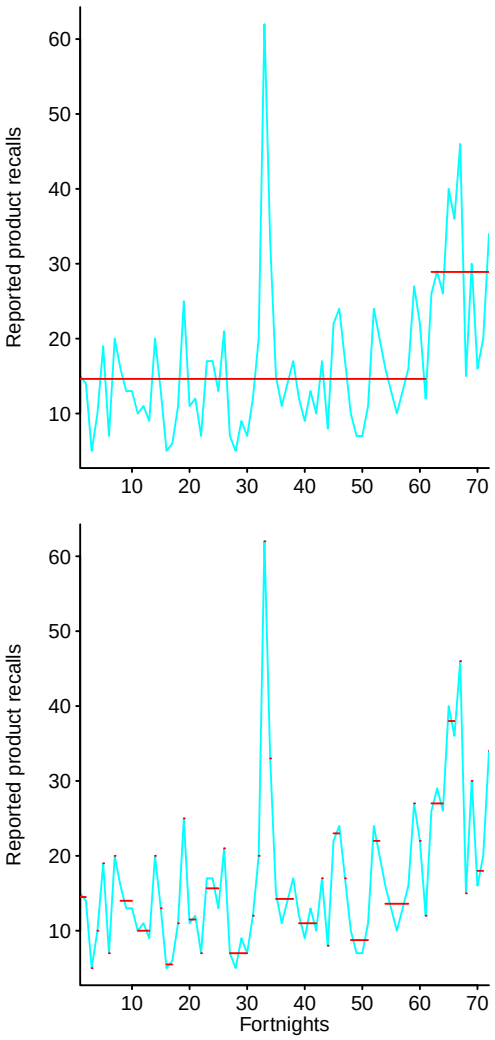


Figure 11.24: Change-points detected by `cpt.mean`, upper using `method="AMOC"` and lower using `method="PELT"`. Data from Alemzadeh et al.[30] Github–Local

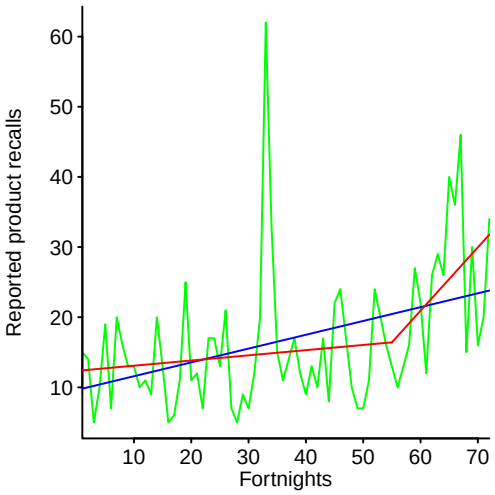


Figure 11.25: Fitted regression model (blue) and adjusted model with one change-point (red). Data from Alemzadeh et al.[30] Github–Local

```
pred=predict(al_mod)
lines(pred, col=pal_col[3]) # add fitted line to plot

seg_mod=segmented(al_mod, npsi=1) # adjust fitted model with one change-point

plot(seg_mod, col=pal_col[1], add=TRUE) # add fitted lines to plot
```

When the location of the change-point is known, or the `segmented` function fails to find a reasonable fit, an abrupt change can be modeled using `glm` to effectively fit multiple equations; one equation over each discontinuity separated interval. While each equation may be fitted by an independent call to `glm`, it may be possible to build a single model incorporating every discontinuity.

Figure 11.26 shows an abrupt change in the sales volume of 4-bit microprocessors (green). Straight lines have been fitted to the two periods before/after April 1998 (red), with the yearly sales cycle modeled by a sine wave (blue).

The technique for fitting a model that handles discontinuous patterns of behavior makes use of an interaction between the explanatory variable (`date` in this case), and a dummy variable whose 0/1 value depends on `date`, relative to the change-point. The code for the straight line model (red line) is:

```
y_1998=as.Date("01-04-1998", format="%d-%m-%Y")  # estimated discontinuity point

p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998), data=proc_sales)
```

and the `summary` output is: Github–Local

```
Call:
glm(formula = bit.4 ~ date * (date < y_1998) + date * (date >=
    y_1998), data = proc_sales)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-19756.9   -6372.8    -558.7    6533.2   19086.4

Coefficients: (2 not defined because of singularities)
                           Estimate Std. Error t value Pr(>|t|)
(Intercept)               7.050e+04  5.802e+04   1.215   0.2265
date                      7.072e-01  5.368e+00   0.132   0.8954
date < y_1998TRUE        -8.357e+04  5.873e+04  -1.423   0.1572
date >= y_1998TRUE               NA         NA      NA       NA
date:date < y_1998TRUE    1.045e+01  5.466e+00   1.912   0.0581 .
date:date >= y_1998TRUE          NA         NA      NA       NA
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 80003408)

    Null deviance: 2.0763e+10  on 131  degrees of freedom
Residual deviance: 1.0240e+10  on 128  degrees of freedom
AIC: 2782.6

Number of Fisher Scoring iterations: 2
```

Sales follow a seasonal trend that can be approximated using a sine wave having a 12-month frequency, adding this to the straight line model as follows:

```
season_p4=glm(bit.4 ~ date*(date < y_1998)+date*(date >= y_1998)+
                                sin(rad_days)+cos(rad_days), data=proc_sales)
```

11.2.10 Low signal-to-noise ratio

Measurements sometimes contain a large amount of noise, relative to the signal present, i.e., a low signal-to-noise ratio. Fitting a model to such data can be difficult, because many equations do an equally (not very) good job.

The two plots along the upper row in figure 11.27 show data generated from a quadratic equation containing noise, along with two fitted models (red and blue lines). The equation used to generate the two sets of data is:

$y = x^2 + K \times (5 + rnorm(length(x)))$

where: $K = 10^3$ (left column), and $K = 10^2$ (right column).

It is not possible to tell by looking at the upper left plot whether a quadratic (blue), or an exponential (red), is a better fit; the output from `summary` is not much help; see Github–regression/noisey-data.R. The upper right plot contains less noise, and it is easier to see that the exponential fit does not follow the data as well as the quadratic.

Sometimes the peaks (or troughs) in the plotted data can be an indicator of the shape of the data. The upper left plot includes a quadratic and exponential fit to the three largest values at each x-value (the fitted model does not seem to have less the uncertainty in this case).

The *ratio test* is a technique that can help rule out some equations as possible candidates for modeling. If $f(x)$ is the function being fitted to the data and this data was generated by the function $g(x)$, the ratio $\frac{g(x)}{f(x)}$ will converge to a constant as x becomes small/large enough such that the signal dominates the noise.

The two plots along the lower row in figure 11.27, show ratio tests for quadratic (blue), cubic (red) and exponential (green) equations. The exponential equation shows no sign of converging to a constant, while quadratic is closer to doing this than cubic (which can be ruled out because it does a poor job of fitting the data).

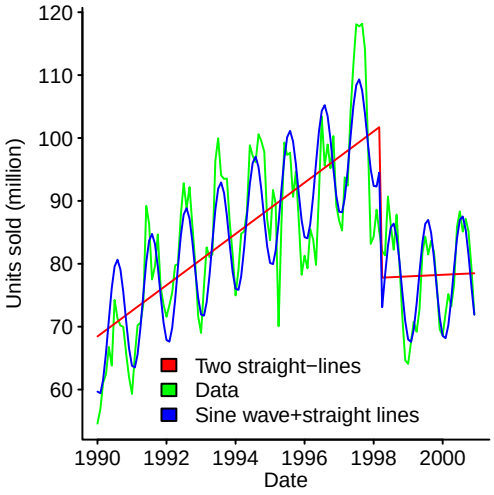


Figure 11.26: Monthly unit sales (in millions) of 4-bit microprocessors. Data kindly supplied by Turley.[1854] Github–Local

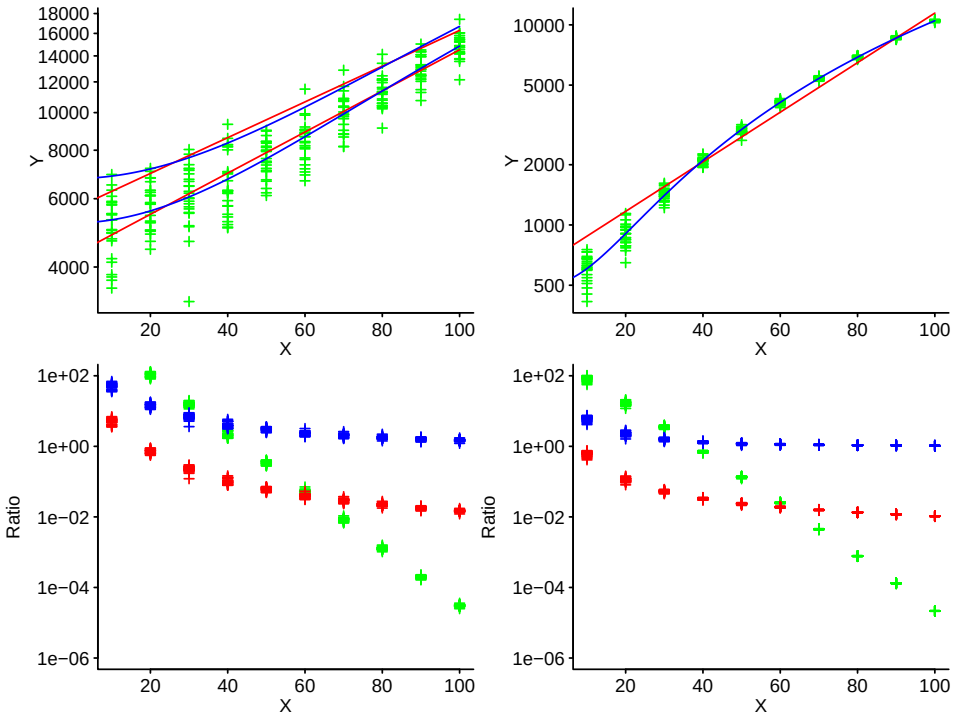


Figure 11.27: Quadratic relationship with various amounts of added noise, fitted using a quadratic and exponential model. Github–Local

The ratio test rules out an exponential equation being a good candidate for fitting a model to the data in figure 11.27.

A study by Vasilescu, Serebrenik, Goeminne and Mens[1884] investigated contributions to the Gnome ecosystem, from the point of view of workload (measured as the number of file touches, e.g., commits), breaking it down by projects, authors and number of activity types, e.g., coding, testing, documentation, etc.

Figure 11.28, upper plot, shows, for individual authors, workload and the number of activity types they engaged in. There is a large amount of noise in the data (or variance not explained by the explanatory variable used for the x-axis). Figure 11.28, lower plot, shows a ratio test, with an exponential failing to level off, the linear equation slowly growing, and the quadratic looking like it is trying to grow.

Perhaps the behavior would become clearer with more activity types, but the quadratic is the only candidate not ruled out.

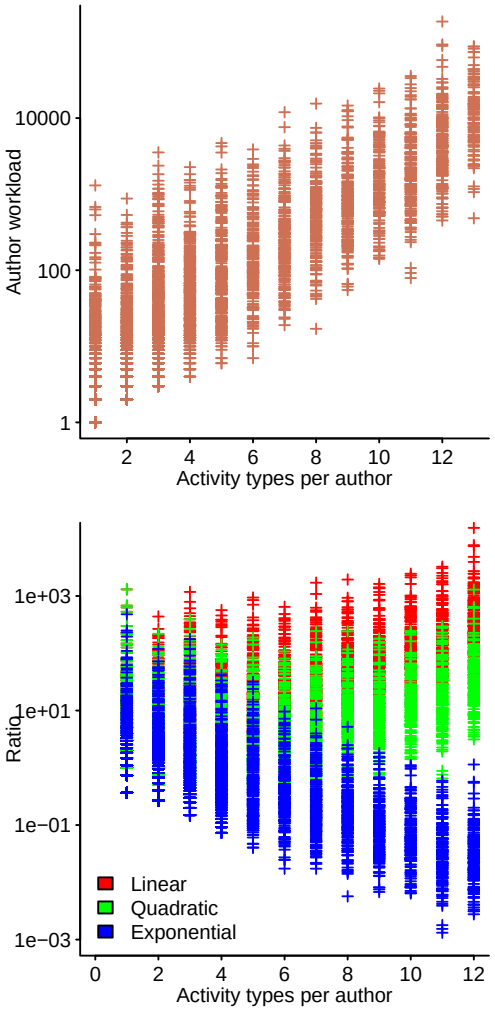


Figure 11.28: Author workload against number of activity types per author (upper) and ratio test (lower). Data from Vasilescu et al.[1884] Github–Local

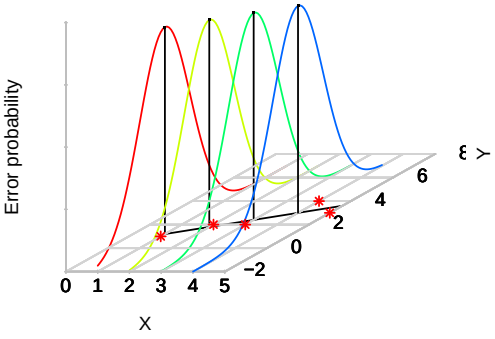


Figure 11.29: Fitted regression line to points (in red) and 3-D representation of assumed Normal distribution for measurement error. Github–Local

11.3 Moving beyond the default Normal error

Measurements sometimes have properties that do not meet the requirements assumed by the mathematics on which `glm`s default argument values are based. Some measurement properties that non-default argument values can handle, include the response variable having values that:

- can never go below zero, e.g., count data,
- can never be greater than some maximum value, e.g., some percentages can never be greater than 100,
- span several orders of magnitude and contain an additive error.

By default, `glm` uses a Normal distribution for the measurement error. Figure 11.29 shows a fitted regression line with four data points (red stars adjacent to a black line); the colored Normal curves over each point represents the probability distribution of the measurement error that is assumed to have occurred for that measurement (the center of each error distribution curve is directly above the fitted line at each explanatory variable measurement point).

`glm`'s `family` argument has the default value `family=gaussian(link="identity")`, which can be shortened to `family=gaussian` (the default link function for `gaussian` is `link="identity"`).

The Normal distribution includes negative values and when a measurement cannot have a negative value, using an error distribution that includes negative values can distort the fitted model. One alternative is the Poisson distribution, which is zero for all negative values. The following call to `glm` specifies that the measurement error has a Poisson distribution:

```
a_model=glm(a_count ~ x_measure, data=some_data, family=poisson)
```

After Normal, the Poisson and Beta distributions are the most common measurement error distributions used by the analysis in this book.

Calling `glm`, with a non-default value for the `family` argument, requires knowing something about the mathematics behind generalised regression model building. The equation actually being fitted by `glm` is:

$$l(y+\varepsilon) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

which differs from the one given at the start of the chapter in having $l(y+\varepsilon)$ on the left-hand-side, rather than y. This l is known as the *link function*, which for the Normal distribution is the identity function (this leaves its argument unmodified, and the equation ends up looking like the one given at the start of this chapter).

Once a regression model is fitted, the value of the response variable is calculated from:

$$y = l^{-1}(\alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots +) - \varepsilon$$

where: l^{-1} is the inverse of the link function used, e.g., the inverse of log is e raised to the appropriate power.

Every error distribution has what is known as a *canonical link* function, which is the function that pops out of the mathematical analysis for that distribution. By default, `glm` uses the canonical link function for each error distribution, and allows some alternatives to be specified. The canonical link function for the Poisson distribution is `log`.

When the link function is not `identity`, prediction values and confidence intervals need to be mapped as follows:

```
a_pred=predict(a_model, se.fit=TRUE)
inv_link=family(a_model)$linkinv      # get the inverse link function

lines(x_values, inv_link(a_pred$fit)) # fitted line
# confidence interval above and below
lines(x_values, inv_link(a_pred$fit+1.96*a_pred$se.fit))
lines(x_values, inv_link(a_pred$fit-1.96*a_pred$se.fit))
```

The analysis in the following sections involve measurements that require the use of a variety of measurement error distributions and link functions.

11.3.1 Count data

Count data has two defining characteristics, it is discrete and has a lower bound of zero. The discrete distribution taking on non-negative values, supported by `glm`, is the Poisson distribution.

In practice, when measurement values are sufficiently far away from zero (where far may be more than 10) there is little difference between models fitted using the Normal and Poisson distributions. For measurements close to zero, the main difference between models fitted using different distributions is the confidence intervals (which are usually not symmetric, and may be larger/smaller).

The canonical link function for the Poisson distribution is `log`, and the following two calls to `glm` are equivalent:

```
p_mod=glm(y ~ x, data=sample, family=poisson)
p_mod=glm(y ~ x, data=sample, family=poisson(link="log"))
```

The `log` link function means that the equation being fitted is actually:

$$y = e^{\alpha+\beta x} + \varepsilon$$

To fit the equation: $y = \alpha + \beta x + \varepsilon$, for a Poisson error distribution, the `identity` link function has to be used, as follows (experience shows that `glm` sometimes fails to converge when `family=poisson(link="identity")` is specified and that `start` values have to be specified):

```
p_mod=glm(y ~ x, data=sample, family=poisson(link="identity"))
```

A study of the effectiveness of security code reviews by Edmundson, Holtkamp, Rivera, Finifter, Mettler and Wagner[531] asked professional developers, with web security review experience, to locate vulnerabilities in web code. The number of vulnerabilities found can only be a non-negative integer value and in this study were single digit values.

The values fitted to a discrete distribution consists of a series of discrete steps, as the upper plot of figure 11.30 shows (fitted line and 95% confidence intervals). While this plot is technically correct, it is ambiguous: are the values specified by the top left edge, or the bottom right edge of the staircase?[xvi] Plots using continuous lines are simpler for readers to interpret and so are used in this book.

The dashed lines in figure 11.30, lower plot, were fitted using `glm`'s default values,[xvii] while the argument `family=poisson(link="identity")` was used to fit the model represented by the smooth lines.

The two fitted lines are virtually identical (the green dashed line is drawn over the continuous red line), but the 95% confidence intervals do differ. This pattern of behavior is very common, unless the response variable has many values near zero.

Is the difference between fitting a model using the technically correct Poisson distribution, or a Normal distribution, worth the effort (for the analyst, not the use of any additional computing resources)?

Sometimes the Poisson distribution is used because a `log` link function transforms the response variable, while keeping an additive measurement error.

When fitting models containing multiple explanatory variables (discussed later) and a response variable containing count data, it can be more difficult to detect differences between using a Poisson and Normal distribution. While use of the Poisson distribution may involve more effort, it removes uncertainty and is always worth trying.

The Negative Binomial distribution is perhaps the second most commonly encountered count distribution. A study by Jones[930] included counting the number of `break` statements in C functions. Figure 11.31 shows the number of functions containing a given number of `break` statements, along with a fitted Negative Binomial distribution.

A `break` statement can occur zero or more times within a loop or `switch` statement, and these statements can occur zero or more times within a function definition. A Negative Binomial distribution can be generated by drawing values from multiple Poisson distributions (whose characteristics have been drawn from a Gamma distribution); might the number of `break` statements in each function different Poisson distribution?

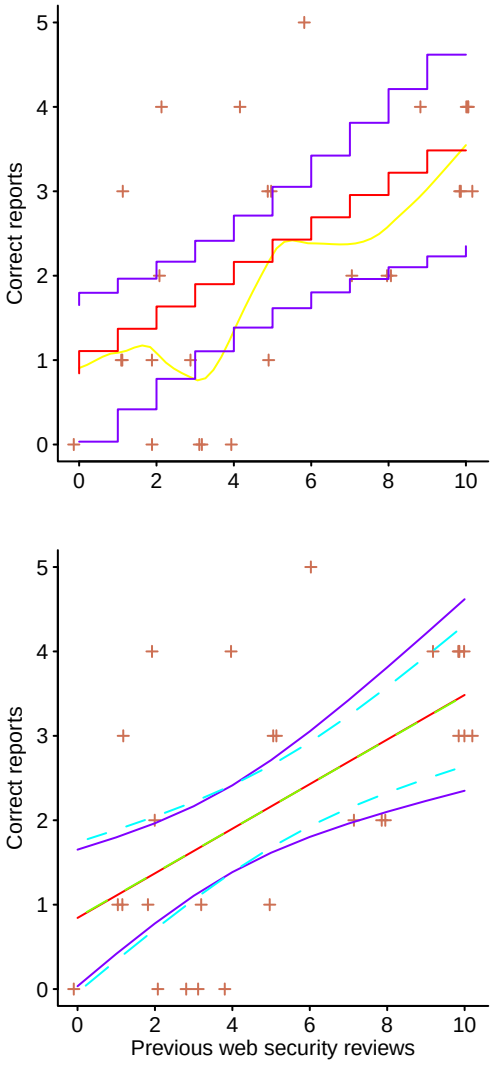


Figure 11.30: Number of vulnerabilities detected by professional developers with web security review experience; upper: technically correct plot of model fitted using a Poisson distribution, lower: simpler to interpret curve representation of fitted regression models assuming measurement error has a Poisson distribution (continuous lines), or a Normal distribution (dashed lines). Data extracted from Edmundson.[531] Github–Local

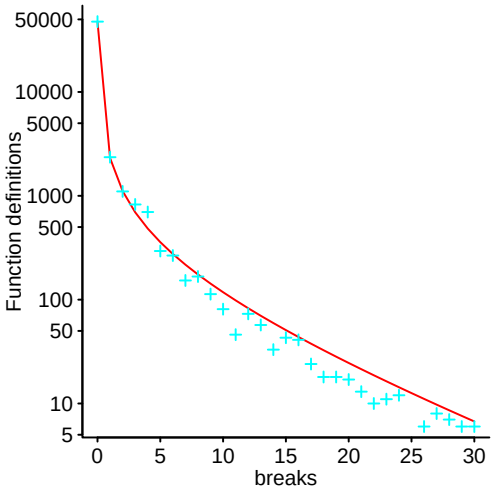


Figure 11.31: Number of functions containing a given number of `break` statements and a fitted Negative Binomial distribution. Data from Jones.[930] Github–Local

[xvi] The choice is selectable via the `type` argument to `plot/lines`.

[xvii] To achieve an acceptable p-value, three outliers were removed.

The gamlss package[1766] supports a wide variety of probability distributions, including the NBI distribution (Negative binomial type I distribution; there is also a type II) used in the following code:

```
library("gamlss")

breaks=rep(j_brk$occur, j_brk$breaks)
nbi_bmod=gamlss(breaks ~ 1, family=NBI)

plot(function(y) max(jumps$breaks, na.rm=TRUE)*     # Scale probability distribution
                        dNBI(y, mu=exp(coef(nbi_bmod, what="mu")),
                                sigma=exp(coef(nbi_bmod, what="sigma"))),
        from=0, to=30, n=30+1, log="y", col=pal_col[1],
        xlab="breaks", ylab="Function definitions\n")
points(jumps$occur, jumps$breaks, col=pal_col[2])
```

While zero is a common lower bound, other lower bounds are sometimes encountered; see section 9.3.1. Both the gamlss.tr and VGAM packages support a wide variety of truncated distributions; gamlss and related packages are used in this book because of the volume and quality of their documentation.

A study by Starek[1763] investigated API usage in Java programs. Figure 11.32 shows the number of APIs used in Java programs containing a given number of lines of code. The API count starts at one, not zero, and many programs use a few APIs, suggesting that a Poisson distribution may be applicable; the range of the number of APIs used does not suggest a log scale.

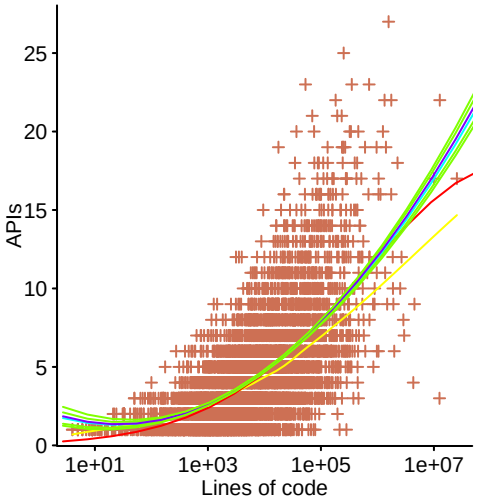


Figure 11.32: Number of APIs used in Java programs containing a given number of LOC; lines are fitted models based on a zero-truncated Poisson (red), Poisson and Normal distributions (blue, with confidence intervals in green), yellow line is loess fit. Data from Starek.[1763] Github–Local

In the following code, gen.trun creates a zero-truncated Poisson distribution (derived from PO in the gamlss.tr package) having the identity function as the link for its mean (rather than the default log link).

```
library("gamlss")
library("gamlss.tr")

gen.trun(par=0, family=PO(mu.link=identity))

tr_mod=gamlss(APIs ~ l_size+I(l_size^2), data=API, family=POtr)
```

Figure 11.32 shows the fitted model in red. The other lines are fitted models using a Poisson distribution that is not zero-truncated and a Normal distribution, along with 95% confidence intervals. The yellow line is a loess fit. Other explanatory variables could be added to the model to improve the fit to the data.

11.3.2 Continuous response variable having a lower bound

Measurements of the response variable may be drawn from a continuous distribution, e.g., measurements involving length or time. The continuous distribution taking non-negative values, supported by glm, is the Gamma distribution.

In practice, when most measurement values are sufficiently far away from zero (where far away could be a large single digit value) there is little difference between models fitted using the Normal and Gamma distributions. For measurements closer to zero the main difference between models fitted using different distributions is in the confidence intervals (which are usually not symmetric, and may be larger/smaller).

The canonical link function for the Gamma distribution is inverse, and the following two calls are equivalent:

```
G_mod=glm(y ~ x, data=sample, family=Gamma)  # Yes, capital G
G_mod=glm(y ~ x, data=sample, family=Gamma(link="inverse"))
```

The inverse link function means that the equation being fitted is (the identity link function is supported):

$$y = \frac{1}{\alpha + \beta x} + \varepsilon$$

Figure 11.33 comes from a code review study (discussed in section 13.2) and shows meeting duration when reviewing various amounts of code. Meeting duration must be greater

than zero, and a Gamma measurement error distribution is assumed to apply (the variables are assumed to have a linear relationship, and the identity link function is used). The red line is the model fitted using a Gamma error distribution (plus confidence bounds), the green line is the Gaussian distribution fit.

The data contains a few points with high leverage, and the loess fit suggests that there may be a change-point, so a more involved analysis is appears necessary.

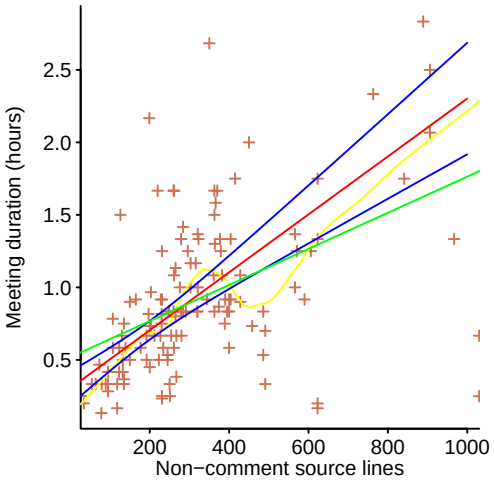


Figure 11.33: Code review meeting duration for a given number of non-comment lines of code; fitted regression model, assuming errors have a Gamma distribution (red, with confidence interval in blue), or a Normal distribution (green). Data from Porter et al.[1507] Github–Local

11.3.3 Transforming the response variable

When plotting sample points, values along one or both axes are sometimes transformed to compress or spread out the points, for the purpose of improving data visualization.

A regression model is fitted to a pattern (represented by an equation) and if a plot using transformed axis, contains visible pattern(s) of behavior, it is worth investigating a model that uses similarly transformed values.

Applying a non-linear transform to the response variable changes its error distribution, and a regression model built using this transformed response variable may not be a natural fit to the processes that produced the measurements. Explanatory variables are assumed not to contain any error and transforming them does not change this assumption.

For example, in the following regression model the error, ε, is additive:

$$y = \alpha + \beta x + \varepsilon$$

while fitting a log-transformed response variable:

$$\log y = \alpha + \beta x + \varepsilon$$

produces a model where the error is multiplicative, i.e., the error is a percentage of the measured value:

$$y = e^{\alpha+\beta x} e^{\varepsilon}$$

The error in a model fitted using a log link function is additive, because the equation fitted is:

$$\log(y + \varepsilon) = \alpha + \beta x$$

which becomes (the error randomly fluctuates around zero, and negating it changes nothing):

$$y = e^{\alpha+\beta x} + \varepsilon$$

If the response variable is transformed, the decision on whether to transform it directly, or via a link function, is driven by whether the error is thought to be additive or multiplicative; as always, domain knowledge is crucial.

A log link can be specified for `glm`'s default Normal distribution by passing: `family=gaussian(link="log")`. If this use fails to converge, the Poisson distribution is a good approximation to the Normal distribution (except when many sample values are close to zero) and can be substituted when the response variable takes integer values; see fig 7.36.

One advantage of log transforming a response variable is that it reduces the influence of outliers (because the range of values is compressed). Figure 11.34 illustrates the impact of removing one highly influential value (circled in red) from the data used to fit a model using a log link function (blue lines, dashed is after removal), and a model fitted using a log transformed response variable (red lines, dashed is after removal);[xviii] the vertical shift is the difference between treating measurement error as additive and multiplicative.

The visual appearance of outliers and influential observations plotted using log axis can be deceiving, i.e., they may not appear to be that far removed from the general trend; see fig 8.40. As always, assumptions based on visual appearance need to be checked numerically.

Many data analysts continue to fixate on fitting data whose measurement error has a Normal distribution. The Box-Cox transformation continues to be used to map a response variable to have a more Normal distribution-like error. The `boxcox` function in the `MASS` package, and the `powerTransform` function in the `car` package, provide support for this functionality.

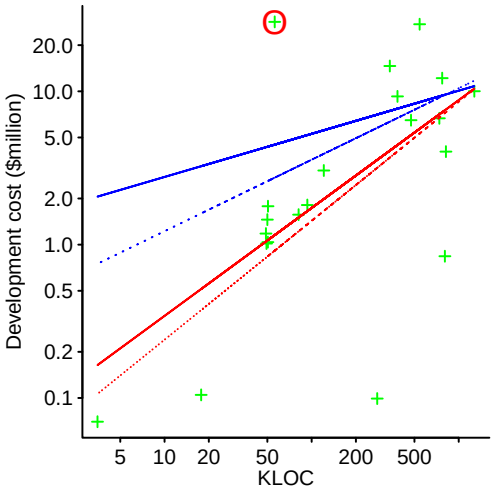


Figure 11.34: Annual development cost and lines of Fortran code delivered to the US Air Force between 1962 and 1984; lines show fitted regression models (red: log transformed, blue: using a log link function) before(solid)/after(dotted) outlier removed (circled in red). Data extracted from NeSmith.[1366] Github–Local

[xviii] The visual difference is less dramatic if the axes are switched.

A traditional approach to simplifying a problem is to map a continuous variable to a number of discrete values, e.g., small/medium/large. Throwing away information may simplify a problem, but the cost can be a considerable loss of statistical power and residual confounding.[1615] Using a computer removes the need to simplify just to reduce the manual effort needed to perform the analysis. See Github–regression/melton-statics.R for an example where building a regression model provides a lot more information about the characteristics of the continuous data, compared to mapping values to large/small and running a chi-squared test.

Adjusting a fitted model to handle uncertainty in the explanatory variables, when the model contains a multiplicative error, requires specifying the measurement error for every value of an explanatory variable. The following option assigns a 10% error: `measurement.error=maint$lins_up/10`.

A study by Jørgensen[945] investigated maintenance tasks and obtained developer effort and code change data. Figure 11.35 shows the effort (in days) and number of lines inserted and updated for 89 maintenance tasks. The original fitted regression line is in red, and the SIMEX adjusted line is in blue. The call to `simex` is:

```
maint_mod=glm(EFFORT ~ lins_up, data=maint,
                        family=gaussian(link="log"), x=TRUE, y=TRUE)

y_err=simex(maint_mod, SIMEXvariable="lins_up",
                measurement.error=maint$lins_up/10, asymptotic=FALSE)
```

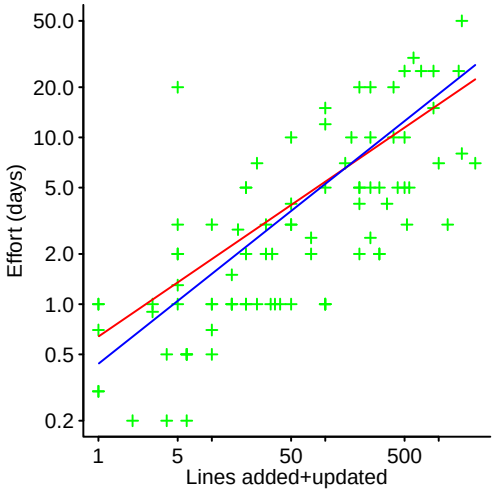


Figure 11.35: Maintenance task effort and lines of code added+updated, with fitted regression model (red), and SIMEX adjusted for estimated 10% error (blue). Data from Jørgensen.[945] Github–Local

11.3.4 Binary response variable

When the response variable takes one of two possible values, e.g., (false, true) or (0, 1), it has a binomial distribution. If the value of the response variable switches from 0 to 1 (or 1 to 0), as the explanatory variable increases (or decreases), and then always has that value for further increases (decreases) in the explanatory variable, there is no need to build a regression model (simply find the switch point). When the response variable can have two possible values over some range of the explanatory variable, regression modeling fits an equation that minimises some metric for the residual error.

A study by Höfer[837] investigated the various aspects of the implementation a problem by students and professional developers (working in pairs). Figure 11.36 shows the number of lines of test code changed by students and professionals (measurement denoted by the grey plus is treated as an outlier and not included in the model building), along with fitted regression lines, a straight line and a logistic equation. (For an analysis of Microsoft's C/C++ compiler price differential under MSDOS and Windows, see Github–economics/upgrade-languages.R).

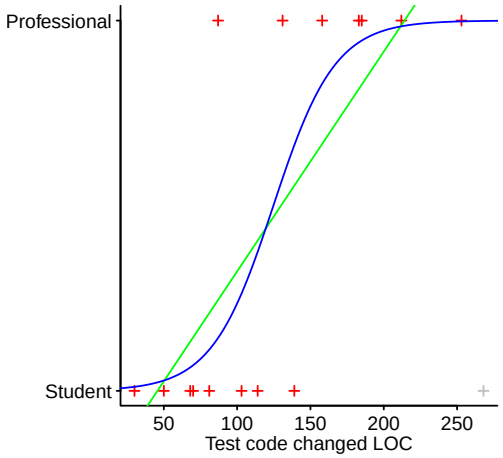


Figure 11.36: Regression modeling 0/1 data with a straight line and a logistic equation. Github–Local

The canonical link function for the Binomial distribution is `logit`, and the following two calls are equivalent:

```
b_mod=glm(y ~ x, data=sample, family=binomial)
b_mod=glm(y ~ x, data=sample, family=binomial(link="logit"))
```

The equation for the `logit` link function is:

$$\log \frac{y}{1-y} = \alpha + \beta x$$

where the response has the form of a log-odds ratio. This equation can also be written as:

$\log \frac{p}{q} = \alpha + \beta x$, where: p proportion of successes, q proportion of failures ($q = 1 - p$).

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

The values returned by `predict`, for a fitted binomial model, are in the range $0 \ldots 1$. The person doing the analysis has to decide the value that divides this continuous range, such that predictions are either zero or one. One approach is to treat predicted values greater than 0.5 as predicting one, and predicted values less than or equal to 0.5 as predicting zero. A more sophisticated approach looks at the distribution of predictions, and makes an informed trade-off between the true/false positive rate (often calculated using *recall* and *precision*).

A ROC curve (receiver operating characteristics; named after a technique originally used to measure the performance of radio receivers) is a visualization technique showing the trade-offs between two rates, i.e., the true positive rate and false positive rate; it is a common technique for displaying the trade-offs from predictions returned by machine learning models. The `ROCR` package supports the creation and plotting of ROC curves.

The columns in table 11.1 show an example of the impact of selecting particular cut-off values, for distinguishing between true/false (for 10 data points). Reading left-to-right, at a cut-point of 0.9 there is one correct prediction (a true positive), at a 0.81 cut-point another correct prediction, while at 0.72 an incorrect prediction (a false positive) is made (at this cut-point the response rate for correct predictions is 40% and 20% for incorrect predictions).

t	t	f	t	f	t	f	t	f	f
0.90	0.81	0.72	0.60	0.53	0.44	0.39	0.28	0.16	0.09

Table 11.1: Example list of prediction outcome occurring at various cut-point values. Github–Local

Figure 11.37 shows the ROC curve for this data.

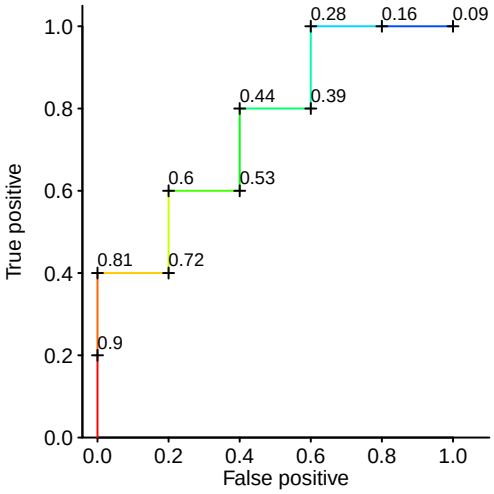


Figure 11.37: ROC curve for the data listed in table 11.1. Github–Local

11.3.5 Multinomial data

When a discrete response variable takes on more than two values, it has a *multinomial* distribution.

nominal: when a response variable can take N distinct values and π_i is the probability of the i^{th} value occurring ($\sum_{i=1}^{N} \pi_i = 1$), then the *baseline-category logit model* (with one explanatory variable, x, in this example) is:

$$\log \frac{\pi_n}{\pi_N} = \alpha_n + \beta_i x, \text{ for } n = 1, \ldots, N-1.$$

Fitting a model results in $N-1$ equations, with separate coefficients for each.

The `mlogit` package supports the building of multinomial logit models for response variables containing nominal data.

ordinal: fitting an independent logit model to each pair of adjacent values (as is done for nominal models) fails to make use of all the available information; the logit function can be extended to include the ordering information present in ordinal data.

Given an ordinal response variable, Y, that can appear in one of $j = 1, \ldots, N$ possible categories, then Y has a multinomial distribution; its cumulative probability is given by:

$P(Y_i \leq j) = \pi_{i1} + \pi_{i2} + \cdots + \pi_{iN}$, where: π_{ij} is the probability that the i^{th} measurement appears in response category j, and $\sum_{j=1}^{N} \pi_{ij} = 1$

The *cumulative logits* treats $P(Y \leq j)$ as the response variable in a model fitting process that uses a logit link function (other, related functions can be used).

The `ordinal` package supports the building of *cumulative link models*, also known as *ordinal regression models*.

A study by Luthiger and Jungwirth[1176] investigated the importance of fun as a motivation for software development. The survey, which had 1,330 responses from people working on open source projects, asked for an estimate of the percentage of their spare time people spent on activities involving open source development. Possible answers were restricted to intervals of 10%, an ordinal scale. The `clm` functions fits a cumulative link model; `predict` returns a vector of predictions, one for each ordinal value (six vectors in this example):

```
library("ordinal")

f_mod=clm(q42 ~ q31, data=fasd) # Best fitting model is q42 ~ q5+q29+q31

pred=predict(f_mod, newdata=data.frame(q31=1:6))

plot(-1, type="n", xlim=c(1, 6), ylim=c(0, 0.6),
        xlab="Answer given to q31", ylab="Probability\n")

dummy=sapply(1:10, function(X) lines(1:6, pred$fit[ ,X], col=pal_col[X]))
```

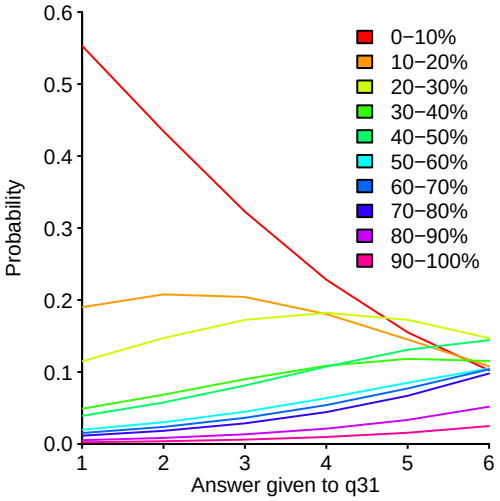


Figure 11.38: Probability of subject response being within a given percentage interval, based on their response to question q31. Data kindly provided by Luthiger[1176] . Github–Local

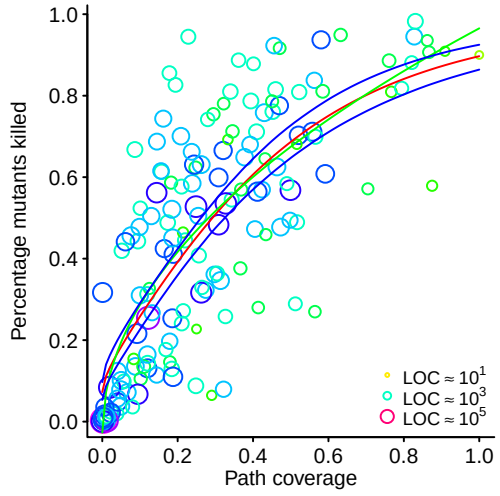


Figure 11.39: Percentage of mutants killed at various percentage of path coverage for 300 or so Java projects; fitted Beta regression (red), with 95% confidence intervals (blue) and `glm` (green) regression models. Data from Gopinath et al.[712] Github–Local

Figure 11.38 shows the probability of a subject giving an answer within a given 10% band, given their answer to question q31 (the formula: `q42 ~ q5+q29+q31`, is a better fit, but is not easily plotted in 2-D).

11.3.6 Rates and proportions response variables

When dealing with a response variable that is a rate or proportion, there is a fixed lower and upper bound, e.g., 0 and 100. Measurements within a fixed interval often share two characteristics: they exhibit more variation around the mean and less variation towards the lower and upper bounds,[xix], and they have an asymmetrical distribution. These characteristics can be modeled by a Beta equation. A regression model where the response variable is fitted to a Beta equation is known as a *Beta regression model*.

The `betareg` package contains functions that support the fitting of Beta regression models. When fitting basic models, calls to `betareg` have the same form as calls to `glm`; both functions include options that are not supported by the other.

Figure 11.39 shows fitted curves from a beta regression model (red), bootstrapped confidence intervals (blue), and a call to `glm` (green); the study that produced the data is discussed elsewhere, see fig 6.54. The equation fitted is: $mutants_{killed} \propto \sqrt{coverage}$, and was chosen because it is something simple that works reasonably well. Searching for the best fitting exponent, using `nls` (the `betareg` package does not support fitting non-linear models), shows that 0.44 is a better fit than 0.5 for this sample.

The `summary` output for a Beta regression model includes extra information, as follows: Github–Local

```
Call:
betareg(formula = y_measure ~ I(x_measure^0.5))

Standardized weighted residuals 2:
    Min      1Q  Median      3Q     Max
-2.6881 -0.6403 -0.1279  0.6399  3.1829

Coefficients (mean model with logit link):
                Estimate Std. Error z value Pr(>|z|)
(Intercept)      -2.5460     0.1891  -13.46   <2e-16 ***
I(x_measure^0.5)  4.7093     0.3502   13.45   <2e-16 ***

Phi coefficients (precision model with identity link):
      Estimate Std. Error z value Pr(>|z|)
(phi)   4.9641     0.5386   9.217   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Type of estimator: ML (maximum likelihood)
Log-likelihood: 80.37 on 3 Df
Pseudo R-squared: 0.6323
Number of iterations: 13 (BFGS) + 1 (Fisher scoring)
```

The `phi` coefficient (the Greek letter ϕ) is the second coefficient of the fitted Beta distribution, $B(\mu, \phi)$.

The `predict` function returns the expected value of the response variable, $E(y) = \mu$, but does not support a `se.fit` option (when passed a Beta regression model). The bootstrap can be used to calculate a confidence interval from the predictions made by many models, as follows:

```
library("betareg")
library("boot")

boot_reg=function(data, indices)
{
cov_data=data[indices, ]
b_mod=betareg(mut_cov ~ I(path_cov^0.5), data=cov_data)
# A vector must be returned, i.e., no data frames
return(predict(b_mod, newdata=data.frame(path_cov=x_vals)))
}
```

[xix] The measurement sample is heteroskedastic.

```
cov_boot=boot(pm_info, boot_reg, R = 4999)

ci=apply(cov_boot$t, 2, function(X) quantile(X, c(0.025, 0.975)))
lines(x_vals, ci[1, ], col=pal_col[3])
lines(x_vals, ci[2, ], col=pal_col[3])
```

The default link function used by the `betareg` function is logit, the same default link used by `glm`, for the argument `family=binomial`.

11.4 Multiple explanatory variables

Linear regression can be used to fit models containing more than one explanatory variable; *multiple regression* is the term used for modeling with more than one explanatory variable (the term *bivariate regression* is sometimes applied to the single explanatory variable+response variable case). In theory there is no limit on the number of explanatory variables, but in practice available processing resources, and the need to hold data in storage set an upper bound.

Visualization is much more complicated when there are multiple explanatory variables. Chapter 8 contains examples for visualizing two variables, and the general approach is to break down multiple regression visualization into pairs of variables.

System performance is affected by many factors; figure 11.40 shows SPECint 2006 results for processors running at various frequencies (upper), color coded by memory chip frequency (center) and name of processor family (lower).

The SPECint results include 36 columns of information relating to the benchmarked system. Which of these columns contains information that can be used to succinctly model the performance of a system, and what equation best describes the form of their contribution?

The R formula notation includes a symbol that denotes all columns in the data frame as explanatory variables, except the one specified as the response variable; the dot symbol is used as follows:

```
spec_mod=glm(Result ~ ., data=cint)
```

Given enough cpu power and memory, it can be more productive to start by considering all explanatory variables and remove underperforming variables, rather than starting with the explanatory variables believed to be the most important and then adding more variables.

The `stepAIC` function, in the `MASS` package, automates the process of removing underperforming explanatory variables from a fitted model, to create a model having a minimum AIC (the `step` function, in the base system, is a rather minimal implementation).[xx]

When some domain knowledge is available (e.g., performance often correlates with clock rate and is not usually affected by date of execution), experimentation of fitting models containing explanatory variables considered to be most likely to have a large impact on the response variable, can help refine the analyst's appreciation of the impact of different explanatory variables on overall performance.

For this SPEC dataset, there is so much detail recorded in the `Processor` column of the Spec results, that each entry is often unique; making it possible to create an almost perfect, but completely uninformative, model using just this one explanatory variable.

The following model explains 80% of the variance in the `Result` values:

```
spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)
```

where: `Processor.MHz` is the processor clock rate, `mem_rate` the peak memory transfer rate and `mem_freq` the frequency at which memory is clocked.

The + binary operator, in the above formula, specifies that explanatory variables are added together. The `summary` output shows that the equation fitted by `glm` is:

$$Result = -2.4 \cdot 10^{1} + Processor.MHz 7.3 \cdot 10^{-3} + mem\_rate 2.5 \cdot 10^{-3} + mem\_freq 1.0 \cdot 10^{-2}$$

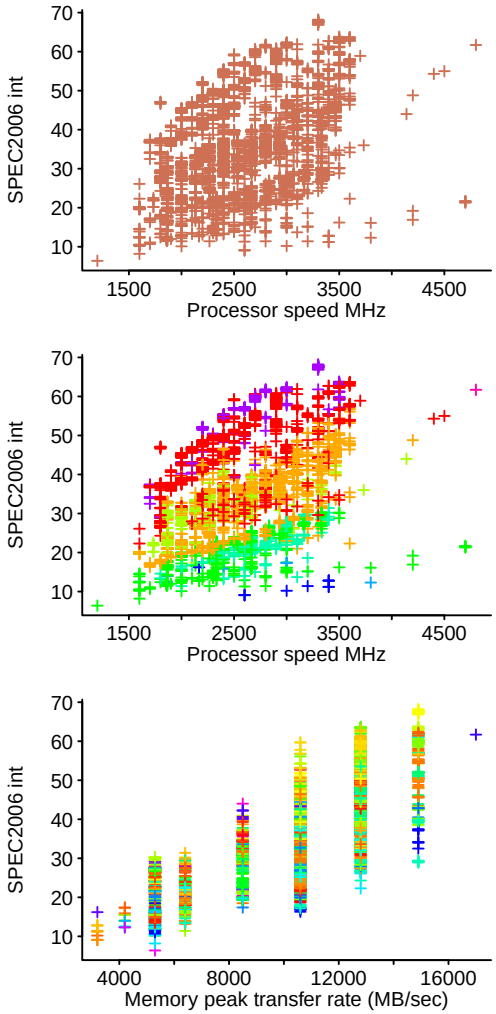


Figure 11.40: SPECint 2006 performance results for processors running at various clock rates, memory chip frequencies and processor family. Data from SPEC.[1742] Github–Local

[xx] This fishing expedition approach to model building requires that p-values be suitably reduced, e.g., using a Bonferroni corrected value.

With a single explanatory variable, it is easy to visually compare model predictions against measured values; with multiple variables things are not so simple. One approach is to analyse the impact of each variable, on predictions, in turn.

The `crPlot` and `crPlots` functions, in the `car` package, produce a *component+residual* plot (also known as a *partial-residual* plot); the y-axis contains the predicted value plus the residual, the x-axis contains the value of the explanatory variable:

```
library("car")

spec_mod=glm(Result ~ Processor.MHz+mem_rate+mem_freq, data=cint)

crPlots(spec_mod, term= ~ ., layout=c(3, 1), col=point_col,
         cex.lab=1.5, cex.axis=1.5, ylab="Component+Residuals\n", main="")
```

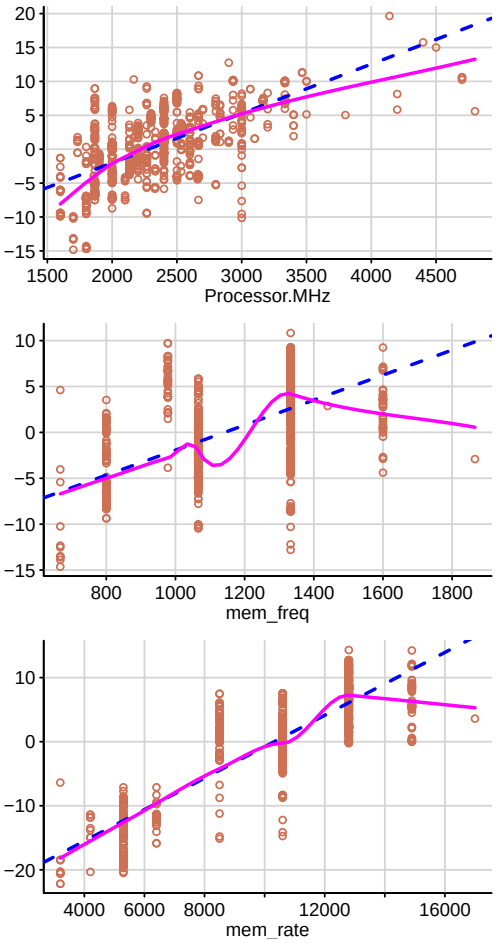


Figure 11.41: Component+residual plots for three explanatory variables in a fitted SPECint model. Github–Local

Figure 11.41 shows the component+residual plots produced by the above code. The red dotted line is derived from the fitted model, and the green line a loess fit; if the form of an explanatory variable, in the formula used to fit a model, is close to reality, the two lines will be closely intertwined. For the SPEC model there is consistent divergence of the two lines, over ranges of the measurement interval, for two variables and perhaps some for a third.

Experience of hardware characteristics suggests that performance does not increase forever, as clock rates are increased. Adding quadratic forms of the explanatory variables to the model is a step up in complexity, to try out with a fitted model (an exponential is more realistic, in that its maximum converges to a limit, but this form of modeling requires the use of non-linear regression, which is covered later).

Adding quadratic terms, for two of the three explanatory variables, to the fitted model explains another 4% of the variance, but significantly reduces the error at higher processor and memory frequencies; see figure 11.42.

Some of the systems benchmarked contained error correcting memory, which might be expected to slightly reduce performance. The `update` function can be used to add, or remove, explanatory variables from a previously fitted model. The following code adds the variable `ecc` to the previously fitted model, `spec_mod`:

```
ecc_spec_mod=update(spec_mod, . ~ . + ecc)
```

The advantage of using `update` is a reduction in the system resources needed to fit the model, compared with starting from the beginning again.

The `summary` output shows that systems containing error correcting memory have slightly better performance. Before jumping to the conclusion that adding error correction improves system performance, it is worth noting that this kind of memory tends to be used in high-end systems, where it is likely that money has been spent to improve performance and reliability.

The choice of cpu and memory frequency is based on information that is not present in the SPEC result data, the intended price point the computing system is designed to be sold at, and the trade-off in the cost/performance of the components needed to build it.

What contribution does each explanatory variable make to a fitted model? Some ways in which individual contributions can be measured include:

- the amount of variance, in the response variable, explained by an explanatory variable.

  The `calc.relimp` function, in the `relaimpo` package, calculates the contribution made by each explanatory variable to the variance explained by the fitted model,

- the impact each explanatory variable can have on the range of values taken by the response variable (with all other explanatory variables maintaining a fixed value).

  For very simple models, [xxi] one way of calculating the maximum impact on the value of the response variable is by multiplying the minimum/maximum value taken by an explanatory variable by the corresponding coefficient in the fitted model. For instance, `range(cint$Processor.MHz)*7.3*10^-3` evaluates to `11.68 35.04`, a difference of `23.36`.

  The `visreg` function, in the `visreg` package, produces a visual representation of the impact of each explanatory variable on the response variable.

[xxi] Those that are linear in the explanatory variable, with no interactions between variables.

Nomograms are a visual method for calculating the value of a response variable when each explanatory variable has a particular value: the DynNom function in the DynNom package supports interactive exploration of model behavior in a web browser.

Normalising values prior to fitting a model is sometimes suggested (e.g., using the scale function); the relative values of the model coefficients can then be directly compared. This method only works when all explanatory variable values are drawn from a Normal distribution.

The relaimpo package supports a variety of functions[745] for calculating the relative contribution made to a model by each explanatory variable it contains. For instance, the calc.relimp function calculates: first, the variance explained by a model containing just each variable, last, the variance explained when a variable is added to a model that already contains the other variables, betasq, the standardized coefficients of the model (i.e., one fitted after normalising the data; effectively a metric for the contribution of each explanatory variable to the response variable value), and lmg (named after the initials of its creators), the variance explained by each variable; the boot.relimp function returns confidence intervals for these values.

```
library("relaimpo")

spec_mod=glm(Result ~ Processor.MHz+I(Processor.MHz^2)+mem_rate
                          + I(mem_rate^2)+mem_freq, data=cint)

# How much does each explanatory variable contribute?
calc.relimp(spec_mod, type = c("first", "last", "betasq", "lmg"))
```

The calc.relimp output is: Github–Local

```
Response variable: Result
Total response variance: 81.56
Analysis based on 1346 observations

5 Regressors:
Processor.MHz I(Processor.MHz^2) mem_rate I(mem_rate^2) mem_freq
Proportion of variance explained by model: 83.77%
Metrics are not normalized (rela=FALSE).

Relative importance metrics:

                      lmg     last   first betasq
Processor.MHz      0.06189 0.017807 0.04609 0.6888
I(Processor.MHz^2) 0.04556 0.005698 0.02962 0.2201
mem_rate           0.29380 0.028909 0.55554 2.0853
I(mem_rate^2)      0.29050 0.006363 0.58253 0.4768
mem_freq           0.14598 0.067531 0.28997 0.1258

Average coefficients for different model sizes:

                          1X        2Xs        3Xs        4Xs        5Xs
Processor.MHz      4.308e-03  1.290e-02  1.568e-02  1.823e-02  1.666e-02
I(Processor.MHz^2) 6.560e-07 -4.894e-07 -9.880e-07 -1.728e-06 -1.788e-06
mem_rate           2.343e-03  1.562e-03  2.236e-03  3.303e-03  4.540e-03
I(mem_rate^2)      1.280e-07  1.488e-07  8.108e-08 -1.286e-08 -1.158e-07
mem_freq           2.429e-02  2.012e-02  1.558e-02  1.457e-02  1.600e-02
```

the second set of columns, under the line starting Average coefficients, lists the model coefficients for each explanatory variable, if that variable were to appear in a model containing X variables (values are averaged over all combinations of other variables). The values in the last column (5Xs in this case) are the same as those produced by the summary function for the fitted model.

How do changes in the value of each explanatory variable individually affect the value of the response variable? The visreg package supports functions for plotting the relationship between the response variable and individual explanatory variables (with the other variables held constant at their median value).

Figure 11.42 shows the individual contribution made by each explanatory variable to the value of the response variable (along with confidence intervals in grey), for the following model of SPECint performance:

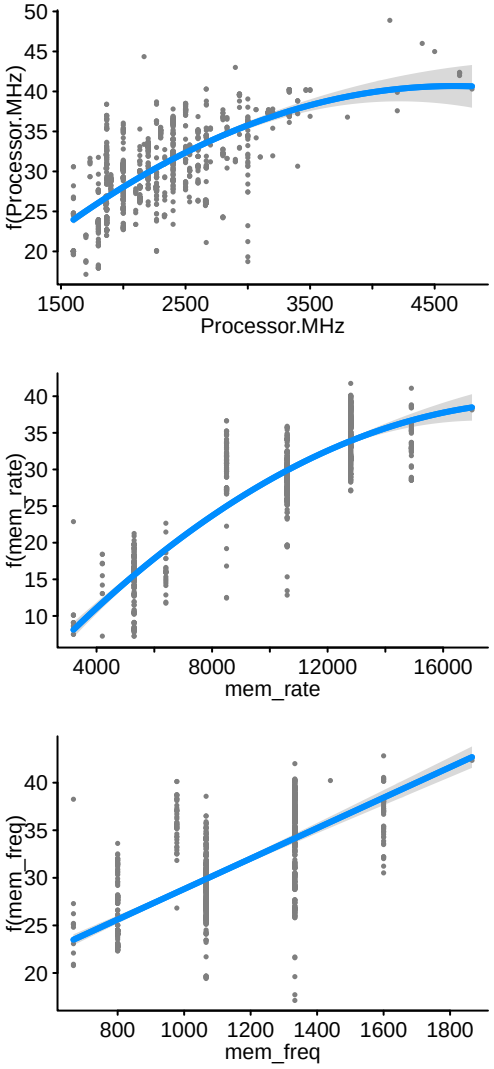


Figure 11.42: Individual contribution of each explanatory variable to the response variable in a quadratic model of SPECint performance. Github–Local

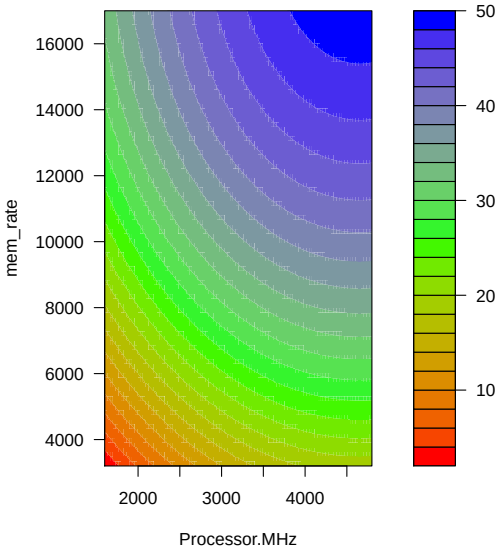


Figure 11.43: Contour map of Result values predicted by a fitted model of SPECint performance, over range of Processor.MHz and mem_rate values. Github–Local

```
library("visreg")

spec_mod=glm(Result ~ Processor.MHz + I(Processor.MHz^2)+mem_freq
                                  +mem_rate+I(mem_rate^2), data=cint)
visreg(spec_mod)
```

Figure 11.43 shows a contour plot created using the `visreg2d` function.

Sometimes including an explanatory that has no correlation with the response variable improves the performance of a model; why does this happen? An explanatory variable may correlate with the residual of a model, and adding this new variable has the effect of improving a model by reducing its residual.

11.4.1 Interaction between variables

In the models fitted so far, each explanatory variable has been independent of the others. The `glm` function and many other regression modeling functions provide mechanisms for specifying interactions between explanatory variables, using binary operators in the formula, such as `:`, `*` and `^`.

Operator	Effect
+	causes both of its operands to be included in the equation.
:	denotes an interaction between its operands, e.g., `a:b` or `a:b:c`.
*****	denotes all possible combinations of `+` and `:` operators, e.g., `a*b` is equivalent to `a+b+a:b`.
^	denotes all interactions to a specific degree, e.g., `(a+b+c)^2` is equivalent to `a+b+c+a:b+a:c+b:c`.
.	denotes all variables in the data-frame specified in the `data` argument except the response variable.
-	specifies that the right operand is removed from the equation, e.g., `a*b-a` is equivalent to `b+a:b`.
-1	specifies that an intercept is not to be fitted (many regression fitting functions implicitly include an intercept).
I()	"as-is", any operators in the enclosed expression are not treated as formula operators, the behavior is that applying outside a formula.

Table 11.2: Symbols that can be used within a formula to express relationships between explanatory variables.

As with all data analysis, the choice of interactions between explanatory variables should be driven by domain knowledge. When there is a lot of uncertainty about which interactions are significant, it may be easiest to start by specifying all pairs of interactions between variables (or triple interactions if there are not too many variables), and to then simplify, either automatically using `stepAIC`, or through manual inspection of `summary` output of the fitted models.

Stepwise regression techniques, such as that provided by `stepAIC`, can return models that suffer from a variety of problems, such as overfitting. There are techniques available to help avoid these problems; the `train` function in the `caret` package supports some of these techniques. The `glmulti` package automates the process of finding an optimal, in a sense specified by the user (e.g., minimise AIC or some other measure), explanatory variable interaction; a list of variables is specified, and the function permutes through the possibilities, e.g., `glmulti("y", c("a", "b", "c", "d"), data=some_data)`.

A study by Moløkken-Østvold and Furulund[1306] investigated the impact of daily communication between the customer and contractor on the accuracy of effort estimates, for 18 software projects. Figure 11.44 shows estimated vs. actual effort broken down by communication frequency (i.e., daily or not daily), along with individually fitted straight lines.

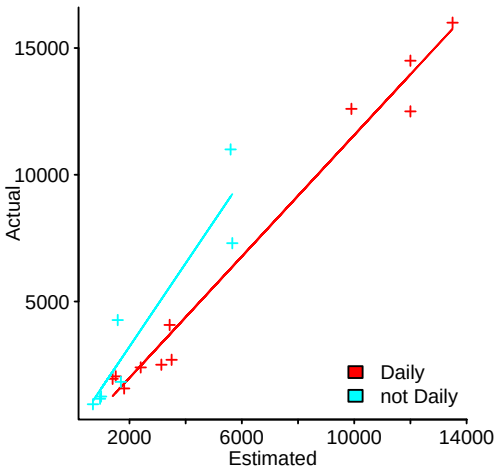


Figure 11.44: Estimated and actual effort broken down by communication frequency, along with individually fitted straight lines. Data from Moløkken-Østvold et al.[1306] Github–Local

It is possible to fit one regression model that simultaneously fits both straight lines to this data; the following code shows one possibility:

```
sim_mod=glm(Actual ~ Estimated+Estimated:Communication, data=sim)
```

The fitted equation is (based on `summary` output):

$$\begin{aligned} Actual &= -270.1 + 1.18 Estimated + 0.51 Estimated \times D \\ &= -270.1 + (1.18 + 0.51D) Estimated \end{aligned}$$

where: *Actual* is the actual and *Estimated* the estimated effort, and D has one of two values:

$$D = \begin{cases} 1 & \text{daily communication} \\ 0 & \text{not daily communication} \end{cases}$$

Is this formula the best fit possible using the available data? The formula used was selected by your author, because of a belief that the benefit of communication will increase as project size increases.

There are six data points for each of the 18 projects, computationally small enough for the brute force approach of examining all possible models; but with only 18 projects, some formula possibilities cannot be fitted because they contain more variables than available data points (a unique solution requires fewer variables than data points).

The formula in the following code fits four explanatory variables individually, plus each variable paired with every other variable (one at a time). `stepAIC` is used as a quick way of removing explanatory variables that are not paying their way (automatic model selection is fraught with problems, with perhaps the largest being that it allows analysts to stop thinking about the data):

```
sim_mod=glm(Actual ~ (Estimated+Communication+Contract+Complexity)^2, data=sim)

min_sim=stepAIC(sim_mod)
summary(min_sim)
```

This book fits regression models as a means of building understanding, and minimising AIC is often a useful step along the way. Another way of removing low impact variables from a model is to consider the p-value of each fitted component.

The `summary` output for ordinal and nominal explanatory variables, lists p-values for each value that these variables take in the data. The `Anova` function, in the `car` package, lists p-values at the variable level, and its output for the above model is: Github–Local

```
Analysis of Deviance Table (Type II tests)

Response: Actual
                        LR Chisq Df Pr(>Chisq)
Estimated                 45.879  1  1.258e-11 ***
Communication             17.272  1  3.240e-05 ***
Contract                   6.767  3    0.07971 .
Complexity                 1.543  1    0.21423
Estimated:Communication    2.546  1    0.11060
Communication:Contract     5.020  3    0.17034
Contract:Complexity        3.197  2    0.20224
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The variable `Complexity` has the highest p-value, and repeatedly removing the component having the highest p-value, for successively smaller models (smaller in the sense of containing fewer components) leads to the following model:

```
sim_mod=glm(Actual ~ Estimated+Communication+Communication:Contract, data=sim)
```

which fits the following equation:

$$\begin{aligned} Actual = -274.8 + 1.21 Estimated + 2625 \times !D + \\ C_{fp}(1862 \times !D - 197.6 \times D) + \\ C_{tp}(-2270 \times !D - 462.2 \times D) + \\ C_{ot}(-2298 \times !D - 234.3 \times D) \end{aligned} \tag{11.6}$$

where the new variables are: C_{fp} is a fixed price contract, C_{tp} is a target price contract and C_{ot} other kind of contract.

$$C_{fp} = \begin{cases} 1 & \text{fixed price contract} \\ 0 & \text{not fixed price contract} \end{cases} \quad C_{tp} = \begin{cases} 1 & \text{target price contract} \\ 0 & \text{not target price contract} \end{cases}$$

$$C_{ot} = \begin{cases} 1 & \text{other contract} \\ 0 & \text{not other contract} \end{cases}$$

This model explains a greater percentage of the variance in the data than the first model fitted, it also has a slightly smaller AIC. While it makes use of extra information (i.e., the kind of contract), a more noticeable difference is that `Communication` has a constant effect (i.e., it does not increase with estimated size); the case of fixed price contracts, with no daily communication cries out for attention.

Following the numbers has produced a model that is a better fit to the data, but not to expectations (which may, of course, be wrong).

11.4.2 Correlated explanatory variables

The mathematics behind many approaches used to fit linear regression models assumes that explanatory variables are independent of each other. If a linear relationship exists between one or more pairs of explanatory variables (i.e., a relationship of the form: $PV_1 = a + b \times PV_2$, where PV_1 and PV_2 are explanatory variables, a is any constant and b a non-zero constant), then this needs to be taken into account by the model building technique used.[xxii]

Multicolinearity is said to occur, when a linear relationship exists between two or more explanatory variables, the term *colinearity* is often used when only two variables are involved.

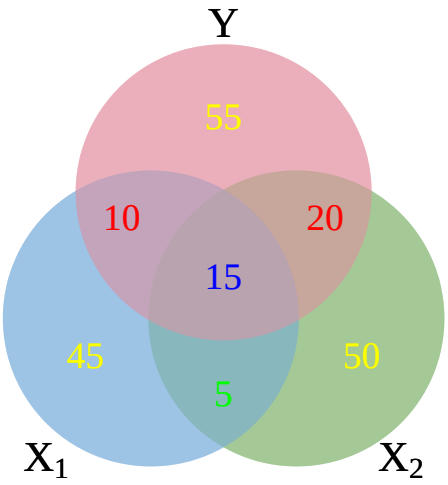


Figure 11.45: Illustration of the shared and non-shared contributions made by two explanatory variables to the response variable Y. Github–Local

Figure 11.45 illustrates how the variance in Y explained by combining X_1 and X_2 may be less than the sum of the variance explained by each individually, because the two variables are not independent; there is a shared contribution.

The impact of multicolinearity is to increase the standard error in the calculated value of the fitted model coefficients (i.e., the β_n), potentially resulting in a model that is not considered acceptable or is unreliable (in the sense that small changes in the data result in large changes in the coefficients of the fitted model). The increased uncertainty, in some variables, will make it more difficult to isolate the effects of individual explanatory variables and will increase the width of the confidence intervals for the predicted values of the response variable.

The *Variance Inflation Factor* (VIF) is a measure of the uncertainty created by the presence of multicolinearity. The impact of VIF is the same as reducing the sample size. When no multicolinearity is present, VIF has a value of one. The impact on the standard error is:

$$\varepsilon_{standard} \propto \sqrt{\frac{VIF}{observations}}$$

When is a VIF value too large? A large VIF is more likely to be acceptable with a large sample, compared to a small one, e.g., the standard error is proportionally the same for 10,000 observations having a VIF of 400 and for 100 observations having a VIF of 4.

Suggested maximum VIF values appear in print, e.g., 5 or 10 are sometimes suggested. As always, think about what the VIF value means in the context of how the results will be used; pick a value that makes sense given the sample size, the error in the measurements and the level of error that is acceptable in the business context.

The `car` and `rms` packages support a `vif` function, that takes the model returned by a call to, for instance, `glm` and returns the VIF for each explanatory variable.

A study by Kroah-Hartman[1046] investigated the amount of change in the Linux kernel source code occurring between each release. Figure 11.46 shows the number of lines added, modified and removed, plus overall growth, number of files and total number of lines at each initial release of the Linux kernel from version 2.6.0 to 3.9 (two outliers have been excluded).

[xxii] It is ok for a nonlinear relationship to exist in linear models, e.g., $PV_1 = a + b \times PV_2^2$.

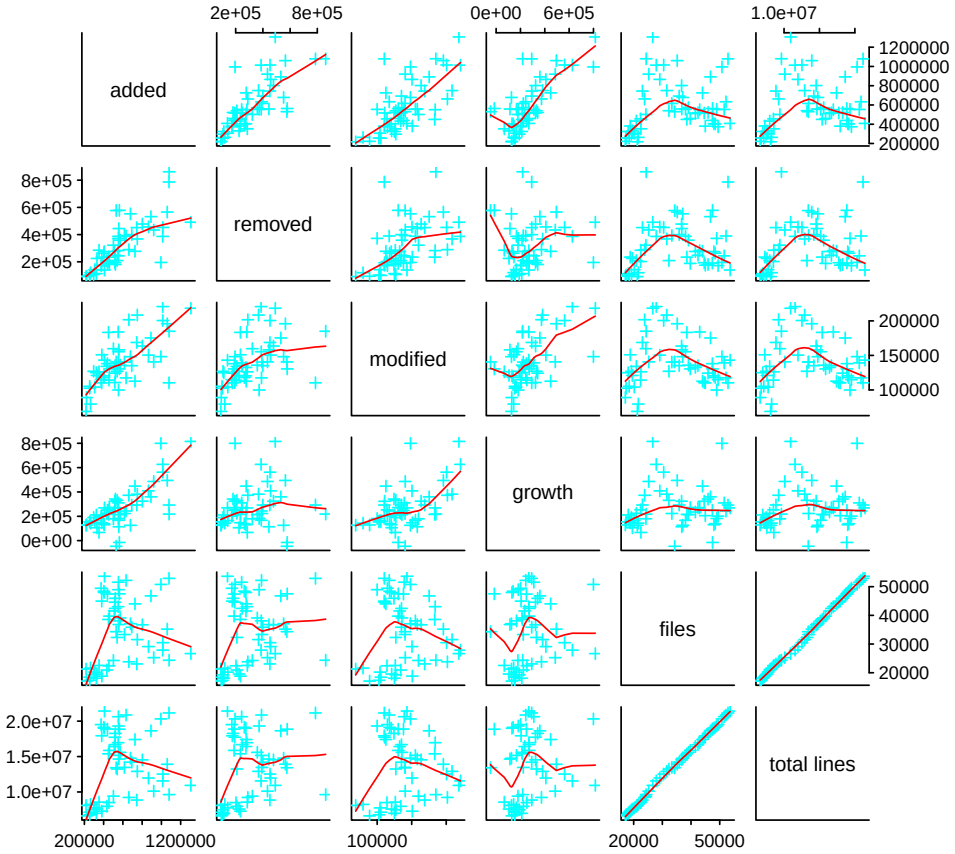


Figure 11.46: `pairs` plot of lines added/modified/removed, growth and number of files and total lines in versions 2.6.0 through 3.9 of the Linux kernel. Data from Kroah-Hartman.[1046] Github–Local

Building a model of the Linux kernel growth is complicated by the potentially high correlation between some measured variables, including:

- the growth, in lines of code, between releases is the difference between lines added and lines removed; these three variables are perfectly correlated in that knowing two of them enables the third to be calculated,
- lines added appears strongly correlated with lines removed. Perhaps existing functionality is being rewritten, rather than unrelated functionality being added,
- the decision about whether a line has been modified or removed/added is made algorithmically (rather than asking the developer who made the change). The amount of misclassified lines is not known,
- system level measurements are also correlated, e.g., number of files and total lines of code.

Modeling the number of modified lines, using the Kroah-Hartman data, finds that both lines added and lines removed individually explain around half of the deviance (61% and 41% respectively). However, combining them in a model does not produce any improvement; the following output from `summary` was obtained by including the argument `correlation=TRUE`. Github–Local

```
Call:
glm(formula = lines.modified ~ lines.added + lines.removed, data = amr_out)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-72376  -12049     321   11274   54964

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)    8.705e+04  8.625e+03  10.093 9.40e-14 ***
lines.added    9.958e-02  2.093e-02   4.759 1.64e-05 ***
lines.removed -1.500e-02  3.117e-02  -0.481    0.632
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 639477748)

    Null deviance: 6.2276e+10  on 53  degrees of freedom
```

```
Residual deviance: 3.2613e+10  on 51  degrees of freedom
AIC: 1253.1

Number of Fisher Scoring iterations: 2

Correlation of Coefficients:
              (Intercept) lines.added
lines.added   -0.54
lines.removed -0.06       -0.76
```

The correlation between the model coefficients appears at the end of the output and shows a high negative correlation between `lines.added` and `lines.removed`; the variable `lines.added` is a better predictor of `lines.modified` and has been selected over `lines.removed` (whose p-value is significantly larger than when just this variable appeared in a model).

A call to the `vif` produces the following: Github–Local

```
  lines.added lines.removed
      2.39311       2.39311
```

With only two explanatory variables, there is no ambiguity about which variables are involved in a linear relationship, but with more than two variables things are not always so obvious. The correlation table produced by `summary` can be used to identify related variables; the `alias` function generates just this information, when the argument `partial=TRUE` is specified.

Approaches to dealing with multicolinearity, to reduce any undesirable impact it may have on fitting a model, include:

- removing one or more of the correlated explanatory variables. The choice of which explanatory variables to remove might be driven by:
  - the cost of collecting information on the variable(s),
  - a VIF driven approach. The process involves fitting a model using the current set of explanatory variables, removing the explanatory variable with the largest VIF (removing one variable affects the VIF of those that remain and may reduce the VIF of other variables to an acceptable level) and iterating until all explanatory variables have what is considered to be an acceptable VIF,
- combining the strongly correlated variables in a way that makes use of all the information they contain.

The disadvantages of excluding explanatory variables from a model include:

- ignoring potentially useful information present in the excluded variable,
- creating a model that gives a false impression about which explanatory variables are important, i.e., readers will assume that the variables appearing in the model are the only important ones, unless information about the excluded variables is also provided,
- it provides the opportunity for the analyst to select the model that favours the hypothesis they want to promote (by selecting which explanatory variables appear in the model).

The SPEC power benchmark[1741] is designed to measure single and multi-node server power consumption, while executing a known load. The results contain 515 measurements of six system hardware characteristics, such as number of chips, number of cores and total memory, as well as average power consumption at various load factors.

A model of average power consumption, at 100% load, containing a linear combination of all explanatory variables, shows very high multicolinearity for the number of chips (its VIF is 27.5 and several other variables have a high VIF; see Github–hardware/SPECpower.R). Removing this variable reduces the VIF of the remaining variables, but the AIC drops from 6798.7 to 7182.1. Whether this decrease in model performance is important depends on the reason for building the model, e.g., prediction or understanding. Do the values of the model coefficients, after removing this variable, provide more insight than the coefficient values of the original model? These kinds of questions can only be answered by a person having detailed domain knowledge. This example shows how removing a variable solves one problem and raises others.

A study of fault prediction by Nagappan, Zeller, Zimmermann, Herzig and Murphy[1345] produced data containing six explanatory variables having an exact linear relationship

with other explanatory variables. The `glm` function detects the existence of this relationship and makes a decision about explanatory variables to exclude from the model (the value returned for their fitted coefficients is `NA`); see Github–regression/change-burst-sum.R.

Two explanatory variables having an exact linear relationship will have a correlation of ±1, as a call to `alias` will show.

11.4.3 Penalized regression

Penalized regression handles multicolinearity by automatically selecting how much each explanatory variable should contribute to the model; explanatory variables are penalized, based on their relative contribution to the model. The `penalized` package supports penalized regression.

The traditional technique for fitting a regression model involves minimising some measure of a specified error, where the error is defined to be the difference between actual and predicted values, e.g., the sum of squared error, whose equation is:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Penalised regression modifies this equation to include a penalty (the λ in the equation below), for the P coefficients in the model (β in the equation below).

$$SSE_{enet} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \lambda_1 \sum_{j=1}^{P} |\beta_j| + \lambda_2 \sum_{j=1}^{P} \beta_j^2$$

This technique of using both first- and second-order penalties is known as the *elastic net*. When only the first order term, λ_1, is used, it is known as the Least Absolute Shrinkage and Selection Operator method (*lasso*). When only the second order term, λ_2, is used, it is known as *ridge regression*.

In theory the penalization penalties, λ_1 and λ_2, are chosen by the analyst. In practice, software packages provide a function that automatically finds values (using the bootstrap) that minimise the error.

The lasso tends to pick one from each set of correlated variables and ignores the rest (by setting the corresponding βs to zero). Ridge regression has the effect of causing the coefficients, β, of the corresponding correlated variables to converge to a common value, i.e., the coefficient chosen for k perfectly correlated variables is $\frac{1}{k}$th the size chosen, had just one of them been used.

The calculation of mean squared error (MSE) adds contributions from both variance and bias. The default regression modeling techniques are unbiased, i.e., they attempt to minimise bias. It is possible to build models with lower MSE by trading off bias for variance (see Github–hardware/SPECpower.R for an example; in this case the use of penalised regression makes little difference to the final model).

11.5 Non-linear regression

The regression models fitted in earlier sections are linear models because the coefficients of the model (e.g., β_1 in the equation at the start of this chapter) are linear (the form of the explanatory variables is irrelevant). In a non-linear regression model one or more of the coefficients have a non-linear form, e.g., θ_1 in the following equation:

$$y = \alpha_1 + \beta_1 x^{\theta_1} + \varepsilon$$

Table 11.3 lists some commonly occurring non-linear equations, and figure 11.47 illustrates example instances of these equations.

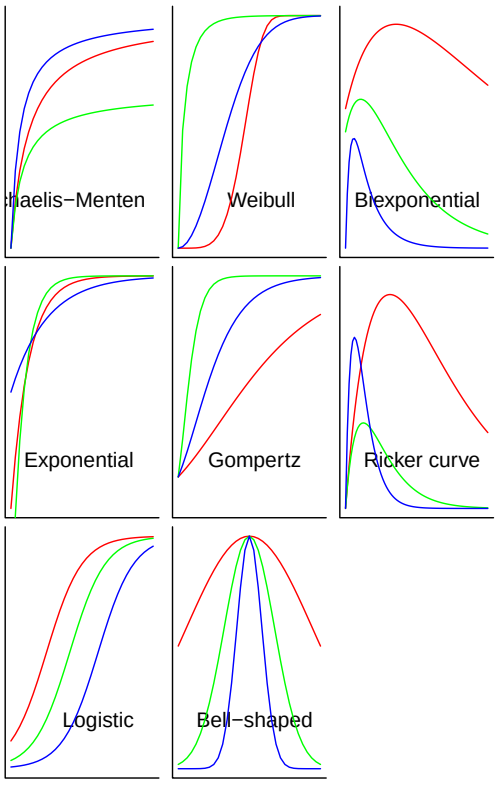


Figure 11.47: Example plots of functions listed in table 11.3. These equations can be inverted, so they start high and go down. Github–Local

The `nls` function (Nonlinear Least Squares) is part of the base system and can be used to build non-linear regression models; it requires that the response variable error have a Normal distribution (`glm`'s default behavior). The `gnm` package (Generalized nonlinear models) contains support for other forms of error distribution.

From the practical point of view there are several big differences between using `glm` and using `nls`, including:

Shape	Name	Equation
Asymptotic growth to a limit	Michaelis-Menten	$y = \frac{ax}{1+bx}$
Asymptotic growth to a limit	Exponential	$y = a(1-be^{-bx})$
S-Shaped	Logistic	$y = a + \frac{b-a}{1+e^{(c-x)/d}}$
S-Shaped	Weibull	$y = a - be^{-cx^d}$
S-Shaped	Gompertz	$y = ae^{be^{-cx}}$
Humped	Bell-shaped	$y = ae^{-\lvert bx\rvert^2}$
Humped	Biexponential	$y = ae^{-bx} - ce^{-dx}$
Humped	Ricker curve	$y = axe^{-bx}$

Table 11.3: Some commonly encountered non-linear equations, see figure 11.47.

- `nls` may fail to fit a model; the techniques used to find the coefficients of a non-linear model are not guaranteed to converge,
- `nls` may return a fitted model that differs from the actual solution; the techniques used to find the coefficients of a non-linear model may become stuck in a local minimum, that is good enough, and fail to find a better solution,
- `nls` often requires the analyst to provide estimate(s) for the initial value of each model coefficient, that is close to the final values (using the `start` argument),
- names for the model coefficients being estimated have to explicitly appear in the formula (i.e., implicit names are not created automatically),
- the operators appearing in the expression to the right of ~ have their usual arithmetic interpretation, i.e., the R formula specific behaviors listed in table 11.2 do not apply.

The biggest problem with fitting non-linear regression models, is finding a combination of starting values that are good enough for `nls` be able to converge to a fitted model. Possible techniques for finding these values include:

- using a "self-start" function, if available (e.g., `SSlogis` for Logistic models); these attempt to find good starting values to feed into `nls`, and functions, in turn, may require starting values (but at least there is a known method for calculating them),
- fitting a linear model that is close enough to the non-linear model and working with the coefficients of the fitted linear model as possible starting values,
- using the argument `trace=TRUE`, which outputs the list of model coefficients that are being used internally, as a source of ideas,
- picking a few points in the plotted data that a fitted line is likely to pass through and calculating values that would result in the equation being fitted, passing close to these points.

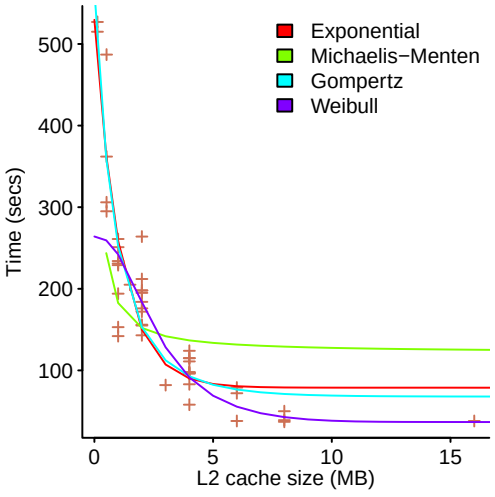


Figure 11.48: Time to execute a computational biology program on systems containing processors with various L2 cache sizes. Data kindly provided by Hazelhurst.[794] Github–Local

A study by Hazelhurst[794] measured the performance of various systems running a computational biology program. Figure 11.48 shows four non-linear equations fitted to one processor characteristic (L2 cache size). The calls to `nls` are as follows:[xxiii]

```
b_mod=nls(T1 ~ c+a*exp(b*L2), data=bench, start=list(a=300, b=-0.1, c=60))

mm_mod=nls(T1 ~ (1+b*L2)/(a*L2), data=bench, start=list(b=3, a=0.004))

gm_mod=nls(T1 ~ a/exp(b*exp(-c*L2)), data=bench,
                start=list(a=80, b=-1, c=0.1), trace=FALSE)
Asym = 0.0125
Drop = 0.002
lrc = -1.0
pwr = 2.5
# 1/SSweibull does not have the desired effect, so have to invert the response.
getInitial(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
wb_mod=nls(1/T1 ~ SSweibull(L2, Asym, Drop, lrc, pwr), data=bench)
```

At the start of this chapter, various linear models were fitted to the growth of Linux, see fig 11.7. Polynomials containing integer powers were used, perhaps the data is better fitted by a polynomial containing non-integer powers. The following call to `nls` attempts to fit such an equation, it uses starting values extracted from the quadratic model fitted earlier:

[xxiii] It is difficult to separate inspiration from suck it and see, in this process.

```
m1=nls(LOC ~ a+b*Number_days+Number_days^c, data=h2,
                        start=list(a=3e+05, b=-4e+2, c=2.0))
```

The summary and AIC output is: Github–Local

```
Formula: LOC ~ a + b * Number_days + Number_days^c

Parameters:
    Estimate Std. Error t value Pr(>|t|)
a -1.679e+05  2.969e+04  -5.656 2.61e-08 ***
b  7.319e+02  3.463e+01  21.131  < 2e-16 ***
c  1.806e+00  4.616e-03 391.211  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 231800 on 498 degrees of freedom

Number of iterations to convergence: 5
Achieved convergence tolerance: 4.299e-06

[1] "AIC =  13805.2100165816"
```

showing that the equation:

$$sloc = (-1.68 \cdot 10^5 \pm 3 \cdot 10^4) + (7.32 \cdot 10^2 \pm 3.5 \cdot 10^1) Number\_days + Number\_days^{1.81 \pm 4.6 \cdot 10^{-3}}$$

is a slightly better fit than a cubic equation (i.e., a lower AIC) and also predicts continuing growth (unlike the cubic equation).

It is possible that further experimentation will find a polynomial model with a lower AIC. However, the purpose of this analysis is to understand what is going on, not to find the equation whose fitted model has the lowest AIC.

A more practical issue is to create a model that makes what are considered to be more realistic future predictions. The growth in the number of lines in the Linux kernel will not continue forever, at some point the number of lines added will closely match the number of lines deleted. One commonly seen growth pattern, starts slow, has a rapid growth period, followed by a levelling off converging to an upper limit, i.e., an S-shaped curve. The Logistic equation is S-shaped and is often used to model this pattern of growth; the equation involves four unknowns (third row in table 11.3).

Fitting a Logistic equation the hard and easy (when it works) way:

```
# suck it and see...
m3=nls(LOC ~ a+(b-a)/(1+exp((c-Number_days)/d)), data=h2,
                start=list(a=-3e+05, b=4e+6, c=2000, d=800))
# no thinking needed, SSfpl works out of the box for this data :-)
m3=nls(LOC ~ SSfpl(Number_days, a, b, c, d), data=h2)
```

The AIC for the fitted Logistic equation is slightly worse than the cubic polynomial (13,273 vs. 13,220), but a lot better than the quadratic fit and it predicts a future trend that is likely to occur, eventually.

While the `predict` function includes parameters to request confidence interval and standard error information, support for both is currently unimplemented for models fitted using `nls`. The `confint` function, in the `MASS` package, when passed a model built using `nls`, returns the confidence intervals for each model coefficient; bootstrapping can also be used to find confidence intervals.

Figure 11.49 shows the fitted model predicting a slow down in growth, with the maximum being reached at around 10,000 days. Who is to say whether this prediction is more likely to occur, over the specified number of days, than the continuing increase predicted by the quadratic model? Given that the one explanatory variable used to fit the models, time, does not directly impact the production of source, it is no surprise that the predictions of future behavior made by the various models vary so wildly.

One technique for getting a rough idea of the accuracy of the future predictions made by a model, is to fit models to subranges of the data, and then check the predictions made against the known data outside the subrange. Figure 11.50 shows logistic equations fitted to subranges of the data, e.g., all data up to 2900, 3650, 4200 number of days and all days.

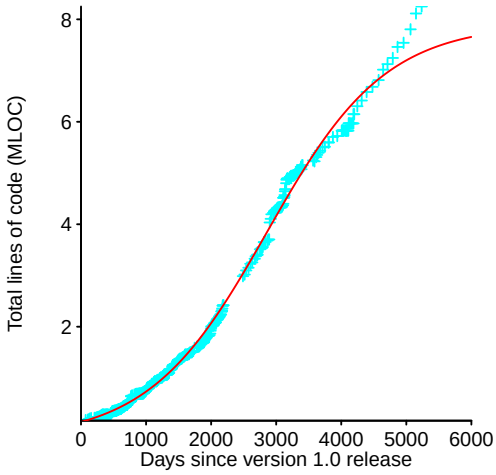


Figure 11.49: A logistic equation fitted to the lines of code in every non-bugfix release of the Linux kernel since version 1.0. Data from Israeli et al.[902] Github–Local

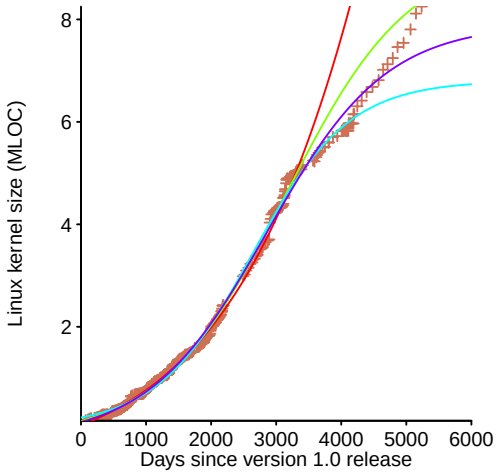


Figure 11.50: Predictions made by logistic equations fitted to Linux SLOC data, using subsets of data up to 2900, 3650, 4200 number of days and all days since the release of version 1.0. Data from Israeli et al.[902] Github–Local

The lesson to learn from figure 11.50 is to be careful what you ask for, asking for a logistic equation fitted to the data may get you one. The fitting process is driven by your expectations (in the form of a formula), and the data it is given.

The processes generating the data fitted by a Logistic equation may not in themselves follow this pattern, the contributions of independent processes may combine to create an emergent pattern. A study by Grochowski and Fontana[744] showed that increases in the density of data stored on hard disks could be viewed as a sequence of technologies that each rapidly improved, e.g., magneto-resistive and antiferromagnetically-coupled. Figure 11.51 shows the areal density (think magnetic domains) of various models of hard disk on first entering production. Improvements in each technology can be fitted with its own Logistic equation, as can the overall pattern of performance improvements.

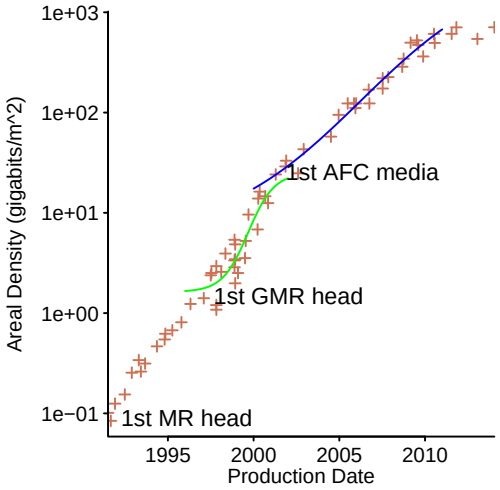


Figure 11.51: Increase in areal density of hard disks entering production over time. Data from Grochowski et al.[744] Github–Local

A codebase showing some evidence of having completed its major expansion phase is glibc, the GNU C library (i.e., its growth rate has levelled off); see figure 11.52. The `summary` of the fitted model is (the `SSfpl` function automatically estimates initial values for a Logistic equation): Github–Local

Plugging the fitted model coefficients into the Logistic equation give:

$$KLOC = -28 + \frac{1115-(-28)}{1+e^{(3652-Days)/935}}$$

Since these measurements were made, the C Standard's committee, JTC1 SC22/WG14, have started work on revising the existing specification; the model's prediction that glibc will max out at around 1,115,000 lines is unlikely to remain true for many more years.

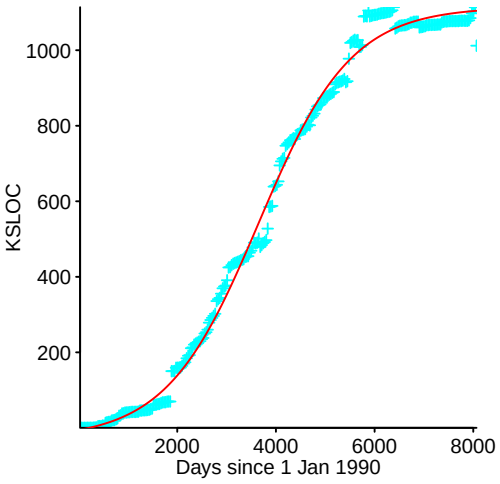


Figure 11.52: Lines of code in the GNU C library against days since 1 January 1990. Data from González-Barahona.[704] Github–Local

A study by Chen, Groce, Fern, Zhang, Wong, Eide and Regehr[345] investigated fault experiences in a C compiler and JavaScript engine, by having them process randomly generated programs. Some programs failed to be correctly processed (1,298 in gcc and 2,603 in Mozilla's SpiderMonkey), and many of these failures could be traced back to the same few underlying mistakes in the code, i.e., some fault experiences were encountered more often than others. Figure 11.53 shows the number of failing programs that could be traced back to the same mistake, the curved green line is a regression fit (a biexponential, or double exponential); the two straight lines are the exponentials that are added to form the bi-exponential.

The `nls` has a `SSbiexp` starter function, which performs poorly for this data (or, at least, your author could not make it do well).

The sample contains count data, with many very small values, implying a Poisson error distribution. The `gnm` function, in the `gnm` package, has an option to select an error distribution.

The formula notation used by `gnm` is based on function calls,[1856] rather than the binary operators used by `glm` and `nls`. The formula argument in the following call (used to fit the model plotted in figure 11.53), contains two exponentials (specified using the `instances` function), the literal `1` is a placeholder for an unknown constant multiplied (the `Mult` function) by an exponential (the `Exp` function); as with calls to `nls`, starting values are required:

```
library("gnm")

fail_mod=gnm(count ~ instances(Mult(1, Exp(ind)), 2)-1,
             data=wrong_cnt, verbose=FALSE,
             start=c(2000.0, -0.6, 30.0, -0.1),
             family=poisson(link="identity"))
```

See fig 6.25 for a discussion of one possible reason the biexponential is such a good fit.

Various natural processes can be modeled using a sum of (possibly) many exponentials, and specific techniques have been created to fit data to this specific non-linear case; some of these technique have the advantage of being able to operate with a very approximate starting estimates of the exponent.

The `mexpfit` function, in the `pracma` package, implements one such technique. Support is rudimentary, at the time of writing, but `mexpfit` can save a lot of time by providing a workable estimate for a call to `gnm`.

```
library("pracma")

me_mod=mexpfit(wrong_cnt$ind, wrong_cnt$count, p0=c(-0.9, -0.1))
print(me_mod)  # no summary support, at the time of writing
```

11.5.1 Power laws

Plotting values drawn from a power law distribution using a log scale for both axis, produces a straight line. This straight line characteristic is not unique to power laws, it can also appear to occur with samples drawn from other distributions, e.g., an exponential distribution;[xxiv] see section 7.1.3.

The `poweRlaw` package includes functions for fitting and checking whether a power law is likely to be a good fit for a sample.[371]

When the model being fitted contains one explanatory variable, thought to have the form of a power law, functions from the `poweRlaw` package can be used. However, this package does not support more complicated models, and so other regression modeling functions have to be used when a power law is one of multiple components in a model, e.g., `nls`.

A study by Queiroz, Passos, Valente, Hunsen, Apel and Czarnecki[1546] analysed the conditional compilation directives (e.g., `#ifdef`) used to control the optional features in 20 systems written in C. Researchers in this area use the term *feature constant* to denote macro names used to control the selection of optional features and *scattering degree* to describe the number of `ifdefs` that refer to a given feature constant, e.g., if the macro `SUPPORT_X` appears in two `ifdefs`, it has a scattering degree of two.

Figure 11.54 shows the total number of feature constants (y-axis) having a given scattering degree (x-axis) in these 20 systems, lines are a power law (red) and exponential (blue) of fitted models; the numbers are the p-values for the fit (higher is better, i.e., fail to reject the hypothesis). This analysis is a fishing expedition involving 20 systems, and a power law is suggested by the visual form of the plotted data; with multiple tests it is necessary to take into account the increased likelihood of a chance match.

If 0.05 is taken as the p-value cutoff, for one test, below which the distribution hypothesis is rejected, then $(1 - 0.95^{20}) \rightarrow 0.64$ is the cutoff when 20 tests are involved. Some systems have p-values above the cutoff for one of the power law or exponential fitted models, and so the given distribution is not rejected for these systems.

The `poweRlaw` package supports discrete and continuous forms of heavy tailed distributions. The scattering degree is an integer value, and the following code fits both a discrete power law and exponential to the data (the continuous forms are `conpl` and `conexp` respectively):

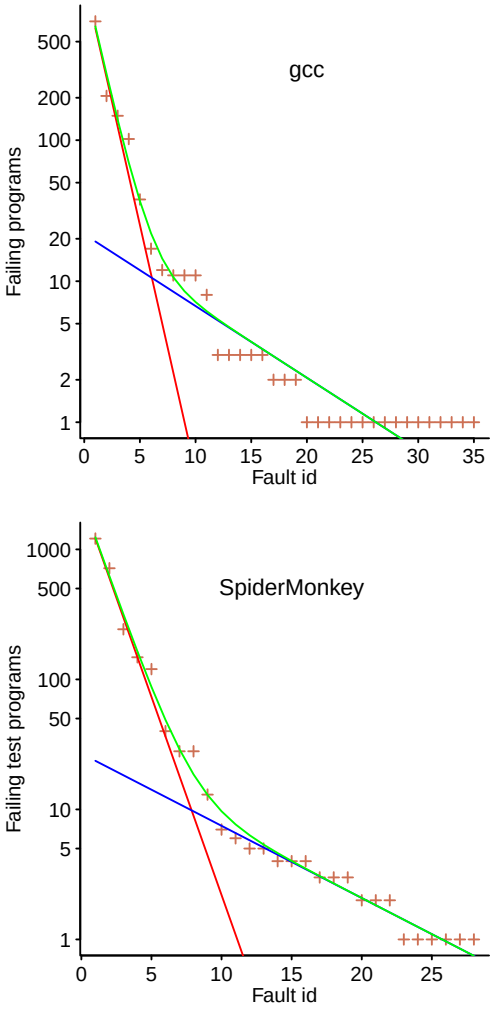


Figure 11.53: Number of failing programs caused by unique fault experiences in gcc (upper) and SpiderMonkey (lower). Fitted model in green, with two exponential components in red and blue. Data kindly provided by Chen.[345] Github–Local

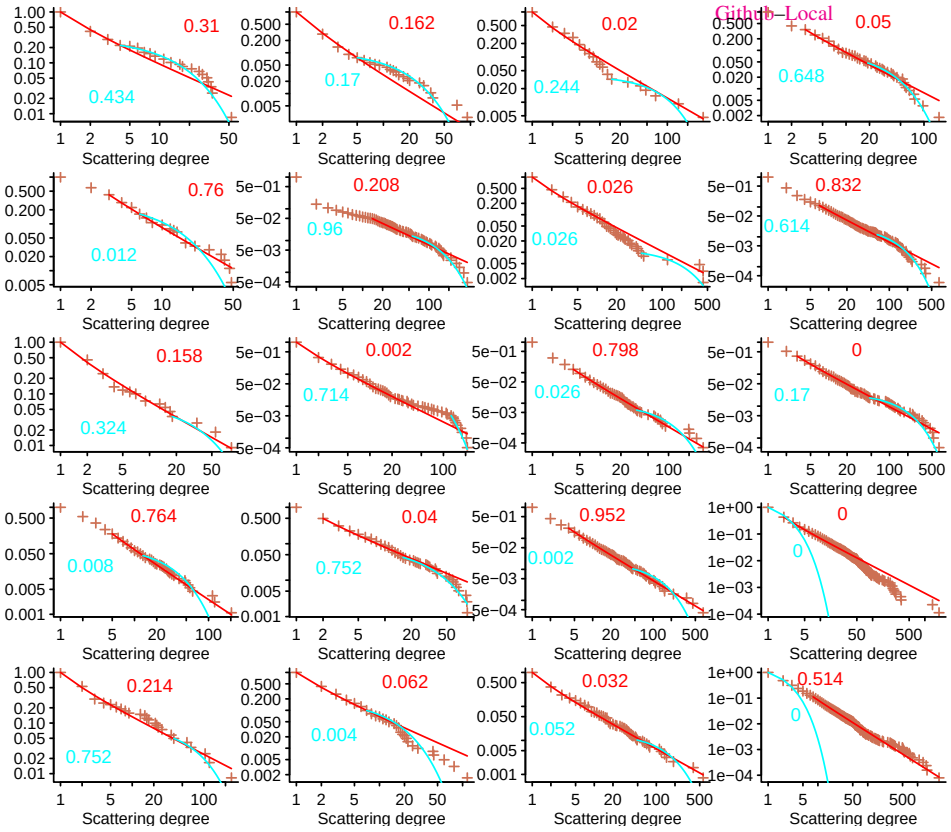


Figure 11.54: Power law (red) and exponential (blue) fits to feature macro usage in 20 systems written in C; fail to reject p-value for 20 systems is 0.64. Data from Queiroz et al.[1546] Github–Local

[xxiv] Papers[1163] claiming to have found a power law, purely on the basis of a plot showing points scattered roughly along a straight line, are a common occurrence.

```
library("poweRlaw")

# Fit scattering degree
# displ is the constructor for the discrete power law distribution
pow_mod=displ$new(FS$sd)
exp_mod=disexp$new(FS$sd)  # discrete power exponential

# Estimate the lower threshold of the fit
pow_mod$setXmin(estimate_xmin(pow_mod))
exp_mod$setXmin(estimate_xmin(exp_mod))

# Plot sample values
plot(pow_mod, col=point_col, xlab="Scattering degree", ylab="")
lines(pow_mod, col=pal_col[1]) # Plot fitted line
lines(exp_mod, col=pal_col[2])

# Bootstrap to test hypothesis that sample drawn from a power law
bs_p=bootstrap_p(pow_mod, threads=4, no_of_sims=500)
text(40, 0.5, bs_p$p, pos=2, col=pal_col[1]) # Display value
```

The power law equation includes a minimum value of x, scattering degree in this case, below which it does not hold. The `estimate_xmin` function estimates the value, x_{min}, that minimises the error between the fitted model and the data. The `new` function, called by the constructor, sets x_{min} to the minimum value present in the data. It is common for power laws to fit a subset of the data.

11.6 Mixed-effects models

Mixed-effects models are used to model measurements of multiple correlated measurements of the same subjects (e.g., before/after measurements of the same subject), and clusters of related subjects. The regression techniques discussed so far assume that measurements are not correlated with each other.

In a mixed-model the explanatory variables are classified as either a *fixed-effect*, or a *random-effect* (sometimes called a *covariate*). Technically the effects are not fixed and are not random[xxv]. One way to think about classifying the two kinds of explanatory variables, is to look at the impact they have on the response variable:

- fixed effects influence the mean value of the response variable, and are associated with the entire population,
- random effects influence the variance of the response variable, and are associated with individual subjects.

A study by Balaji, McCullough, Gupta and Agarwal[124] measured the power consumption of six different Intel Core i5-540M processors executing the SPEC2000 benchmark at various clock frequencies; the six processors are a sample of the entire population of Intel Core i5-540M processors. The power consumption characteristics might be modeled by combining the data from all six processors; the following is the `summary` output for this model: Github–Local

```
Call:
glm(formula = meanpower ~ frequency, data = power_bench)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5746  -0.1882   0.0413   0.1902   2.2965

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  2.12594    0.01506   141.2   <2e-16 ***
frequency    1.95248    0.00767   254.6   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

[xxv] Some authors point this out, and then proceed to use what they consider to be more technically correct terms, this book follows common usage because it is common; these terms crop up as named parameters in functions, and appear in output information.

```
(Dispersion parameter for gaussian family taken to be 0.1429928)

    Null deviance: 10692.7  on 9980  degrees of freedom
Residual deviance:  1426.9  on 9979  degrees of freedom
AIC: 8916.2

Number of Fisher Scoring iterations: 2
```

This model does not provide any information about how performance varies between processors. The identity of the processor measured could be included in the model (see Github–regression/hotpower-proc.R), but this is a model of the sample, and it cannot be used to deduce anything about the population from which it was drawn.

How might the sample of processors be modeled in a way that provides an estimate of population variability? Possible techniques include:

- building a regression model for each processor, and average these six models in some way, e.g., use the coefficients from each of the six models to build a regression model that is a model of models,

  Electronic circuit theory tells us that processor power consumption is proportional to clock frequency, and figure 11.55 shows the results of fitting a separate straight line to the data for each processor.
- building a *mixed-effects model*. A mixed-effects model (also known as a *hierarchical model*) might be viewed as a model of models; mathematically it uses a more direct approach, making more effective use of the available data than the method described above.

A number of different packages are available for fitting mixed-effects models, this book uses `lme4`, whose workhorse functions are the `glmer` and `lmer` functions.[xxvi]

The `lme4` package extends the formula notation to support the specification of random effects. In the following code:

```
library("lme4")

# Express in Gigahertz (otherwise lmer does not converge)
power_bench$frequency=power_bench$frequency/1000000

p_mod=lmer(meanpower ~ frequency + (1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency-1 | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (frequency   | processor), data=power_bench)
p_mod=lmer(meanpower ~ frequency + (1 | processor) + (frequency-1 | processor),
                                                              data=power_bench)
```

- first call to `lmer`: `frequency` is the fixed-effect and `(1 | processor)` is the random-effect; the `1` specifies variation in the intercept value, and the source of this variation is the `processor` variable, i.e., the column having this name in the data frame. When plotted the model might look something like the upper plot of figure 11.56, with six lines intersecting the y-axis at different points, but all having the same slope,
- second call to `lmer`: `(frequency-1 | processor)` specifies there is variation in the slope, and the source of this variation is the `processor` variable (this can also be written as: `(frequency+0 | processor)`). When plotted the model might look like the lines in the middle of figure 11.56, where all lines intersect the y-axis at the same point but have different slopes,
- third call to `lmer`: `(frequency | processor)` specifies that variation in the `processor` variable may cause both the intercept and the slope to vary, and the intercept and slope are correlated (can also be written as: `(1+frequency | processor)`). When plotted, the models might look like the lines in the lower plot of figure 11.56, where the lines have different intersections and slopes,
- fourth call to `lmer`: the operands `(1 | processor)+(frequency-1 | processor)` differs from the third call in that the intercept and slope are not correlated.

The following is the `summary` output from a mixed-effects model, where the processor is a random-effect on both the intercept and slope: Github–Local

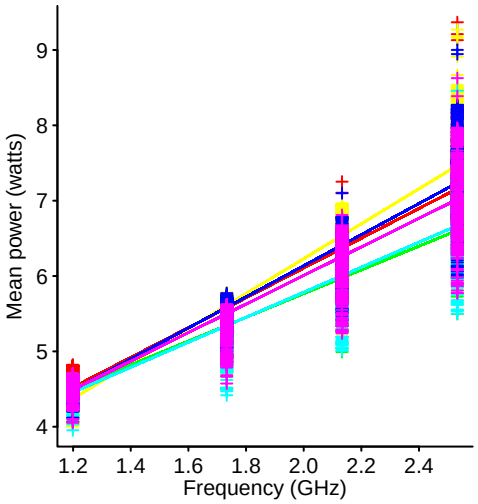


Figure 11.55: Power consumption of six different Intel Core i5-540M processors running at various frequencies; colored lines denote fitted regression models for each processor. Data from Balaji et al.[124] Github–Local

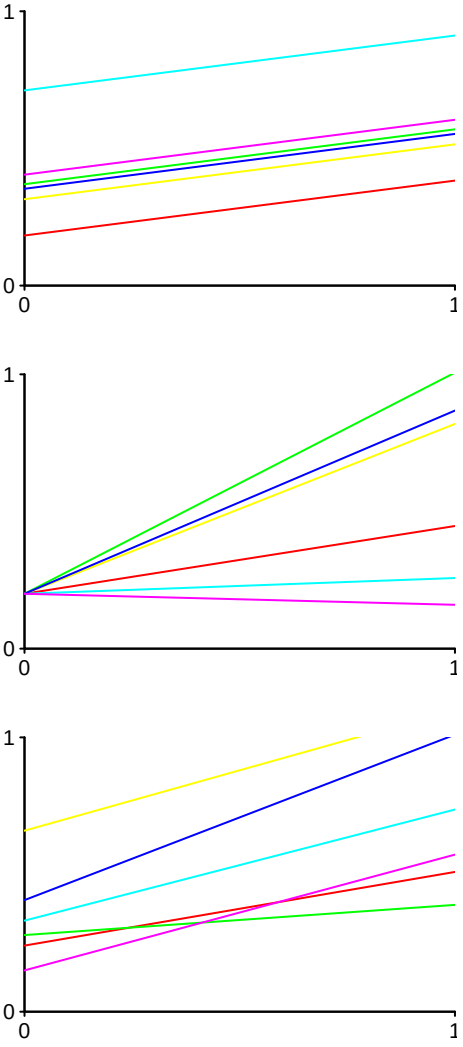


Figure 11.56: Example showing the three ways of structuring a mixed-effects model, i.e., different intersections/same slope (upper), same intersection/different slopes (middle) and different intersections/slopes (lower). Github–Local

[xxvi] A call to the `glmer` function that uses the default family distribution, i.e., gaussian, generates a warning that this usage is deprecated and `lmer` should be used.

```
Linear mixed model fit by REML ['lmerMod']
Formula: meanpower ~ frequency + (frequency | processor)
   Data: power_bench

REML criterion at convergence: 6300.3

Scaled residuals:
    Min      1Q  Median      3Q     Max
-4.0533 -0.4866  0.1453  0.4994  6.6744

Random effects:
 Groups    Name        Variance Std.Dev. Corr
 processor (Intercept) 0.13202  0.3634
           frequency   0.07552  0.2748   -0.99
 Residual              0.10941  0.3308
Number of obs: 9981, groups:  processor, 6

Fixed effects:
            Estimate Std. Error t value
(Intercept)   2.1740     0.1490   14.59
frequency     1.9156     0.1124   17.04

Correlation of Fixed Effects:
          (Intr)
frequency -0.993
convergence code: 0
Model failed to converge with max|grad| = 0.0314161 (tol = 0.002, component 1)
```

The values for `(Intercept)` and `frequency`, listed under `Fixed effects:`, are very similar to the combined data model fitted earlier. Annoyingly, the `summary` output does not include p-values. These can be obtained using the `Anova` function from the `car` package.

The `Random-effects:` table lists the variation introduced by `processor` (listed in the `Groups` column, on the variables listed in the `Name` column); the `Std.Dev.` column lists the estimated standard deviation in the corresponding coefficient listed in the `Fixed effects:` table. `Residual` lists the residual random effects left after taking into account all the specified random-effects.

As an example, taking `frequency`, there are two sources of uncertainty in its contribution to the response variable (as expressed in its model coefficient), one from fixed-effects, and a random-effect caused by the variation between processors.

Plotting the 95% confidence intervals, for the intercept and slope of a mixed-effects model, provides a visualization of the relative contribution of the sources of variation. Figure 11.57 was generated using the following code, with data from the six processors:

```
library("lattice")
library("lme4")
library("gridExtra")

proc_mod=lmer(meanpower ~ frequency +(frequency | processor),
                                                  data=power_bench)
dp_orig=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

power_bench$shift_freq=power_bench$frequency-min(power_bench$frequency)
proc_mod=lmer(meanpower ~ shift_freq +(shift_freq | processor),
                                                  data=power_bench)

dp_shift=dotplot(ranef(proc_mod, condVar=TRUE), main=FALSE)

# dotplot comes from the lattice package, which uses grid layout
grid.arrange(dp_orig$processor, dp_shift$processor, nrow=2)
```

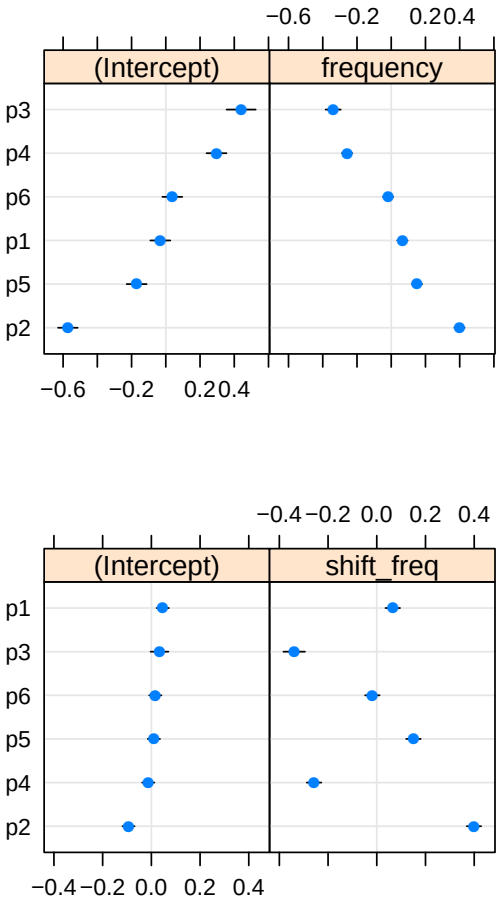


Figure 11.57: Confidence intervals, 95%, for first (upper) and second (lower) call to `lmer`; within-subject intercepts (left column) and slopes (right column) for the mixed-effects models in the adjacent code. Github–Local

Figure 11.57, upper plot, is the model fitted using the original data; the intercept (upper left) and slope (upper right) appear to be correlated. Looking at the straight line fits for each processor in figure 11.55, they appear to share an origin starting at the lowest frequency measured; an intercept included as a random effect has a common origin assumed to start at zero; see figure 11.56. Shifting frequency values down, by the minimum

measured value, and refitting a model produces the confidence intervals in the lower plot. The correlation has disappeared; perhaps including the intercept as a random effect is not worthwhile.

Refitting a model without the intercept as a random effect, produces a model that differs from previous models by a small amount; see Github–regression/hotpower-mix-plot.

There is an upper limit on the number of random effects (i.e., number of unknowns) that can occur in a model. The total number of unknown random effects must be less than the number of observations, otherwise the equations do not have a unique solution. A continuous explanatory variable counts as a single unknown, while a variable holding nominal or ordinal values contributes one unknown for each of the possible discrete values (there is no slope associated with fitting a variable that is not treated as being continuous).

The bootstrap can be used to calculate confidence intervals for a mixed-effects model.

11.7 Generalised Additive Models

The regression modeling techniques discussed so far have required the analyst to specify an equation expressing the detailed relationship between explanatory variables and the response variable (these are said to be *parametric models*). If no equation provides a reasonable fit, or accuracy of prediction is important (rather than understanding), then a *Generalised additive model* (GAM) is an alternative approach. A GAM only requires a list of explanatory variables and a response variable to be specified (these are said to be *nonparametric models*).

A GAM is built by finding the best fit for a sequence of polynomial equations (e.g., some form of spline), that smoothly captures the shape of the data. These smooth equations might be used to make predictions, or when the fitted model is plotted may suggest possible parametric equations. The details of the fitted equations are not a source of understanding, but they may make good predictions.

The `gam` function, in the `mgcv` package, can be viewed as extending the functionality of `glm` to support a variety of nonparametric smoothing functions (the `gam` package is simpler, but does not offer such a wide range of functionality). The following code shows formulas using a potentially different smoothing polynomial for each explanatory variable (first line below), a different smoothing polynomial for some combinations of explanatory variables (second and third line), a combination of a smoothing polynomial and parameterised form (fourth line), or an interaction between a smoothed and non-smoothed variable (fifth line; the `by` parameter, rather than the `:` operator is used):

```
mod=gam(y ~ s(x_1) + s(x_2) + s(x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3), data=foo_bar)
mod=gam(y ~ s(x_1) + s(x_2, x_3) + s(x_3, x_4) + s(x_4), data=foo_bar)
mod=gam(y ~ x_1 + s(x_2) + x_3, data=foo_bar, family="poisson")
mod=gam(y ~ x_1 + s(x_2, by=x_1) + x_3, data=foo_bar, family="poisson")
```

The smoothing function, `s`, supports a variety of options for controlling the fitting process; two that are likely to be encountered are `k`, which specifies an upper limit on the degrees of freedom that can be used in the fitted polynomial, and `bs`, a string identifying the kind of smoother, e.g., `"tp"`, the default, for a thin plate regression spline and `"cr"` for a cubic regression spline.

The value of `k` needs to be large enough to support the degrees of freedom needed by a polynomial capable of representing the underlying pattern in the data; the `gam.check` function provides information about fitted models that can be used to help select a value for `k`.

The fitting procedure used, by the `mgcv` version of `gam`, tries to avoid overfitting by making every degree of freedom pay its way (using, for instance, *penalized regression splines*). Criteria used for measuring the *cost-effectiveness* of more complicated models include generalised cross-validation (GCV; the default) and AIC. The `select` argument provides support for *null space penalization*, see package documentation for details.

A study by Lee and Brooks[1103] built a model to predict the performance and power consumed by applications running on processors having various hardware configurations, e.g., number of registers, size of cache and instruction latency.

The following additive model is based on the one proposed by Lee et al, and explains over 95% of the variance in the data; see Github–regression/lee2006.R. While this model is likely to be useful for prediction, it provides virtually no insight into the impact of various hardware attributes on performance characteristics.

```
l_mod=gam(sqrt(bips) ~ benchmark + fix_lat
                        +s(depth, k=4) + s(gpr_phys, k=10)
                        +s(br_resv, k=6) + s(dmem_lat, k=10) +
                                            s(fpu_lat, k=6)
                        +s(l2cache_size, k=5) + s(icache_size, k=3) +
                                           s(dcache_size, k=3)
                        +s(depth, gpr_phys, k=10)+s(depth, by=width, k=6)
                        +s(gpr_phys, by=width, k=10)
                  , data=lee)
```

The analysis associated with figure 8.33 used two approaches to modeling the number of accesses to a function's local variables. Without knowing anything about what relationships might exist between explanatory and response variables, and being willing to use very high degree polynomials, it is possible to build and use `gam` to build a prediction model.

In the calls to `gam` below, the first assumes there is an interaction between the two explanatory variables (allowing up to 75 degrees of freedom), and the second assumes the variables are independent (allowing up to 50 degrees of freedom for each of them). While the fitted model might make usable predictions, the use of such high degree polynomials suggests that the underlying processes have a non-polynomial form; see Github–sourcecode/local-use/obs-fit.R.

```
locg_mod=gam(norm_occur ~ s(object.access, total.access, k=75),
                                    data=common_loc, family=Gamma)

locp_mod=gam(norm_occur ~ s(object.access, k=50)+s(total.access, k=50),
                                    data=common_loc, family=Gamma)
```

11.8 Miscellaneous

Topics that your author has had to deal with, from time to time.

11.8.1 Advantages of using `lm`

This book promotes `glm` as a one-stop solution, however, the `lm` function has some advantages over `glm`, including:

- requiring less cpu time to fit a model. If many models need to be fitted on a regular basis, the performance difference may be worth considering,
- requiring less memory to fit a model. For extremely large datasets, memory requirements may be excessive for `glm`; possible solutions that continue to use `glm` are discussed below,
- the algorithm used by `lm` is always guaranteed to converge to a solution, singularities generated by a correlation between explanatory variables excluded. There are edge cases where `glm` does not find a solution without being given some reasonable starting values.

The implementation of `lm` is based on the mathematics of *Ordinary Least Squares* (OLS), and the data has to satisfy additional conditions for OLS to be applicable. Perhaps the most important new condition is that the error variance in the measurements be constant (in practice close to constant is usually good enough). The `ncvTest` function, in the `car` package, checks that a fitted model meets this requirement; the `spreadLevelPlot` function provides some visualization; also, see the `lmtest` function.

A user interface issue with models fitted using `glm` is that they do not come with a scale-invariant goodness of fit number, i.e., the R-squared value.

11.8.2 Very large datasets

The `biglm` package supports fitting regression models using data that is too large to fit in memory all at once; the models are built using an incremental algorithm, which only requires a subset of the data to be held in memory at any time. A variety of options are available for creating chunks of data to feed into the model building process, including incremental reading from files and databases.

The `biganalytics` package extends the `bigmemory` package by providing interfaces to various analytic packages, such as `biglm`; see Github–benchmark/bounds_chk.R.

11.8.3 Alternative residual metrics

The error metric used by many regression techniques is based around squaring the difference between the actual and predicted value. This choice has been driven by the theoretical usefulness of the mathematical properties of sum-of-squares. Other error metrics are available to fit models, e.g., the absolute difference between actual and predicted values.

The `rlm` function, in the `MASS` package, supports analyst specified functions for calculating the residual to be minimised when fitting a model. The `robustbase` and `robust` packages support a wide variety of functionality.

11.8.4 Quantile regression

The techniques discussed up to this point are based around predicting the expected value of the mean. Quantile regression is based on the proportion of data points above/below the fitted equation; it is robust to the presence of outliers, and is not influenced by the form of the error distribution.

The `rq` function, in the `quantreg` package, fits quantile regression models. Figure 11.58 was generated using the following code (also see fig 8.13):

```
library("quantreg")

quant_fit=function(tau_val, col_str)
{
rq_mod=rq(log(SLOC) ~ log(Files), data=proj_inf, tau=tau_val) # tau is the quantile
pred=predict(rq_mod, newdata=data.frame(Files=x_bounds))

lines(log(x_bounds), pred, col=col_str)

return(rq_mod)
}

plot(log(proj_inf$Files), log(proj_inf$SLOC),
        col=densCols(log(proj_inf$Files), log(proj_inf$SLOC)), pch=20,
        xlab="log(Files)", ylab="log(SLOC)\n")

x_bounds=exp(seq(0, log(1e5), by=0.1))

rq05_mod=quant_fit(0.05, pal_col[3]) # specify quantile and color
rq50_mod=quant_fit(0.5, pal_col[1])
rq95_mod=quant_fit(0.95, pal_col[2])
```

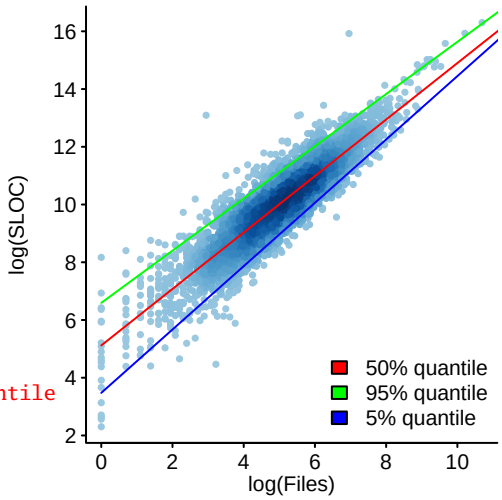


Figure 11.58: Number of files and lines of code in 3,782 projects hosted on Sourceforge; lines are 95%, 50% and 5% quantile regression fits. Data from Herraiz.[817] Github–Local

A line fitted to the 50% quartile has half the measurement points below/above it, while the 95% quartile line divides the measurements such that 95%/5% are below/above (the division of measurements for the 5% quartile is reversed).

11.9 Extreme value statistics

Extreme value statistics deals with the probability of occurrence of extreme values, e.g., use of maximum memory available memory, or minimum response time. The two main techniques are Generalized Extreme Value (GEV) and Generalized Pareto (GP); the `ext Remes` package supports both techniques.

The GEV approach analyses each maximum value that occurs in a specified interval (e.g., maximum daily fault reports within each month), while the GP approach analyses all values above a specified threshold value, e.g., all program runs taking longer than x seconds. The equation fitted by each approach both contain three parameters: μ (the mean value for GEV, and the threshold for GP), σ a multiplier that scales the function, and ξ (greek lower-case xi) a shape parameter (depending on whether ξ is equal/greater/less than zero, the equations simplify to more well-known distributions; Gumbel, Fréchet and Weibull respectively for GEV, and Exponential, Pareto and Beta for GP).

Some of the WG21 (the ISO C++ Standard working group) email reflectors receive a lot of traffic, particularly the Core and Lib reflectors. What is the maximum number of messages on one day that is likely to occur within a 10-year period?

Roger Orr[xxvii] kindly extracted the date of every message posted since February 2016 (configuration changes over the years make it non-trivial to obtain data before this date) to the Core and Lib mailing list.

The `fevd` function, in the `extRemes` package, calculates the parameters for the extreme value distribution that best fits the data. When using GP a threshold has to be chosen, and the `threshrange.plot` can be used to help select a value.

The default value of options assume stationary data (i.e., the mean does not change over time); an equation can be given for each model parameter specifying how it changes with time.

```
library("extRemes")

max_mod=fevd(month_max$V1, type="GEV", period.basis="month")
plot(max_mod, rperiods=c(6, 12, 18, 36, 72, 120), type="rl", col="red", main="")

summary(max_mod)
```

Figure 11.59 shows a GEP fitted model for the maximum number of daily emails expected to occur (y-axis) within a given number of months (x-axis), for WG21's the Lib email list; the pluses are actual occurrences, and dashed lines 95% confidence intervals.[xxviii]

The model used is very simplistic, and does not take into account the growth in members joining these lists and traffic lost when a new mailing list is created for a new committee subgroup.

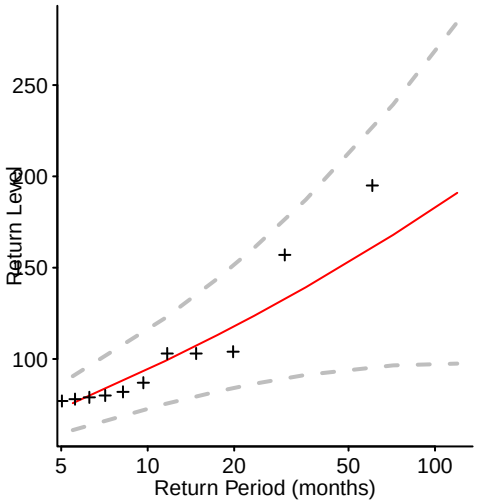


Figure 11.59: Expected maximum number of daily emails to the C++ lib email list expected to occur within a given period of months, with 95% confidence intervals; a GEP fitted model (corresponding `plot` function does not provide any user interface options). Data kindly extracted from the WG21 mailing list archive by Roger Orr. Github–Local

11.10 Time series

Time series analysis deals with measurements that are sequentially correlated. An example of correlated measurements is current room temperature, which is likely to be similar to the temperature 10 minutes ago, and the temperature 10 minutes from now. Techniques developed to analyse time-series can be used to analyse measurements of any quantity, where a correlation exists between successive measurements.

The base system provides basic functions for analyzing time series of continuous values.

A time series contains one or more of the following three components:

- underlying trend: which changes slowly,
- regular recurring pattern of changes (known as *seasonality*): for instance, expected daytime temperature throughout the year,
- random, irregular or fluctuating component.

The `stl` function (Seasonal Trend using Lowess) provides a way of splitting a time series into these three components (the argument must be an object of type `ts`, with a user specified frequency; the `stl` function does not automatically detect the recurrence period), and there is a corresponding `plot` function.[xxix]

Figure 11.60, from a study by Eyolfson, Tan and Lam,[566] shows the three time-series components of the hourly rate of commits to the Linux kernel source tree, over the days of a week (the commits during the same hour of the same day were summed). The `stl`

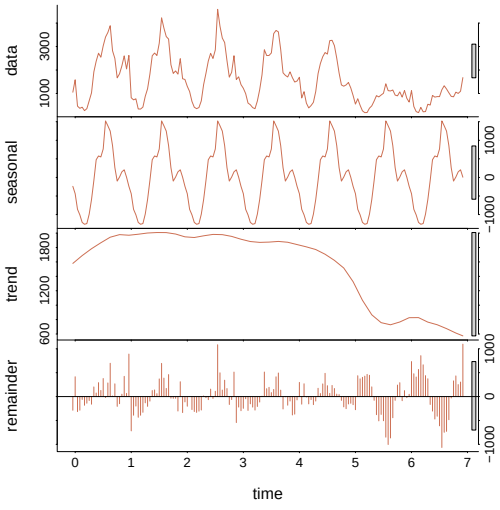


Figure 11.60: The three components of the hourly rate of commits, during a week, to the Linux kernel source tree; components extracted from the time series by `stl`. Data from Eyolfson et al.[566] Github–Local

function assumes a fixed, recurring, pattern of seasonal behavior, a slowly changing trend, with everything else classified as random noise.

```
# A seasonal frequency has to be specified
hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)
plot(stl(hr_ts, s.window="periodic"))
```

Possible outputs from time-series analysis include:

- a fitted model specifying how the value of a quantity at time t depends on its values at earlier measurement times (often at $t-1$),
- a regression model, adjusted for the correlation between sequential measurements,
- a power spectrum showing the dominant frequencies present in the data,
- a hierarchical clustering of multiple time series,
- a list of patterns, motifs, that occur within a time series.

Structure is often added to the linear nature of time by imposing repeating fixed length intervals, such as hours of the day and days of the week. Many time series analysis techniques require measurements to be made at fixed length intervals; analysis of measurements at irregular intervals is not discussed here.

Some library functions use a time series datatype for representing time related measurements. The `ts` function, part of the base system, converts a vector to class `ts` (many time series functions will automatically convert vectors to this class).

The `xyplot` function, in the `lattice` package, can be used to create a time series strip chart, see fig 8.19.

11.10.1 Cleaning time series data

Many time series techniques implicitly assume that measurement data occurs at regular intervals. A measurement process may only record events when they occur and if no event occurred in within an interval there may be no data-point for that interval. The cleaning process includes ensuring that every interval contains a value (which may be zero or inferred from surrounding values).

A study by Buettner[275] gathered project staffing information for several commercial development software projects. On large commercial projects the amount of work done at weekends is likely to be zero (except for the weeks prior to major deliveries), and the autocorrelation of project activity is likely to show a recurring pattern involving two consecutive days separated by seven days, i.e., weekends and weekdays.

Figure 11.61 shows the autocorrelation of the number of defects found on a given day, for one development project. The seven-day recurring pattern contains a three consecutive day pattern, are the developers only working a four-day week? It turns out [xxx] that contractors on some projects work a two-week cycle, with extra hours worked one week and then not working the Friday of the following week. The extent to which regular staffing level differences, between Friday and other weekdays, has to be taken into account, will depend on the kind of analysis performed (weekends can be handled by excluding them from the analysis, focusing on where most effort occurs, i.e., week days).

Measurements made on public holidays, such as the New Year, are very likely to differ from normal work days. Removing public holidays from the data will scramble the association with day of the week. The extent to which day of the week is a more important factor in the analysis, than public holidays, has to be considered.

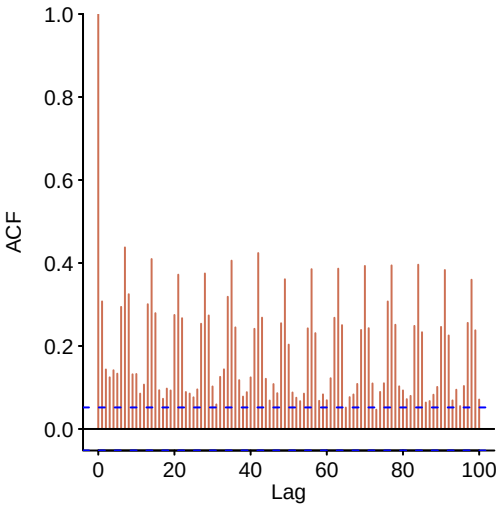


Figure 11.61: Autocorrelation of number of defects found on a given day, for development project C. Data kindly provided by Buettner.[275] Github–Local

11.10.2 Modeling time series

The expected mean of a time series can be modeled using one or both of the following two approaches (series whose variance is serially correlated are discussed later):

[xxvii] Roger is the convenor of the UK's BSI C++ panel.
[xxviii] The `plot` function supports very few visual presentation configuration options.
[xxix] The `decompose` function, part of the base system, implements the same functionality in a less sophisticated way.
[xxx] Email discussion with Buettner.

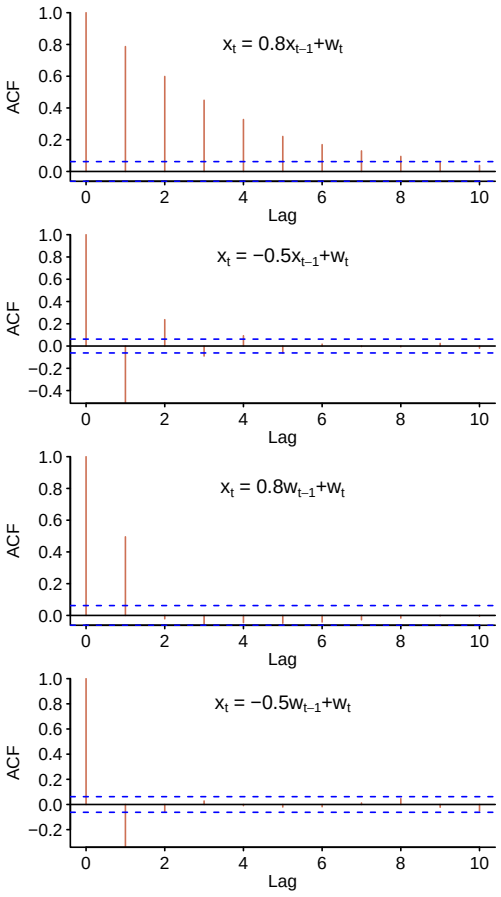


Figure 11.62: Autocorrelation of two AR models (upper plots) and two MA models (lower plots); the same models are used in figure 11.63. Github–Local

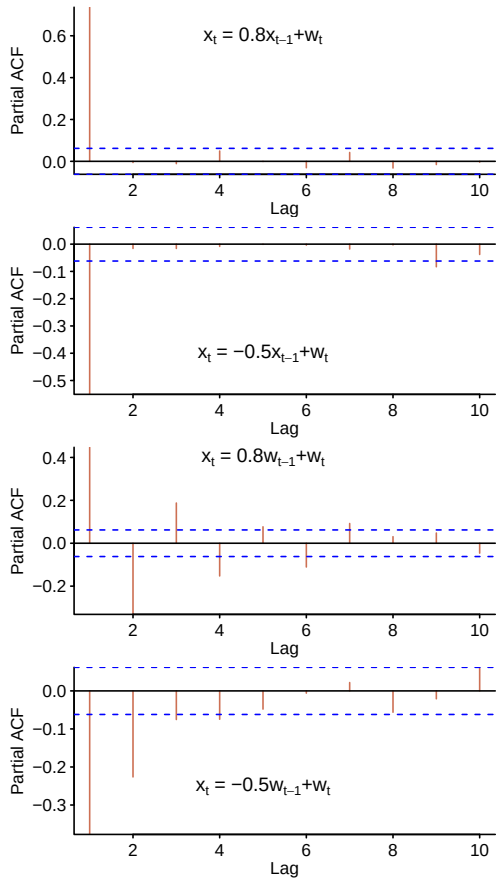


Figure 11.63: Partial autocorrelation of two AR models (upper plots) and two MA models (lower plots); the same models are used in figure 11.62. Github–Local

- the *Autoregressive model* (AR), models the value at time t as a weighted combination of values from earlier time steps, plus some amount of added noise, w_t, for instance:

  $x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t$

  is an autoregressive model of order 2, abbreviated AR(2); it is based on values going back two time steps (with weights ϕ_1 and ϕ_2).

  The `ar` function fits data to an autoregressive model.

- the *Moving Average model* (MA), models the value at time t as the sum of noise, w_t, and a weighted combination of the noise from earlier time steps, for instance:

  $x_t = w_t + \theta_1 w_{t-1}$

  is a moving average model of order 1, abbreviated MA(1); it uses a value from one time step back (with weight θ_1).

  The `arima` function, with the first two values of the `order` argument set to zero, fits data to a moving average model.

The autocorrelation function, `acf`, returns and plots the correlation of a time series with itself at successive lag intervals (i.e., the correlation of the measurement at time t with the measurement at time $t+n$; the default sequence of lags is `n=1:25`); see figure 11.62. For the AR(1) model, $x_t = \phi x_{t-1}$, the impact of serial correlation on values separated by k lags (time intervals) decreases by ϕ^k.

The lag 0 autocorrelation is always one, and the two dotted blue lines are 0.05 p-value bounds. Each lag is a hypothesis test, and with 25 hypothesis tests (the default) at least one calculated value is expected to exceed a 0.05 p-value with probability $1-0.95^{25} \to 0.72$; also, successive measurements are correlated, so neighbouring lag points are likely to show similar significance levels.

The partial autocorrelation function (the `pacf` function) calculates and plots the correlation at lag k, after removing the effect of any correlation generated by terms at shorter lags; see figure 11.63. The partial autocorrelation at lag k is the k^{th} coefficient of an AR(k) model.

The previous two plots illustrate how short range correlations in an AR model have a long range impact on the values returned by `acf`, but an MA model does not have a long range impact, while the opposite behavior is seen in the values returned by `pacf`. An ARMA model always behaves in the most unhelpful way.

An ARMA model (*Autoregressive Moving Average*) is a combination of an AR and MA model, e.g., ARMA(2, 1) is the sum of an AR(2) and MA(1) model, such as the following:

$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + w_t + \theta_1 w_{t-1}$

The `ARMAacf` function takes a specification of an ARMA model, and returns what `acf` would return when passed a time series following this model (the `pacf=TRUE` option switches the behavior to that of `pacf`).

A time series is said to be *stationary* if the expected mean value does not change over successive measurements, i.e., $E[t_i] = E[t_{i+k}]$. The mathematics behind both the basic AR and MA model fitting techniques assume a stationary time series (more sophisticated techniques are available; ARIMA (*Autoregressive Integrated Moving Average*) handles some non-stationary time series: supported by the `arima` function).

Many software engineering processes include non-stationary components, e.g., varying number of developers working on a project, increasing number of customers, system updates, etc.

Time series analysis techniques are not limited to measurements involving time, they can be applied to any data that has serial correlation between measurements.

A study by Hindle, Godfrey and Holt[832] investigated the indentation of the first non-whitespace character on a line, for code written in a variety of languages. Figure 11.64 shows the autocorrelation of a list, ordered by indentation, of the total number of lines having a given indentation.

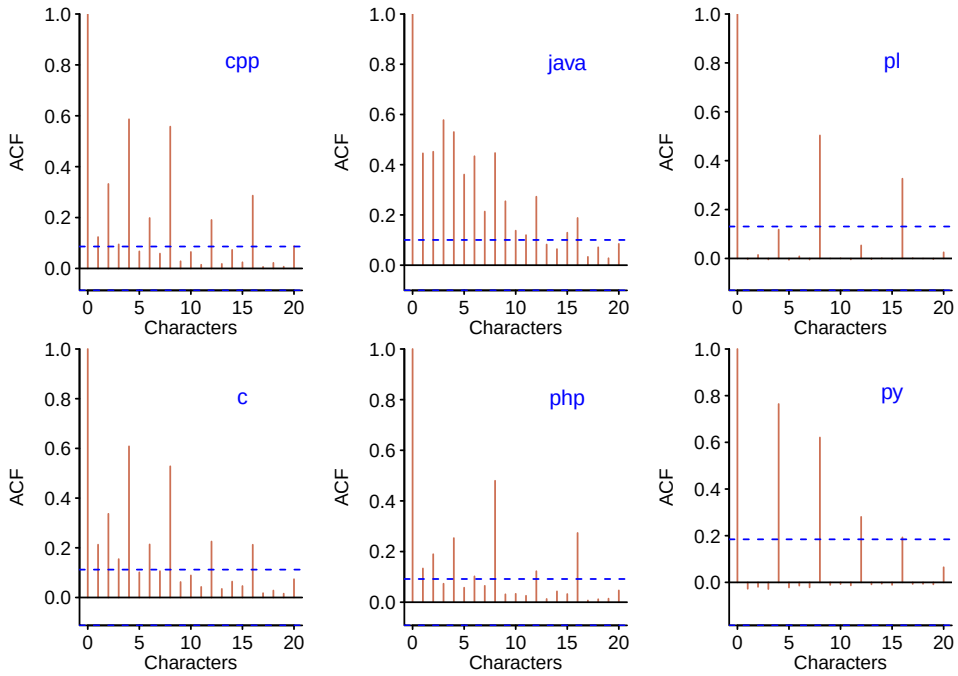


Figure 11.64: Autocorrelation of indentation of source code written in various languages. Data from Hindle et al.[832] Github–Local

11.10.2.1 Building an ARMA model

ARMA modeling takes as input a time series; if this time series is non-stationary, it has to be converted to a stationary form before model building can begin. Common reasons for a time series not being stationary and possible transforms to a stationary series include:

- a non-zero trend: for instance, the following equation contains an increasing time dependent trend:

  $x_t = \alpha + \beta t + w_t$

  Differencing can be used to remove trends, but care needs to be taken because this can introduce signals that are not in the original data. For instance, differencing the above equation gives:

  $\Delta x_t = x_t - x_{t-1} = b + w_t - w_{t-1}$

  an MA(1) process, which the original series does not contain.

  Subtracting the trend $\alpha + \beta t$ leaves just w_t; see the lower plot of figure 11.65,

- non-constant variance; known as *volatility* in the analysis of financial time series.

  If the growth in variance, over time, approximately follows the growth of the mean (i.e., a relatively consistent percentage change at each time step, e.g., $y_t = (1+x_t)y_{t-1}$), then a log transform produces a time series with approximately constant variance (i.e., $\Delta(\log y_t) \approx x_t$, assuming $\log(1+x_t) \approx x_t$).

  A log transform requires special processing of any zero values; possible solution include adding a small amount to every value [xxxi] and setting non-finite log-transformed values to zero (both have some impact on a fitted regression model). The 7digital data has increasing variance (more developers are employed and time to implement features decreases), and many zeroes; see Github–time-series/agile-day-starts.R.

- seasonality: this is a cyclic trend, e.g., changes recurring every year. Implementations of ARMA often include support for including a seasonal component in the model, e.g., the `seasonal` to the `arima` function,

The Augmented Dickey-Fuller test is a well known technique for checking whether a time series is stationary, others include the Phillips-Person test and the KPSS-test (supported by the `adf.test`, `pp.test` and `kpss.test` functions in the `tseries` package). These tests all have low power (i.e., fail to detect that a time series is non-stationary; they all fail to detect that the untransformed 7digital data is not stationary, and sometimes give contradictory results; see Github–time-series/agile-day-starts.R.

The plots produced by `acf` and `pacf` provide useful information about the likely structure and order of an ARMA model:

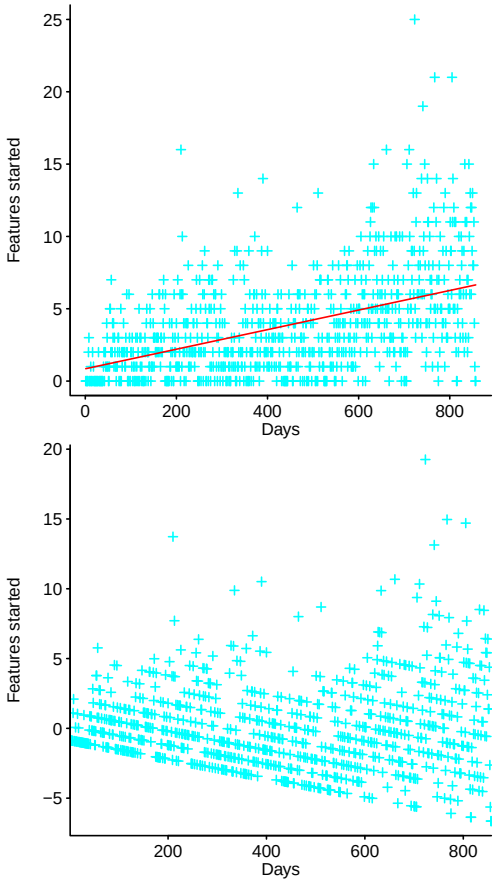


Figure 11.65: Number of features started for each day and fitted regression trend line (upper) and number of features after subtracting the trend (lower). Data kindly supplied by 7Digital.[1] Github–Local

[xxxi] The value should not be so small that its log is a large negative value.

- if the `acf` plot shows a decreasing trend, while the `pacf` shows a sharp cut-off (see figure 11.62), an AR model is a good place to start,
- if the `acf` plot shows a sharp cut-off, while the `pacf` shows a decreasing trend (see figure 11.63), an MA model is a good place to start,
- if both plots show a decreasing trend, then some combination of AR and MA model is likely to be needed.

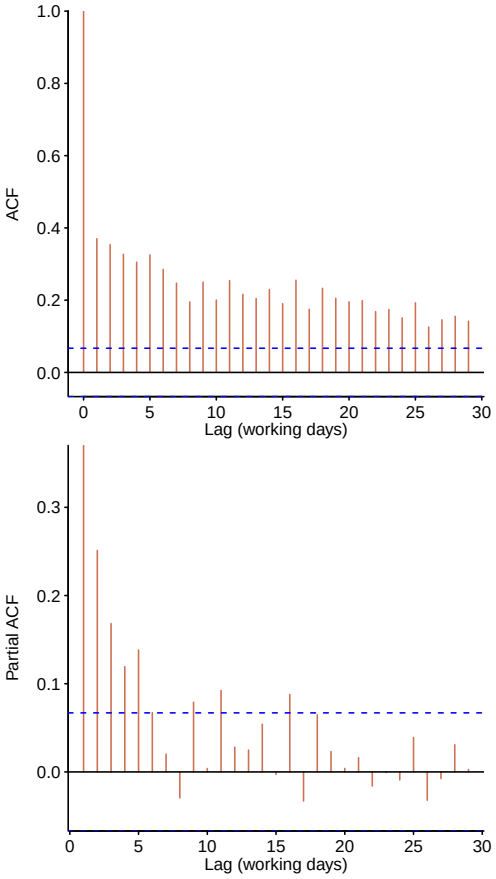


Figure 11.66: Autocorrelation (upper) and partial autocorrelation (lower) of the number of features started on a given day (after differencing the log transformed data), over the entire period of the 7digital data. Data kindly supplied by 7Digital.[1] Github–Local

Figure 11.66 was produced using the following code:

```
lwd=log(weekdays+1e-1)  # handle days with zero values

acf(lwd, xlab="Lag (working days)")
pacf(lwd, xlab="Lag (working days)")
```

Models fitted, by calling `arima` with various values of its `order` argument, can be compared using AIC (`arima` only returns the series mean, when a difference value of zero is passed to `order`; the `Arima` function, in the `forecast` package, is not limited in this way). The `auto.arima` function, in the `forecast` package, can be used to automatically find ARIMA model values that minimise AIC.

```
library("forecast")

arima(lwd, order=c(5, 0, 1))
auto.arima(lwd, max.order=7)
arima(lwd, order=c(1, 0, 1))
```

The two best fitting models, for the feature start data, are ARMA(5, 1) and ARMA(1, 1). The output from the last call to `arima` above is: Github–Local

```
Call:
arima(x = lwd, order = c(1, 0, 1))

Coefficients:
         ar1      ma1  intercept
      0.9627  -0.7993      0.561
s.e.  0.0158   0.0380      0.241

sigma^2 estimated as 1.789:  log likelihood = -1465.67,  aic = 2939.35
```

The `Coefficients:` table lists the model coefficients and their standard error. The `intercept` column is the mean value of the time series. The equation for one of the models is:

$$x_t - 0.5610 = 0.9627(x_{t-1} - 0.5610) + w_t - 0.7344 w_{t-1}$$

which simplifies to:

$$x_t = 0.5610(1 - 0.9627) + 0.9627 x_{t-1} + w_t - 0.7993 w_{t-1}$$

The constant (log transformed) increment per time step is: $0.5610(1-0.9627) \rightarrow 0.0209253$.

Which of these two models provides the better explanation of the data? Features take different amounts of time to implement, and work can only start on a new feature when enough people have been freed up, through completion of work on other features. The coefficients of the AR component, of the ARMA(5, 1) model, can be interpreted as a probability that people working on a feature started a given number of days earlier will become available to start work on a new feature; see table 11.4.

	AR	Duration
ar1	0.19	0.32
ar2	0.11	0.16
ar3	0.09	0.11
ar4	0.07	0.07
ar5	0.10	0.05

Table 11.4: AR coefficients of ARMA(5, 1) model and percentage of features taking a given number of days to implement. Data kindly supplied by 7Digital.[1] Github–Local

This may be a just-so story, but stories are useful tools, but your author cannot think of one for the ARMA(1, 1) model.

Handling seasonal trends: A seasonal ARIMA model can include AR, difference and MA components at an offset equal to the number of measurement intervals in the season. By default, the `auto.arima` function, in the `forecast` package, will return seasonal components (if any are found). The `seasonal` option can be used to specify seasonal components to the `arima` function.

The following code estimates a seasonal ARIMA model, for hourly commits to the Linux kernel source tree (see fig 11.60):

```
library("forecast")

hr_ts=ts(linux_hr, start=c(0, 0), frequency=24)

auto.arima(hr_ts)
arima(linux_hr, order = c(2,1,1), seas = list(order = c(1,0,1), period=24))
```

The coefficients of the first and second fitted models, below, differ because of differences in the algorithms used by the functions that fitted them, but are within each other's standard error (the third set of coefficients is for a slightly simpler model): Github–Local

```
Series: hr_ts
ARIMA(2,1,2)(1,0,0)[24] with drift

Coefficients:
         ar1      ar2     ma1    ma2    sar1    drift
      -0.892  -0.5348  0.5087  0.221  0.6751  13.0240
s.e.   0.244   0.1621  0.2694  0.223  0.0708  50.1094

sigma^2 estimated as 143870:  log likelihood=-1233.06
AIC=2480.12   AICc=2480.83   BIC=2501.95

Call:
arima(x = linux_hr, order = c(2, 1, 1), seasonal = list(order = c(1, 0, 1),
    period = 24))

Coefficients:
          ar1      ar2     ma1    sar1     sma1
      -0.8124  -0.2862  0.4483  0.8909  -0.4516
s.e.   0.2995   0.1177  0.3058  0.0546   0.1404

sigma^2 estimated as 129070:  log likelihood = -1229.02,  aic = 2470.03

Call:
arima(x = linux_hr, order = c(2, 1, 0), seasonal = list(order = c(1, 0, 1),
    period = 24))

Coefficients:
          ar1      ar2    sar1     sma1
      -0.3632  -0.1174  0.8980  -0.4727
s.e.   0.1007   0.0964  0.0519   0.1391

sigma^2 estimated as 129942:  log likelihood = -1229.71,  aic = 2469.42
```

The `sar1` is the seasonal AR coefficient and `sma1` the seasonal MA coefficient.

The output from `auto.arima` is a suggested model. In this case the `ar` and `sar` coefficients are pulling in opposite directions, and the standard error for the `ma1` coefficient is very high. Removing the MA component produces a model (second call to `arima` above), where the coefficients are not almost cancelling each other out; the model is (24 is the seasonal period):

$$x_t = -0.4x_{t-1} - 0.1x_{t-2} + 0.9x_{t-24\times1} - 0.5w_{t-24\times1}$$

What happened 24 hours ago contributes more to the predictor, than what happened in the previous hour or two.

Predictions made using a fitted ARMA model: A fitted ARIMA model can be used to predict what may occur after a measurement at time t; the relatively large noise component present in some ARMA models means that the confidence bounds of the predicted values may quickly become very wide. The R's system supports a `predict` function that accepts models fitted by the `arima` function; the `Arima` and `forecast` functions, from the `for`

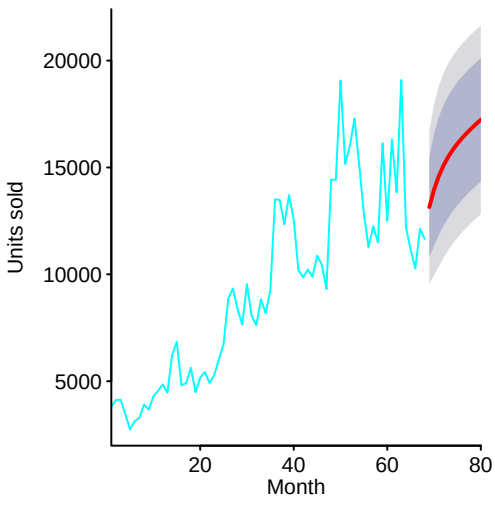


Figure 11.67: Monthly sales of spreadsheets in the UK, starting January 1987, with 12-months of sales predictions (shaded light blue are 80% confidence intervals, grey shaded 95%). Data from Givon et al.[684] Github–Local

ecast package, support more options for fitting and forecasting of time-series data. In the following code, the include.drift=TRUE option specifies that trend information be included in the model; for this data, the increasing volume of sales generates an increasing trend:

```
library("forecast")

Ar_mod=Arima(data$Spreadsheets, order=c(1, 0, 1), include.drift=TRUE)

f_pred=forecast(Ar_mod, h=10)
plot(f_pred, col=point_col, main="", xaxs="i",
       xlab="Month", ylab="Monthly sales\n")
```

A study by Givon, Mahajan and Muller[684] investigated UK sales of PC's, wordprocessors and spreadsheets. Figure 11.67 shows monthly sales of spreadsheets, and based in an arima model, 12-months of predicted sales; shaded areas are 80% and 95% confidence intervals.

11.10.3 Non-constant variance

Time-series data containing rapid changes in variance is said to be *volatile*; correlated variance is common during periods of volatility (a time series is *heteroskedastic* if the change in variance is regular, and *conditionally heteroskedastic* if the change is irregular). Techniques for building an autoregressive model, for the variance, include *autoregressive conditional heteroskedastic* (ARCH) and *generalised ARCH* (GARCH) models.

An increase in frequency of commits leading up to a major new release is an example of behavior that can cause a change of variance in a time series.

The autocorrelation of a time-series may not show any correlation, but if its variance changes the square of the zero adjusted values will have a pattern of decreasing correlation in its ACF, as seen in figure 11.68; the code is:

```
acf(t_series)
acf((t_series-mean(t_series))^2)  # Check for changing variance
```

The rugarch package supports the fitting of GARCH models; see Github–time-series/splc-2010-fm.R.

A study by Lotufo, She, Berger, Czarnecki and Wąsowski[1162] investigated the evolution of the Linux variability model, through the lens of commits to Kconfig files. Figure 11.69 shows the number of commits per week made to the Linux kernel source and its associated Kconfig files. The commit bursts occur immediately prior to new releases.

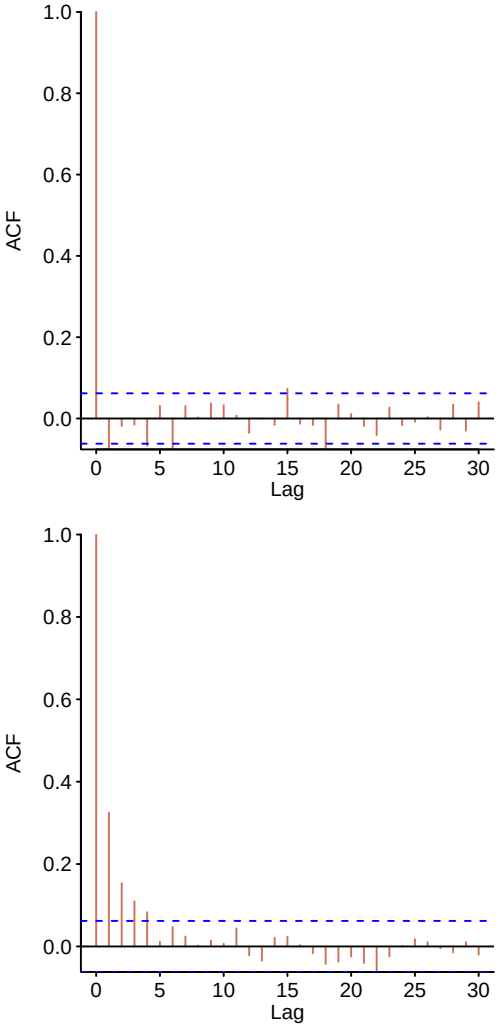


Figure 11.68: Time series whose values are uncorrelated (upper), but whose squared values are correlated (lower); see code for generation process. Github–Local

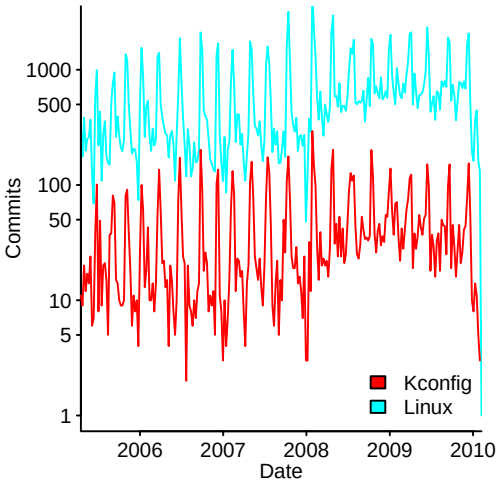


Figure 11.69: The number of commits per week to Linux kernel source and its Kconfig files. Data kindly provided by Lotufo.[1162] Github–Local

11.10.4 Smoothing and filtering

Smoothing a time-series can make it easier to visually identify larger scale patterns, and also provides a simple approximation to predicting immediate future values. Even when data does not contain a systematic trend, or seasonal effects (perhaps because they have been removed), it may still be possible to make a useful estimate of immediate future values based on immediate past values.

Smoothing using the *exponentially weighted moving average* (EWMA; also known as *exponential moving average*, EMA) uses the formula:

$EMA_t = \phi x_t + (1-\phi)EMA_{t-1}$, where: ϕ determines the amount of smoothing.

The *exponential moving standard deviation* (EMS) is given by:

$$EMS_t = \sqrt{\phi EMS_{t-1}^2 + (1-\phi)(x_t - EMA_t)^2}$$

EMA and EMS can be used to detect when a real-time data stream trends outside pre-specified bounds.

Holt-Winters smoothing is a generalization of exponential smoothing, that uses three parameters: estimated level, slope and seasonality; the HoltWinters function can be used to both estimate and apply these parameters.

The filter function can be used to apply AR and MA filters to a time series.

11.10.5 Spectral analysis

A series of measurements in the time domain can be transformed into a sequence in the frequency domain; see fig 1.14.

The `spectrum` function estimates the spectral density of a vector, which is assumed to be a time-series (the default behavior is to call the `spec.pgram` function). The `spec.arma` function takes a specification of an ARMA model, and returns its power spectrum, i.e., behaves like a call to `spectrum` when passed a time series that follows this model.

A stationary time-series does not contain components at specific frequencies, but can be described in terms of an average frequency composition.

11.10.6 Relationships between time series

Time series analysis can be used to find relationships between multiple time series, where each time series comes from measuring separate variables associated with some evolving process.

The simplest technique is cross-correlation, the correlation, at various lags, between two stationary time-series. Figure 11.70 shows the cross-correlation between the number of source lines added/deleted, per week, to the glibc library. In calls to the `ccf` function, the first argument is the one which is shifted, while the second is fixed. In the following call:

```
ccf(lines_added, lines_deleted, col=point_col, xlab="Weeks")
```

the plot shows correlation spikes, above the confidence bounds, occurring between the sequence pairs $\texttt{lines\_added}_{t+2}$/$\texttt{lines\_deleted}_t$ and $\texttt{lines\_added}_{t+8}$/$\texttt{lines\_deleted}_t$ (i.e., changes involving `lines_deleted` is correlated with changes to `lines_added` two and 10 weeks later; a positive lag means the first argument follows the second, a negative lag that it leads the second); there are small spikes at: $\texttt{lines\_added}_{t-8}$/$\texttt{lines\_deleted}_t$ and $\texttt{lines\_added}_{t-13}$/$\texttt{lines\_deleted}_t$. Your author has no explanation for this correlation.

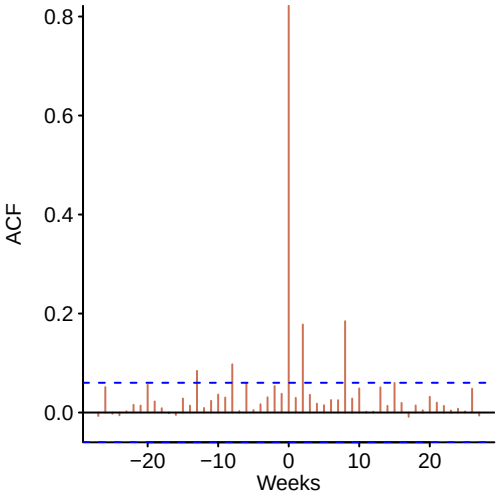


Figure 11.70: Cross-correlation of source lines added/deleted per week to the glibc library. Data from González-Barahona.[704] Github–Local

Techniques are available for building models of the relationship between two time series.

After making a commit to the Linux kernel, it may be discovered that an associated Kconfig file needs to be updated, i.e., the pattern of commits to Kconfig files will lag that of the commits of Linux source. Figure 11.71 shows the first six months of the two time series in figure 11.69, with the number of Kconfig commits shifted up to align with the kernel commits. The Kconfig commits often lag behind kernel commits.

The following calls to the `lags.select` function report a lag of 2-weeks for binned weekly data, and 7-9 days for daily data (which contains many zeroes):

```
library("tsDyn")

# Oldest comes first, and they need to be the same length
# By week
lags.select(cbind(head(log(linux_week$freq), -1), log(kconfig_week$freq)))
# By day
lags.select(cbind(head(log(linux_day$freq+1e-1), -2),log(kconfig_day$freq+1e-1)))
```

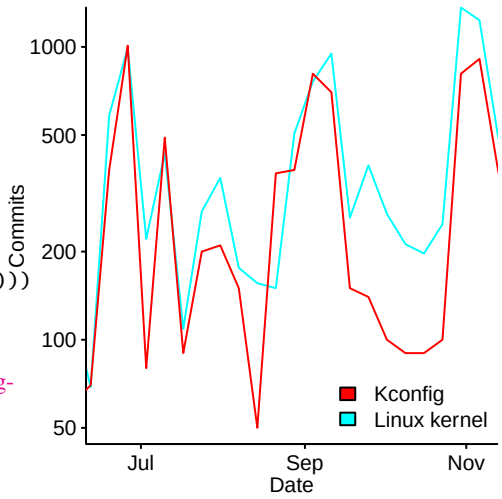


Figure 11.71: The number of commits per week to Linux kernel source and its Kconfig files, during the last half of 2005. Data kindly provided by Lotufo.[1162] Github–Local

What are the interdependencies between Linux source commits and Kconfig commits? The following code fits a Vector Autoregression (VAR) model (see Github–time-series/kconfig-evol.R):

```
library("tsDyn")

# Oldest comes first, using lag returned by lags.select
day_mod=lineVar(cbind(head(log(linux_day$freq+1e-1), -2),
                      log(kconfig_day$freq+1e-1)),
                lag=9)
```

the fitted equations for, *log*, Linux and Kconfig daily commits, are (the error terms have been omitted for brevity):

$$\begin{aligned}
L_commits_t =& 1.6 + 0.2L_commits_{t-1} + 0.07K_commits_{t-1} + 0.08K_commits_{t-2} \\
&+ 0.09L_commits_{t-3} + 0.1L_commits_{t-6} + 0.1L_commits_{t-7} - 0.09L_commits_{t-9} \\
K_commits_t =& -1.1 + 0.3L_commits_{t-1} + 0.09K_commits_{t-1} + 0.09K_commits_{t-2} \\
&+ 0.06L_commits_{t-3} + 0.2L_commits_{t-6} + 0.09K_commits_{t-7} - 0.1L_commits_{t-9}
\end{aligned}$$

showing Kconfig commits having a small influence, over a few days, on Linux commits, and Linux commits having a larger and longer term impact on Kconfig commits, than even earlier Kconfig commits.

Other forms of relationship that may exist between two or more time-series include:

Alignment: A time series is a sequence of values, with each value being larger, smaller or equal to the value immediately before it. If two time series are generated by the same, or similar, process they may contain subsequences of values that share the same pattern of up, down and no-change. A non-time series application of this kind of subsequence matching is extracting word sequences that commonly appear in two or more documents.

Dynamic time warping (DTW) is a class of algorithms that compares two series of values by stretching or compressing one of them (treated as the reference series), so it resembles the other (treated as the query series). The `dtw` package contains functions to perform and support DTW alignment of two series.

A study by Herraiz[817] investigated the evolution of various long-lived software systems, and measured the growth of NetBSD and FreeBSD (in lines of code). These two operating systems started from the same base, continue to share developers (see fig 9.22) and code continues to be ported between them. Figure 11.72 shows the alignment, found by a call to `dtw`, between the weekly measurements of the lines of code in each OS (for the first 100 weeks of their development).

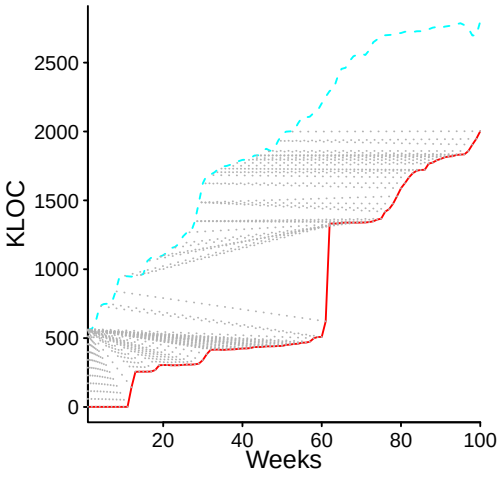


Figure 11.72: Visualization of alignment between lines of code, in NetBSD's (blue) and FreeBSD's (red) first 100 weeks. Data from Herraiz[817] Github–Local

```
library("dtw")

bsd_align=dtw(freebsd_weeks, netbsd_weeks, keep=TRUE,
              step=asymmetric, open.end=TRUE, open.begin=TRUE)
plot(bsd_align, type="twoway", offset=1, col=pal_col, xlab="Weeks")
```

Clustering: The pair-wise similarity of multiple time-series can be used as a clustering metric. Many techniques for measuring the distance between two time-series have been invented (at the time of writing, the `diss` function, in the `TSclust` package, supports 22 distance metrics).

A study by Powell[1517] investigated task effort allocation in a development project at Rolls-Royce. Figure 11.73 shows effort (in person hours) spent on eight major tasks (lower plot, from the bottom up: s/w requirements, top-level design, coding, low level test, requirement test, system acceptance test, management and holiday/non-project), and a hierarchical clustering of each task by its effort time series, with pair-wise distance between time series calculated using correlation (upper) and Euclidean (middle) metrics.

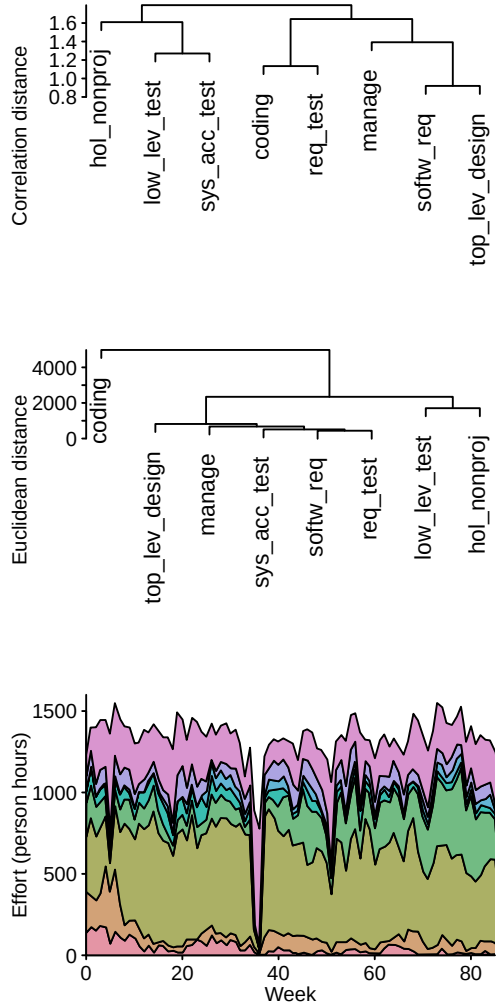


Figure 11.73: Effort distribution (person hours) over the eight main tasks of a development project at Rolls-Royce and a hierarchical clustering of each task effort time series based on pair-wise correlation and Euclidean distance metrics. Data extracted from Powell.[1517] Github–Local

```
library("TSclust")

eff_dist=diss(t(all_effort), METHOD="COR")
plot(hclust(eff_dist), main="", sub="", xlab="", ylab="Correlation distance")
```

11.10.7 Miscellaneous

Regression of time series data: Some of the issues involved in building regression models with serially correlated data are discussed in section 11.2.7.

Stochastic processes: Events involving future uncertainty may be modeled as a stochastic process; see section 3.2.4. The `Sim.DiffProc` package can be used to numerically solve stochastic differential equations of the Itô type;[230] section 3.2.4 discusses this topic in more detail.

The *Ornstein-Uhlenbeck process* is the continuous time version of an AR(1) model,[503] and the AR(1) process corresponding to equation 3.1 is:

$$x_t - x_{t-1} = \hat{x}(1 - e^{-\eta}) + (e^{-\eta} - 1)x_{t-1} + \varepsilon_t$$

Matrix profile[988] is an efficient new technique for finding motifs in time series. The `tsmp` package supports a variety of matrix profile related techniques; see Github–time-series/BSD-dtw.R.

11.11 Survival analysis

Survival analysis is the analysis of data where the response variable has the form of *time-to-event*. Historically this kind of model building has been used to compare the impact of different medical procedures, or drugs, on subject survival rate.

Survival analysis often deals with one kind of event, which causes a transition to a terminal state, e.g., there is returning from the dead. Competing risk models deal with the situation where one of several risk events can cause the transition to the final state. Multistate models handle the situation where some transitions are to states that are not final, i.e., an appropriate event can cause a transition to another state.

In some cases, the event of interest may not occur during the measurement period, in this case the measurement is said to be *censored*; for instance, when measuring the time interval between a function definition being written and the first time it is modified, the measurement data is said to be *right censored*, when one or more functions have not been modified over the time interval for which data is available.

Survival analysis makes greater use of available censored information, to produce estimates containing less error, than other forms of regression modeling; a linear regression model comparing mean time-to-event between groups would have to ignore censored data, while a logistic regression model, using 0/1 to indicate whether a subject survived or not, would again have to ignore censored data.

Possible outputs from survival analysis include:

- a survival function, $S(t)$, the probability of surviving a given amount of time. This can be used to estimate time-to-event for a group of subjects or compare time-to-event between subjects in two or more groups,

- a hazard function, $h(t)$, the hazard rate, that is, the probability of an entity surviving to time t experiencing an event in the next time interval, e.g., having survived 69 years 11 months before reading this sentence the probability that you die in the next month (the interval used to denote an instant is small compared to the time spans involved). The survival and hazard functions can be derived from each other:

  $$h(t) = \frac{f(t)}{S(t)}$$

  where: $f(t)$ is a probability density function, the probability of the event occurring at exactly t time units in the future, e.g., the probability of a baby born 70 years ago living long enough to read this sentence, but not before,

- a regression model specifying the impact of explanatory variables on time-to-event. This may be a non-parametric model, such as the Cox proportional hazard model, because parametric models can be very difficult to build.

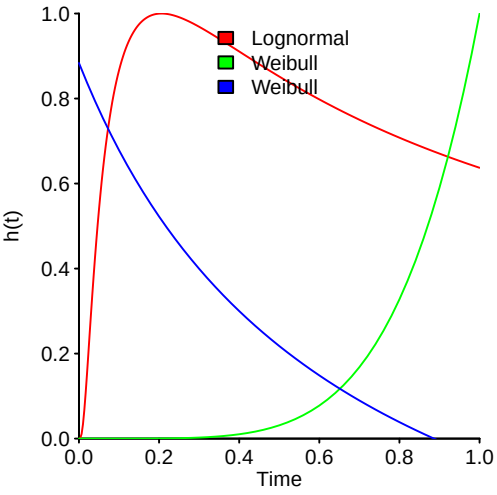


Figure 11.74: Two commonly used hazard functions; Weibull is monotonic (always increases, decreases or remains the same, depending on the equation coefficients), and Lognormal which can increase and then decrease. Github–Local

Time-to-event is always positive and so has a skewed distribution (which means it cannot have a Normal distribution).

The `survival` package contains functions implementing the functionality needed to perform survival analysis.

11.11.1 Kinds of censoring

Ideally censoring is uninformative, i.e., the distribution of censoring times provides no information about the distribution of survival times. When a period of study is decided in advance, the censoring information is uninformative.

When censoring is not under experimenter control, it is said to occur at random. For instance, a subject may decide to stop taking part in a study because they are not happy with their performance.

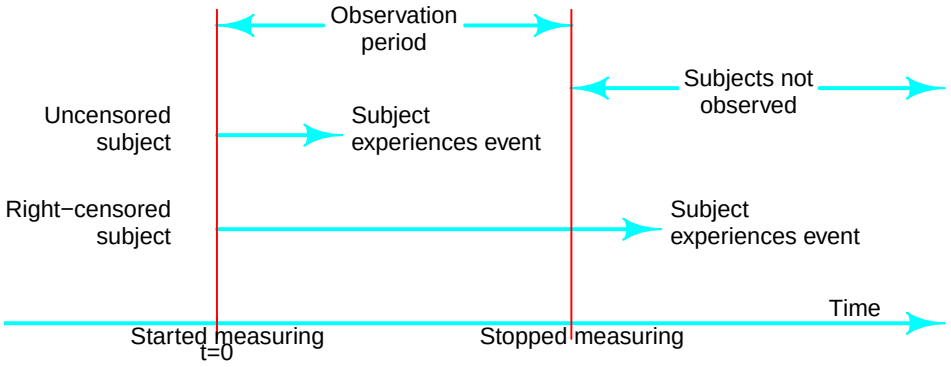


Figure 11.75: Observation period of study, with events inside and outside the study period. Github–Local

Kinds of censoring that can occur include:

- *left truncation*: subject not observed before t_0, experienced an event before that time and is not included in the study (the event may have been such that it rendered the subject unable to join the study, e.g., developer left the company),
- *left censored* (also *left truncated*): a subject included in the study is known to have had the event prior to time t, but with the exact time not being known,
- *right censored*: described at the start of the subsection,
- *interval censored*: when measurements are made at regular intervals, the exact time of an event is not known, only that it occurred between two measurement points,
- *non-detect*: the measurement process may fail to detect an event because the strength of the event is below the detection threshold. This kind of censoring is not covered here, see Helsel.[802]

11.11.1.1 Input data format

The `Surv` function creates a survival object from data, and the object it returns plays the role of the response variable in formula passed to model building functions. The required data format depends on the kind of censoring and presence of time dependencies. The following is an example of the basic information required:

```
id,start_time,end_time,failure_status,explanatory_v1,explanatory_v2
```

where: `id` is a unique identifier denoting each subject (only needed when information on the same subject occurs on multiple lines), `start_time`/`end_time` the starting time (or date) of measurement, and the end time (either when the event occurred, the end of the study or the last recorded time of a subject who was not seen again) and `failure_status` one of two values specifying whether an event occurred or not; followed by an optional list of explanatory variables.

The time of interest is the difference between the start/end time, and the data may contain just this value.

11.11.2 Survival curve

The *Kaplan-Meier* curve is a descriptive statistic of time-to-event measurements; it shows the percentage of subjects who have not experienced an event up to a point in time, along with an optional confidence interval; see figure 11.76.

The median is preferred over mean as the measure of central tendency for survival data, because the mean underestimates the true value when samples contain censored data. The median is measured as the point where the Kaplan-Meier curve falls before 0.5 and printing the model returned by `survfit` gives this value along with its 95% confidence intervals.

A study by Businge[285] investigated the number of releases of Eclipse third-party plug-ins (ETP) between 2003 and 2010; the history of each ETP was traced from the year of its first release and any releases in subsequent years were noted.

The Eclipse framework includes a published list of officially recognised APIs, but each Eclipse SDK release also includes support for APIs considered to be for internal use, i.e.,

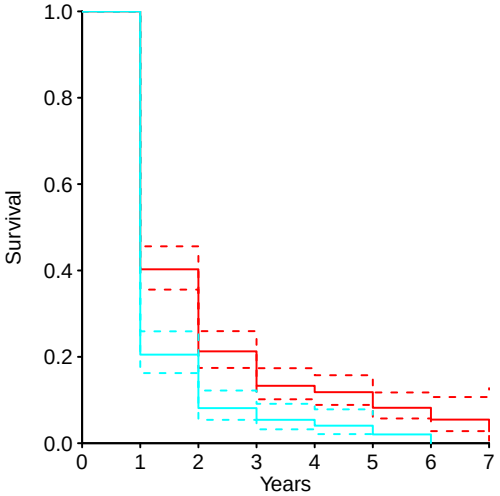


Figure 11.76: The Kaplan-Meier curve for survivability of new releases: (blue) ETPs using only official APIs, (blue) ETPs calling internal APIs (red); dotted lines are 95% confidence intervals. Data from Businge.[285] Github–Local

not to be used by applications. The status difference between official/internal APIs is that internal APIs can be changed without notice, while the official APIs are intended to have some degree of permanence (they may change on major releases but are not intended to change on minor releases; starting in 2004 all yearly releases were minor).

At some point there are no new releases of an ETP in a year and this cessation of new releases could be regarded as the *death* of development of the ETP (some ETP development died for one year only to be resurrected the following year; for simplicity the small number of such recurring events are ignored).

For this analysis ETP yearly release counts are divided into two groups, those that only made use of official APIs, and those that made use of one or more internal APIs; table 11.5 shows the number of ETPs using only the official API.

	2003	**2004**	**2005**	**2006**	**2007**	**2008**	**2009**	**2010**
2003	35	10	3	1	1	2	0	0
2004		33	4	4	2	2	0	0
2005			41	10	4	3	1	1
2006				61	7	1	0	2
2007					37	12	4	6
2008						38	7	2
2009							25	3
2010								16

Table 11.5: Total number of distinct ETPs released in a year; left column lists year of first release and releases in subsequent years. Data from Businge.[285]

Figure 11.76 shows the Kaplan-Meier curve for ETPs using only official APIs (blue) and ETPs that use internal APIs (red); the dotted lines are 95% confidence intervals. The following is the essential code (calling the `Surv` function to create a survival object, containing time and censored information on each subject, is the first step in most survival analysis using R):

```
library("survival")

api_surv=Surv(all_API$year_end-all_API$year_start,
                  event=(all_API$survived == 0), type="right")
api_mod=survfit(api_surv ~ all_API$API)
plot(api_mod, col=pal_col, conf.int=TRUE,  xlim=c(0,7), xlab="Years")
```

The `summary` function can be used to obtain values of the survival curve at each time measurement point.

Comparing two survival curves: Are the two survival curves statistically different? The `survdiff` function can be used to answer this question. The p-value returned by the call (bottom right) shows that the two survival curves are very unlikely to be the same: Github–Local

```
Call:
survdiff(formula = Surv(year_end - year_start, event = (survived ==
    0), type = "right") ~ API, data = all_API)

        N Observed Expected (O-E)^2/E (O-E)^2/V
API=0 381      334      372      3.83        29
API=1 289      260      222      6.41        29

 Chisq= 29  on 1 degrees of freedom, p= 7e-08
```

By default, `survdiff` performs a *log-rank test*, which gives equal weight to all events. Passing the argument `rho=1` causes greater weight to be given to earlier events, while the argument `rho=-11` gives greater weight to later events. The hazard function is returned by `survfit` functions when is it passed the argument `type="fh"`.

Why, on average, do new releases of an ETP using internal APIs occur over a greater number of years? Is it because there are changes to the internal APIs that break the ETP, requiring the ETP to be updated to handle the change and a new version released, or is it because authors who use internal APIs are more committed to creating the best possible product and so continue to refine their ETP over more years?

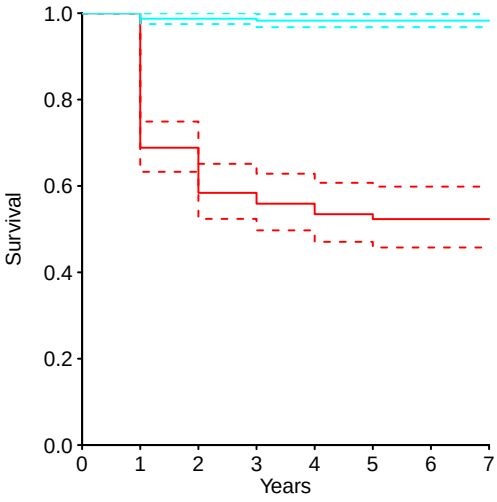


Figure 11.77: The Kaplan-Meier curve for survivability of ETPs ability to be built using SDK released in subsequent years: (blue) ETPs using only official APIs, (red) ETPs calling internal APIs; dotted lines are 95% confidence intervals. Data from Businge.[285] Github–Local

Perhaps, suspecting that changes to the SDK were a significant factor, Businge[285] investigated the source compatibility of ETPs with the Eclipse SDK across releases 1.0 to 3.7, i.e., releases in every year from 2001 to 2011. Every ETP was built using each of these 11 SDK releases (yes, even SDKs created before an ETP was first released). To allow easy comparison with the ETP analysis above, the following analysis only considers SDK builds released after an ETP was first made available.

Figure 11.77 shows the survival of ETPs' ability to build under Eclipse SDKs released in each successive year. ETPs using internal APIs (red) are much more likely to fail to build (precompiled plug-ins may still function, if they don't call any changed internal API) when a new Eclipse SDK is released, compared to ETPs using only the official APIs (blue).

This analysis suggests that developers using internal APIs in their ETP, are more likely to be forced to release an update, if they want their ETP to continue to function with later releases. However, this data does not address the possibility that developers who make use of internal APIs are more committed to creating the best possible product.

11.11.3 Regression modeling

Survival data implicitly contains information that is not present in other forms of regression modeling: the probability of an event occurring at a given time, i.e., a hazard function. Estimating the appropriate hazard function for survival data requires knowing the coefficients of the explanatory variables in the regression model, while estimating the coefficients of the explanatory variables requires knowing the hazard function.

When building a model, R functions will attempt to fit the shape of the hazard function specified (by the analyst), but if this hazard function is incorrect, the returned model may be substantially incorrect. In practice, parametric models have been found to be very sensitive to the explanatory variables provided as input to the model fitting process.

There is no single statistic available for definitively selecting the best model, i.e., hazard function and appropriate explanatory variables.

The Cox proportional-hazards model does not require the specification of a hazard function, which breaks the circularity of needing to select regression coefficients for such a function and removes some of the dangers associated with use of an incorrect hazard function (the Cox modeling approach is not guaranteed to always build a reasonably accurate model). If there is any doubt about the appropriate parametric distribution, the Cox model is a safe choice.

While the Cox proportional hazards model has many advantages, a potentially big disadvantage is that without specifying a hazard function, it is not possible to make predictions outside the interval covered by the measurements.

The `flexsurv` package supports the fitting of complex parametric distributions, and the `censReg` package supports fitting regression models to censored data.

11.11.3.1 Cox proportional-hazards model

The Cox proportional-hazards regression model has been found to provide reasonably good estimates for the coefficients of the explanatory variables and hazard ratios (not absolute values, ratios) for a wide variety of data. The Cox model is popular because it is robust, and will closely approximate the correct parametric model. If the correct parametric model has a Weibull hazard function (whose shape parameter is unknown), the Cox model will give similar results to those obtained from this parametric model. If the parameters of the Weibull hazard function are known, a model built using them will outperform a Cox model.

The Cox likelihood (known as a *partial likelihood*) is based on the observed order of events, rather than the interval between them (so it only considers subjects' experiencing an event).

In the equation for the basic Cox model, time is not included as an explanatory variable, x_{ki}, i.e., the variables cannot be time dependent. The equation is:

$$h_i(t) = h_0(t)e^{\beta_1 x_{1i}+\cdots+\beta_k x_{ki}}$$

where: $h_i(t)$ is the hazard function for subject i at time t, $h_0(t)$ is a baseline hazard function, the contents of the exponent expression are explanatory variables and their regression coefficients (β_0 is included as part of the baseline hazard).

This equation can be written as a log ratio of the hazard functions:

$$\log \frac{h_i(t)}{h_0(t)} = \beta_1 x_{1i} + \cdots + \beta_k x_{ki}$$

or, as a hazard ratio for two subjects, i and j (where, $h_0(t)$, the baseline hazard function cancels out):

$$\frac{h_i(t)}{h_j(t)} = e^{\beta_1(x_{1i}-x_{1j})+\cdots+\beta_k(x_{ki}-x_{kj})}$$

In this proportional hazards model, the effect of each explanatory variable is multiplicative on the hazard function. In accelerated failure time (AFT) models the multiplicative effect is on the survival function.

The `coxph` function, in the `survival` package, builds Cox proportional-hazard models; the basic usage follows the pattern used by `glm`, with the object returned by `Surv` playing the role of the response variable. For example:

```
p_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource,
                                                          data=ISR_disc)
```

The `cox.zph` function can be used to check the assumption that the explanatory variables are not time dependent (at least during the measurement period).

If two or more events occur at the same time the associated data is said to be *tied*. The default value of the option `ties="efron"`, can handle some tied data, but if many events occur at the same time (e.g., the ETP data in table 11.5), calls to `coxph` might need to use `ties="exact"`.

The techniques for formula specification and refinement used with `glm` can also be applied to models created with `coxph`, e.g., starting with a complicated model and using `stepAIC` to simplify it.

A study by Arora, Krishnan, Telang and Yang[78] investigated the time taken by vendors to release patches, to fix vulnerabilities reported in their product; explanatory variables included information about the software vendor, whether the vendor was privately notified about the vulnerability, or the vendor first found out about it through a public disclosure.

The following is the `summary` output from a model fitted by `coxph` to the data for public disclosure vulnerabilities: Github–Local

```
Call:
coxph(formula = Surv(patch_days, !is_censored) ~ log(cvss_score) +
    opensource + y2003 + smallvendor + small_loge + log(cvss_score):y2002 +
    y2002:smallvendor + y2003:smallvendor, data = ISR_np)

  n= 945, number of events= 824

                          coef exp(coef) se(coef)      z Pr(>|z|)
log(cvss_score)        0.23283   1.26217  0.08570  2.717  0.00659 **
opensource             0.42235   1.52555  0.09167  4.607 4.08e-06 ***
y2003                  0.83643   2.30811  0.10459  7.997 1.27e-15 ***
smallvendor           -0.40940   0.66405  0.17331 -2.362  0.01816 *
small_loge             0.02926   1.02969  0.01346  2.173  0.02975 *
log(cvss_score):y2002  0.23048   1.25920  0.04961  4.646 3.39e-06 ***
smallvendor:y2002      0.59685   1.81638  0.19540  3.054  0.00226 **
y2003:smallvendor      0.58999   1.80396  0.22502  2.622  0.00874 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

                      exp(coef) exp(-coef) lower .95 upper .95
log(cvss_score)           1.262     0.7923    1.0670    1.4930
opensource                1.526     0.6555    1.2747    1.8258
y2003                     2.308     0.4333    1.8803    2.8332
smallvendor               0.664     1.5059    0.4728    0.9326
small_loge                1.030     0.9712    1.0029    1.0572
log(cvss_score):y2002     1.259     0.7942    1.1425    1.3878
smallvendor:y2002         1.816     0.5505    1.2384    2.6640
```

```
y2003:smallvendor         1.804     0.5543    1.1606    2.8039

Concordance= 0.647  (se = 0.011 )
Likelihood ratio test= 199  on 8 df,   p=<2e-16
Wald test            = 184.9  on 8 df,   p=<2e-16
Score (logrank) test = 198.3  on 8 df,   p=<2e-16
```

The first half of the output is similar to the `summary` output produced by a model fitted using `glm`. The table of numbers in the middle are 95% confidence intervals, which are printed by default. The bottom section of the output lists the R-squared of the fit[xxxii] (0.19 in this case, showing that only a small amount of the variance in the data is described by the model), and p-values for various tests of the null hypothesis that the coefficients are zero (abbreviated to a single letter, `p`).

The explanatory variable coefficients are proportions, not absolute values (the Cox model is a proportional-hazards model). The coefficients specify the expected impact of the respective explanatory variable, when the values of all the other variables are kept constant. The explanatory variables cannot be used to independently calculate response variable values, they can only be used to predict the change in a known value of the response variable, i.e., the value of the response variable known to occur for specific values of the explanatory variables.

Taking the values in the `log(cvss_score)` row as an example, the value 1.26217 appears in its `exp(coef)` column; what impact will a ± 1 change in the value of $\log(cvss\_score)$ have on the response variable (i.e., time taken to produce a patch)? The percentage change in the response variable is: $\pm(1.2621-1)\times 100 \rightarrow \pm 26.21\%$; a value of less than one, in the `exp(coef)` column, reverses the sign of the percentage change, e.g., an increase in the value of the explanatory variable is predicted to decrease the value of the response variable.

Model adequacy can be checked using Cox-Snell residuals, and influential observations searched for using *score residuals* (which specify how each regression coefficient would change if a particular observation was removed; see Github–survival/vulnerabilities/patch-cph.R).

Frailty of subjects: Unobserved differences in subject performance may result in some variation in the hazard function they experience;[90] *frailty* is the term used to denote these random (multiplicative) changes in hazard function. Introducing a random effect, v_j, the frailty of group j that x_i belongs to, modifies the Cox model as follows:

$$h_i(t) = h_0(t)v_j e^{\beta_1 x_{1i}+\cdots+\beta_k x_{ki}}$$

The previous analysis of time-to-patch, implicitly assumes there is no difference between vendors, in their ability to respond and fix reported vulnerabilities. The `frailty` function can be included in a formula, to specify explanatory variables that identify particular groups of subjects sharing the same frailty.

```
fp_mod=coxph(Surv(patch_days, !is_censored) ~ log(cvss_score)+opensource
                                            +frailty(vendor), data=ISR_disc)
```

The `summary` output includes the information: `Variance of random effect=0.374`; see Github–survival/vulnerabilities/patch-frailty.R.

The v_j in the above equation is assumed to have a mean and variance that is calculated as part of the model building process (in this case it is 0.374). The main consequence of including frailty in a Cox model is to explicitly allocate some of the variance present in the data to a specific explanatory variable (the model coefficients of explanatory variables may also change).

The `frailtypack` package provides a wider range of frailty related options and functionality, than is available in the `survival` package. The `coxme` package supports the fitting of mixed-effects Cox models (frailty can be handled as a specific kind of random effect).

11.11.3.2 Time varying explanatory variables

The behavior of an explanatory variable may change over time. Options for handling this behavior includes, excluding all affected subjects from the analysis or using a technique that handles time dependent behavior.

[xxxii] The value printed is the Cox & Snell pseudo R-squared, which can be less than one; the maximum possible value for the data is given in the `summary` output.

The Arora et al study (discussed earlier) investigated the impact of public disclosure of vulnerabilities, on the time taken by vendors to release patches for their product. Possible event sequences were:

- vendor was privately notified about a vulnerability and some time later a simultaneous announcement of the vulnerability and a vendor patch was made (213 of 755 private notifications),
- vendor was privately notified about a vulnerability, but information about the vulnerability was later made public before a patch was available for release (the vendor's patch being released some time later in 542 of 755 private notifications); this is a time dependent change of a significant attribute.
- the vendor learned about a vulnerability when information about it was made public, and sometime later released a patch (945 cases),

If privately notified and public disclosure fix rates are compared using a Kaplan-Meier curve, any privately notified vulnerabilities that become public before a patch is available have to be treated as censored (ignoring them biases fix rates towards a lower value; see figure 11.78).

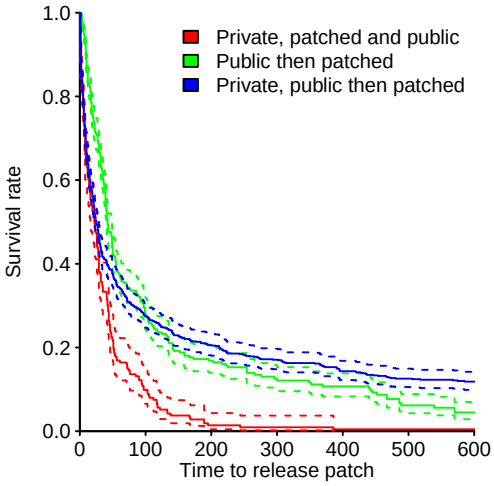


Figure 11.78: Kaplan-Meier curves for time-to-release a patch for a reported vulnerability, with private, public, and private then public notification. Data from Arora et al.[78] Github–Local

The Cox models discussed earlier, were fitted using vulnerability data where the vendor found out about the vulnerability via public disclosure.

Building a regression model using all the vulnerability data, requires handling time dependent explanatory variables; the data has to be restructured to make the time dependencies explicit. The time dependency, for this data, is a possible change of state, from the vulnerability not being public, to the information being public.

The original data looks something like the following:

```
notify,publish,patch,vendor,employee,os
2000-10-16,2000-11-18,2000-12-20,"abc",1000,unix
```

When vulnerability information is made public before a patch is released, extra information is involved. For this data, five columns are added: one to uniquely identify each vulnerability, the start/end dates of the interval during which the information was private or disclosed, a flag specifying private/disclosed, and a flag for whether an event (i.e., release of a patch) occurred in the interval. The first interval starts on the date the vendor was notified and ends on the date the vulnerability is made public, a second interval occurs for vulnerabilities that change state from private to disclosed before a patch is available and starts on the date of disclosure and ends on the date a patch became available, as follows:

```
id,start,end,priv_di,notify,publish,patch,event,vendor,os
1,2000-10-16,2000-11-17,1,2000-10-16,2000-11-18,2000-12-20,0,"abc",unix
1,2000-11-18,2000-12-20,0,2000-10-16,2000-11-18,2000-12-20,1,"abc",unix
```

Treating `priv_di` as an explanatory variable (1 for private disclosure to vendor and zero for public disclosure), enables the impact of disclosure on patch time to be included in a model.

When all the measurement data needs to be split on the same date, the `survSplit` function can be used to create the necessary rows, otherwise (as in this case) specific data mangling code has to be written.

The call to `coxph`, or `survreg`, has to include the term `cluster(id)`, which ties together (by vulnerability id in this case) the rows associated with the same subject. The call to `coxph` looks something like the following:

```
td_mod=coxph(Surv(patch_days, !is_censored) ~ priv_di*cvss_score
                                             +cluster(id), data=ISR_split)
```

It is not possible for both `cluster` and `frailty` to appear in the same formula (`cluster` is based on GEE model building, while `frailty` is based on mixed-effects model building).

The `summary` output for the time dependent model is: Github–Local

```
Call:
coxph(formula = Surv(patch_days, !is_censored) ~ priv_di + cvss_score +
    y2 + small_loge + priv_di:cvss_score + priv_di:c_o + priv_di:dis_by_s +
```

```
    priv_di:os + priv_di:y2 + priv_di:smallvendor + priv_di:small_loge +
    cvss_score:c_o + cvss_score:dis_by_s + cvss_score:s_app +
    c_o:opensource + dis_by_s:opensource + os:s_app + y2:s_app,
    data = ISR_split, cluster = ID)

  n= 2242, number of events= 2081

                          coef exp(coef)  se(coef) robust se      z Pr(>|z|)
priv_di               2.798750 16.424106  0.216150  0.209360 13.368  < 2e-16 ***
cvss_score            0.153926  1.166404  0.016806  0.017733  8.680  < 2e-16 ***
y2                    0.277421  1.319722  0.044042  0.044590  6.222 4.92e-10 ***
small_loge            0.037114  1.037811  0.007262  0.008817  4.210 2.56e-05 ***
priv_di:cvss_score   -0.114788  0.891555  0.017327  0.016795 -6.835 8.22e-12 ***
priv_di:c_o           0.644347  1.904743  0.228463  0.211989  3.040 0.002369 **
priv_di:dis_by_s      0.475405  1.608665  0.116261  0.106601  4.460 8.21e-06 ***
priv_di:os           -0.331847  0.717597  0.098936  0.086976 -3.815 0.000136 ***
priv_di:y2           -0.614162  0.541094  0.063296  0.061954 -9.913  < 2e-16 ***
priv_di:smallvendor  -0.440845  0.643492  0.138900  0.099310 -4.439 9.03e-06 ***
priv_di:small_loge   -0.082120  0.921161  0.016589  0.014449 -5.683 1.32e-08 ***
cvss_score:c_o       -0.060084  0.941685  0.012861  0.011990 -5.011 5.42e-07 ***
cvss_score:dis_by_s  -0.061114  0.940716  0.008798  0.011002 -5.555 2.78e-08 ***
cvss_score:s_app     -0.096771  0.907764  0.014972  0.014853 -6.515 7.27e-11 ***
c_o:opensource        0.443978  1.558896  0.137952  0.118459  3.748 0.000178 ***
dis_by_s:opensource   0.414151  1.513086  0.091161  0.102359  4.046 5.21e-05 ***
os:s_app              0.815803  2.260991  0.077450  0.093536  8.722  < 2e-16 ***
y2:s_app              0.291007  1.337774  0.047420  0.045599  6.382 1.75e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Concordance= 0.654  (se = 0.007 )
Likelihood ratio test= 590.1  on 18 df,   p=<2e-16
Wald test            = 376.9  on 18 df,   p=<2e-16
Score (logrank) test = 586.8  on 18 df,   p=<2e-16,   Robust = 454.6  p=<2e-16

  (Note: the likelihood ratio and score tests assume independence of
     observations within a cluster, the Wald and robust score tests do not).
```

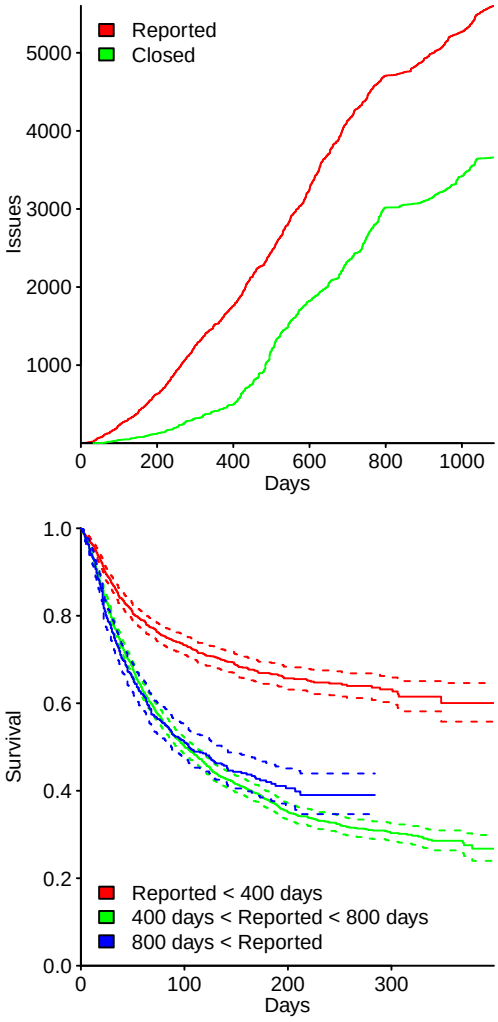


Figure 11.79: Cumulative number of issues reported and closed, and issue survival curves for three intervals. Data from Lunesu.[1172] Github–Local

There are two parts to the contribution made by `priv_di`; as a standalone variable it has a large impact, but its interactions with other variables create a large impact in the opposite direction (the model building process tries to minimise its error metric, not make it easy for analysts to understand what is going on).

The component of the fitted equation of interest is:

$$e^{priv\_di(2.8-0.11cvvs\_scorei+0.64c\_o+0.48dis\_by\_s-0.33os-0.61y2-0.44smallvendor-0.08small\_loge)}$$

where: *priv_di* is 0/1, *cvvs_score* varies between 1.9 and 10 (mean 7), *c_o* 0/1 NA other[xxxiii] (mean 0.13), *dis_by_s* 0/1 disclosed by SecurityFocus (mean 0.38), *os* 0/1 vulnerability in O/S (mean 0.26), *y2* years since 2000 (mean 1.9), *smallvendor* 0/1 small vendor flag (mean 0.25) and *small_loge* zero for small vendors, otherwise the log of number of employees (mean 5).

Applying some hand waving to average away variables:

$$e^{priv\_di\times(2.8-0.11\cdot7+0.64\cdot0.13+0.48\cdot0.38-0.33\cdot0.26-0.61\cdot1.9-0.44\cdot0.25-0.08\cdot5)}$$

$$\rightarrow e^{priv\_di\times(2.8-0.77+0.08+0.18-0.09-1.2-0.11-0.4)} \rightarrow e^{0.49priv\_di}$$

produces a (hand waved mean) percentage increase of $(e^{0.49}-1)\times 100 \rightarrow 63\%$, when `priv_di` changes from zero to one. The percentage change for patches, for vulnerabilities with a low `cvvs_score` is around 90% and for a high `cvvs_score` around 13%, i.e., the patch time of vulnerabilities assigned a low priority improves a lot when they are publically disclosed, but patch time for those assigned a high priority is only slightly affected.

The process of calculating the 95% confidence bounds, based on the values in the `summary` output, is fiddly and left to the reader.

Time varying changes may occur, but the information needed to model these changes may not be available.

[xxxiii] No information is available on what this variable represents.

A study by Lunesu[1172] investigated the maintenance activities of a large software company; between 2005 and 2010 there were 5,854 issues. The response time for an issue (i.e., the time between an issue being opened to it being closed) depends on the rate issues are reported, and the resources available to handle issues.

Extra resources were added to handle the growing number of issues (after around 400 days), and the number of issues reported decreased after around 800 days (it is not known whether this resulted in issue handling resources being decreased, or the same level of resources applied to fewer issues). The level of resources used to resolve issues cannot be included in a model, because this information is not available.

Figure 11.79 shows the cumulative number of issues reported and closed, over time (upper); the lower plot shows the survival curve for issues reported in the first 400 days, reported between 400 and 800 days and reported after 800 days. As expected, the extra resources added after 400 days reduced open issue survival times, but the reduction in reported issues after 800 days does not appear to have had much impact on survival rates (perhaps because it is easier to move existing staff to other work, than to add new staff).

11.11.4 Competing risks

When more than one possible kind of event can occur (i.e., there are multiple terminal states), a competing risk model might be used or each distinct event type might be analyzed separately from the other event types (data involving other events is flagged as censored at the time the other event occurs).

The Kaplan-Meier plot for a single event, in a competing risk context, may give a misleading impression of the actual situation for events that rarely occur. The *cumulative incidence curve* (CIC) is a commonly used alternative, which includes information on every event (when there is only one event, $CIC = 1 - KM$). CIC does not assume that competing risks are independent and estimates the marginal probability of an event.

The `cmprsk` package supports the modeling of competing risks.

A study by Di Penta, Cerulo and Aversano[492] investigated the history of mistakes in source code flagged by various static analysis tools. Newly written source code containing a flagged construct was tracked through subsequent versions. Possible competing events include the removal of the code containing the flagged construct and the flagged construct being modified such that it is no longer flagged, e.g., a bug fix.

Figure 11.80 shows the cumulative incidence curves (created by the `cuminc` function, in the `cmprsk` package) for problems reported in the Samba and Squid source by the splint static analysis tool.

```
library("cmprsk")

plot_cif=function(sys_str)
{
t=cuminc(rats$failtime, rats$type, cencode=0, subset=(rats$SYSTEM == sys_str))

plot(t, col=pal_col, cex=1.25,
        curvlab=c("was removed", "disappeared"),
        xlab="Snapshot", ylab="Proportion flagged issues 'dead'\n")

text(max(t[[1]]$time)/1.5, 0.9, sys_str, cex=1.5)
}

plot_cif("samba")
plot_cif("squid")
```

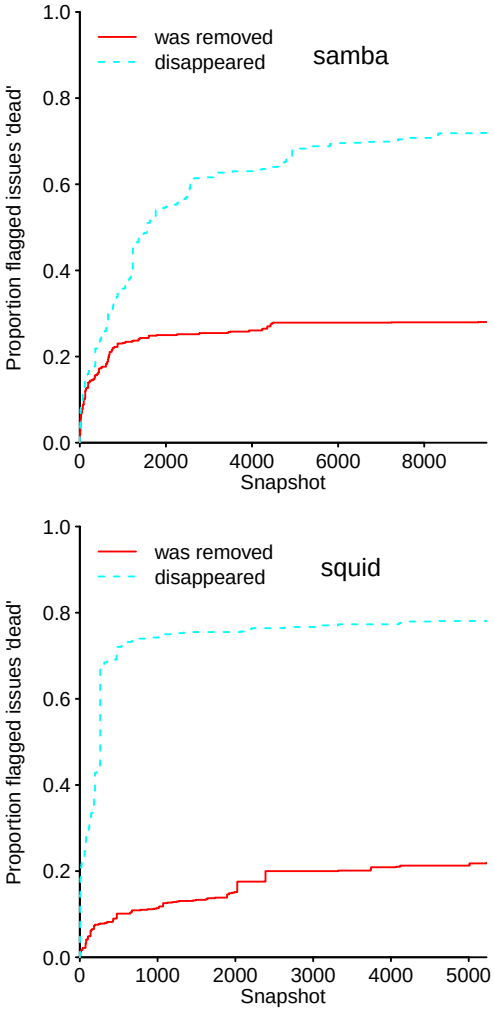


Figure 11.80: Cumulative incidence curves for problems reported by the splint tool in Samba and Squid (time is measured in number of snapshot releases). Data from Di Penta et al.[492] Github–Local

11.11.5 Multi-state models

Multistate models deal with time to event processes, where there are multiple events with potential changes of state between them.

The `mstate` package supports the building of multi-state Cox models and competing risk models, the `msm` package supports the building of multi-state Markov models.

A study by Goeminne and Mens[692] investigated the evolution in the use of four database frameworks in 2,818 Java projects. A project might start using, say, Spring, then add support for another framework or switch from using Spring to a different framework.

What is the probability of changes, over time, in the number of database frameworks, used by a project?

Table 11.6 shows the estimated likelihood of a project migrating from using i database frameworks to using j frameworks, within 365 days.

from	to 1	to 2	to 3	to 4
1	0.89	0.09	0.02	0.00
2	0.07	0.74	0.17	0.03
3	0.02	0.07	0.77	0.14
4	0.01	0.01	0.05	0.93

Table 11.6: Estimated likelihood that within 365 days, a project using i database frameworks will migrate to using j frameworks. Data kindly provided by Goeminne.[692]

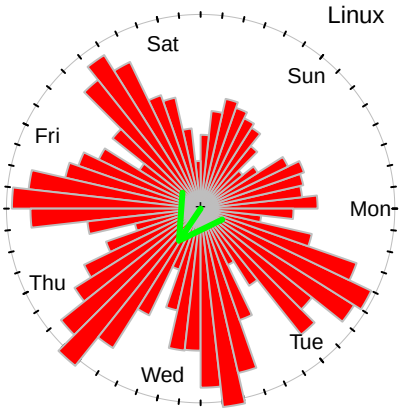
The following call to `msm` fits a multi-state Markov model for the number of database frameworks (`dbs`) used by projects (`X`) over time (`date`). The matrix `Q` is an initial estimate of the state transition probabilities of a project currently using i frameworks migrating to use j frameworks; any transition that cannot occur must contain zero in the corresponding non-diagonal element (specifying `gen.inits=TRUE` results in an approximate estimate being calculated internally; see Github–survival/icsme2015era.R).

```
library(msm)

Q=matrix(nrow=4, ncol=4,
                c(0,   0.1, 0.1, 0.01,
                  0.1, 0,   0.1, 0.01,
                  0.1, 0.1, 0,   0.1,
                  0.1, 0.1, 0.1, 0))

# Fit a multi-state Markov model
db_msm=msm(dbs ~ date, subject=X, data=uses,
                          qmatrix=Q, gen.inits=TRUE, exacttimes=TRUE)

# Extract the estimated transition probability matrix from the fitted
# model at time 365 (a year)
pmatrix.msm(db_msm, t=365)
# Estimate mean time spent in each transient state of the model
sojourn.msm(db_msm)
```



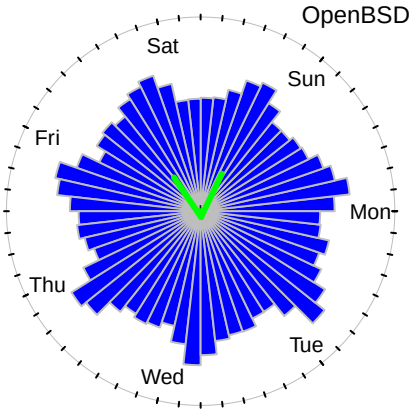


Figure 11.81: Rose diagram of number of commits in each 3-hour period of a day for Linux and FreeBSD. Data from Eyolfson et al.[566] Github–Local

11.12 Circular statistics

Some measurements are based on a circular scale, with values wrapping around to the minimum value when incremented past the maximum value, e.g., time of day, or months of the year. Circular statistics[1477] is the name given to the analysis of data measured using such a scale. Circular statistics has only become widely studied in the last 40 years or so, and techniques for handling operations that are well-established in other areas of statistics are still evolving. The `circular` package supports the analysis of data measured using a circular scale.

Differences between measurements based on circular and linear scales include:

- plotting uses a polar representation, rather than x/y-axis (the `circular` package includes support for `plot`, `lines`, `points` and `curve` functions),
- the mean has two components: mean direction ($\overline{\theta}$, an angle) and mean resultant length ($\overline{R}$; if this value is zero, the data has no mean), returned by the `mean` and `rho.circular` functions respectively (the `trigonometric.moment` function provides another way of obtaining this information). The `median.circular` function returns a median (multiple medians may exist, but only one is returned),
- the term variance, on its own, is ambiguous. The *circular variance*, V, is defined as $V = 1 - \overline{R}$ and varies between zero and one. Another measure is *angular variance* (returned by the `angular.variance` function), which varies between zero and two.

The *circular standard deviation* is returned by the `sd.circular` function (it is not calculated by taking the square root of the variance; its formula is: $\sqrt{-2\log\overline{R}}$),

- the von Mises distribution plays a role similar to that filled by the Normal distribution on linear measurement scales.

The mean resultant length, $\overline{R}$, is a measure of how spread out data points are around the circle. If the points have a symmetric distribution $\overline{R}$ equals zero and if all the points are concentrated in one direction $\overline{R}$ equals one; for unimodal distributions the term *concentration* is applied to $\overline{R}$ to denote the extent to which measurements concentrate around the mean direction.

Figure 11.81 is a Rose diagram of the number of commits to Linux and FreeBSD for each 3-hour period of the days of the week (the same data is plotted using a linear scale in figure 5.6).

The `rose.diag` function plots Rose diagrams. By default, the area of each segment is proportional to the number of measurement points in the segment (the behavior used when plotting histograms).

```
library("circular")

# Map to a 360-degree circle
HoW=circular((360/hrs_per_week)*week_hr, units="degrees", rotation="clock")
rose.diag(HoW, bins=7*8, shrink=1.2, prop=5, axes=FALSE, col=col_str)
axis.circular(at=circular(day_angle, units="degrees", rotation="clock"),
        labels=c("Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"))

text(0.8, 1, repo_str, cex=1.4)
arrows.circular(mean(HoW), y=rho.circular(HoW), col=pal_col[2], lwd=3)
```

The arrow at the center shows the direction of the mean, and the length of its shaft is its resultant length. Linux has fewer commits at weekends, compared to weekdays, and a mean direction near the middle of the week looks reasonable. The number of commits to FreeBSD does not seem to vary between days; the mean length is 0.03 (it almost does not have a mean), compared to Linux's mean length of 0.2.

If the measurement scale is very granular (e.g., measuring commit time once by day, rather than hour or minute), then $\overline{R}$ will be underestimated, and introduce errors in the calculation of the various location measures that use this value; see Github–statistics/circular/circle-bin.R. The correction to $\overline{R}$, for calculating circular standard deviation, when measuring in units of days rather minutes, is to multiply the calculated value of $\overline{R}$ by 1.034 (the correction for higher order moments involves much larger values).

11.12.1 Circular distributions

Circular distributions that are often encountered include the uniform distribution, wrapped Cauchy, wrapped von Mises and the Cartwright distributions.

Figure 11.82 shows differences in the shapes of three popular, symmetrical, single peak, wrapped circular distributions.[xxxiv] The *Jones-Pewsey distribution* includes them all, and others, as special cases.

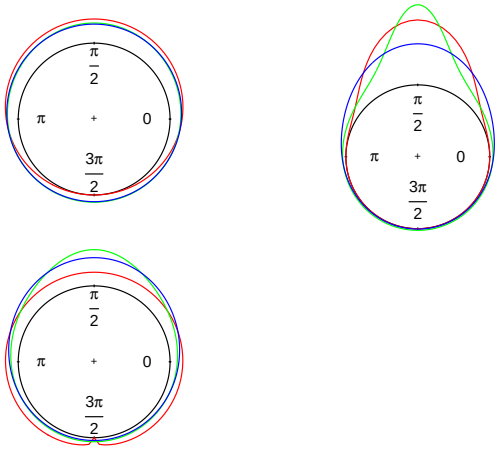


Figure 11.82: The Cartwright (red; `dcarthwrite`), wrapped Cauchy (green; `dwrappedcauchy`) and wrapped von Mises (blue; `dvonmises`) circular probability distributions for various values of their parameters. Github–Local

Figure 11.83 shows asymmetric extended forms of some common circular distributions. The `circular` package does not include support for asymmetric distributions, but code is available in Pewsey et al.[1477]

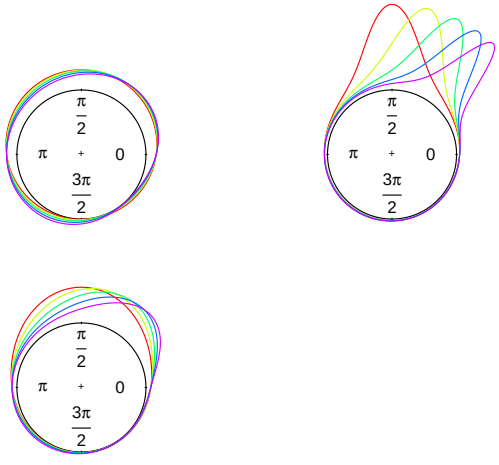


Figure 11.83: Asymmetric extended wrapped forms of the Cardioid (upper), von Mises (middle) and Cauchy (lower) probability distributions for various values of their parameters. Github–Local

The *discrete circular uniform distribution* consists of m points, equally spaced around a unit circle, with each point occurring with probability $\frac{1}{m}$.

The *continuous circular uniform distribution* treats all directions as being equally likely; the probability of point occurring between the angles ϕ and ψ is: $P(\phi < \theta < \psi) = \frac{\phi-\psi}{2\pi}$. The `circular` package supports the `dcircularuniform` and `rcircularuniform` functions, but not `p` and `q` forms.

The choice of which circular uniformity test to use, for measurements on a continuous scale, i.e., many possible measurement points around the circle, depends on how the data is thought to deviate from uniformity. The two uniformity deviation possibilities are:

[xxxiv] The implementation of the Cartwright distribution, up to at least version 0.4.93 of the `circular` package, uses the spelling `carthwrite`.

- a single peak over some range of values, i.e., a unimodal distribution. In this case the Rayleigh test is the most powerful known test; available in the `rayleigh.test` function,
- multiple peaks in the distribution of values around the circle. There are three tests that are more powerful than the Rayleigh test, when the data distribution could be more complicated than a single peak, but no single one is superior to the others; support for these tests is available in the `kuiper.test`, `watson.test` and `rao.spacing.test` functions.

Unless there is a good reason to think that the measurements could have a single peak, one (or all) of the omnibus tests should be used.

Your author was once a member of a four-person compiler implementation team, all born in February; we all agreed that the best compiler implementers are born in February. An alternative, easier to test hypothesis, is that most compiler implementers are born in February. Your author ran a survey on his compiler oriented blog,[935] asking readers their birth month and whether they had spent more than four months working on a compiler project (132 responded, of which 82 had worked on a compiler).

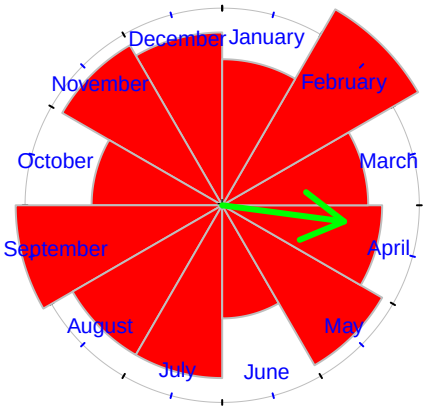


Figure 11.84: Number of readers of author's blog, whose birthday falls within a given month and who have worked on a compiler. Data from Jones.[935] Github–Local

Figure 11.84 shows that while February was the most common birth month, with 15% of implementers it is substantially below a majority (weighting by the number of days in a month, or the yearly percentage of births in each month, does not have much impact).

How likely is it that compiler implementer birthdays are uniformly distributed over the months? When the measurements have been grouped into a few bins, e.g., months of the year, a grouped data test has to be used; see Github–statistics/circular/grp-data-boot.R for examples using bootstrap. All tests for uniformity fail.

A study by Eyolfson, Tan and Lam[567] investigated the correlation between commit time and the likelihood of a fault being experienced because of a mistake in the commit code, for Linux and PostgreSQL. Figure 11.85 shows the number of non-fault commits (upper) and number of commits in which a fault was detected (lower), made in each hour of combined weekdays (the pattern of commits on weekdays differs from weekend days, and the following analysis is based on weekdays only).

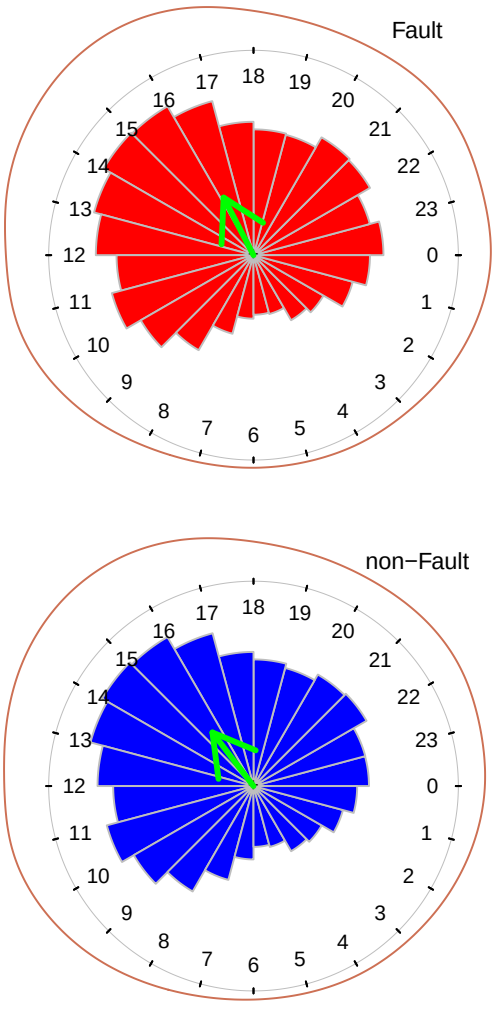


Figure 11.85: Number of commits (upper) and number of commits in which a fault was detected (lower) by hour of day of the commit, for Linux. Data from Eyolfson et al.[567] Github–Local

What differences, if any, exist between the two sets of daily commit times and in particular are commits made at certain times of the day more likely to cause a fault experience?

- testing for a common mean direction: The `watson.williams.test` function assumes that both samples are drawn from a von Mises distribution; the *Watson large sample non-parametric test* does not even require the samples to share a common shape; see Github–statistics/circular/common-mean.R. When any sample has a size less than 25, a bootstrap version of these tests should be used. The daily commit times do not share a common mean direction (15.5 hours for fault commits and 16.2 hours for non-fault commits); the mean result lengths are 0.33 and 0.32 respectively,

- testing for a common concentration: Are the points concentrated around a common direction? The *Wallraff test* is not supported by the `circular` package, but is described in Pewsey et al;[1477] see Github–statistics/circular/common-concen.R. The two commit samples do not share a common concentration.

11.12.2 Fitting a regression model

When one or more variables are measured on a circular scale the technique used to build a regression model depends on whether the circular variable is an explanatory or response variable.

When the response variable is measured on a linear scale, existing techniques and functions can be used; there may be one or more circular or linear explanatory variables,

When the response variable is measured on a circular scale, the `lm.circular` function, in the `circular` package, can be used

11.12.2.1 Linear response with a circular explanatory variable

Circular explanatory variables can be modeling using periodic functions, and the regression modeling techniques discussed in earlier sections; sine and cosine functions can be

combined to model any periodic function. As always, a model containing the fewest number of distinct parameters is desired.

The cosine function can be modified in various ways to change its shape, including:

$y = \alpha + \beta \cos(\omega x + \phi)$

and higher order harmonics can be added:

$y = \alpha + \beta_1 \cos(\omega x + \phi) + \beta_2 \cos(2\omega x + \phi) \cdots$

The shape of the peaks and troughs can be modified by adding a sine wave to the angular argument. In the following a positive λ sharpens the peaks and flattens the troughs while a negative λ has the opposite effect.

$y = \alpha + \beta \cos(\omega x + \phi + \lambda \sin(\omega x + \phi))$

A skewed period (which is what asymmetrical distributions have) can be modeled by adding a cosine wave to the angular argument (provided $-\pi/6 \leq \lambda \leq \pi/6$; outside this range it also affects other shape characteristics):

$y = \alpha + \beta \cos(\omega x + \phi + \lambda \cos(\omega x + \phi))$

These are all non-linear equations, which can be fitted using the `nls` function.

Figure 11.85 shows the number of commits to the Linux kernel, per hour; it is asymmetric, and the following code fits an extended cosine regression model (the `gam` values were estimated from the height of the cycle, and `omega` from fitting 24 hours into 2π radians):

```
basic_mod = nls(freq ~ gam0+gam1*cos(omega*hour-phi+nu*cos(omega*hour-phi)),
                       start=list(gam0=800, gam1=700, omega=0.3, phi=1, nu=0),
                       data=week_basic)
```

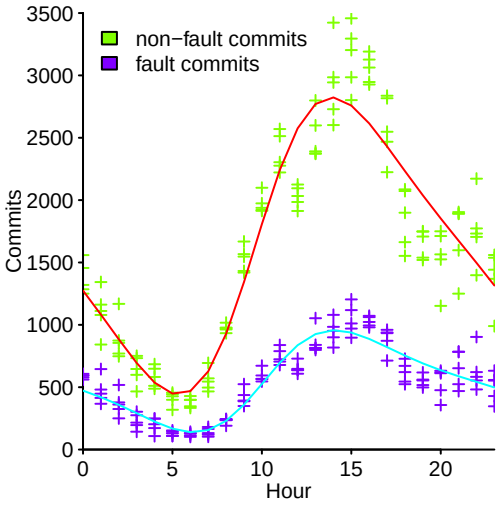


Figure 11.86: Number of non-fault related commits, and commits related to fixing a reported fault, per hour for weekdays, for linux; with fitted models. Data from Eyolfson et al.[567] Github–Local

Figure 11.86 shows the number of non-fault related commits, and fault related commits, per hour for every week day; with fitted models.

Both fits handle the skewed period but not the sharp peak and flat trough. A sine contribution can be added to help handle this shape and improve the fit, the call to `nls` is below:

```
basic_2mod = nls(freq ~ gam0
                        +gam1*cos(omega*hour-phi+nu*cos(omega*hour-phi))
                        +gam2*cos(2*omega*hour-phi+nu*sin(omega*hour-phi)),
                             start=list(gam0=800, gam1=700, gam2=100,
                                        omega=0.3, phi=1, nu=0),
                   data=week_basic)
```

Figure 11.87 overlays the fitted curve for non-fault and fault (red) commits over the non-fault hourly commits for each workday.

The `lm.circular` function supports circular response variables.

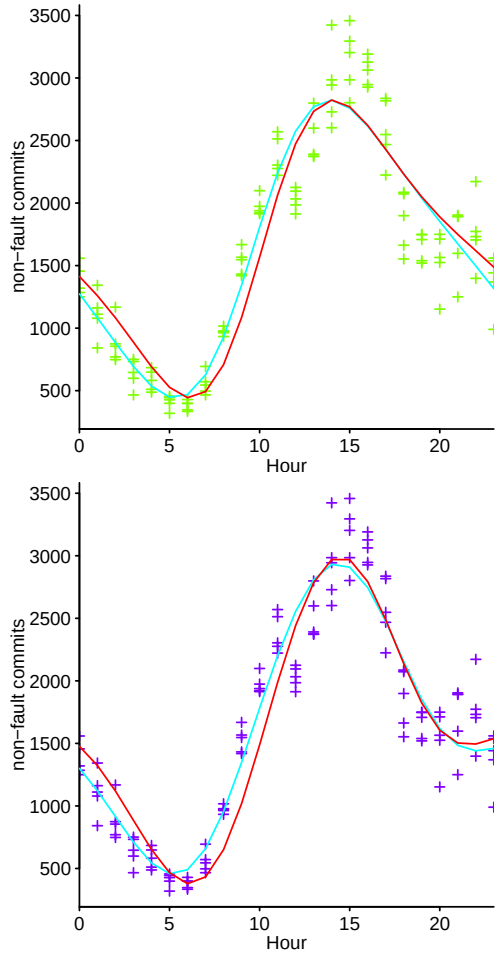


Figure 11.87: Number of commits per hour for each weekday, fitted using $\cos(\ldots\cos\ldots)$ (upper), and $\cos(\ldots\cos+\sin\ldots)$ (lower), for Linux; in both cases the fitted fault model (red) has been rescaled to allow comparison. Data from Eyolfson et al.[567] Github–Local

11.13 Compositional data

A study by Machiry, Tahiliani and Naik[1184] measured the performance of two application test generators, by comparing the number of lines of program source code covered by the tests generated by each tool (50 Android apps were tested); human performance was also measured. The application source lines covered by human and tool generated tests was recorded.

One measure of human vs. tool performance is to compare just those source lines that are covered by tests. What percentage, for each application, is covered by both human and tool generated tests, and what percentage uniquely covered by human or tool tests?

Figure 11.88 shows three quantities in one plot. For each of the 50 applications, the source line coverage common to human and Dynodroid tests (as a percentage of all covered lines), percentage only covered by Dynodroid generated tests and coverage of human only tests. Normalizing coverage counts, to a percentage of source lines covered, allows performance across different applications to be compared.

Fitting three regression models, one for each kind of coverage, using application source lines as the explanatory variable fails to make use of all the available information, i.e., the

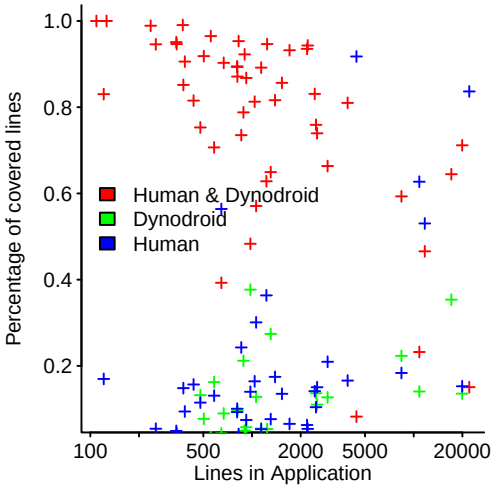


Figure 11.88: Lines of source against percentage test coverage achieved by both Human & Dynodroid tests, only by Dynodroid tests and only by Human tests, for each of the 50 applications. Data from Machiry et al.[1184] Github–Local

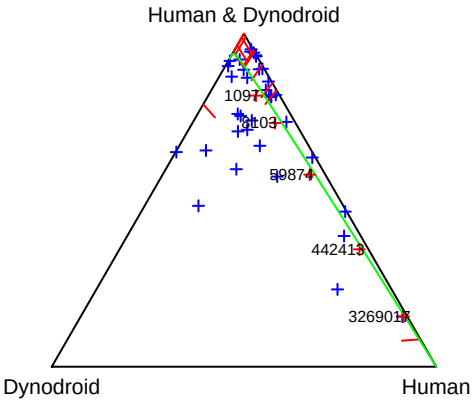


Figure 11.89: Ternary plot composed from source lines covered by both Human & Dynodroid tests, by only by Dynodroid tests and only by Human tests (measurements in blue); fitted regression line (green) and prediction points (red) for various total source lines (numeric values). Data from Machiry et al.[1184] Github–Local

relationship between the three percentages. A method of combining the three percentages into a single entity, that can be used as a response variable, is required. The *isometric log-ratio transformation*, ilr, is one possibility, and the `compositions` package supports the `ilr` function.

Figure 11.89 shows the same information in a ternary plot (in blue), along with a fitted regression model (green line). The explanatory variable is application source, and the red plus signs show predictions for various totals (tick marks on the axis are measurement points where one of the three components is zero). The quality of fit is very poor, with potentially many outliers and non-constant variance; other explanatory variables, not present in the data, may enable the building of a better fitting model.

The clustering of points near the Human & Dynodroid vertex shows that tests created by these two generators tend to cover the same source lines. More points are near the Dynodroid axis than the Human axis, suggesting that Dynodroid generated tests cover fewer unique source lines.

The following code was used to fit the regression model:

```
library("compositions")

covered=acomp(dh, parts=c("LOC.covered.exclusively.by.Dyno..D.",
                          "LOC.covered.exclusively.by.Human..H.",
                          "LOC.covered.by.both.Dyno.and.Human..C."))

plot(covered, labels="", col=point_col, mp=NULL)
ternaryAxis(side=0, small=TRUE, aspanel=TRUE,
                Xlab="Dynodroid", Ylab="Human", Zlab="Human & Dynodroid")

dh$l_total_lines=log(dh$Total.App.LOC..T.)

comp_mod=lm(ilr(covered) ~ I(l_total_lines^2), data=dh) # fit model

d=ilrInv(coef(comp_mod)[-1, ], orig=covered) # extract model coefficients
straight(mean(covered), d, col="green")      # line of fitted model
```

The `acomp` function normalises the columns passed as an argument using a ratio scale, and returns an object having class `acomp` (named after Aitchison, who pointed out the useful mathematical properties that a ratio scale bring to compositional analysis).

The `ilr` function is not currently handled by `glm`, so `lm` has to be used. Understanding the following code requires a lot more background knowledge than is appropriate here; see van den Boogaart and Tolosana-Deldago[1872] for more details.

Chapter 12

Miscellaneous techniques

12.1 Introduction

This chapter covers techniques which produce results that do not have the explicit equational form available with regression models.

12.2 Machine learning

Machine learning is the name given to a collection of techniques for automatically building a black-box prediction model, learned from training examples.

Users of machine learning do not need to understand the data (although it helps if they do), and as such, this approach to model building is ideal for clueless button pushers. From time to time, we are all clueless button pushers; machine learning is an easy-to-use tool that can help find a path through the fog.

This book's emphasis is on understanding the processes involved in software engineering, not building black-box prediction models.

The quality of the predictions made by models built using machine learning, depend on the quality of the training data used. It is worth noting that: the blacker the prediction box, the faster feedback is needed on prediction accuracy. Following black box predictions, without regular feedback on their accuracy is a recipe for disaster.

A sample containing information about many variables is always useful to have; domain knowledge might be used to select an appropriate subset. However, when all the variables are included in the analysis at the same time the *curse of dimensionality* arises.

A common metric used by machine learning algorithms is the distance between points. Each measurement can be viewed as a point in an n-dimensional space, where n is the number of attributes associated with each measured item. For ease of comparison in the following analysis, every side in this n-dimensional space is assumed to have length one, and so its volume is also one. In 3-dimensions the volume of a sphere of diameter one is $\frac{4}{3}\pi 0.5^3 \rightarrow 0.52$, that is the sphere occupies 52% of the unit cube, i.e., if the unit cube contains multiple points, there is a 52% probability that a point at the center of the unit cube is within 1-unit distance of another point. As the number of dimensions increases the sphere/unit cube volume ratio increases to a peak at five dimensions, and then decreases rapidly. Figure 12.1 shows how the volume of a sphere changes, relative to the volume of the unit cube, as the number of dimensions increases.

As the number of dimensions increases, the distance from a point to the point nearest to it approaches the distance to the point furthest from it;[191] an effect that can occur with as few as 10-15 dimensions. This behavior means that any algorithm relying on distance between points effectively ceases to work at higher dimensions.

Text analysis Software engineering produces often produces large quantities of text written in natural language, e.g., English. Like source code, this natural language text is raw material from which useful information might be extracted.

Automated extraction of semantics from natural language is an unsolved problem, for the general case. Approximate answers can sometimes be obtained to specific kinds of

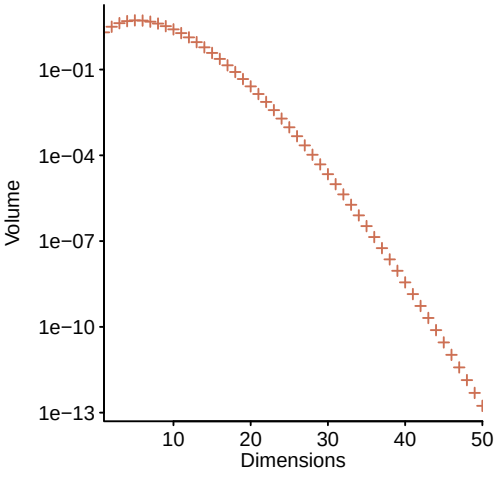


Figure 12.1: Volume of unit sphere in 1 to 50 dimensions, e.g., sphere has volume $\frac{4}{3}pi$ in three dimensions. Github–Local

semantic questions. Some algorithms are based on using prelearned examples, e.g., sentiment analysis is a popular technique for estimating whether text expresses a positive and negative opinion, but the results depend on the training data and tool used[943] (researchers are starting to collate software engineering specific training data[1146]). Dependence on training data is an important issue for any approach based on using pretrained models.

Available R packages for text analysis include `tm` (along with extension packages, such as `tm.plugin.mail` for processing emails and `tm.plugin.webmining` for mining web pages), and `spacyr` provides an interface to the spacy.io natural language processing system. For an example, see Github–faults/reopened_text.R.

12.2.1 Decision trees

As the name suggests decision tree models take the form of a tree like structure. Each node of the tree contains either an expression whose result is used to select which of two branches to follow, or a value denoting the result. The `rpart`[i] package supports the creation of binary decision trees.

Tree models are popular for use cases where a model is needed that can be interpreted by the people making a decision, based on what they observe, e.g., Doctors. Decision trees are the canonical example of a machine learning model that is not a black-box.

Each tree node contains a binary relationship, which selects the branch to follow to the next node, with the process continuing until a leaf node is reached. The model building process decides whether a leaf node should be split into a condition node and two leaf nodes using a method known as *cost complexity pruning*; any node split that does not improve the overall fit by a factor of `CP` is not attempted.

A study by Shihab, Ihara, Kamei, Ibrahim, Ohira, Adams, Hassan and Matsumoto[1695] investigated reopened faults in the Eclipse project. Of the 18,312 bug reports, 3,903 were resolved (i.e., closed at least once) and 1,530 of these could be linked to code changes. Of the 1,530 that could be linked to code changes, 246 had been reopened at the time of the study. Shihab et al cast their net very wide, extracting 22 factors that could possibly be associated with reopened faults.

Figure 12.2 shows the first few levels of a fitted decision tree, which is visibly very cluttered. The names of the people reporting and fixing problems is part of the fitted model, resulting in some overly long lines (names have been truncated to two characters, so something is visible). The `rpart` package provides basic plotting functionality, and `rpart.plot` package provides much more functionality; the following is the essential code:

```
library("rpart")
library("rpart.plot")

dt=rpart(remod ~ time+week_day+month_day+month+time_days+description_size+
              severity+priority+pri_chng+ num_fix_files+num_cc+prev_state+
              fixer_exp+fixer_name+reporter_exp+reporter_name,
        data=raw_data, weight=data_weight,
        method="class", x=TRUE, model=TRUE, parms=list(split="information"))
rpart.plot(weighted_model, cex=1.2, split.font=1, under.col=point_col,
                box.palette=c("green", "red"), branch.col="grey",
                under=TRUE, type=4, extra=100, branch=0.3, faclen=2)
```

[i] Recursive partitioning.

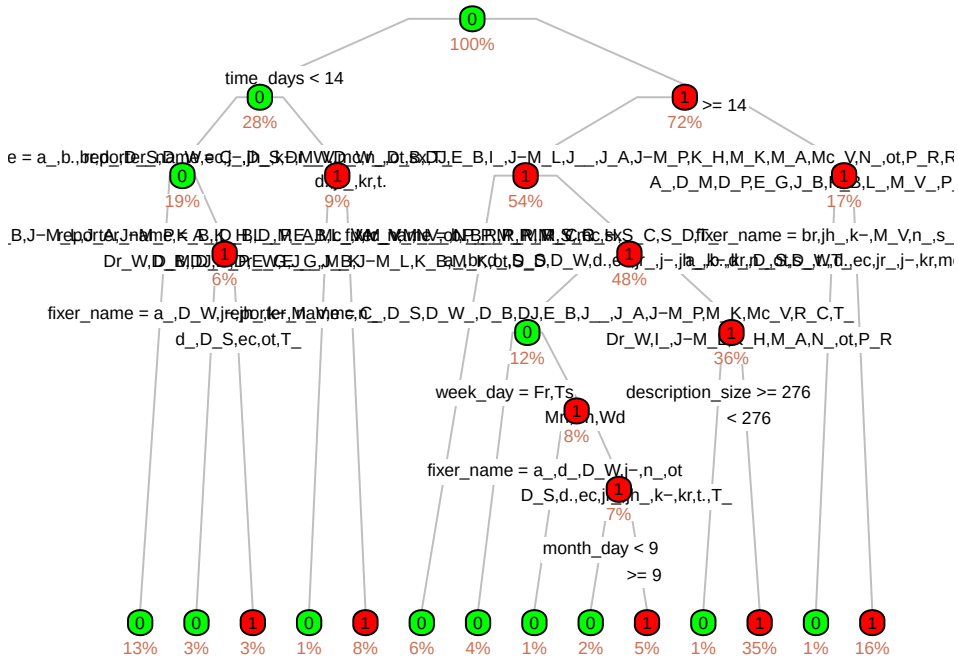


Figure 12.2: Top levels of the decision tree fitted to the reopened fault data (overly long lines are names of people who reported and fixed the fault). Data from Shihab et al.[1695] Github–Local

Which variables contribute most to the model? The `summary` (the `cp=0.4` argument removes lots of details from the output; the `printcp` function does not provide any information on variable importance):

```
> summary(weighted_model, cp=0.4)
Call:
rpart(formula = remod ~ time + week_day + month_day + month +
    time_days + severity + priority + pri_chng + num_fix_files +
    num_cc + prev_state + description_size + fixer_exp + fixer_name +
    reporter_exp + reporter_name, data = raw_data, weights = data_weight,
    method = "class", model = TRUE, x = TRUE, parms = list(split = "information"))
  n= 1530

          CP nsplit rel error    xerror       xstd
1 0.17010632      0 1.0000000 1.0792683 0.01972982
2 0.02814259      1 0.8298937 0.8900876 0.01966538
3 0.02751720      2 0.8017511 0.8905566 0.01966642
4 0.02095059      5 0.6957473 0.8858662 0.01965584
5 0.01485303      6 0.6747967 0.8952470 0.01967656
6 0.01414947      7 0.6599437 0.8586617 0.01958573
7 0.01407129      9 0.6316448 0.8685116 0.01961284
8 0.01219512     10 0.6175735 0.8708568 0.01961901
9 0.01000000     13 0.5795810 0.8771107 0.01963491

Variable importance
      fixer_name    reporter_name        time_days         week_day
              32               32               13                6
description_size        fixer_exp     reporter_exp        month_day
               4                3                2                2
        priority         severity       prev_state    num_fix_files
               1                1                1                1
           month
               1

Node number 1: 1530 observations
  predicted class=0  expected loss=0.4990637  P(node) =1
    class counts:  1284 1279.2
   probabilities: 0.501 0.499
```

The variables making the largest contribution to the model, and a measure of their relative importance, appear at the end (at least when a large `cp` argument is passed).

For the identity of people fixing and reporting problems to play such a large role in the model, either this subset of people do work that is more likely to need to be looked at again in the future , or the faults they close have some characteristic that causes the faults they close to be reopened. This issue cannot be analyzed further using the available data.

The columns of numbers in the middle of the output contain two measures of error, for various values of CP. Decision trees are susceptible to overfitting, and the xerror column estimates the error using ten-fold cross validation (the error listed in the rel error column does not use cross validation and gives a rosier estimate). The output above suggests that building a model using the CP value listed in the second row is likely to produce more accurate results than other values (the default value of CP is 0.01).

See Github–odds-and-ends/wcre2012-delaystudy.R for an example of a decision tree analysis of data containing many variables.

12.3 Clustering

Clustering is the process of grouping together items (into one or more clusters), such that items in a given cluster are more similar to each other than items in other clusters. Some commonly used measures of similarity include distance between items, density of items, and distance from a set of items chosen to be representative of some collection of characteristics of interest.

A clustering approach to data analysis requires a method for measuring item similarity, and an algorithm for using this information to group the appropriate items within the same cluster. Examples of attempts to understand data, via clustering, include: location based distance (fig 2.19), similarity between Linux distributions based on packages they contain (fig 4.17), trading relationships between companies (fig 4.40), developers contributing to the same Apache project (fig 4.32), density of variables experiencing a given number of read/writes (fig 7.46) and correlation between attributes of Github pull requests (fig 8.9).

A study by Kanda, Ishio and Inoue[971] reverse engineered the evolutionary history of nine large software systems by comparing the similarity of the files containing the source code used to build successive releases. Figure 12.3 shows an unrooted tree representation of the phylogenetic tree estimated from the paired similarity of corresponding files contained in various releases of OpenBSD, FreeBSD and NetBSD.

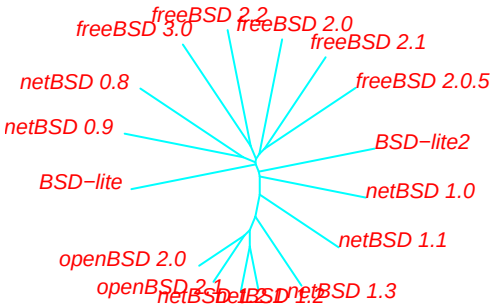


Figure 12.3: Unrooted tree denoting a phylogenetic tree estimated from the paired similarity of the corresponding source files contained in some releases of the major variants of BSD unix. Data kindly supplied by Kanda.[971] Github–Local

12.3.1 Sequence mining

Recurring subsequences may occur in a collection of related item (or event) sequences. A common application of sequence mining is recommendation systems, such as finding items that shoppers often buy together, or in sequence (e.g., as children grow up), or shared buying patterns between groups of shoppers.

Common sequences often occur in calls to a given API, e.g., open/read/close.

The arules package supports the mining of association rules and frequent item sets.

Given N sequences of items, the fraction of these sequences containing a particular item, say X, is known as the *Support* for X. Given that X appears in a sequence, what is the likelihood that Y will also appear in that sequence (written $\{X \Rightarrow Y\}$)? The following are two common answers:

$$Confidence\{X \Rightarrow Y\} = \frac{Support\{X,Y\}}{Support\{X\}}$$

$$Lift\{X \Rightarrow Y\} = \frac{Support\{X,Y\}}{Support\{X\} \times Support\{Y\}} = \frac{Confidence\{X \Rightarrow Y\}}{Support\{Y\}}$$

A study by Fowkes and Sutton[627] investigated the sequence of API calls made within the method bodies of 17 Java systems. The following call to apriori searches for sequences of method calls (in the drools business management system) having a given *support* and *confidence*; see Github–odds-and-ends/drools.R:

```
library("arules")

drools=read.transactions(paste0(ESEUR_dir, "odds-and-ends/drools.csv"),
                         format="single", cols=c(1, 2))

rules=apriori(drools, parameter=list(support=0.0001, confidence=0.1))

summary(rules)
inspect(head(rules, n=3, by = "confidence"))
```

The number of rules, of a given length, found were (`lhs` and `rhs` refer to the two sides of the $\Rightarrow$ symbol):

```
rule length distribution (lhs + rhs):sizes
   2    3    4    5
1478  252   32    5
```

If many results are returned, some form of visualization can help reduce the effort needed to appreciate the distribution of results. The `arulesVis` package supports a variety of ways of visualizing association mining results. Figure 12.4 shows what is known as a *two-key plot* of the above results; the colored orders indicates the number of items contained in each rule.

The `apriori` algorithm treats each item, in a sequence, as being independent. The order of method calls may be significant, and the `arulesSequences` package adds sequence mining functionality to the `arules` package.

A required call may be missing from a sequence of method calls; the `recommenderlab` package provides support for developing and testing recommendation algorithms.

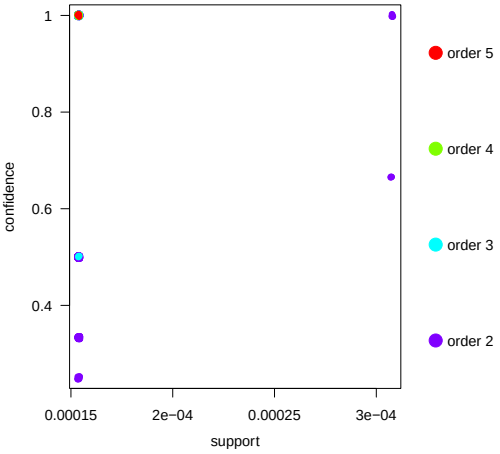


Figure 12.4: A two-key plot of associating mining results; order indicates number of items in rules. Data from Fowkes et al.[627] Github–Local

12.4 Ordering of items

Arranging items in order can reveal information about the form of the calculation used to select the relative positions of ordered items. Given multiple orderings of the same items, ordering patterns of subsequences of items, common to multiple sequences may indicate a shared semantics for those items.

A ranking is created when items are placed in an order relative to each other. Items are rated when, for instance, people are asked to provide a relative rating based on an ordinal scale, e.g., the extent to which a person like/disliked a book. The analysis of rating data is discussed in section 13.4.

12.4.1 Seriation

Seriation is the process of placing items into a linear order, based on a metric derived from some item characteristics. The number of possible item orderings grows as $n!$, making it impractical to evaluate every possibility for non-trivial sample sizes; heuristic algorithms have to be used. The `seriation` package supports a variety of functions that attempt to find an optimal linear ordering of items; see fig 4.32 and fig 7.12.

A study by Jones[934] investigated the extent to which developers create similar data structures to hold information listed in a specification, e.g., grouping together identifiers containing related information in the same data structure. The hypothesis was that shared cultural and professional experiences would result in subjects defining data structures containing similar contents.

Subjects were given a list of items from the "Department of Agriculture", and asked to design a C/C++ API containing this information. The results, from each subject, were the `structs` or `classes` defined and their fields/members (with each field containing one item of API information, e.g., "Date crop harvested" and "Organically produced").

Ordering the data highlights API information that tends to be colocated in the same data structure, and the subjects making similar choices.

Figure 12.5 shows the items placed in the same data structure as the information item "Antibiotics used", by each subject (colored squares indicate presence). The matrix passed to `seriate` contains boolean values (indicating presence in the same data structure), and the subjects/fields are ordered so that shared usage appears adjacent. The `bertinplot` function provides a particular visualization of the ordered data.

```
library("seriation")

fser=seriate(fmat, method="BEA",  control = list(rep = 10))
bertinplot(fmat, fser, options=list(panel=panel.squares, spacing=0,
                        gp_labels=gpar(cex=0.6)))
```

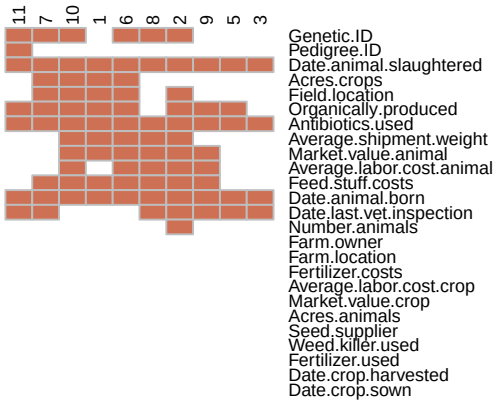


Figure 12.5: A Bertin plot for items included in the same data structure as the item "Antibiotics used", for each numbered subject, after reordering by `seriate`. Data from Jones.[934] Github–Local

A single item's pattern of association, with all the other items, can be generalised by counting occurrences of every pair of items in the same data structure. A Robinson matrix has the property that the value of its matrix elements decrease, or stay the same, when moving away from the major diagonal; this matrix has been used to study commonality in subjects' categorization behavior.[1605] Figure 12.6 shows a visualization of a Robinson matrix for the Jones data.

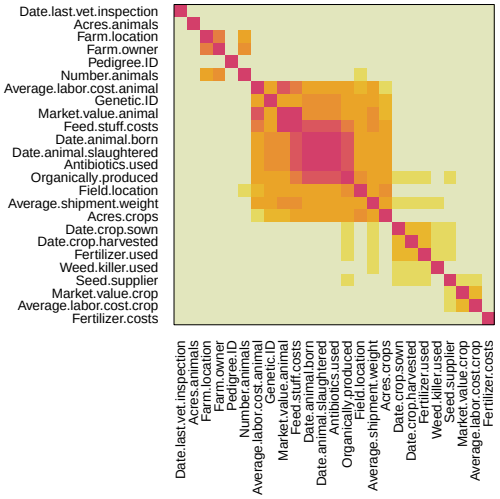


Figure 12.6: A visualization of the Robinson matrix based on number of times pairs of items co-occur in the same data structure (the closer to the diagonal the more often they occur together). Data from Jones.[934] Github–Local

```
library("seriation")

fdist = as.dist(1 - fmat/max(fmat)) # Normalise counts
fser = seriate(fdist, method="BBURCG")

pimage(fdist, fser, col=pal_col, key=FALSE, gp=gpar(cex=0.8))
```

12.4.2 Preferred item ordering

A list of items may have a preferred ordering. The ordering may be the result of ranking, rating or a comparison between pairs of items.

Bradley-Terry statistics are a traditional technique for calculating an ordering for a list of items, based on results from pairwise comparisons, e.g., football match results (there is an extension for analyzing results from simultaneous comparisons of more than two items). This section is based around use of the `BradleyTerry2` package for simple paired comparisons, and the `PlackettLuce` package for the more complicated cases.

An alternative approach to analyzing item ordering is discussed in section 9.6.1.

Given a *contest* between i and j, the probability that i beats j is assumed to have the form:

$P(i > j) = \dfrac{\alpha_i}{\alpha_i + \alpha_j}$, where α_i and α_j might be thought of as some measure of *ability* of i and j.

Bradley-Terry statistics uses logistic regression to model this equation, and fits the βs in the following equation:

$P(i > j) = \dfrac{e^{\beta_i}}{e^{\beta_i} + e^{\beta_j}}$, where: $\alpha_i = e^{\beta_i}$ and $\alpha_j = e^{\beta_j}$.

A study by Jones[931] investigated developer beliefs about binary operator precedence. Subjects saw an expression, such as `a + b % c`, and were asked to insert parenthesis such that the behavior (as interpreted by the compiler) remained unchanged. Based on subject answers, what is the relative precedence of the binary operators used in the study (which may be different from the actual precedence)?

The parenthesis specifies the operator that can be treated as *winning* a precedence contest. The `BTm` function, in the `BradleyTerry2` package[1855] takes the results from paired comparisons, and returns the coefficients of a fitted model (internally, the `glm.fit` function is used to fit a logit regression model).

```
library("BradleyTerry2")

prec_BT=BTm(cbind(first_wl, second_wl), first_op, second_op, data=nodraws)

summary(prec_BT)

# A less interesting plot than the one specifically created
plot(qvcalc(BTabilities(prec_BT)), col=point_col, main="",
       xlab="Operator", ylab="Relative order")
```

The output produced by `summary` has the same form as that produced for other regression models. Note: the `summary` function does not list the factor with value zero (the subtraction operator, for this data). The `BTabilities` function lists all factors and their values, and the call to `plot` uses this information to visualize the estimated coefficients and their corresponding standard error.

Figure 12.7 shows the estimated β coefficients (along with a corresponding standard error), and can be used to estimate subjects beliefs about relative binary operator precedence. The probability of, for instance, equality, ==, *winning* a precedence choice against

binary plus, +, is (based on the values returned by the fitted model): $\frac{e^{-2.08}}{e^{-2.08}+e^{0.26}} \to 0.088$, and the probability of binary plus, +, *winning* against == is: 0.912.

In a sports match, the home team is often considered to have an advantage over the away team. Perhaps, when developers are uncertain they are more likely to select the first binary operator, of a pair. A model can include information on first position *wins*, for each operator pair (the effect is very small for this data); see Github–developers/jones_prec.R for details.

The two item model can be extended to where three or more items are ranked by ordering them according to some preference criteria. For instance, for a ranking of three items:

$$P(i > j > k) = \frac{\alpha_i}{\alpha_i + \alpha_j + \alpha_k} \times \frac{\alpha_j}{\alpha_j + \alpha_k}$$

The `PlackettLuce` package supports the analysis of rankings of more than two items, tied ranks (i.e., items having the same rank), and rankings where some items do not appear in every ranking.

A study by Biegel, Beck, Hornig and Diehl[196] investigated the ordering of definitions appearing in 5,372 `classs`, from 16 Java programs. The Java coding conventions (JCC)[1795] recommends a particular ordering of the four kinds of definitions, and on average over 80% of classes in these projects follow the JCC ordering recommendation: *class variable* (or *field*), *instance variable* (or *static initializer*), *constructor* and *method*.

Java definitions include access control information, via the use of the keywords: `private`, `protected`, `public` and no keyword (the default behavior given in the language specification is used). The ordering of definitions in the source code, by identifier visibility (i.e., access control), can be treated as a ranking. The declarations in a source file may not contain an instance of every kind of access control, i.e., some rankings will include a subset of items.

Figure 12.8 shows the relative ordering (ranking) of method definitions by access control keyword, over all projects investigated; see Github–sourcecode/member-order/vis-key_order-pref.R, which also includes details of other kinds of declarations. If the methods defined in a class have three kinds of visibility (only 3% of cases in the study), the probability of, for instance, the visibility order being `public`, `protected`, then `private` is (using β values from the fitted model):

$$\frac{e^{0.36}}{e^{0.36}+e^{-0.36}+e^{-0.67}} \times \frac{e^{-0.36}}{e^{-0.36}+e^{-0.67}} \to 0.54 \times 0.58 \to 0.31$$

When teams are ranked, with varying team membership, the `hyper2` package may be of use in obtaining a ranking of the individuals.

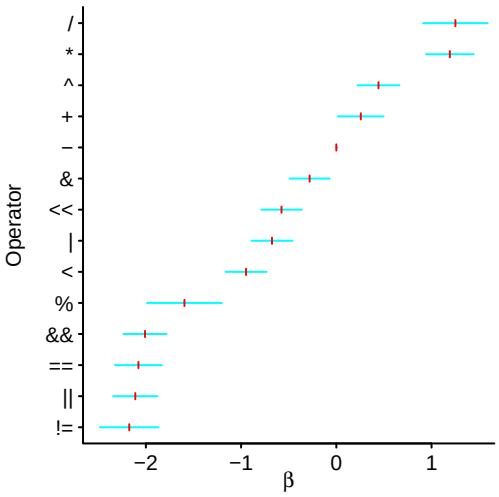


Figure 12.7: Relative ordering of binary operator precedence (i.e., value of β), and corresponding standard error, based on subject responses to binary operator precedence questions. Data from Jones.[931] Github–Local

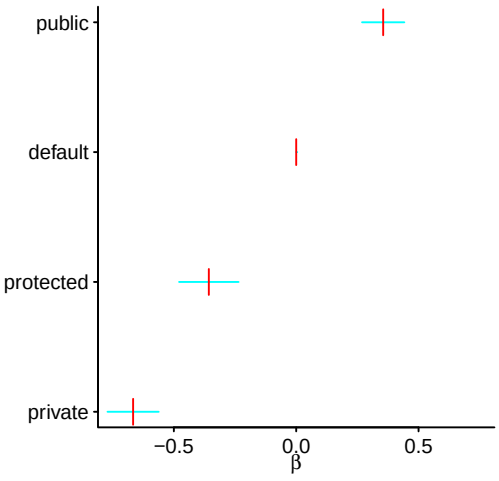


Figure 12.8: Fitted values of β for access control (visibility) of method definitions within a Java class. Data from Biegel et al.[196] Github–Local

12.4.3 Agreement between raters

A measurement may be based on human judgement, e.g., assigning a product rating. Different people may make different judgements of the same characteristic/entity, and a way of evaluating the agreement between the different judgements is needed.

Cohen's Kappa is a measure of inter-rater agreement between two raters, it varies from zero (no agreement) to one (perfect agreement). Fleiss's Kappa is a measure of inter-rater agreement between three or more raters.

The `kappa2` function, in the `irr` package, supports Cohen's Kappa (weighting is supported by passing the argument `weight="squared"`); the `kappam.fleiss` function calculates Fleiss's Kappa.

A study by Schach, Jin, Yu, Heller and Offutt[1640] categorised the kinds of maintenance activity performed on various systems. Table 12.1 lists the categories assigned by two raters to 215 maintenance categories involving the first 20 versions of Linux. The Cohen's Kappa for these two raters is 0.805; see Github–group-compare/agreement.R.

12.5 Simulation

Simulating a process or system, via an executable model, is a means of gaining information about the operational characteristics of the process or system (assuming the characteristics of the model are sufficiently accurate). Varying the characteristics of a simulation

	Adaptive	Corrective	Perfective	Other	Total
Adaptive	2	0	0	0	2
Corrective	0	82	16	0	98
Perfective	0	5	99	2	106
Other	0	0	0	9	9
Total	2	87	115	11	215

Table 12.1: Maintenance categories assigned by two raters (row and column) for the first 20 versions of the Linux kernel at the change-log level. Data from Schach et al.[1640]

model can provide possible answers to what-if questions. Some of the kinds of simulation methods available include:

- discrete event simulation: the system modeling is based on the discrete events that can occur; supported by the `simmer` package, e.g., staff scheduling, see Github–projects/impl-sim.R,
- agent-based modeling: a set of agents, having specified attributes, interact with each other in a computer simulation. After a given number of time steps the state of the system is measured, with the results from many simulation runs combined to calculate probabilities for the various end-states. NetLogo is a widely used system for agent-based modeling and the `RNetLogo` package provides an R interface,
- system dynamics: represents a system using causal loops between all the interacting components, essentially an analogy representation of differential equations. This approach has been used to simulate software project staffing,[3,275,1188]
- differential equations: if a system can be described using a set of differential equations, the `deSolve` package can be used to numerically solve these equations.

  Fitting sales data to the Bass diffusion model was discussed in section 3.6.3. Duggan[513] investigated the impact, on sales volume, of the spatial separation of potential customers living in 10 regions. Figure 12.9 shows the connections between regions, and the percentage of total population they contain, given as initial conditions in the numerical solution; see Github–odds-and-ends/RJ-2017.R for implementation details.

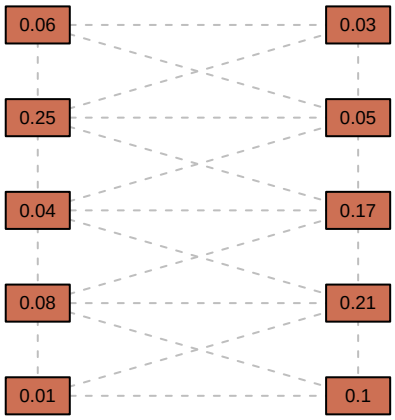


Figure 12.9: Region populations and their connections: initial conditions used in Duggan's[513] numerical solution of the Bass equation. Github–Local

Chapter 13

Experiments

13.1 Introduction

Does doing X have a significant effect on S? Traditionally X might have been a new kind of fertiliser (or drug) and S the crop yield (or being cured of some illness). In software engineering the effect sought is often a performance improvement, and X the latest snake oil. Detecting discontinuities in data is discussed in section 11.2.9.

In an observational study the researcher is a passive observer, simply recording what happened, or is happening. In an experimental study the researcher actively attempts to control the values of the explanatory variables (a common technique is to vary the values of one explanatory variable, while the others are held constant; in some cases, the environment in which events occur form a natural experiment, in that the explanatory variables of interest vary in a way that an experimenter might vary them).

A controlled experiment is the technique used to obtain the data needed to test a hypothesis. The controlled experiment that most developers are likely to be familiar with is benchmarking.

Dramatic changes in performance, after doing X, are relatively common in software development, and in such cases using statistics to confirm that a noticeable change has occurred is almost a formality. At the other extreme, differences that require statistical analysis to be detected are often not worth being concerned about in practice.

Advice for running experiments often follows the waterfall model of software development, with lots of upfront planning and little or no feedback from actual use until the end of the process. This advice has its roots in the environment in which experiments are carried out by the readership of many statistics text books, where running an experiment is costly (in money or time), or is a once only opportunity.[i]

Some experimental questions in software engineering are amenable to iteration. Running quick, inexpensive experiments, can be a cost effective technique for filtering possible questions of interest, and obtaining information on which, of the myriad of variables, have a worthwhile impact on the response variable(s).

Finding the right question to ask is sometimes the most useful output from running an experiment.

An important point to remember is, that, it is better to have an inexact answer to the right question, than an exact answer to the wrong question.

Like all software development activities, experiments have to pay their way. Some of the answers needed for a cost-benefit analysis include: the cost of running an experiment, capable of producing the information of interest, within acceptable confidence intervals; the usefulness of the data likely to be obtained, by running an experiment, using a given amount of resources (such as time and money).

Many software engineering tasks are performed within complex environments. Controlling and measuring all the variables in the environment has been perceived as being time-consuming and expensive that few researchers have been willing to attempt realistic controlled experiments. Consequently, much of the hypothesis testing performed in commer-

[i] Experiments in the social sciences, major producer of experimental studies, are often grant funded, with limited opportunities for rerunning experiments that failed to produce data that can be published.

cial environments has been based on convenience samples, obtained from experimentally uncontrolled, production software projects.

Running an experiment with minimal funding means that experimental subjects are often unpaid volunteers, from a pool containing who ever is available.

Software engineering is not known as a research area where experiments are commonly performed. A study[1717] of 5,453 papers in software engineering journals, published between 1993 and 2002, found that only 1.9% reported controlled experiments (of which 72.6% used students, only, as subjects), and the statistical power of many of these experiments fell below expected norms.[523]

13.1.1 Measurement uncertainty

The term *measurement error* is often applied to measurements involving physical quantities. It is based on the assumption that any different between the measured and actual value is caused by errors made by the measurement process.

In software engineering, some quantities can be measured exactly, e.g., lines of code in a source code file. However, the process that generated the code (e.g., a software developer) may produce a different number of lines, if repeated using a different developer or even the original developer (reimplementing the program); see fig 5.23. The code that is measured is a sample drawn from a population, and measurements involving this code needs to be treated as having an accompanying uncertainty.

Goodhart's law is an observation about human behavior, rather than a law: "Any observed statistical regularity will tend to collapse once pressure is placed on it for control purposes." If the measurements collected were actively used to control or evaluate the development team (for instance), then developers have a motivation to cause the measurements to move in a direction favorable to themselves.

A study by Perry, Staudenmayer and Votta[1472] investigated various software development activities (e.g., working time on an activity, and activities performed throughout the day), as measured by the developers involved (i.e., self reports) and as measured by external observers. Differences in reported measurement values included, developers reporting activity time 2.8% higher, on average, than that reported by an observer; the measurement agreement rate for activities performed varied between 0.6 and 0.95.

A series of studies[996] of social network data, as reported by those within the network and extracted from externally observed information, found that differences were large enough to render invalid any analysis of a network characteristics based on member supplied information.

Random variability in the performance of, what are intended to be, identical hardware components, is discussed in section 13.3.2.1.

Different tools, or different tool options may produce different results, e.g., the diff algorithms supported by Git,[1394] reported statement coverage,[1990] or clone detection technique.[1834]

A study by Li[1130] investigated the call graphs built by four Python tools. Figure 13.1 shows the number of nodes in the call graphs built by four tools, broken down by number of nodes common to each tool.

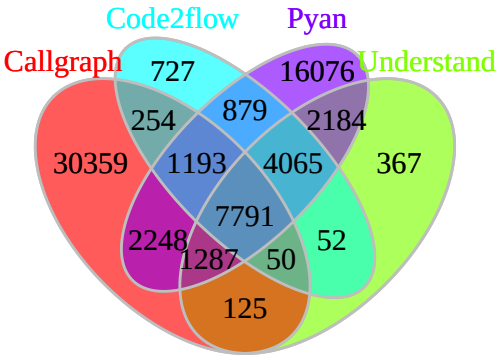


Figure 13.1: Number of nodes in the Python call graphs built by four tools, broken down by number of nodes common to each tool. Data from Li.[1130] Github–Local

Programs sometimes consume increasing amounts of memory, the longer they are run; sometimes known as *software aging*. One consequence of changes in the behavior of the environment in which a benchmark is executed, is to introduce systematic noise into the results.

A study by Cotroneo, Iannillo, Natella and Pietrantuono[410] investigated the impact of running two applications on three versions of Android, installed on four different phones, running with low/high available memory, and three kinds of event interactions (i.e., $2*3*4*2*3$ combinations of App, environment, and usage); there were an average of 272 App executions per combination, i.e., 39,187 total executions. A fitted regression model finds that the amount of memory used consistently changed by a small amount with each successive App execution, with the amount depending on environment and event use (less than 10 bytes); see Github–benchmark/2005-11523.R.

13.2 Design of experiments

Randomization is the foundation on which any claims of causation, involving experimental results, are based, i.e., doing X caused Y. Without randomization, the most that can be said is that a correlation has been observed between doing X, and Y occurring.

The purpose of running an experiment is to obtain data that can be used to help answer one or more questions. Experiments have to be carefully designed, to ensure that the data obtained is representative of the processes involved for the question(s) being asked. Design issues include:

- recruiting the appropriate subjects from the available population (sampling is discussed in section 10.2),
- creating an experimental task(s) that shares all the important characteristics of the tasks associated with the questions of interest,
- creating an environment for the subjects, in which they can give a suitable performance, during the experiment.

  Subject characteristics can sometimes interfere with good experimental design, e.g., human subjects have memories of their previous experiences that they cannot choose to erase,
- controlling all variables that could have a significant impact on the response variable(s) of interest. Failure to take into account, and control, variables having a significant impact, can cause a tiny effect to appear to be a large effect and vice versa. One person's tongue-in cheek-advice on how to bias an experiment to get the desired outcome[1288] is another person's list of thoughtless mistakes.

  When factors cannot be controlled, they need to have their impact contained. One technique for handling the problem of uncontrolled variables is to group subjects into blocks based on the variable that is suspected of influencing the response (a process known as *blocking*), randomization of subjects then occurs within each block. The identity of the block becomes another explanatory variable during analysis of the results.

A study by Basili, Green, Laitenberger, Lanubile, Shull, Sørumgård and Zelkowitz[140] compared the performance of subjects when using perspective based reading (which instructs reviewers to read a document from a specified perspective, e.g., a designer, tester or user), against the reading technique currently used by the professional developers who were the subjects.

The researchers thought it likely that, training subjects to use the new technique would alter their performance when using whatever technique they currently used, and decided to measure subject performance using their existing review technique first, before giving subjects training in the new, perspective based reading, technique.

Having all subjects use the perspective-based reading technique second, means it is not possible to separate out ordering effects in the results, e.g., effects such learning during the experiment, and any random distraction effects that only occurred at certain times.

Factors outside the control of these researchers, which could affect the results, include:

- the time taken for subjects to become proficient at using a new technique; old habits die hard. How much practice do subjects need, for them to be able to give a performance that makes it possible to reliable compare the new technique? In this study subjects were taught PBR two days after the first part of the experiment, they trained on a test document, reviewed one document, received more training and then reviewed another document.
- the kind of review technique used by subjects in the first half of the experiment. A change in performance is expected, but it is not known what technique any change is relative to (it is assumed that adhoc techniques are being used),
- the characteristics of the seeded faults. Were more faults found in the NASA documents because readers were familiar with reading that kind of document, or perhaps the characteristics of the seeded faults was such that they were harder to detect in one kind of document than another?

The experimental output included, for each subject, the number of faults detected (which has a known upper limit, and a yes/no detection status), and the number of false positives in each document reviewed (which has no upper limit, in theory); see Github–faults/basili/pbr-experiment.R for details of fitting a regression model.

Basing all experimental choices on random selection does not automatically create samples that maximise the information that can be obtained.

A study by Porter, Siy, Mockus and Votta[1507] investigated software inspections. The structure of the inspection process was manipulated by varying the number of reviewers (1, 2 or 4), number of meeting (1 or 2), and for multiple meetings whether reported faults were repaired between meetings (88 inspections occurred, involving 130 meetings and 17 reviewers).

Selecting the treatment to use, for a review, from successive entries on a randomised list of all possible treatment structure combinations (created at the start of the study), would ensure that the results are balanced across the variables of interest. However, in this study the choice of treatment to use was randomly selected from all possibilities, as each unit of code became available for review, resulting in some combinations of reviewers/meetings/repaired not being used, and some used very often. The results contain an unbalanced set of experimental conditions, making it difficult to fit a reliable model; see Github–experiment/porter-siy/inspection.R, Github–experiment/porter-siy/meeting.R and fig 11.33.

The complexity of computing platforms means their behavior can quickly change. A study by Barrett, Bolz-Tereick, Killick, Mount and Tratt[136] investigated the performance of Just-in-time compilers. The computer system (three different systems were measured) was rebooted and a benchmark run 2,000 times, this process was repeated 30 times, i.e., 60,000 executions of every benchmark on three systems. Every effort was made to reduce measurement noise, e.g., as many background processes as possible were disabled.

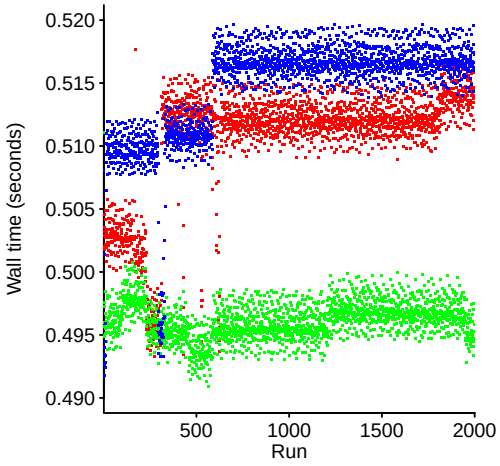


Figure 13.2: Time taken by 2,000 runs of a Javascript BinaryTree benchmark, with JIT enabled, on a quad-core Intel i7-4790; three colors are three iterations of the process: reboot machine, execute 2,000 runs. Data from Barrett et al.[136] Github–Local

Figure 13.2 shows the wall time taken for three sequences of 2,000 executions of a Javascript BinaryTree benchmark, running on a quad-core Intel i7-4790. The abrupt changes in performance match a change in the processor core used to execute the code; the benchmark process could have been locked to a given core, but would that be representative of real-life use? As discussed later (see fig 13.22) performance variability between system reboots can be larger than between a sequence of runs during one uptime.

13.2.1 Subjects

Experimental subjects might be people or artefacts, e.g., hardware or source code. An essential requirement for generalising the results from an experiment, to a larger population of subjects, is that the characteristics of the sample of experimental subjects are representative of the applicable characteristics of the population of interest. Statistical issues around sampling are discussed in section 10.2.

The major issues involved in having computer hardware as an experimental subject are covered in section 13.3, while the major issues in human cognitive performance are covered in chapter 2. A limiting factor, when designing an experiment involving human subjects, is typically the amount of time the subjects are likely to be willing to make available to participate.[1716]

When people are the subjects in experiments, a variety of human factors introduce uncertainty into the results; people constantly adapt to their environment, including the environment of an experiment (e.g., they learn and retain memories of their experiences; see section 2.5), they also experience fatigue, and their attention ebbs and flows during an experiment.

Professional developers, working within different ecosystems, may share a set of basic skills and knowledge, such as being able to fluently use at least one programming language.

Much of the published research involving human subjects, in software engineering experiments, has used students. Students are a convenience sample for many researchers, with results based on student subjects being accepted for publication by some journals. Industry is well aware that students' software engineering skills are not representative of professional developers (who have a few years experience); industry is where many graduates find employment after graduation, and the abilities of these new employees is plain for everyone in industry to see.

- students' commercial software skills and knowledge is likely to be very poor, in comparison to professional developers.[1326] This lack of experience and know-how means that student subjects need to spend time on activities that are second nature to professionals, or they simply make noncommercial judgement calls,

- students, typically, have very little experience of writing software, perhaps 50 to 150 hours (and many have no basic coding skills[1152,1234,1867]), while commercial software developers are likely to have between 1,000 to 10,000 hours of experience. This lack of programming fluency means that student programming performance is likely to contains a large learning component, as well as student performance being much lower than professional developers,[1305]

In other research areas students subjects may be more representative of the target population, because they have had many years of experience performing the activities and tasks used in those areas, e.g., processing text written in English and everyday image processing, which are activities used in cognitive psychology experiments.

When experimental results are intended to be applied to the population of university students studying a software related subject, subjects drawn from this population can be representative.

In the US, UK and some other countries, students pay to attend university, and like all businesses universities have to respond to customer demand. When a student decides to study a computing related subject, the University's interests are in ensuring that the student meets its minimum entry requirements and can pay; the likelihood of that person being offered employment in a software related job is not a consideration (in the UK computer science graduates have a much higher unemployment rate, six-months after graduation, compared to those studying other STEM subjects[1673]). Given the high failure rate for computing degrees[1978] and introductory programming courses,[1933] many students on such courses may not even have any software development skill or ability.

Note on terminology: many academic studies use the phrase *expert* to describe subjects who are final-year undergraduates or graduate students, with the term *novice* used to describe first-year undergraduates. In a commercial software development environment a recent graduate is considered to be a *novice* developer, while somebody with five or more years of commercial development experience might know enough to be called an *expert*.

Amazon's Mechanical Turk is becoming popular as a resource for finding subjects and running experiments (in 2015 the population of workers was estimated to be 7,300[1781]). Subjects can stop taking part in a MTurk experiment at any time and care needs to be taken to ensure that the characteristics of subjects who remain does not bias the results.[2020]

Instances of source code can be measured exactly, but different people write different code, i.e., measurements of source code contain implicit variability; see figure 5.23.

13.2.2 The task

Generalizing experimental results to daily work conditions, requires that the characteristics of the tasks performed by subjects share the essential characteristics of the work tasks. That is, the task needs to mimic realistic activities (the technical term is being *ecologically valid*):

- being representative of real world intended usage requires information about how a system will be used in practice, along with the inputs it is likely to experience; lack of resources to perform an analysis of real world usage often means that a convenience sample is used. In a rapidly changing environment, it may not be possible to specify usage patterns in sufficient detail, and perhaps one of the important real world behaviors that needs to be benchmarked is adaptability to change.

  A study by Gregg and Hazelwood[738] provides an example where data usage characteristics are the deciding factor in a cost/benefit trade-off. The time taken to perform a matrix multiply, when the CPU uses a local GPU (the SGEMM implementation in the nVidia CUBLAS package[1397] was used) was measured. Figure 13.3 shows that moving data between the CPU and GPU consumes a significant amount of time, relative to the work done on the data once inside the GPU. Estimating whether GPU usage is worthwhile, depends on the size of matrices encountered in the real world use case, the performance may be slower because of the data transfer overhead, or faster if the matrices are large enough to consume the larger amount of compute resources.

  Another example of the impact of variations in the input data relates to fig 10.9,

- Products that appear to be very similar can have very different performance characteristics (this is one reason why they exist as different products; the other common reason is marketing). Using a product that is identical to the one used in production avoids this problem.

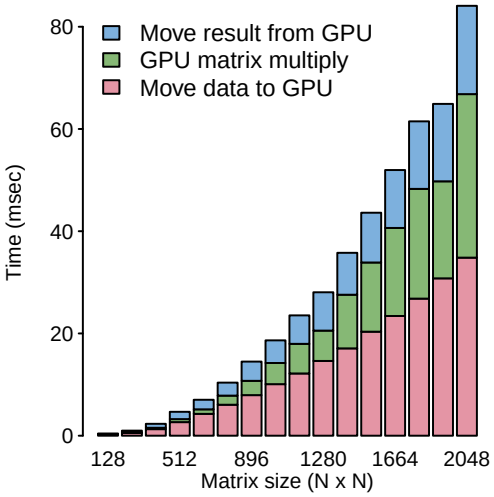


Figure 13.3: Time taken to transfer and multiply 2-dimensional matrices of various sizes on a GTX 480 GPU. Data kindly supplied by Gregg.[738] Github–Local

In a study by Bird,[201] a performance optimization expert took the existing generic code of a library and created tuned versions for each of five different processors (IBM's Blue Gene P and four different members of Intel's x86 product line). The performance of the generic, and all tuned versions of the code, was measured on all processors. Figure 13.4 shows relative performance, with the x-axis listing the processor the code was tuned for, and the y-axis the processor on which it was run; the numbers are relative performance difference, compared to running code on the processor for which it was specifically written.

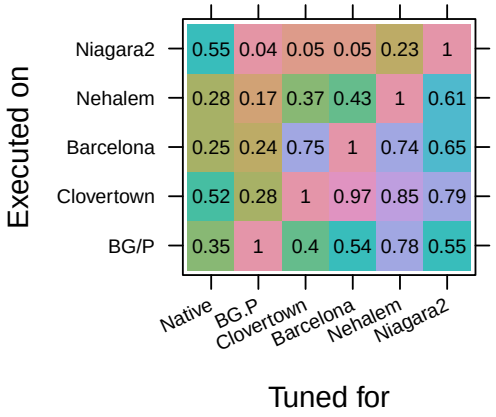


Figure 13.4: Relative performance (y-axis) of libraries optimized to run on various processors (x-axis). Data from Bird.[201] Github–Local

Availability of resources is often a constraining factor, for running an experiment that mimics real world usage. For example, benchmarking backup/restore tools, or desktop search applications, requires realistic file system contents (e.g., the file system must contain a realistic number of files, directory depth, disk fragmentation, etc); getting to a position of being able to generate realistic file systems is a non-trivial task,[16] let alone realistic file content characteristics.[1814]

13.2.3 What is actually being measured?

Subjects may not solve the problems they are presented with, in an experimental, in ways that were intended by the person who designed the experiment.

The history of research into human memory provides an example of how early experimental results were misinterpreted.[928] These early experiments asked subjects to remember sequence of digits, the results suggested that short term memory has a capacity limit of 7 ± 2 items.[1284] After many years and more experiments, the 7 ± 2 digit limit model was replaced by a model based on a limit of 2 seconds of sound[113] (in English this corresponds to around 7 digits, 5.8 in Welsh[542] and around 10 digits in Chinese:[851] the number of digits that can be held in memory when people use these languages).

Software developers are problem solvers and get plenty of practice in finding patterns that can be used to achieve a goal. Unless an experiment is carefully constructed, it is naive to assume that developers will use any of the techniques anticipated by the person designing the experiment.

Your author once ran several experiments,[931] expecting to find a *two seconds of sound* effect in developers short term memory of source code (sequences of simple assignment statements were used). A great deal of effort was invested in creating code sequences whose spoken form required either more or less than two seconds of sound, but the results did not contain any evidence of the expected effect, i.e., a difference in performance caused by the length of sound in the spoken form of source code statements.

At the end of one experiment, a subject mentioned a strategy used to help improve his performance: remembering the first letter of each variable (your author had not noticed that the variables in each list had unique first letters). Use of this strategy reduced the amount of STM the subject needed to use, providing one explanation why the expected effect was not found. The last task, on subsequent experiments, asked subjects to list any strategies they had used during the experiment.

What appear to be small differences, can have a large impact. Human written source code contains very similar constructs where, what might be thought to be small differences in semantics, have usage patterns which are very different. For instance, many languages allow numeric literals to be specified using decimal and hexadecimal notation; figure 13.5 shows that the distribution of literal values written using each notation is different (at least in C source).

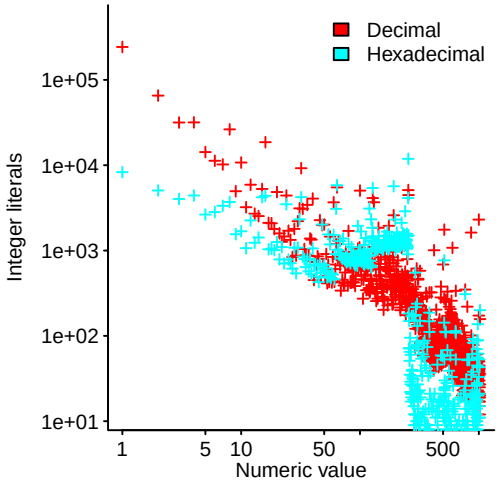


Figure 13.5: Number of integer constants, appearing in the visible form of C source code, having the lexical form of a decimal-constant (the literal 0 is also included in this set) and hexadecimal-constant that have a given value. Data from Jones.[930] Github–Local

A study[753] of how subjects split identifiers, in source code, into components, and expanded them, or not, into words, measured individual performance against what was considered to be the definitive expansion of each identifier. The results of this experiment could also be used to measure the performance of the researchers, in creating a list of identifier expansions that maximised the likelihood of developers correctly decoding the intended information.

Subject motivation is an important factor in obtaining reliable experimental data. Subjects who feel they are being coerced may respond by providing spurious responses, or simply attempt to find short-cuts that minimise the time then need to spend taking part in the experiment, without attracting attention by making too many mistakes.[791] Your author always requests that subjects put as much effort into performing the task, as they would at work: not more, not less.

13.2.4 Adapting an ongoing experiment

The costs of running an experiment makes it tempting to stop, as soon as what is thought to be enough data has been obtained (to produce a sufficiently reliable result). For instance, a researcher may process subjects in batches, running statistical tests after each batch of results, to find out whether the numbers look good/bad enough to stop. One study[184] of A/B testing estimated that 73% of experimenters stopped their experiment once a 90% confidence level was reached.

As each subject is analysed, differences in subject performance cause the aggregated values to fluctuate, and it is possible that some cut-off value (e.g., a p-value cutoff level) is achieved; however, later data may cause it to fluctuate to a less extreme value.

When running an experiment on a live system (or on a stream of subjects), an analysis of the ongoing measurements may suggest that certain changes to the experiment may have a worthwhile impact. Adaptive designs is the term used for experimental designs that support modification of an experiment as results become available.

An adaptive design might be used to reduce the number of different combinations that need to be measured, e.g., benchmarking a computing system supporting many options.[1535]

13.2.5 Selecting experimental options

The behavior of some systems may be configured by selecting from a variety of options; an estimate of the impact of individual options, on system performance, may be required. One way of obtaining this information, is to measure system performance for all possible combinations of option values. This approach might be practical for a few options, each having relatively few values, e.g., Apache supports nine build time yes/no options giving 2^9 possible configurations (out of these 512, only 192 are valid). However, large systems often support so many options, that building and executing every configuration would be impractical, e.g., SQLite supports 3,932,160 valid options.

A study by Lee and Brooks[1103] investigated the impact of configuration option values on the performance and power consumption of a computer architecture by simply selecting three values for each of the 23 design space options; see Github–experiment/lee2006/lee.R.

Randomly selecting option values is an inefficient use of resources, because some option values are over/under used; see Github–experiment/SQL_PWR.R.

An experiment in which all possible permutations of option values are tested, is known as a *full factor design* (options go by the experimental term *factors*). The `fac.design` function, in the `DoE.base` package, takes a specification of factor levels and returns a list of all combinations that need to be run to perform a full factor design; see Github–experiment/design_fac.R.

A study by Citron and Feitelson[366] investigated the performance impact of adding, what they called a Memo-Table (essentially a cache designed to store and reuse the results of previously executed instruction sequences), to the IBM Power4 cpu architecture. The configuration options for the Memo-Table were: Size (1k or 32k), Associativity (1-way or 8-way), Mapping (indexing by program counter or operand+opcode) and Replacement method (random or least recently used).

Four configuration parameters, each having two possible values, gives $4^2 \rightarrow 16$ possible configurations. Citron and Feitelson benchmarked all 16 possibilities, enabling them to check for interactions between all factors. In many experiments the number of interactions between factors is small, and a common cost saving is to only consider interactions between pairs of factors (rather than, say, between three factors).

Factor having just two possible values is a common case, and is known as a two-factor factorial design: it is a *full two-factor design* when all combinations used, and a *fractional two-factor design* when a subset is used.

The `FrF2` function, in the `FrF2` package, generates a list of the combinations of factor values that need to be run, to analyse N factors having a resolution of R (the ability to separate out main effects and interactions between factors; to be able to separate out main effects a resolution of 3 is required, a resolution of 4 enables detection of separate pairs of interactions). In the following list, `1` indicates the option is enabled, `-1` that it is disabled; the output from some functions uses +/−, rather than 1/−1:

```
> library("FrF2")
> FrF2(nfactors=4, resolution=3, alias.info=3)
   A  B  C  D
1  1  1  1  1
2  1 -1  1 -1
3 -1  1 -1  1
4  1 -1 -1  1
5 -1 -1 -1 -1
6  1  1 -1 -1
7 -1  1  1 -1
8 -1 -1  1  1
class=design, type= FrF2
```

The price paid for running a fractional, rather than full, factorial design experiment, is that it is not possible to distinguish interactions between some combinations of factors. For instance, after running the eight combinations listed above, it is not possible to distinguish between an effect caused by a combination of the AB factors, and one caused by the combination CD; this combination is said to be *aliased*. The complete list of aliased factors is:

```
> design.info(FrF2(nfactors=4, resolution=3, alias.info=3))$aliased
$legend
[1] "A=A" "B=B" "C=C" "D=D"

$main
[1] "A=BCD" "B=ACD" "C=ABD" "D=ABC"

$fi2
[1] "AB=CD" "AC=BD" "AD=BC"

$fi3
character(0)
```

To distinguish between an effect caused by any of these combinations, all 16 factor combinations have to be run.

Factorial designs require the number of runs to be a power of two, so the number of different runs grows very quickly as the number of factors increases.

A Plackett and Burman design requires that the number of runs be a multiple of four and at least one greater than the number of factors (they are non-regular fractional factorial 2-level designs). The down-side of these designs is that the results from experiments using them will only support the analysis of the main factors, i.e., any interactions between factors will not be detected; also Plackett and Burman designs can contain complex aliasing between the main factors and (possible) interactions between pairs of factors. The pb function, in the FrF2 package, generates Plackett and Burman designs.

Multiple fractional factorial designs can be combined to isolate effects, i.e., remove aliasing between combinations of factors. Some signs in the original design are switched, creating what is known as a *fold over* of the original; switching the signs of all factors is known as a *full fold over*. The fold.design function generates a foldover design from an existing design.

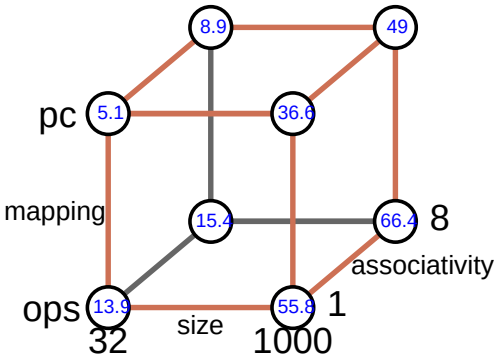


Figure 13.6: A cube plot of three configuration factors and corresponding benchmark results (blue) from Memory table experiment. Data from Citron et al.[366] Github–Local

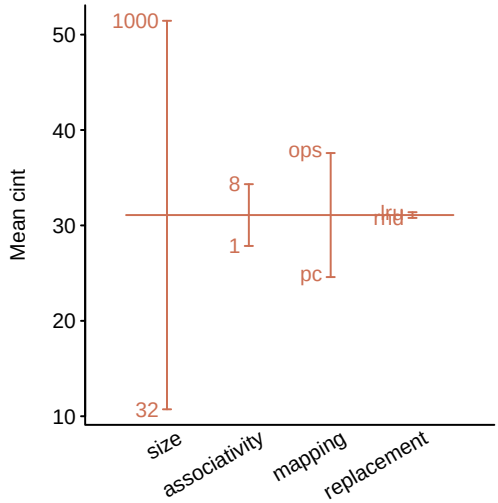


Figure 13.7: Design plot showing the impact of each configuration factor on the performance of Memo table on benchmark performance. Data from Citron et al.[366] Github–Local

13.2.6 Factorial designs

A variety of techniques are available to help visualise the results from an experiment using a factorial design. The analysis is the same for full and partial fractional designs, the only difference is the number of interactions between factors that will be available for analysis.

The study by Citron and Feitelson,[366] discussed earlier, used the SPEC CPU 2000 benchmark, and the values measured were integer floating-point performance, power consumption, processor timing, and die area of the chip.

For simplicity consider three of the factors (ignoring, for the time being, the replacement method), the results can be visualised as a cube with each of the eight vertices representing one combination of factor values; the values at each vertice of figure 13.6 are the SPEC benchmark cint performance figures.

Imagine taking two opposite faces of the above cube, say the two for `size` on the left and right going into the page, and finding the mean `cint` value for both faces, the difference between these two values is known as the *main effect* for `size`; the main effect for the other factors is similarly calculated.

A design plot is a visualisation of these main effects, with a central horizontal line showing the overall mean value of the response variable. Figure 13.7, created using the `plot.design` function, shows the impact that each factor can have on the value of `cint`, offset from the mean value.[ii]

For an example involving more changeable parameters; see Github–experiment/lee2006/lee.R.

At a finer level of granularity, an interaction plot shows the interaction between pairs of factors. Figure 13.8 shows how the mean value of `cint` varies as `size` is changed, for a given value of `mapping` (see legend).

For an equation based analysis, a fitted regression model can be used to investigate the interactions between factors. For instance, the following model specifies interactions between all variable pairs; see Github–experiment/MemoPower03.R for details:

```
Memo_glm=glm(cint ~ (size+associativity+mapping)^2, data=Memo)
```

A study by Pallister, Hollis and Bennett[1439] investigated the power consumed by various embedded programs when compiled with gcc using various command lines parameters. The design used contained partial aliases, which many plotting functions cannot handle; the `halfnormal` function, from the `DoE.base` package, has an option to orthogonalize the design; see Github–experiment/pallister/gcc-power.R.

Plackett and Burman designs do not contain enough information to fit a regression model, a bespoke method has to be used, e.g., the `DanielPlot` function, in the `FrF2` package (in the plot produced, the x-axis shows effect size, the y-axis contains diagnostic information). If the data is the result of random variation (i.e., changing factor values has no effect), differences between pairs of factor averages have a (roughly) normal distribution; plotting values from a normal distribution using a normal probability scale produces a straight line. If many of the points displayed by `DanielPlot` appear to form a straight line, then the corresponding factors are likely to have had little effect on the results; any factors well off the line are of interest.

A study by Debnath, Mokbel and Lilja[464] investigated the impact of seven system configuration settings on PostgreSQL performance, on the TPC-H benchmark. High and low values were chosen for the configuration values and a Plackett and Burman design with full fold-over was used. Figure 13.9 shows the half-normal plot from the 16 runs; factors P4 and P7 do not fall on a straight line that passes close by the other factors, and they also exhibit the largest effect.

13.3 Benchmarking

Benchmarking is the process of running an experiment to obtain information about the performance of some aspect of hardware and/or software. Common reasons for benchmarking, in software engineering, include comparing before/after performance and obtaining numbers to put in a report. For a more general audience, benchmarking information is often used as input to a selection process and as such often has a marketing orientation. Accurate measurements are not always necessary, showing that a system is good enough, may be good enough.

The time taken to add and multiply values was used to compare the performance of early computers.[206, 1252, 1941, 1942] Studies by Knight[1026, 1027] calculated performance based on a weighted average of instruction times, based on the kinds of instructions executed by commercial and scientific programs. Based on a study of over 300 computers available between 1944 and 1967, the rental cost for performing an operation decreased with increasing computer performance; see figure 13.10, the lines are fitted power laws with exponents between two and three; see Github–benchmark/EvolvingCompPerf_1963-1967.R.

Obtaining accurate benchmark data for many questions relating to computing platforms it may to be economically infeasible.[iii] A consequence of the continual reduction in the size

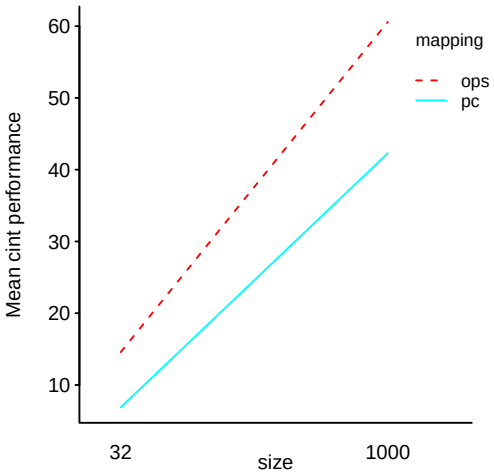


Figure 13.8: Interaction plot showing how `cint` changes with `size`, for given values of `mapping`. Data from Citron et al.[366] Github–Local

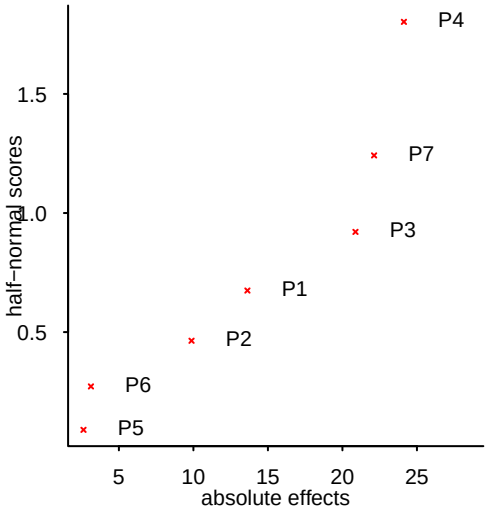


Figure 13.9: Half-normal plot of data from a Plackett and Burman design experiment. Data from Debnath et al.[464] Github–Local

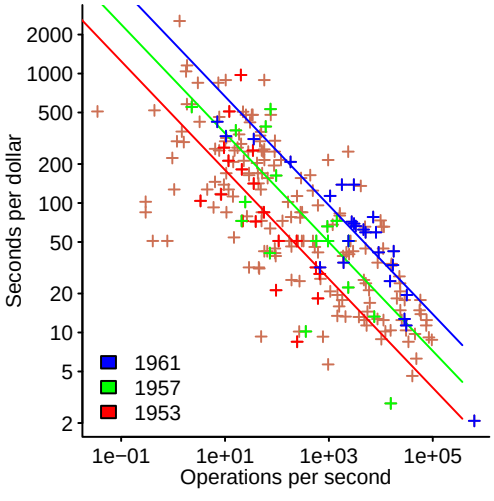


Figure 13.10: Performance and rental cost of early computers, with straight line fits for a few years. Data from Knight.[1026] Github–Local

[ii] `plot.design` makes some unexpected display decisions when the explanatory variables are not factors.

[iii] Obtaining accurate benchmark results has always been an expensive and time-consuming process, but at least it was once possible to rely on devices sharing the same part number to have the same performance characteristics.

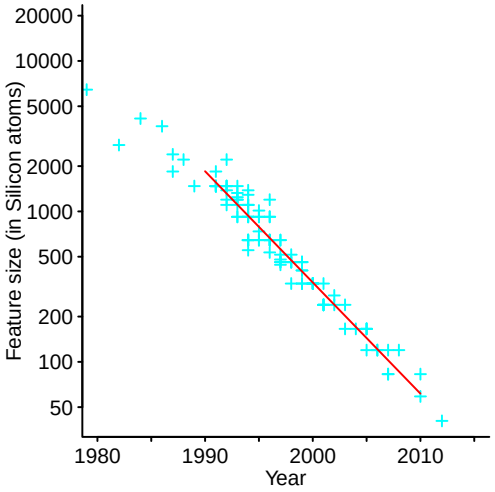


Figure 13.11: Feature size, in Silicon atoms, of microprocessors; line is a fitted regression of the form: $Silicon\_atoms \propto e^{-0.17Year}$. Data from Danowitz et al.[436]

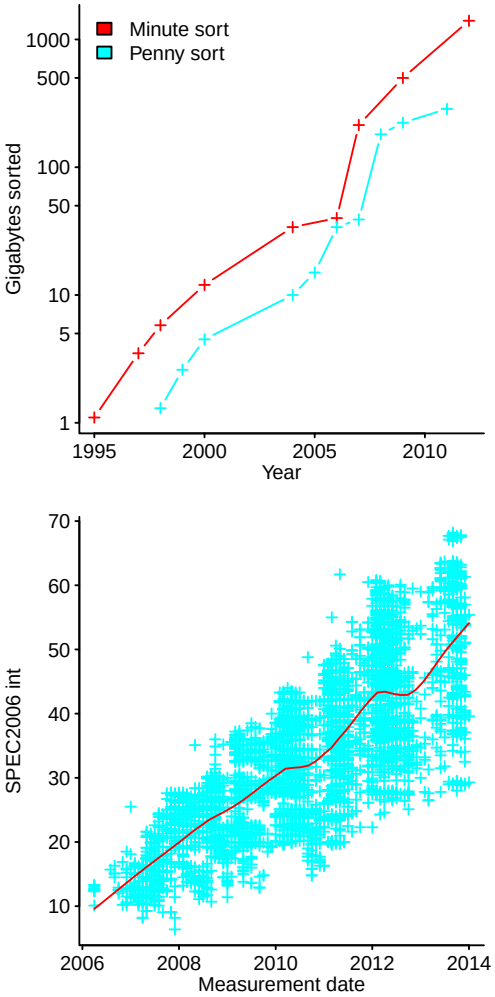


Figure 13.12: Maximum number of records sorted in 1 minute and using 1 penny's worth of system time (upper), and SPEC2006 integer benchmark results (lower, with loess fit). Data from Gray et al[731] and SPEC.[1742] Github–Local

of components within microprocessors (see figure 13.11 and fig 11.51), is that individual components are now so small that variations in the fabrication process (e.g., differences in the number of atoms added or removed during the fabrication process) can noticeably change their geometry, leading to large variations in the runtime electrical characteristics of supposedly identical devices.[187]

Manufacturers offer computing systems having a range of performance characteristics; figure 13.12, lower plot, shows all published results for the integer SPEC2006 benchmark. While hardware performance has now improved to the point where for many uses it appears to be good enough, it is still possible to buy hardware that is under-powered for the job it is expected to perform; figure 13.12, upper plot, shows the orders of magnitude improvements in cost and performance of sorting over 15 years.

Benchmark results published for general consumption are sometimes little more than marketing claims. The author(s) may have reasons for wanting to create a favourable impression for one system, in preference to others, or may just have done a sloppy job (perhaps because of inexperience, incompetence or lack of resources to do a decent job). A class action suite alleged that:[673] "Intel used its enormous resources and influence in the computing industry to, in Intel's own words, "falsely improve" the Pentium 4's performance scores. It secretly wrote benchmark tests that would give the Pentium 4 higher scores, then released and marketed these "new" benchmarks to performance reviewers as "independent third-party" benchmarks. It paid software companies to make covert programming changes to inflate the Pentium 4's performance scores and even disabled features on the Pentium III so that the Pentium 4's scores would look better by comparison."[iv] Using an established benchmark does not guarantee the results are free of vendor influence. One class action suite alleged[1612] "Samsung intentionally rigged the GS4 to operate at a higher speed when it detected certain benchmarking apps. In versions of the GS4 using the Qualcomm Snapdragon 600 processor, Samsung wrote code into the firmware (embedded software) of the GS4 to automatically and immediately drive Central Processing Unit ("CPU") voltage/frequency to their highest state, and to immediately engage all four of the processing cores of the CPU."[v]

In some consumer goods markets, product benchmark results receive a lot of publicity, with potential customers thought to be influenced by the results achieved by similar products. A study by Shimpi and Klug,[1698] of Android benchmarks, found that some mobile phone vendors detected when a particular benchmark was being run and raised the devices thermal limits (allowing the system clock rate to run faster for longer; a 4.4% performance improvement was measured).

Researchers are happy to complain about poor benchmarking practices, but are not always willing to name names.[1874]

In published benchmark results the Devil is in the detail, or more often in the lack of detail, as illustrated by the following:

- Bailey[116] lists twelve ways in which parallel supercomputer benchmarks have been written in a way likely to mislead readers, including: quoting 32-bit, not 64-bit results, quoting figures for the inner kernel of the computation, as if they applied to the complete application, and comparing sequential code against parallelized code.
- Citron[365] analysed the ways in which many research papers using the SPEC CPU2000 suite have produced misleading results, by only using a subset of the benchmark programs (of 115 papers surveyed, 23 used the whole suite). In one case a reported speed up of 1.42 is reduced to 1.16 when the whole suite is included in the analysis (reduction from 1.43 to 1.13 in another and from 1.76 to 1.15 in a third).

The primary purpose of this section is to highlight the many sources of variability present in modern computing systems. The available evidence suggests that large variations in benchmark results are now the norm. Large variations in measured performance do not prevent accurate results being obtained, the impact is to increase the time and money needed, i.e., it is simply a case of making enough measurements. Advice on how to perform benchmarks is available elsewhere.[585,758]

People interested in consistent performance will want to minimise the variation in benchmark results (which did occur for some programs), while those interested in actual benchmark performance will be interested that significant changes in the mean occurred for some programs.

[iv] The class action was settled[1785] with Intel agreeing to pay $15 to Pentium 4 purchasers, $4 million to a non-profit entity and an amount not to exceed $16.45 million to the lawyers who brought the suit.

[v] The class action was settled[1612] with Samsung agreeing to pay $2.55 million, with each member of the class action estimated to receive $38.38.

When comparing different systems, benchmark performance may be normalised to produce a relative performance ranking. The geometric mean needs to be used when comparing normalized values, otherwise the results can be inconsistent.

	R	M	Z	R/M	M/R	R/Z	Z/R	M/Z	Z/M
E	417.00	244.00	134.00	1.71	0.59	3.11	0.32	1.82	0.55
F	83.00	70.00	70.00	1.19	0.84	1.19	0.84	1.00	1.00
H	66.00	153.00	135.00	0.43	2.32	0.49	2.05	1.13	0.88
I	39449.00	33527.00	66000.00	1.18	0.85	0.60	1.67	0.51	1.97
K	772.00	368.00	369.00	2.10	0.48	2.09	0.48	1.00	1.00
Arithmetic	8157.40	6872.40	13341.60	1.32	1.01	1.50	1.07	1.09	1.08
Geometric	586.79	503.13	498.68	1.17	0.86	1.18	0.85	1.01	0.99

Table 13.1: Benchmark results for three processors (R, M, Z), running five benchmarks (E thru K), with normalisation using different processors, along with arithmetic and geometric means. Data from Fleming et al.[615] Github–Local

Table 13.1 shows the results of five benchmark programs (E to K) from three systems (i.e., columns `R`, `M` and `Z`); two other sets of columns list normalised values, with different processors used as the reference. The bottom two rows list the arithmetic and geometric mean of the columns; note: for the arithmetic mean, both ratios `R/M` and `M/R` are greater than one, while for the geometric mean one ratio is less than one (the same pattern is occurs for the other ratios). A ranking based on the arithmetic mean depends on the processor used as the base for normalization, while the geometric mean produces a consistent ranking; see section 10.3.3.

13.3.1 Following the herd

When choosing a benchmark, there is a lot to be said for doing what everybody else does, advantages include:

- can significantly reduce the cost and time needed to obtain benchmark data,
- an established benchmark is likely to be usable out-of-the-box. It takes time for a benchmark to become established; an analysis[1462] of one Java source code corpora was able to build 86 of the 106 Java systems in the corpus, with 56 of these having to be patched to get them to build,
- it is easier to sell the results to audiences, when the benchmark used is known to them.

The disadvantage of following the herd is that there may be fitness-for-purpose issues associated with using the benchmark, i.e., herd behavior is adapted to environments that may be substantially different from the environment in which the system is intended operate. For instance, the SPEC benchmark is often used to compare compiler performance, but SPEC's intent is for it to be used for benchmarking processor performance.

Commonly used benchmarks suffer from vendors tuning their products to perform well on the known characteristics of the benchmark. The SPEC benchmark has been used over many years for compiler benchmarking and compiler vendors often use it in-house for performance regression testing.

13.3.2 Variability in today's computing systems

In the good old days, computer performance tended to be relatively consistent across identical, but physically different, components, i.e., the same model of cpu or memory chip. Also, software tended to have relatively few options that could significantly alter its performance characteristics.

Modern hardware may contain components that are fabricated using handfuls of atoms, with process variations, of an atom or two here and there, producing surprisingly different performance characteristics,[1296] but externally looking like identical devices. Further reductions, in the number of atoms used to fabricate devices, will lead to greater variations in the final product. Today's consumer of benchmark results has to chose between:

- accepting a wide margin of error,
- executing a benchmark very many times, to ensure the sample size is large enough to achieve the desired statistical confidence interval for the results.

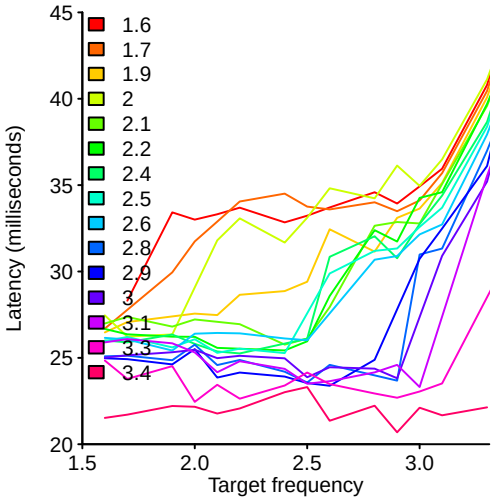


Figure 13.13: Mean time for an Intel IvyBridge to transition from a given frequency (colored lines) to another frequency (x-axis). Data kindly provided by Mazouz.[1226] Github–Local

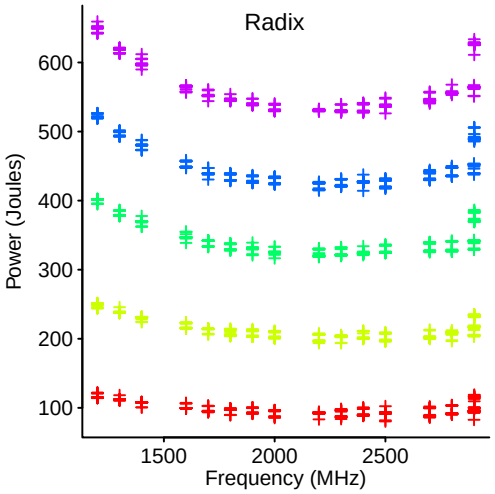


Figure 13.14: Total system power consumed when sorting 10, 20, 30, 40, 50 million integers (colored pluses), using Radix sort on the same processor running at different clock frequencies. Data from Götz et al.[719] Github–Local

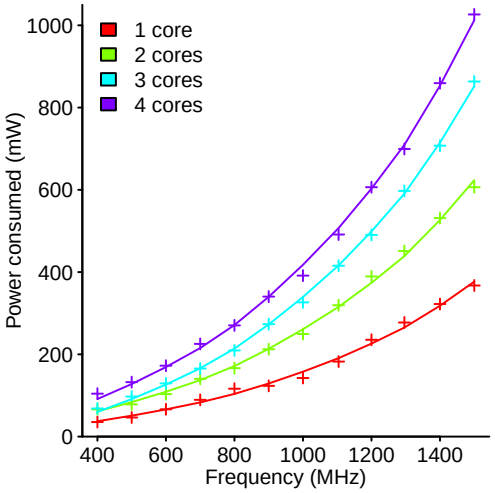


Figure 13.15: Power consumed by an Exynos-7420 A53 processor at various frequencies, and one to four cores under load, with fitted regression lines. Data kindly provided by Frumusanu.[633] Github–Local

Both approaches require checking that a wide range of, possible unknown, factors are controlled for, by those running the benchmark.

Intrinsic variability in system performance impacts development teams that regularly monitor the performance of their products during ongoing development. For instance, Mozilla regularly measures the performance of the latest checked-in version of Firefox source code, if an update results in a performance decrease exceeding a predefined limit, the update is rolled back. Successful implementation of such a policy requires careful control of external factors that could impact performance.

Performance variation has to be addressed from a system wide perspective,[vi] hardware/-software interaction can have a significant performance impact and there are often multiple, independent, sources of variation. At the systems level differences in component characteristics (e.g., differences in system clock frequency drift in multiprocessor systems[875]) can interact to produce emergent effects.

DVFS (Dynamic Voltage and Frequency Scaling) provides an example of how the complexities of system component interactions make it difficult to reliably predict performance. As its name suggests, DVFS allows processor voltage and frequency to be changed during program execution; an analysis of system power consumption[1998] concludes that total power consumed, executing a program from start to finish, is minimised by running the processor as fast as possible (assuming there is no waiting for user input).

A change in voltage or frequency is not instantaneous, it can take many clock cycles. A study by Mazouz, Laurent, Pradelle and Jalby[1226] investigated the latency of CPU frequency transition. Figure 13.13 shows the mean time, in milliseconds, taken by an Intel IvyBridge to transition from one clock frequency (colored lines) to another (x-axis); original paper shows median time.

A study by Götz, Ilsche, Cardoso, Spillner, Aßmann, Nagel and Schill[719] investigated how the total system power consumed by implementations of various algorithms varied with cpu clock frequency, with the intent of finding the frequency that minimised power consumption.

Figure 13.14 shows that total power consumption does not always decline with frequency, there is a frequency below the maximum that minimises power consumed.[459] The power minimisation frequency depends on the implementation of the sorting algorithm, with the difference between minimum and maximum depending on the number of items being sorted. Predicting the power consumed[199] by a program is a non-trivial problem.

Programming languages are starting to support constructs that provide developers with options for dealing with power consumption issues.[1633]

13.3.2.1 Hardware variation

This section outlines some evidence for large variations in hardware component performance. Much of the data used in the analysis was obtained using programs executing on components manufactured five to ten years before this book was published; variability has likely increased in the subsequent years. The hardware components covered include:

- CPU: performance, power consumption and instruction counts,
- main storage: hard disc performance and power consumption,
- memory: performance and power consumption,

When computing devices are connected to the mains power supply, there is rarely any need to be concerned about the characteristics of the supply. Batteries have characteristics that can affect the performance of devices connected to them, such as the level of power delivery being dependent on the current charge state and power draw frequency characteristics. In a mobile computing environment, power consumption can be just as important as runtime performance, if not more so; there are limits to the amount of electrochemical energy that can be stored.[1218] Peltonen et al[1465] is a public dataset of power consumption on 149,788 mobile devices, containing 2,535 different Android models; Jongerden[944] analyses various models of battery powered systems and Buchmann[271] covers rechargeable batteries in detail.

In mobile devices, a large percentage of power is consumed by the display; optimization of display intensity and choice of color,[1759] while an app is running, is not discussed here.

[vi] The following two sections separately discuss performance variation whose root cause in hardware or software; this is for simplicity of presentation.

CMOS (complementary metal-oxide-semiconductor) is the dominant technology used in the fabrication of the chips contained in computing devices; until a so-called beyond-CMOS device[1383] technology becomes commercially viable, this section only considers the characteristics of CMOS devices.

CPU: The processors executing code are often considered to be interchangeable with any other, mass-produced, etched slices of silicon stamped with the same model number; while never exactly true, deviations from this interchangeability assumption were once small enough to only be of interest within specialised niches, e.g., hardware modders interested in running systems beyond rated limits.

The micro-architecture of modern processors has become so complicated that apparently minor changes to an instruction sequence can have a major impact on performance;[874] a trivial change to the source code, or the use of a different compiler flag may be enough.

Power consumption and clock frequency are directly connected; increasing clock frequency increases power consumption (a good approximation for processor power consumption is $P = \alpha F V^2 + I_0 V$, where: α is a device dependent constant, F is clock frequency, V is voltage supplied to the cpu,[vii] and I_0 is leakage current). Processors clocked at the same frequency execute instructions at the same rate. However, variations in the number of atoms implementing internal circuitry produces variations in power consumption. Some processors reach their maximum operating temperature more quickly than others; to prevent device destruction through overheating, power consumption is reduced by reducing the clock rate. Different processors have different sustained performance rates because of differences in their power consumption characteristics. Vogeleer[458] discusses the modeling of low level temperature/power relationships for the kind of processors used to run applications.

A study by Frumusanu[633] measured the power and voltage, at various frequencies, of an Exynos-7420 A53 processor idling and at load. Figure 13.15 shows measured power consumption, involving one to four cores under load, at various frequencies and a fitted regression model.

Any benchmark made using a single instance of a processor is a sample drawn from a population that could vary by something like 5-10% or more when executing code and several hundred percent when idling. The extent to which results based on this minimum sample size is of practical use will depend on the questions being asked. If the power consumption characteristics of the population of a particular CPU is required, then it is necessary to benchmark a sample containing an appropriate number of *identical* processors. Methodologies for benchmarking power consumption[1743] require detailed attention to many issues.

A study by Wanner, Apte, Balani, Gupta and Srivastava[1925] measured the power consumed by 10 separate Amtel SAM3U microcontrollers at various ambient temperatures. Figure 13.16 shows a 5-to-1 difference, between supposedly identical processors, in power consumption when in sleep-mode (upper plot), and around 5% difference when operating at 4MHz.

Section 11.6 discusses the building of mixed-effects models for power variations of the Intel Core processor.

A study by Marathe, Zhang, Blanks, Kumbhare, Abdulla and Rountree[1202] investigated the variation in performance of 2,386 Intel Sandy Bridge XEON processors while operating under a running average power limit (RAPL). Figure 13.17 shows the time taken by 2,386 processors to complete the Embarrassingly parallel benchmark and their clock frequency, with a RAPL of 65 Watts.

How accurate are power consumption measurements? These measurements are often implemented by periodically sampling the voltage across a known resistance. A study by Saborido, Arnaoudova, Beltrame, Khomh and Antoniol[1623] investigated the measurement error introduced by different sampling rates, on mobile devices. Figure 13.18 shows the power spectrum of the Botanica App, executing on a BeagleBone Black running Android 4.2.2, sampled at 500K per second. By using a very high sampling rate, it is possible to see the noticeable peak in power consumed by very short-lived events, something that low frequency sampling would not detect (the paper lists error estimates for lower sampling rates).

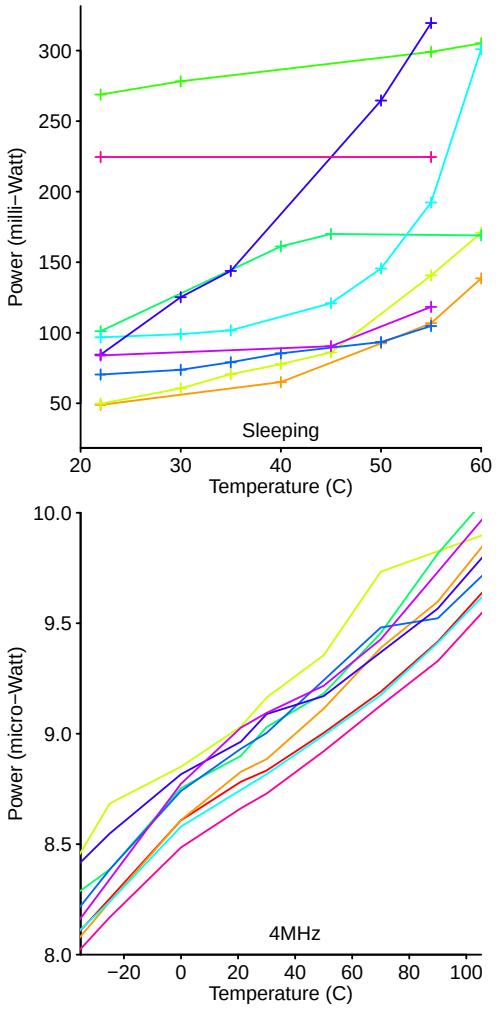


Figure 13.16: Power consumed by 10 Amtel SAM3U microcontrollers at various temperatures when sleeping or running. Data from Wanner et al.[1925] Github–Local

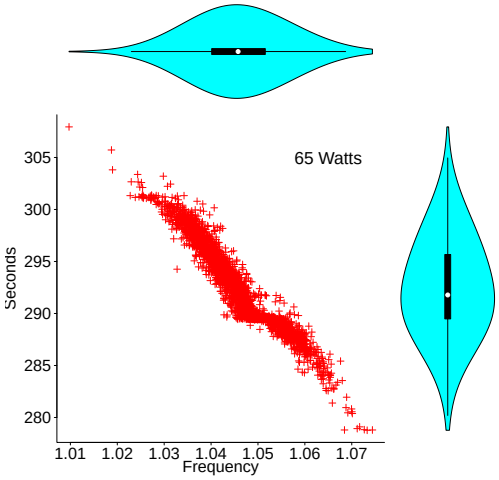


Figure 13.17: Time taken to execute the EP benchmark and clock frequency of 2,386 Intel processors, with a RAPL of 65 Watts. Data kindly provided by Rountree.[1202] Github–Local

[vii] On some processors there is a linear relationship between voltage and frequency,[458] i.e., $P = \beta V^3 + I_0 V$, or $P = \gamma F^3 + I_0 V$.

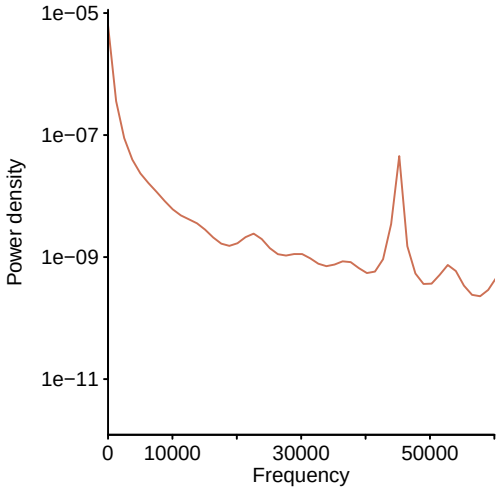


Figure 13.18: Power spectrum of the electrical power consumed by the Botanica App executing on a BeagleBone Black running Android 4.2.2. Data from Saborido et al.[1623] Github–Local

In theory, counting the instruction executed by a program is a means of obtaining, power independent, answers to questions about comparative program performance. Some processors include hardware support for counting the number of operations performed, e.g., instruction opcodes executed and cache misses; however, the purpose of these counters is to help manufacturers debug their processors, not provide end-user functionality. Consequently, counter values are not guaranteed to be consistent across variants of processors within the same family[1934] and fixing faults in the counting hardware does not have a high priority (e.g., counting some instructions twice or not at all; Weaver and Dongarra[1934] found that in most cases the differences were a fraction of a percent of the total count, but for some kinds of instructions, such as floating-point, the counts were substantially different). A study by Weaver and McKee[1935] found that it was possible to adjust for the known faults in the hardware counters; see Github–benchmark/iiswc2008-i686.R.

How are calls into operating system routines, which may execute at higher privilege levels, counted? Also, the execution time of some instructions may depend on other instructions executed at around the same time (modern processors have multiple functional units and allocate resources based on the instructions currently in the pipeline), the time taken to execute other instructions can be very unpredictable, e.g., time taken to load a value from memory depends on the current contents of the cache and other outstanding load requests. A study by Melhus and Jensen[1259] showed that address aliasing of objects in memory could have a huge impact on the relative values of some hardware performance counters.

Hardware counters need not be immune to *observer effects*; in particular, the values returned can depend on the number of different hardware counters being collected.[1342]

Main storage: Traditionally main storage has meant hard disks (sometimes backed-up with tape), but solid state devices (SSD) are rapidly growing in capacity[1800] and use; for extremely large capacity magnetic tape is used: this niche use is not discussed here.

Data is read/written to a hard disk by moving magnetic sensors across a rotating surface. These spatial movements create a correlation between successive operations, e.g., the time taken to perform the second read will depend on its location on the disk relative to the first. Disks spin at a constant rate, and in the same time interval more data can be read from the area swept out near the outer edge of a platter than from one near the spindle (the staircase effect in the upper plot in figure 13.19 is a result of zoning). This location dependent performance characteristic makes disk benchmark performance dependent on the history of the data that has been added/deleted.

Further correlations are created by data buffering, by the operating system and the device itself, and access requests being reordered to optimise overall throughput.

Storage farms organise files so that those most likely to be accessed are stored on the outer tracks, while files less likely to be accessed are stored on the inner tracks. A growing percentage of disks are used in data centers and at some point manufacturers may decide to concentrate on designing drives for this market.[254]

The continuing increase in the number of bits that can be stored within the same area of rotating rust has been achieved by reducing the size of the magnetic domain used to store a bit. Like silicon wafer production, variations in the fabrication process of disc platters can now result in large differences in the performance of supposedly identical drives.

A study by Krevat, Tucek and Ganger[1044] measured the performance of disk drives originally sold in 2002, 2006, 2008 and 2009. Figure 13.19, upper plot, shows the read bandwidth of nine disks from 2002, each displayed using a different color and there is little variation between different disks (fitting a regression model finds that disk identity is not a significant predictor of performance, p-values around 0.2). The lower plot shows the read bandwidth of nine disks from 2006, each displayed using a different color; the visibility of different colors shows the variation between different disks (fitting a regression models finds that disk identity is a significant component of performance prediction, p-values around 10^{-16}).

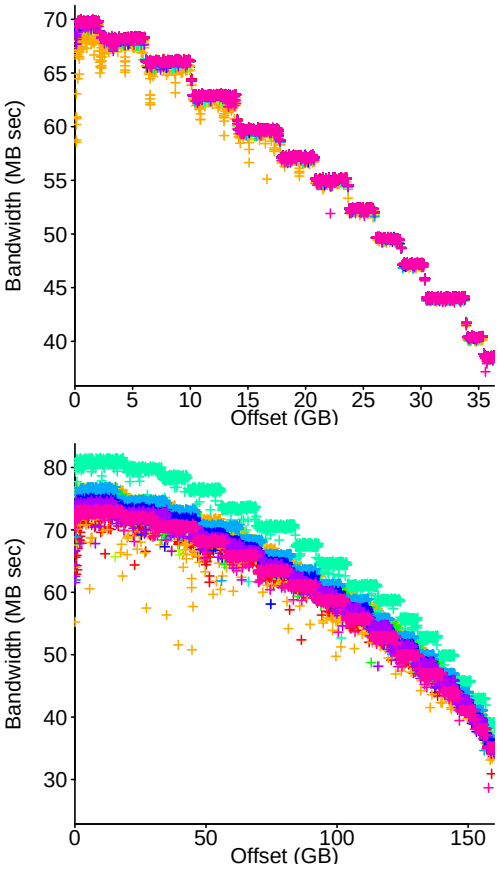


Figure 13.19: Read bandwidth at various offsets for new disks sold in 2002 (upper) and 2006 (lower). Data kindly provided by Krevat.[1044] Github–Local

To increase recording density, drive manufacturers are now using Shingled Magnetic Recording (SMR), where tracks overlap like rows of shingles on a roof. Singled discs have very different performance characteristics,[14] but little data is publicly available at the time of writing.

SSDs are sufficiently new that little performance data is publicly available at the time of writing.

A study by Kim[998] ran eight different benchmarks on SSD cards from nine different vendors. The range of performance values was different for both vendors and benchmark.

Building a regression model, using normalised benchmark scores, finds that one vendor's products have a sufficiently consistent performance that they can be included in a model (these products appear to have the best performance, the other vendors appear to have the same performance); see Github–benchmark/hyojun/hyojun.R.

Memory: Memory chips tend to be thought about in terms of their capacity and not their performance (such as, read/write delays or power consumption). Performance is governed by access rate and by the number of bytes transferred per access, with accesses usually made via some form of memory control chip (the capabilities of this controller have a significant impact on performance). Many motherboards provide options to select memory chip timing characteristics.

A study by Bircher[199] investigated the power consumed by the various hardware of a server, while it executed the SPEC CPU2006 benchmark. Figure 13.20, upper plot, breaks down average power by CPU (red) and memory (blue), while the lower plot breaks the power down by the major subcomponents of the server.

There is often a performance hierarchy for memory, with on (cpu) chip cache providing faster access to frequently used data. The interaction between the size of the various memory caches, and an algorithm's use of storage, can result in performance characteristics that change as the size of the objects processed changes.

A study by Khuong and Morin[995] measured the performance of several search algorithms, when operating on arrays of various sizes (items were stored appropriately in the array, for the algorithm used). Figure 13.21 shows the time taken by different algorithms to find an item in an array, for arrays of various sizes; the grey lines show the total size of the processor L1, L2 and L3 caches.

The following are some examples of memory chip characteristics that have been found to noticeably fluctuate:

- a study by Gottscho, Kagalwalla and Gupta[718] measured power consumption variability of 13 DIMMs, of the same model of 1G DRAM from four vendors. The variation about the mean, at one standard deviation, was 5% for read operations, 9% for write and 7% for idling; see Github–benchmark/J20_paper.R,
- a study by Gottscho[717] measured the power consumption of 22 DDR3 DRAMs, manufactured in 2010 and 2011, from four vendors. Read operations consumed around 60% of the power needed for write operations, with idle consuming around 40%; the standard deviation varied from 10% to 20%. The power consumed also varied with value being read/written, e.g., writing 1 to storage containing a 0 required 25% more power than writing a 0 over a 1; see Github–benchmark/MSTR10-DIMM.R for data,
- a study by Schöne, Hackenberg and Molka[1650] found that memory bandwidth was reduced by up to 60%, as the frequency of the cpu was reduced, that memory performance characteristics varied between consecutive generations of Intel processors and between server and desktop parts.

The variability of memory chip performance is likely to increase, as vendors further reduce power consumption and improve performance by lengthening DRAM refresh times; optimising each computer by tuning it to the unique characteristics of the particular chips present in each system.[1104]

Chandrasekar[322] provides a detailed discussion of DRAM power issues, including code for a tool to obtain detailed information about the memory chips installed on a system.

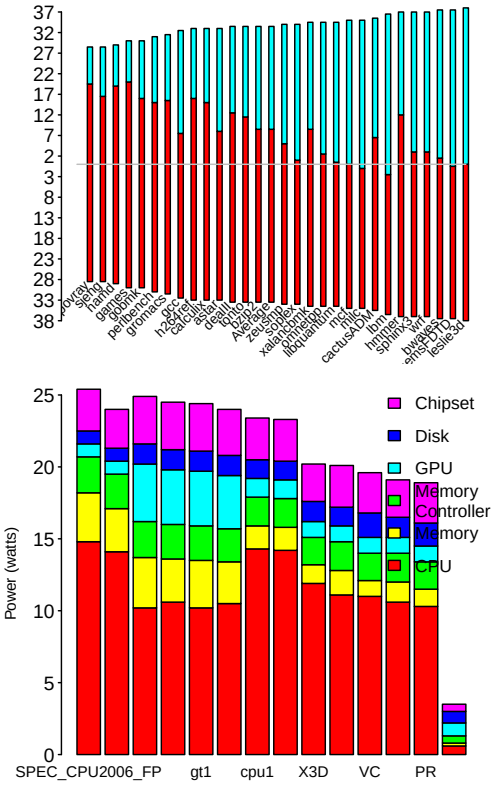


Figure 13.20: Average power consumed by one server's CPU (four Pentium 4 Xeons; red) and memory (8 GB PC133 DIMMs; blue) running the SPEC CPU2006 benchmark (upper) and breakdown by system component when executing various programs. Data from Bircher.[199] Github–Local

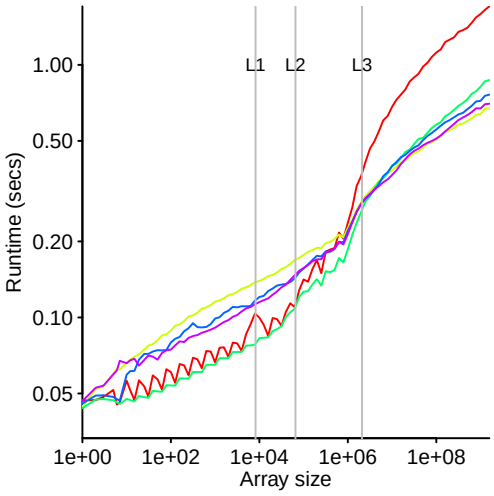


Figure 13.21: Time taken to find a unique item in arrays of various size, containing distinct items, using various search algorithms; grey lines are L1, L2 and L3 processor cache sizes. Data from Khuong et al.[995] Github–Local

13.3.2.2 Software variation

This section outlines some of the evidence for large variations in software performance, briefly covering the following software components and processes:

- The environment: interaction with the environment, file system, support libraries and aging,
- Configurations,
- Creating an executable: compiler optimization and link order,
- Tools.

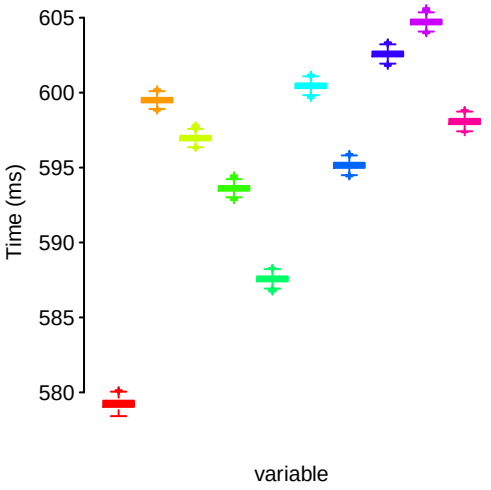


Figure 13.22: FFT benchmark executed 2,048 times followed by system reboot, repeated 10 times. Data kindly provided by from.[963] Github–Local

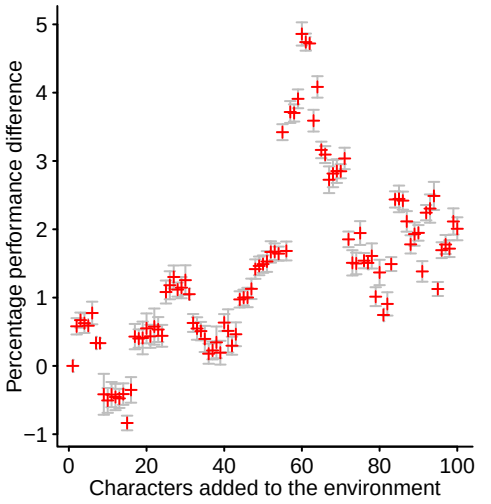


Figure 13.23: Percentage change, relative to no environment variables, in perlbench performance as characters are added to the environment. Data extracted from Mytkowicz et al.[1341] Github–Local

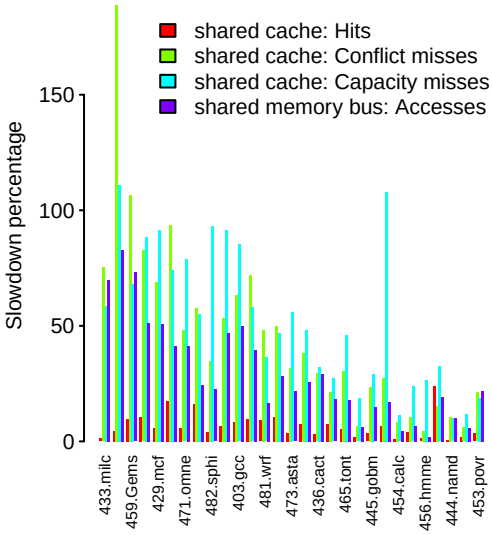


Figure 13.24: Changes in SPEC CPU2006 performance caused by cache and memory bus contention, for one dual processor Intel Xeon E5345 system. Data kindly provided by Babka.[101] Github–Local

The environment: Programs execute within an environment that often contains a complicated ensemble of interconnecting processes and services that cannot be treated as independent standalone components. One consequence of this complexity,[925] and interconnectedness, is that the order in which processes are initiated during system startup can have a noticeable impact on system performance.

The impact of a system's prior history, on program performance, is seen in a study by Kalibera, Bulej and Tůma,[963] who measured the execution time of multiple runs of various programs. Figure 13.22 shows 10 iterations of the procedure: reboot computer and make 2,048 performance measurements. The results show performance variation after each reboot is around 0.1%, but rebooting can cause a shift of 3% in the average performance (the ordering of processes executed during system startup varies across reboots, due to small changes in the time taken to execute the many small scripts that are invoked during startup; execution ordering affects placement of data in memory, which can have an impact on performance). A later study[836] found that the non-determinism of initial program execution, in this case, could be reduced by having the operating system use cache-aware page allocation.

Environmental interactions are not always obvious. A study by Mytkowicz, Diwan, Hauswirth and Sweeney[1341] increased the number of bytes occupied by a Linux environment variable between runs of the `Perlbench` program. The results from each of 15 executions were recorded, an environment variable increased in size by one character, and the procedure repeated 100 times. Figure 13.23 shows the percentage change in performance, relative to the environment variable containing zero characters, at each size of environment variable, along with 95% confidence intervals of the mean of each 15 runs.

Incremental operating system updates can produce a change in program performance. A study by Flater[613] compared the performance of cpu intensive and I/O bound programs on two different versions of Slackware, running on the same hardware (versions 14.0 and 14.1, using Linux kernels 3.12.6 and 3.14.3 respectively). The results show consistent differences in performances of up to 1.5% (rebooting did not have any significant impact on performance).

Many systems allow multiple programs to share system resources, by executing at the same time. Sharing becomes a performance bottleneck when one program cannot immediately access resources when it requests them; access to memory is a common resource contention issue on multi-processing systems. A study by Babka[101] investigated the performance of multicore processors having a shared cache. Figure 13.24 shows changes in SPEC CPU2006 performance caused by cache and memory bus resource contention, on a dual processor Intel Xeon E5345 system.

A study by Mazouz[1225] investigated the performance of the SPEC OpenMP 2001 programs, compiled using gcc 4.3.2 and icc 11.0, running on multicore devices. It is possible for a program's code to execute on a different core after every context switch. Allowing the operating system to select the core to continue program execution is good for system level load balancing, but can reduce the performance of individual programs because recently accessed data is less likely to be present in the cache of any newly selected core. *Thread affinity* is the process of assigning each thread to a subset of cores, with the intent of improving data locality, i.e., recently accessed data is more likely to available in accessible caches.

Figure 13.25 shows the time taken to execute one program in 2, 4, and 6 threads, with thread affinity set to compact (threads share an L2 cache), no affinity (allow the OS to assign threads to cores) and scatter (distribute the threads evenly over all cores), each repeated 35 times.

Configuring the system being benchmarked to only run one program at a time solves some, but not all, cache contention issues. Walking through memory, in a loop, may result in a small subset of the available cache storage being used (main memory is mapped to a much smaller cache memory, which means that many main memory addresses are mapped to the same cache address). Figure 13.26, from a study by Babka and Tůma,[102] shows the effect of walking through memory using three different fixed width strides; for 32 and 64 byte strides accesses to even cache lines is faster than odd lines, with the pattern reversed for a 128 byte stride.

Operating systems generally have background processes that spend most of their time idling, but wake up every now and again. When a background processes wakes up, it will consume system resources and can have an impact on the performance reported by a benchmark, i.e., background processes are a source of variation.

A study by Larres[1088] investigated how the performance of one version of Firefox changed as various operating system features were disabled (the intent being to reduce the likelihood that external factors added noise to the result). The operating system features modified were: 1) every process that was not necessary was terminated, 2) address-space randomization was disabled, 3) the Firefox process was bound exclusively to one cpu, and 4) the Firefox binary was copied to and executed from a RAMDISK.

Every program in the Talos benchmark (the performance testing framework used by Mozilla) was run 30 times. Figure 13.27 shows the performance of various programs running in original and stabilised (i.e., low-noise) configurations.

File systems: These provide the housekeeping structure for keeping track of information on a storage device. The traditional view of a file, as a leaf in a directory tree, has become blurred, with many file system managers now treating compressed archived files (e.g., zip files) as-if they had a directory structure that can be traversed; Microsoft's .doc format contains a FAT (File Allocation Table, just like a mounted Windows file system) that can refer to contents that may exist outside the *file*, other vendor applications can be more complicated.[781]

A study by Zhou, Huang, Li and Wang[2022] investigated the performance interplay between file systems and Solid State Disks (SSD), by running a file-server benchmark on a Kingston MLC 60 GB SSD. Four commonly used Linux filesystems (ext2, ext3, reiserfs and xfs) were mounted in turn using various options, e.g., various block sizes, noatime, etc.

Figure 13.28 shows the number of operations per second for a file-server benchmark (see paper and data for other benchmarks). A linear regression model involving the filesystem and mount options is a poor fit to the data; see Github–benchmark/filesystem-SSD.R.

A study by Sehgal, Tarasov and Zadok[1668] compared the power used when four commonly used filesystems were mounted in various ways, e.g., fixed vs. variable sector size, different journal modes, etc. Various server workloads running on Linux were measured; web server power consumption varied by a factor of eight, mail server by a factor of six and file and database by a factor of two.

Creating an executable: Many applications are built by translating source code to an executable binary, with the translation tools often supporting many options, e.g., gcc supports over 160 different options for controlling machine independent optimization behavior. Compiler writers strive to improve the quality of generated code, and it is to be expected that the performance of each release of a compiler will be different from the previous one; there have been around 150 released versions of gcc in its 30-year history.

A study by Makarow[1195] measured the performance of nine releases of gcc, made between 2003 and 2010, on the same computer using the same benchmark suite (SPEC2000), at optimization levels 02 and 03.

Figure 13.29 shows the percentage change in SPEC number, relative to version 4.0.4, for the 12 integer benchmark programs compiled using six different versions of gcc. SPEC has a long history of being used for compiler benchmarking, and it is possible that the versions of gcc used for this comparison have already been tuned to do well on this benchmark, meaning there is little, benchmark specific, improvement to be had in the successive versions used in this study.

The following `summary` output is from a mixed-effect model with the random effect on the intercept and slope: Github–Local

```
Linear mixed model fit by REML ['lmerMod']
Formula: value ~ gcc_version + (gcc_version | Name)
   Data: lme_02

REML criterion at convergence: 400.6

Scaled residuals:
    Min      1Q  Median      3Q     Max
-2.7256 -0.2748 -0.0683  0.3039  4.3372

Random effects:
 Groups   Name        Variance Std.Dev. Corr
 Name     (Intercept) 1192.792 34.537
          gcc_version    3.155  1.776   -1.00
 Residual                8.632  2.938
```

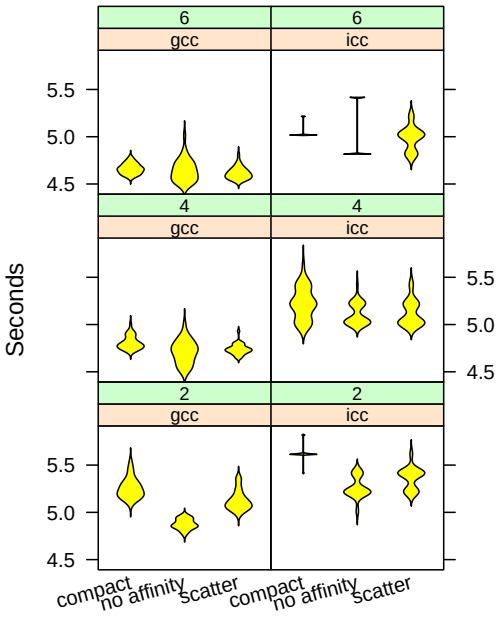


Figure 13.25: Execution time of `330.art_m`, an OpenMP benchmark program, using different compilers, number of threads and setting of thread affinity. Data kindly provided by Mazouz.[1225] Github–Local

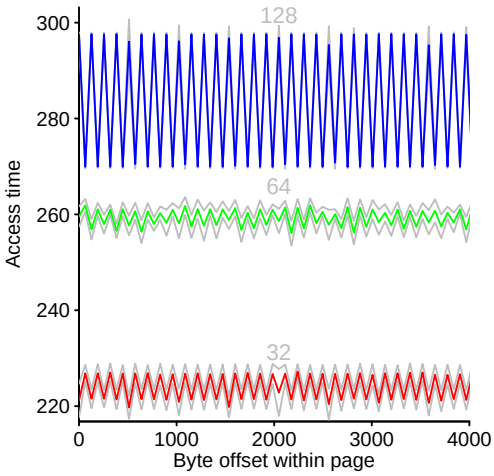


Figure 13.26: Access times when walking through memory using three fixed stride patterns (i.e., 32, 64 and 128 bytes) on a quad-core Intel Xeon E5345; grey lines at one standard deviation. Data kindly provided by Babka.[102] Github–Local

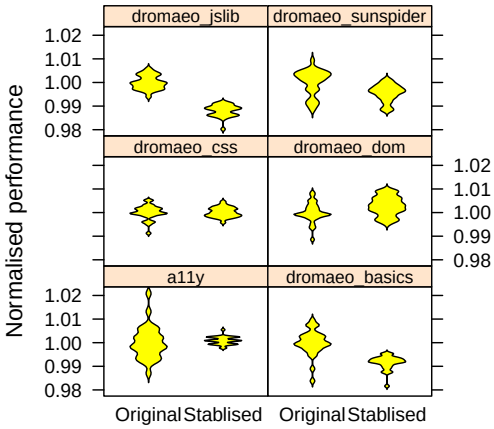


Figure 13.27: Performance variation of programs from the Talos benchmark run on original OS and a *stabilised* OS. Data from Larres.[1088] Github–Local

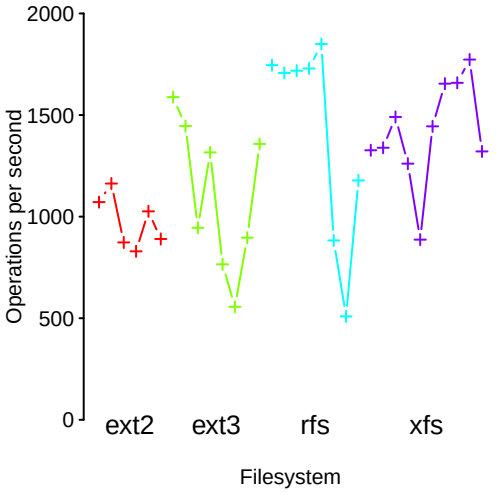


Figure 13.28: Operations per second of a file-sever mounted on one of ext2, ext3, rfs and xfs filesystems (same color for each filesystem) using various options. Data kindly supplied by Huang.[2022] Github–Local

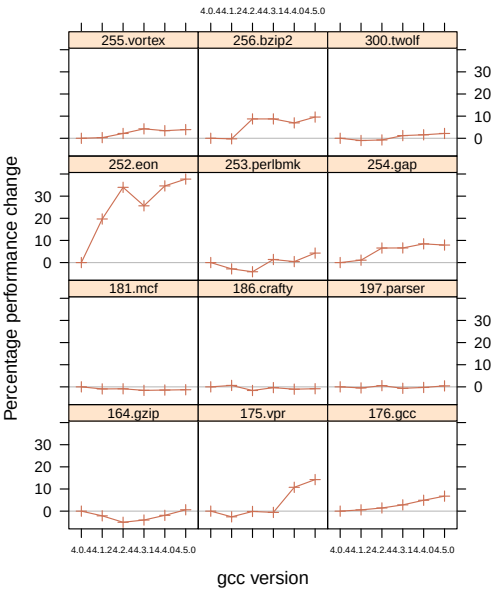


Figure 13.29: Percentage change in SPEC number, relative to version 4.0.4, for 12 programs compiled using six different versions of gcc (compiling to 64-bits with the 03 option). Data from Makarow.[1195] Github–Local

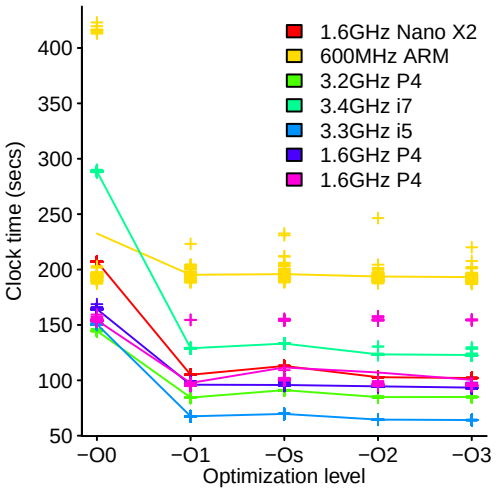


Figure 13.30: Execution time of the xy file compressor, compiled using gcc using various optimization options, running on various systems (lines are mean execution time when compiled using each option). Data kindly supplied by Petkovich.[454] Github–Local

```
Number of obs: 72, groups:  Name, 12

Fixed effects:
            Estimate Std. Error t value
(Intercept) -29.7469    11.0553  -2.691
gcc_version   1.4126     0.5513   2.562

Correlation of Fixed Effects:
            (Intr)
gcc_version -0.997
convergence code: 0
boundary (singular) fit: see ?isSingular
```

The general picture painted by the model results is of a small improvement with each gcc release, which is swamped by the size of the random effects, while the picture painted by figure 13.29, is of some releases having a large impact on some programs.

A study by de Oliveira, Petkovich, Reidemeister and Fischmeister[454] investigated the impact of compiler optimization and object module link order on program performance. Figure 13.30 shows the time taken by the xy file compression program, compiled by gcc using various optimization options, to process the Maximum Compression test set on various systems. The results show that different optimization levels have a different performance impact on different systems (the lines would be parallel if optimization level had the same impact for each system).

Compiling is the first step in the chain of introducing system variability into program performance, the next step is linking. Figure 13.31 shows execution time of `Perlbench` (one of the SPEC benchmark programs), on six systems, when the object files used to build the executable are linked in three different orders and with address randomization on/off. Some systems share a consistent performance pattern across link orderings, and some systems are not affected by address randomization. But there is plenty of variation across all the variables measured.

Tools: Dynamic profiling tools such a `grpof` work by interrupting a program at regular intervals during execution (e.g., once every 0.01 seconds) and recording the current code location (often at the granularity of a complete function). The results obtained can depend on interrupt frequency and the likelihood of being in the process of calling/returning from the profiled function.[612]

13.3.3 The cloud

Cloud computing has become a popular platform for applications that require non-trivial compute resources. The service level agreements offered by cloud providers specify minimum levels of service, e.g., Amazon's June 2013 EC2 terms specify 99.95% monthly uptime.[50] Cloud services general run virtualized instances, which means access to the real hardware may sometimes be shared. Shared hardware access causes performance to vary from one run to the next; what form might the characteristics of this variation take?

A study by Schad, Dittrich and Quiané-Ruiz[1641] submitted various benchmarks, as jobs, to Amazon's Elastic Computing Cloud (EC2), twice an hour over a 31-day period; a variety of resource usage measurements were recorded. Figure 13.32 shows one set of resource usage measurements, the Unix benchmark utility (Ubench; a cpu benchmark) running on small (upper) and large (lower) EC2 instances located both in Europe (red) and the US (green).

Both plots show more than one distinct ranges of performance. This data is an example of the variation experienced in Amazon's EC2 performance over one particular time period, and there is no reason to believe that any subsequent benchmarking will exhibit one, two, three or more distinct performance ranges.

13.3.4 End user systems

Benchmark data supplied by end-users, run on the computing systems they own, is likely to be subject to numerous known and unknown unknowns.

It may be impractical to switch off the many background processes that may be running on, for instance a user's Windows machine, which might include: Internet based

toolbars, anti-virus systems and general OS housekeeping processes. PassMark Software specializes in benchmark solutions for Microsoft Windows based computers, and Wren[1980] kindly provided 10,000 memory benchmark results.

Figure 13.33 shows the results (in sorted order) from 783 systems containing an Intel Core i7-3770K processor (whose official clock speed is 3.5GHz, some users may be overclocking). This is another example (see fig 10.25) of the wide range of performance reported for apparently very similar end-user systems.

User applications can have complex internal structures and modes of operation that invalidate assumptions made by a benchmark. For instance, application data files may not be represented as a contiguous sequence of bytes, but contain internal meta-data and pointers to blocks of data in other files.[781]

13.4 Surveys

This section discusses questionnaire surveys. Organizations use surveys are used to obtain information about customers and the market(s) they are targeting, e.g., characteristics of open source developers;[1596] see fig 8.13. Applications need to run reasonably well on the computers that customers currently use (see fig 8.27), and to coexist (or interoperate) with the versions of libraries and other applications installed on these computers. A lot of software engineering information only exists in the heads' of the people who build software systems, and this information can only be obtained by asking these people questions and analysing their answers.

The `survey` package supports the analysis of samples obtained via surveys.

The characteristics of data encountered in survey samples include:[422]

- missing data: people don't answer all the questions or stop answering after some point,
- misleading answers: giving answers that show those involved in a better light, such as job adverts listing trendy topics and languages to attract more applicants,
- spatial information: how subjects are distributed geographically,

Studies have found[518] that self-assessment of skills and character have a tenuous to modest relationship with actual performance and behavior. The correlation between self-ratings of skill and actual performance in many domains is moderate to meager.

Several studies by your author[931,932,934] included a component that asked developers about how many lines of code they had read and written during their professional career.

This question requires a lot of thought to answer, and there are many ways of adding up the numbers. Does reading the same line twice count as two lines, or one line unless the developer involved had forgotten reading it? How much does visually searching a screen of code (e.g., for a particular identifier) count towards lines read? Counting the number of lines in the programs written by a developer is likely to underestimate the number of lines they have written; a line of code may be written and then deleted, an existing line may be modified slightly.

Figure 13.34 shows the number of lines of code that 101 professional developers estimate they have written. While an exponential model fits the data, the variance explained is small.

A survey of the knowledge, or skill, of members of a population requires subjects to provide correct answers to questions.

Item response theory (IRT) deals with the design, analysis, and scoring of tests and questionnaires. The `ltm` package (latent trait models) supports the analysis of item response data.

What is the probability that a subject, m, will give the correct answer to the i^{th} question, x_{mi}, when the subject has a knowledge/skill level of z_m? The answer given by IRT[156] is:

$$P(x_{mi} = 1|z_m) = c_i + (1 - c_i)g(\alpha_i \times (z_m - \beta_m))$$

where: c_i is the probability that a subject will guess the correct answer, α_i is a measure of how well the question discriminates between subjects having a low/high level, β_i the question difficulty, and $g(.)$ a link function (often the logit function; the default used by the function in the `ltm` package).

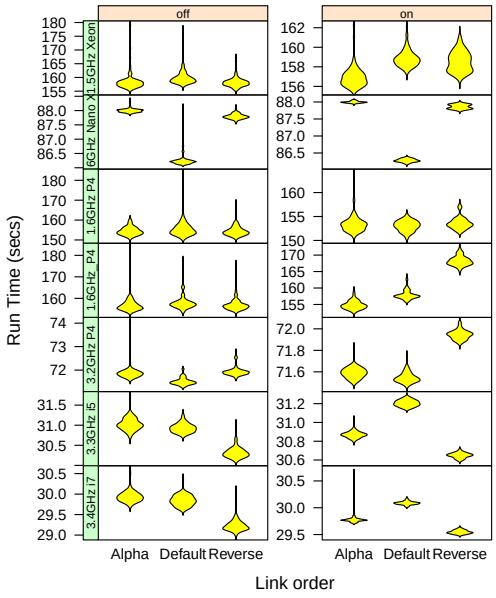


Figure 13.31: Execution time of Perlbench, part of the SPEC benchmark, on six systems, when linked in three different orders and address randomization on/off. Data kindly supplied by Reidemeister.[454] Github–Local

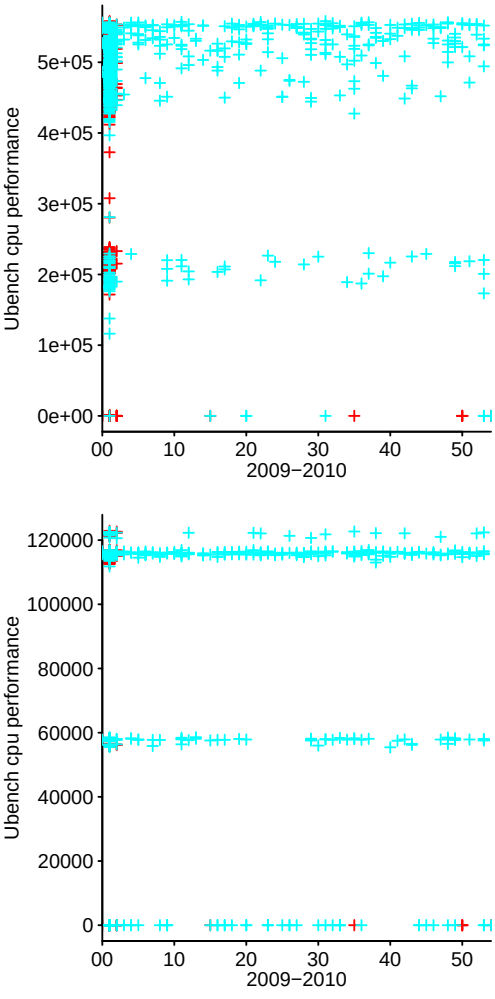


Figure 13.32: Ubench cpu performance on small (upper) and large (lower) EC2 instances, Europe in red and US in green. Data kindly provided by Dittrich.[1641] Github–Local

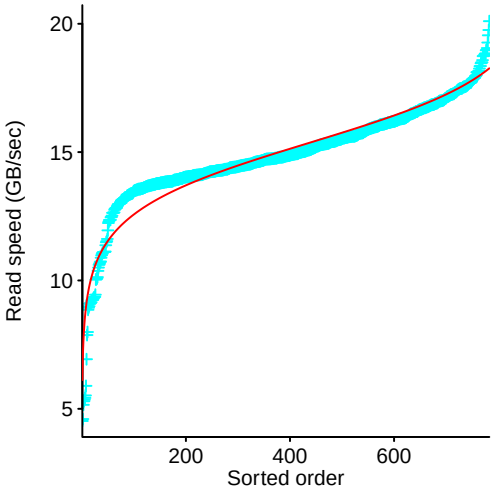


Figure 13.33: Performance of PassMark memory benchmark on 783 Intel Core i7-3770K systems; line is fitted logit model. Data kindly supplied by Wren.[1980] Github–Local

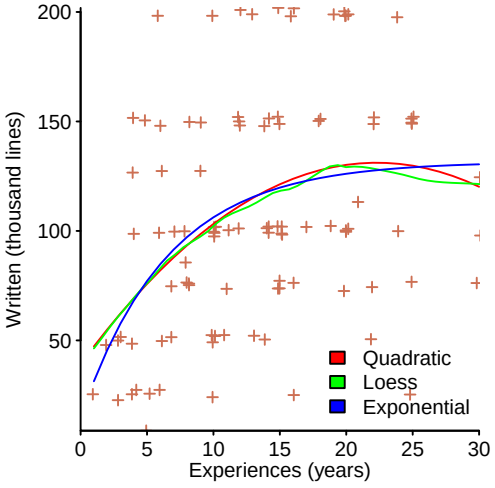


Figure 13.34: Number of lines of code that 101 professional developers, with a given number of years experience, estimate they have written, lines are various regression fits. Data from Jones.[931,932,934] Github–Local

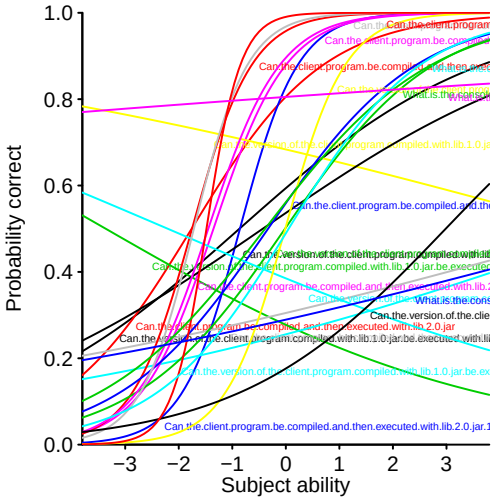


Figure 13.35: Probability that a subject, having a given relative ability, will answer a question correctly: lines are for each question in a fitted Rasch model. Data from Dietrich et al.[497] Github–Local

The Rasch model is simpler, and widely used; it contains a single parameter, β_i, and assumes there is no guessing (i.e., $c_i = 0$), and questions share the same ability to discriminate between subjects at different levels (i.e., $\alpha_i = 1$); the logit function is used as the link function, and the equation is:

$$P(x_{mi} = 1|z_m) = \frac{e^{z_m - \beta_m}}{1 + e^{z_m - \beta_m}}$$

The `ltm` package supports the fitting of Item response models having one (the Rasch model), two (α_i is included) and three (all parameters are included) parameter models; the logit function is the default link function.

A study by Dietrich, Jezek and Brada[497] investigated one aspect of developer knowledge of a language: knowledge of Java type compatibility. The yes/no answers to 22 questions provided by the 184 professional developers can be fitted to an IRT model.

The following code fits a Rasch model (single parameter), and a two parameter model using the `ltm` function (`z1` and `z2` are dummy names used to denote each factor); a three parameter model fails to be fitted by the `tpm` function.

```
library("ltm")

corr_mod1=ltm(corr_df ~ z1)

plot(corr_mod1)

corr_mod2=ltm(corr_df ~ z1+z2)

summary(corr_mod2)

t=tpm(corr_df) # fails to fit
```

Figure 13.35 shows: the probability that a subject having a given level of ability will correctly answer each question.

Section 12.4.2 discusses the analysis of ranked items, i.e., placed in a preferred order.

The results of many studies[62] have found that most subject ratings are based on an ordinal scale (i.e., there is no fixed relationship between the difference between a rating of 2 and 3, and a rating of 3 and 4), that some subjects will be overly generous or miserly in their rating, and that without strict rating guidelines different subjects apply different criteria when making their judgements (which can result a subject providing a list of ratings that is inconsistent with all other subjects).

There is no guarantee that data can be fitted, using this model. A study by Wohlin, Runeson and Brantestam[1974] investigated the faults found in an 18-page document by student and professional developers. Your author was unable to fit an IRT model to the data; see Github–regression/stvr95.R.

When software systems are built by a small group of people, the developers may be called on to solve a wide range of computer related issues, including the testing and tuning of the user interface. The System Usability Scale (SUS)[263,264] is a widely used usability questionnaire that produces a single number for usability. One study[1853] compared five methods of evaluating website usability, and found that SUS produced the most consistent results for smaller sample sizes.

Chapter 14

Data preparation

14.1 Introduction

The most important question to keep asking yourself while examining, preparing and analyzing any data is: Do I believe this data?

Do patterns appear where none are expected, are expected patterns absent, are human errors missing from the raw data, does the data collector believe whatever they are told, does the measurement process create incentives for people to game it?

Books and presentations on data analysis rarely mention that a large percentage of the time spent on data analysis often has to be invested in data preparation (perhaps 80%, or more, of analysis effort), getting data into a form suitable for the chosen statistical analysis techniques (or statistical package).[i]

Perhaps the largest task within data preparation is data cleaning; an often overlooked[1137] aspect of data analysis that is an essential part of the workflow needed to avoid falling foul of the adage: garbage in, garbage out.

Domain knowledge is essential for data cleaning; patterns have to be understood in the context in which they occur. The fact that many data cleaning activities are generic does not detract from the importance of domain knowledge. For instance, software knowledge tells us that 1.1 is not a sensible measurement value for lines of code (this appears in the NASA MDP dataset), talking to developers at a company to discover they don't work at weekends (e.g., the dates in the 7digital data) and knowing that system support staff used the Unix `pwd` command to check that the system was operational (an analysis of job characteristics for a NASA supercomputer[586] has to first remove 56.8% of all logged jobs, which are uses of `pwd`).

A study by Cohen, Teleki and Brown[380] investigated data from 2,751 code reviews, from one company over a 10-month period. During the data cleaning process they removed all code reviews reported taking less than 30 seconds, involving more than 2,000 LOC and processing code at a rate greater than 1,500 lines per hour. Figure 14.1 shows all reported data points, with points inside the triangle being the only measurements retained for analysis.

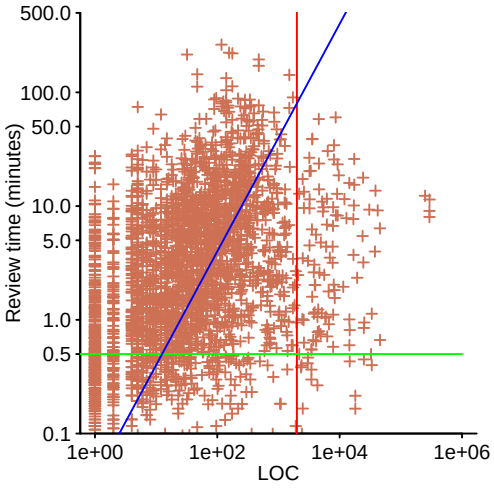


Figure 14.1: Reported LOC and duration of 2,751 code reviews, for one company; reported reviews lasting less than 30 seconds (below green line), involving more than 2,000 LOC (to right of red line), processing at a rate greater than 1,500 LOC per hour (above blue line). Data extracted from Cohen et al.[380] Github–Local

Data cleaning is often talked about as-if it is something that happens before data analysis, in practice, the two activities intermingle; the time spent checking and cleaning the data provides insights that lead to a better understanding of the kinds of analysis that might be applicable, also, results from a preliminary analysis can highlight data needing to be cleaned in some way. Data preparation is discussed here in its own chapter, as-if it was performed as a stand-alone activity, in order to simplify the discussion of material in other chapters.

Once cleaned, data may need to be restructured, e.g., rows/columns contained in different files merged into a single data table, or the row/columns in an existing table reorganised in some way. The required structure of the data is driven by the operations that need to be performed on it (e.g., finding the median value of some attribute), or the requirements of the library function used to perform the analysis.

[i] In early versions, this chapter appeared immediately after the Introduction, but in response to customer demand, it was moved here (the penultimate chapter); people want to read about the glamorous stuff, data analysis, not the grunt work of data preparation.

It may be necessary to remove confidential information from the data, or to anonymize information that might be used to identify individuals (or companies). Datasets do not exist in isolation, and it may be possible to combine apparently anonymous datasets to reveal information;[ii] k-anonymity and l-diversity are popular techniques for handling anonymity requirements (in a k-anonymized dataset each record is indistinguishable from at least $k-1$ other records, while l-diversity requires at least l distinct values for each sensitive attribute). Techniques for anonymizing data are not covered here; Fung et al[636] survey techniques for privacy-preserving data publishing, Templ et al[1821] provide an introduction to statistical disclosure control and the `sdcMicro` package.

The behavior of tools used during software development may result in information being lost during some activities. For instance, the Git distributed version control system does not carry-over information from the originator of push/pull requests, and allows commits to be rebased (which changes their history timeline).[670]

While a lot of software engineering data comes from measurements made using software tools, some is still derived from human written records (which can contain a substantial number of small mistakes[922]).

Data cleaning involves a lot of grunt work that often requires making messy trade-offs and having to make do. Tools are available to reduce the amount of manual work involved, but these sometimes require placing trust in the tool doing the right thing; available tools include:

- OpenRefine[1420] (was Google Refine) reads data into a spreadsheet-like form and supports sophisticated search/replace and data transformations editing,
- the `editrules` package checks data values for consistency with user specified rules involving named columns, e.g., `total_fruit ==total_apples+total_oranges`,
- the `deducorrect` package performs automatic value transformations based on user specified consistency rules, relating to column values that must be met (e.g., failure to meet the condition `total_x > 0` will result in any value in that column having a negative sign removed); this package can also impute missing values,
- there are a variety of special purpose packages that handle domain specific data, e.g., the `CopyDetect` package detects copying of exam answers in multi-choice questions.

14.1.1 Documenting cleaning operations

Documenting the changes made to the original data, during the cleaning process, serves a variety of purposes, including:

- enabling third-parties to check that the changes are reasonable and don't produce an unrealistic analysis,
- enabling potential sources of uncertainty to be checked when multiple analysts publish results based on the same dataset, i.e., if there are differences in the results, it is possible to check whether these differences are primarily the result of differences in data cleaning,
- providing confidence to users of the final results of the analysis, that the researcher doing the work is competent, i.e., that cleaning was performed.

Ideally the operations performed on the original data, to transform it into what is considered a clean state, are collected together as a script for ease of replication.

Some cleaning activities are trivial and yet need to be performed to prevent the analysis being overwhelmed by what appears to be many special cases. For instance, an analysis[78] of different company's response to vulnerabilities reported in their products, started from raw data that sometimes contained slightly different ways of naming the same company. The company names data had to be cleaned to ensure that a single name was consistently used to denote each organization; see Github–data-check/patch-behav.R.

The NASA Metrics Data Program (MDP) dataset contains fault data on 13 projects, and has been widely used by researchers (a literature survey for the period 2000 to 2010[769] found that 58 out of 208 fault prediction papers used it). This dataset contains many problems[730] that need to sorted out, e.g., columns with all entries having the same value (suggesting a measurement or conversion error has occurred), duplicate rows, missing

[ii] 63% of the US population can be uniquely identified using only gender, ZIP code and full date of birth.[700]

values (many occurred in rows calculated from other rows and involved a divide by zero), inconsistent values (e.g., number of function calls being greater than the number of operators) and nonsensical values, e.g., lines of code having fractional values.

Despite the non-trivial work needed to clean the MDP dataset, to remove spurious data, the authors of many papers using this dataset, have either not cleaned it, or only given a cursory summary of the cleaning operations,[217] e.g., " . . . removes duplicate tuples . . . along with tuples that have questionable values . . . ", does not specify what values were questionable. Consequently, even although the original dataset is publicly available it is difficult to compare the results published in different papers, because no information is available on what, if any, data cleaning operations were performed; so much of the data is in need of cleaning that any results based on an uncleaned version of this dataset must be treated with suspicion.

See Github–data-check/NASA_MDP-data_check.R for examples of integrity checks performed on the MDP dataset.

It is possible that the values appearing in a sample are correct, but have been misclassified.

A survey of 682,000 unique Android devices in use during 2015, by OpenSignal,[1421] included the screen height and width reported by the device; see figure 14.2, upper plot. Many devices appear to have greater width than height, particularly those with smaller screens. Perhaps the device owners are viewing the OpenSignal website with their phones in landscape mode; the lower plot in Figure 14.2 switches the dimensions so that height has the larger value; switched values in red.

The fault repositories of open source projects are publicly available, and the repositories of larger projects are a frequent source of data for fault analysis/prediction researchers.

A study by Herzig, Just and Zeller[820] manually classified over 7,000 issue reports extracted from the fault repositories of seven large Java projects. They found that on average 42.6% of reports had been misclassified, with 39% of files marked as defective not actually containing any reported fault (any fault prediction models built using the uncleaned data are likely to be misleading at best and possibly very wrong). An earlier analysis[821] had found that between 6 and 15% of bug fixing changes addressed more than one issue.

Possible reasons for the misclassification include: the status of an issue not being specified when the initial report is filed, resulting in the default setting of *Bug* being used; issue submitters having the opinion that a missing feature is a bug (request for enhancement was the most commonly reclassified status); and bug reporting systems only supporting a limited number of different issue statuses (forcing the submitter to use an inappropriate status).

The study also highlighted how much effort data cleaning consumes; the work was performed independently by two people and took a total of 725 hours (90 working days).

Sometimes the measured values from one or more subjects (e.g., people or programs) are remarkably different from the values measured for other subjects. It can be tempting to clean the data by removing the value for these subjects from the sample. A study by Müller and Höfer[1326] removed data on seven out of 18 subjects, because they considered the performance of these subjects was so poor that they constituted a threat to the validity of the experiment (whose purpose was to compare the performance of students and professional developers). This kind of activity might be classified as outlier removal or manipulating data to obtain a desired result, either way, documenting the cleaning activity makes it possible for readers of the analysis to decide.

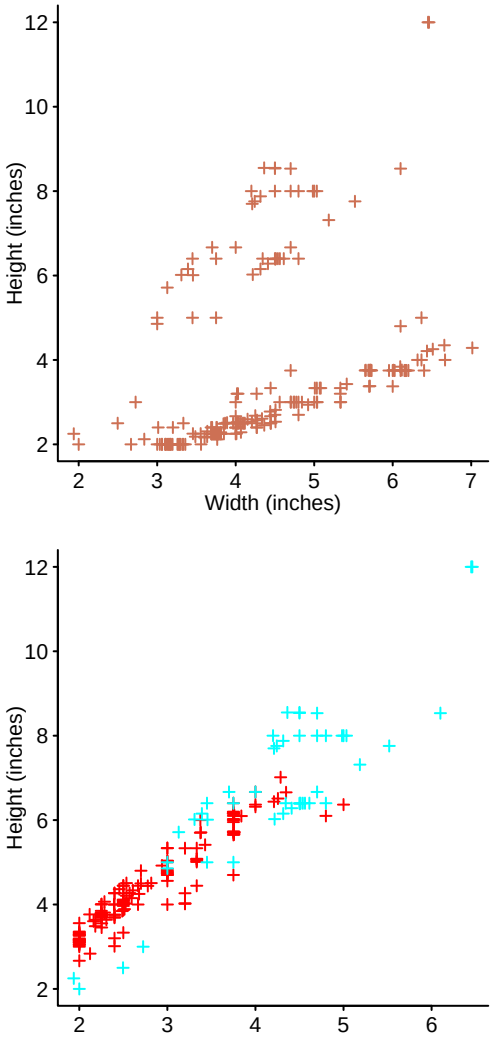


Figure 14.2: Screen height and width reported by 682,000 unique devices that downloaded an App from OpenSignal in 2015 (upper), reported measurements ordered so height always the larger value (lower). Data from OpenSignal.[1421] Github–Local

14.2 Outliers

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism".[789]

In some applications, observations that deviate from the general trend are the ones of interest,[321] e.g., intrusion detection and credit card fraud. This subsection covers the case where deviate observations are unwanted; some later subsections cover software engineering situations where deviate observations are themselves the subject of study.

Methods for handling outliers include:

- using a statistical technique that does not assign too much weight to observations that deviate from the patterns followed by most other observations. Techniques capable of performing the desired statistical analysis are not always available, but when R functions implementing them are available they may be discussed in the appropriate section,
- detecting and excluding outliers from the subsequent analysis. Traditionally outliers have been manually selected and excluded from subsequent analysis. This approach can work well when the sample contains a small amount of data, and the person doing the detection has sufficient domain knowledge. There are a variety of functions that automate the process of outlier selection and handling, some of these are discussed below; see section 11.2.6.

another definition of outlier detection is " . . . the problem of finding patterns in data that do not conform to the expected normal behavior."[321] This definition requires that an expected normal behavior be known, along with a method of comparing values for *outlyingness*.

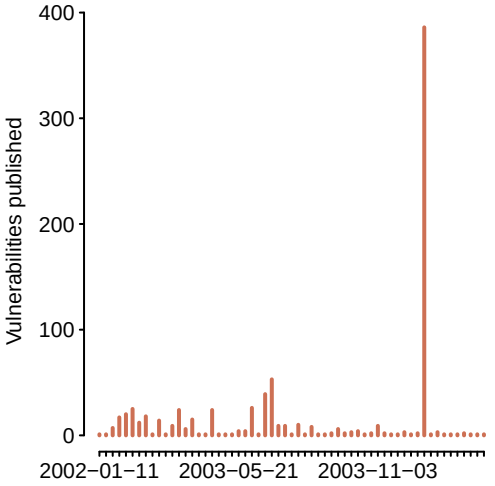


Figure 14.3: Number of reported vulnerabilities, per day, in the US National Vulnerability Database for 2003. Data from the National Vulnerability Database.[890] Github–Local

Figure 14.3 shows a suspicious spike in the number of daily reported vulnerabilities recorded in the US National Vulnerability Database for 2003. What behavior could explain this pattern? Perhaps all vulnerabilities that had been reported, but not yet fully processed, were simply published, for the public to see, at the end of the year?

A study by Zheng, Mockus and Zhou[2018] investigated what they called problematic values, in the task completion time for Mozilla projects. A manual analysis found that for various reasons, some patches for reported faults were being committed in batches (so the commit date did not reflect the date the code for the patch was created). Enough information was available (i.e., there was data redundancy) to build a model that suggested values more likely to be correct (50% more accurate was claimed).

The date when an event occurred may appear unlikely, based on domain knowledge, e.g., staff rarely work at weekends. The following output shows a count of the number of features recorded as being Done, in a company using an Agile process,[1] for each day of the week. Monday is day 0, and the counts for Saturday/Sunday should be zero; the non-zero values suggest a 2-4% error rate, comparable with human error rates for low stress/non-critical work. Github–Local

```
> table(Done_day %% 7)
  0   1   2   3   4   5   6
670 708 669 716 447  12  16
```

Should outliers be removed from the sample used for analysis?

While removing outliers may improve the quality of the model fitted to an equation, does it improve the quality of the fit of the model to reality?

Without understanding the processes that generated the data, there is no justification for removing any value.

The real issue with outliers is the impact they have on the final result. In a large sample, a few unusual values are unlikely to have any real impact data.

However, outliers are handled, any decision to exclude them from analysis needs to be documented.

14.3 Malformed file contents

A sign that data, or its organization in a file, is malformed in some way, is that the variable into which a file has been read does not have the expected contents (e.g., incorrect number of columns, or a surprising type for the data in one or more columns, e.g., a string where a number was expected). The `str` function provides a quick and easy way of checking the types of columns in a data frame.

File formatting issues to watch out for include:

- functions for reading data in R (e.g., `read.csv` and `read.table`) often use the first few lines of the file being read as the format to use when reading the rest of the file, i.e., the number of columns contained in each row and the datatype of the values in each column. If there are one or more rows that do not follow the format selected at the start of the file (e.g., different number of column delimiters; perhaps the result of non-delimited strings such as a missing pair of quote characters), then subsequent values may appear in the other columns, or be converted to a different type,

- termination delimiter missing from a string value; this can result in the contents of the following row being treated as part of the current row (because the newline is treated as part of the string),
- cut-and-pasting of data between media introducing conversion errors, e.g., the digit zero treated as the letter D or G during image to character conversion.

A variety of ad-hoc techniques are available for locating the cause of problems. For instance, the following code will convert all values that do not have the format of a number to NA, which are then easily located using base-library support for processing NAs, with their row index found using the which function:

```
which(is.na(as.number(as.character(data_frame$column_name))))
```

The complete.cases function returns a vector specifying which rows in its data.frame argument are complete, i.e., do not contain any NAs; the na.omit function returns a copy of its argument with any rows containing NA omitted.

14.4 Missing data

Missing data (for instance, a survey where the entry for a person's age is empty) is often the rule, rather than the exception, and books have been written on the subject. Missing data may be *disguised*, in that it appears as a reasonable value[1460] (e.g., zero when the range of possible legitimate values includes zero), or it may not be visible to the measurement process (e.g., intermittent check-ins to version control obscuring the detailed change history[1362]), or the input process provides a two item choice (e.g., male/female), with one item being the default and thus appearing as the missing value when no explicit choice is made.

The starting point for handling missing data is to normalise how it is denoted, to the representation used by R, i.e., NA (Not Available). Normalisation ensures that all missing values are treated consistently; special case handling of NA is built into R and many functions include options for handling NA.

A wide variety of different representations for missingness may be encountered (e.g., special values that cannot occur as legitimate data values, such as: 9999, "#N/A", "missing", or no value appearing between two commas in a comma separated list), and it is not uncommon for different columns within a dataset to use different representations (because they originate from different measurement sources).

The following code illustrates one method for changing a known representation of missing value to NA (the second form would be necessary if 9999 could appear as a legitimate value in a column other than size):

```
data[ data == 9999 ] = NA  # set all elements having value 999 to NA

# set all elements of column size having value 999 to NA
data$size[ data$size == 9999 ] = NA
```

Once missing values have been explicitly identified it is possible to move on to deciding whether to ignore these cases or to replace NA with some numeric value. Some algorithms can handle missing values while others cannot; R functions vary in their ability to handle missing values. A few techniques for selecting the replacement value are discussed below.

The R base I/O functions, such as read.csv, have conventions for handling the case of zero characters appearing between the delimiters on each line of a file. The behavior depends on the type that has been assigned to values in a particular column. For columns assigned a numeric type, zero characters are treated as-if NA appeared between the delimiters, while for columns assigned a string type the zero character case is treated as the empty string rather than NA, i.e., treated the same as the string "". These functions support a variety of options for changing the default the handling of zero characters and the handling of leading/trailing white-space between delimiters.

As an example: reading a file containing the columns below left has the same effect as reading a file containing the columns below right:

```
X,Y_str,Z          X,Y_str,Z
1,"abc",2.2        1,"abc",2.2
2,,3.1             2,"",3.1
,NA,2              NA,NA,2
```

The `table` function counts occurrences of values, and by default does not include `NA` in the count; the `useNA` options has to be used to explicitly specify that `NA` be counted:

```
table(data$some_column, useNA="ifany") # limit the count to one column
```

This one column use can be expanded to cover every column in `data`. If the output is too voluminous, the number of columns processed can be reduced, or the call to `table` replaced by a call to `tabulate`, which provides more options to control behavior:

```
sapply(colnames(data), function(x) table(data[ , x], useNA="ifany"))
```

While there may be documentation specifying how missing values are represented, such details may not be documented. An analysis of a dataset using the above code may show a suspiciously large number of values such as `9999` or `-1` (for an attribute that can never be negative), a result that suggests further investigation is worthwhile.

14.4.1 Handling missing values

When deciding what to do about missing values, it is important to try to understand why the values are missing. The following categories are commonly encountered in the analysis of missing data:

- Missing completely at random (MCAR): As the name suggests, the selection of missing values occurred completely at random. Statistically this is the most desirable kind of missingness, because it means there is no bias in the missing values,
- Missing at random (MAR): This sounds exactly like MCAR, but it is not completely random in the sense that the choice of which values are missing is influenced by other values in the sample. For instance, the level of seniority may correlate with the likelihood that survey questions about salary are answered,
- Missing not at random (MNAR): This missingness could be as random as MAR, with the one difference that the choice of missing values is influenced by values not in the sample. For instance, the name of the developer who originally wrote the code referenced in a fault report may be missing if that developer is friendly with the person reporting the fault, with friendship not being a recorded in the sample.

The following code can be used to get a rough estimate of the correlation between the rows of a `data.frame` that contain missing values (figure 8.8 illustrates a method of visualizing this information):

```
x=is.na(some_data_frame)
# highlight rows having some, but not all, missing values
cor(subset(x, sd(x) > 0))
```

Many analysis techniques handle missing values by ignoring the rows or columns that contain them; if the sample contains many rows and a low percentage of missing values, this behavior may not be a problem. However, if the sample contains a large percentage of missing values, any analysis will either have to make do with a smaller number of measurements, be limited to using techniques that can gracefully adapt to missing data (i.e., don't ignore rows containing one or more missing values) or be forced to use estimated values for the missing data.

The ideal approach is to use an algorithm capable of handling samples that include missing data.

The process of estimating a value to use, where none is present in the sample, is known as *imputing*.

A quick and dirty method of imputing values, that can be effective, is to replace a missing value by the mean of the values in the corresponding column containing the missing value; alternatively, if the data is ordered in some way (e.g., dates), the last value appearing before the missing value might be used.

A more sophisticated approach to imputing values involves filling the missing value entries using other values present in the sample. The `Amelia`, `naniar` and `VIM` packages provide a variety of functions for visualizing datasets containing missing values and imputing values for these entries.

A study by Buettner[275] investigate project staffing, but was not able to obtain complete staffing information. Figure 14.4 shows a loess fit and the 95% confidence bounds.

Some R functions support the use of splines for interpolating values. Splines originated as a method for connecting a sequence of points by a smooth curve, not as a method for fitting a curve minimizing some error metric. Apart from their familiarity, there is no reason to prefer the use of splines over other techniques (implementation issues also exist with the `bs` and `ns` functions, in the `splines` package, when fitting a model with the `predict.glm` function[1888] and then making predictions using new data points).

Data may be missing because the sample may not be large enough to be likely to contain instances of rarely occurring cases (which would be seen in a larger sample). Good-Turing smoothing[643] is a technique for adding non-zero counts to adjust for unseen items.

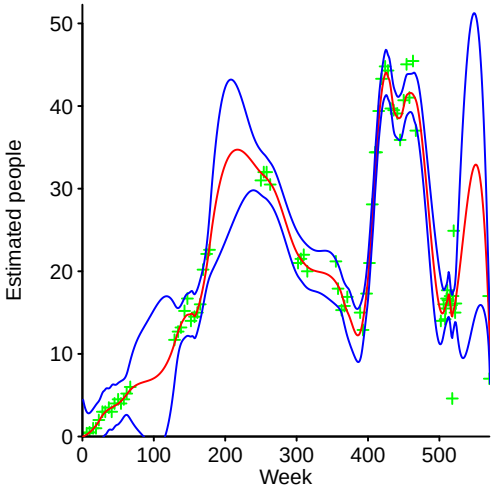


Figure 14.4: Estimated staff working on a project during each week; lines are a fitted loess model and 95% confidence bounds. Data from Buettner.[275] Github–Local

14.4.2 `NA` handling by library functions

R functions vary in their ability to handle `data.frames` containing `NA`, with the behaviors exhibited including:

- behaving in unpredictable ways when `NA` is encountered,
- behaving in predictable ways, that perhaps is surprising to the unknowledgeable, e.g., the value of `NA ==NA` is `NA`, as is `NA !=NA`,

  functions that operate on complete rows or columns have a variety of behaviors when they counter one or more `NA`s, including:

  - supporting a parameter, often called `na.rm`, which can be used to select among various methods for handling any `NA` that occur,
  - ignoring rows containing one or more `NA`, e.g., `glm` ignores these rows by default, but this behavior can be changed using the `na.action` option,
  - making use of information present in rows containing one or more `NA`, e.g., the `rpart` function,

Some regression model building functions return information associated with individual data points, such as residuals. If the function removes rows containing any `NA` before building the regression model, the number of data rows included in the returned model may be less than originally passed in, unless rows containing `NA` is reinserted, e.g., by using the `naresid` function.

14.5 Restructuring data

When the data of interest is spread over several files, it may be necessary to read two or more files and merge their contents into a single `data.frame`.

If two datasets contain shared columns (i.e., column names, column ordering and information held are the same), the `rbind` function can be used to join rows together, returning a single `data.frame`; the `cbind` function performs the join operation for columns.

The `merge` function merges the contents of two `data.frames` based on one or more criteria, e.g., shared column names.

14.5.1 Reorganizing rows/columns

The organization of rows and columns in a `data.frame` may not be appropriate for that used by the library functions used to perform the analysis.

The values in a dataset are may be held in a wide format (i.e., a few rows and many columns), but a long format (i.e., many rows and a few columns) is required, or vice versa.

An example of wide format data is that used in figure 2.5; the IQ test scores have the form:

```
test,gender,1,2,3,4,5,6,7,8,9
verbal,Boy,8455,14171,17596,29308,30490,27544,16037,9857,4635
verbal,Girl,5448,10570,15312,28591,32385,30830,18557,11443,5321
quantitative,Boy,3138,19634,18258,29037,23255,30376,16504,12565,5095
quantitative,Girl,2313,16905,19002,32707,26438,32413,15215,10007,3406
non-verbal,Boy,1390,18144,20713,29245,25720,27077,18095,11369,6077
non-verbal,Girl,1165,14370,18564,30488,29342,30458,18387,10450,5075
CAT3,Boy,2505,14505,19556,29917,29607,30327,17960,9392,2787
CAT3,Girl,1813,10927,17872,31059,32867,33269,18016,9041,2394
```

The `melt` function, in the `reshape2` package, transforms `data.frames` to a long format, such as the following (only the first 11 lines are shown):

```
           test gender stanine count
1        verbal    Boy      X1  8455
2        verbal   Girl      X1  5448
3  quantitative    Boy      X1  3138
4  quantitative   Girl      X1  2313
5    non-verbal    Boy      X1  1390
6    non-verbal   Girl      X1  1165
7          CAT3    Boy      X1  2505
8          CAT3   Girl      X1  1813
9        verbal    Boy      X2 14171
10       verbal   Girl      X2 10570
11 quantitative    Boy      X2 19634
```

which was reorganized using the call (where `b_g_IQ` contains the data):

```
b_g=melt(b_g_IQ, id.vars=c("test", "gender"),
                 variable.name="stanine", value.name="count")
```

It is also possible to convert from long to wide format.

14.6 Miscellaneous issues

14.6.1 Application specific cleaning

The analysis of some kinds of data has acquired established preprocessing procedures; the data is not wrong, but transforming it in some way improves the quality of subsequent analysis. For instance, before analyzing text, common low interest words (such as "the" and "of", known as *stop words*) are removed; also words may be stemmed (a process that removes suffixes with the intent of uncovering the root word, e.g., kicked and kicking both become kick).

14.6.2 Different name, same meaning

Typos in character based data may be detected because of constraints on what can appear in domain specific sequences, e.g., the spelling of words. More difficult to detect problems include different people using different terminology for the same concept, or the same terminology for different concepts.

The SPEC 2006 benchmark results often include a description of the characteristics of the memory used by the computer under test. For historical marketing reasons, two scales are commonly used to specify memory performance; the DDR scale is based on peak bandwidth, while the PC scale uses clock rate. The SPEC result descriptions are not consistent in their choice of scale, and so before any analysis can be performed the values have to be converted to a single form. Also, for marketing reasons, the values are rounded to reduce the number of non-zero digits; an analyst interested in high accuracy would map the *marketing* values to their actual values; see Github–benchmark/scripts/SPEC-memory.awk.

An email address is sometimes the only unique identifying information available, e.g., the list of developers who have contributed to an open source project. The same person may have used more than one email address over the period of their involvement in a project, and it is necessary to detect which addresses belong to the same person.[1956]

14.6.3 Multiple sources of signals

Sometimes a value appearing in a sample could have come from multiple sources, only one of which is of interest. An example of this is the question: when did hexadecimal literals first appear as such in print?

One way of answering this question is to analyze the word n-grams (and associated year of book publication) Google have made available from their English book scanning project.[708]

The regular expression `^[0oO[xX][0-9a-fA-FoOl]]` (ohh, Ohh and ell were treated as the corresponding digits) returned 89 thousand matches.

OCR mistakes have resulted in some words being treated as hexadecimal literals, e.g., Oxford was sometimes scanned as 0xf0fd. The character sequence oxo is common, and looking at some of the contexts in which this sequence occurs suggests that the usage is mainly related to chemical formula (some uses are also likely to be references to a cooking product of this name).

Assuming that hexadecimal notation did not start appearing in books before electronic computers were invented, books prior to say 1945 (i.e., the end of World War II) can be ignored.

Your author also assumed that, if any hexadecimal literal appears in a book, at least one more such literal is likely to appear; applying this final filtering rule, the number of matches was reduced to 7,292; with 319 unique character sequences.

Figure 14.5 shows a comparison of the use of hexadecimal literals in C source with those extracted from Google books n-grams.

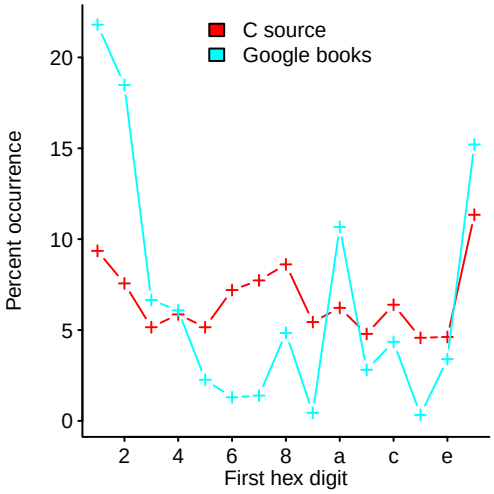


Figure 14.5: Percentage occurrence of the first digit of hexadecimal numbers in C source and estimated from Google book data. Data from Jones[930] and Michel et al.[1276] Github–Local

14.6.4 Duplicate data

Duplicate data can cause some analysis techniques to fail (e.g., regression modeling) or skew the calculated results.

Duplication data is easily generated: the collation of data from multiple sources can result in the same measurements appearing more than once, and there may be multiple measurements of the same event (e.g., logging of computer faults where a single root cause produces the same message at sporadic times after the fault is experienced[1806] and spatially or functionally adjacent units to generate messages;[1132] see Github–data-check/Blue-Gene.log).

The `duplicated` function returns information about rows that are exact duplicates. More subtle duplication may involve the values in a row/column differing by a constant factor from those in another row/column, e.g., one row contains temperature in Celsius while another uses Fahrenheit.

When the data is numeric, *close* duplicates can be highlighted using pairwise correlation; see section 10.5.4.

Some R functions handle duplicate row/columns gracefully (e.g., the `glm` function), while others give unpredictable results (e.g., the `solve` function, which inverts a matrix), the behavior depends on the algorithm used and what if any consistency checks were added by the implementer of the code.

14.6.5 Default values

Sometimes a measurement process returns what is considered to be a reasonable value, if it cannot return the actual value. For instance, IP geolocation services are always able to associate a country with an IP address, but when they are unable to further refine the location within a country, they return a location near the center of the country; for the USA this is close to the town of Potwin in Kansas (population 449) which appears to experience orders of magnitude more Internet related events, for its population size, than other towns in the US.[908]

14.6.6 Resolution limit of measurements

Some kinds of measurement are inherently inexact, e.g., time. When working close to the resolution limit of the measuring process, false signals can be generated by the interaction between the measurement resolution and the processes generating measurement events.

A study by Feitelson[585] measured the runtime of processes, executed on a system, to an accuracy of two decimal digits. Initial analysis of the number of processes whose execution fell within a given time interval found an unexpected behavior, there were many time intervals that did not contain any processes; see figure 14.6, upper plot. Further analysis found that the timer resolution was $1/64$ second, and the gaps were an artefact of the number of digits recorded, recording more digits (see figure 14.6, lower plot) resulted in fewer intervals containing no measurement points.

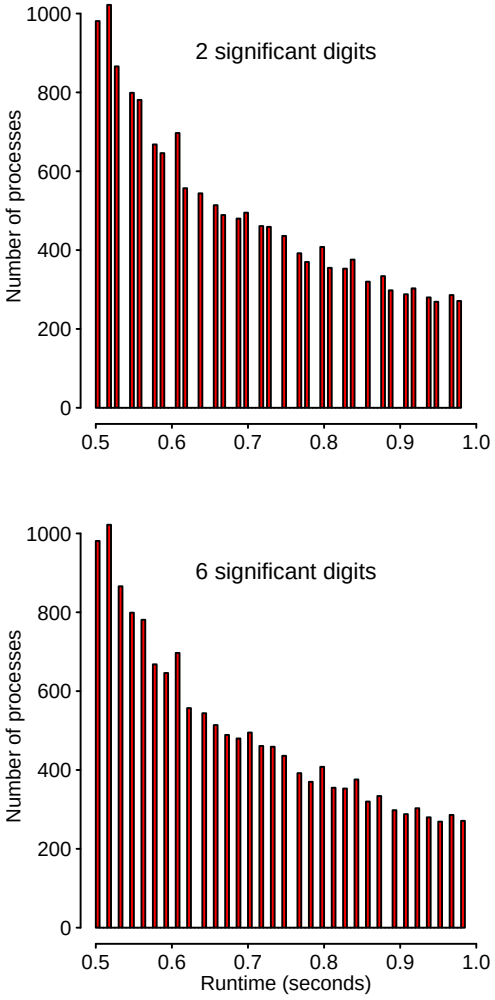


Figure 14.6: Number of processes executing for a given amount of time, with measurements expressed using two and six significant digits. Data from Feitelson.[585] Github–Local

14.7 Detecting fabricated data

A sample is not always derived from accurate measurements, the accuracy failures may be accidental or intentional, or might not involve any actual measurements, e.g., it has been fabricated.

Like all data analysis, detection of fabricated data is based on finding known patterns in the data, i.e., patterns that have previously been found to appear in known fabricated data. As always, the interpretation of why the data contains these patterns is the responsibility of the audience of the results; it is always worth repeating that domain knowledge is key.

One pattern of behavior observed in real world data, with some regularity, is the first digit of numeric values following Benford's law to a reasonable degree of approximation (while a figure of 30% of all datasets has been quoted, the actual figure is likely to be much smaller[1663]). The failure of data to follow Benford's law has been used to detect accounting and election[1610] fraud, identification of fake survey interviews[1654] and scientific research.[496]

While references to Benford's law usually involve the first digit of numeric values, there is a form that applies to the second and perhaps other significant digits.[1382] There has also been work[157] suggesting that the digit at the opposite end of numeric literals, the least significant digit, sometimes has a uniform distribution.

Benford's law specifies that the probability of the first digit having value d is given by: $P(d) = \log_{10}(1 + \frac{1}{d})$

Figure 7.53 shows percentage occurrences for the first digit of numeric literals in C source code.

If a set of independent and identically distributed random variables are sorted, the distribution of digits of the differences between adjacent sorted values is close to Benford's law.[1287] A test based on this fact can detect rounded data, data generated by linear regression and data generated by using the inverse function of a known distribution.[1382]

The `BenfordTests` package contains a variety of function for evaluating the conformity of a dataset to Benford's law.

When generating fabricated data, it is sometimes necessary to produce a random sequence of items. People hold incorrect beliefs[211] about the properties of random sequences and when asked to generate them produce sequences that contain predictable patterns, i.e., they are not random.

One study[1657] was able to build a model that predicted repeated patterns in an individual's *randomly* selected sequence, with around 25% success rate, but when the model built for one person's behavior was used to make predictions about another persons the success rate dropped to around 18%.

Detecting divergence from, or agreement with, these patterns of behavior depends on the authors of the data being unfamiliar with the expected patterns of behavior, or being lazy (i.e., being unwilling to spend the time making sure that the data they generate has the expected characteristics; the creators of the fictitious accounts publicly published by Madoff's companies, before his fraud was uncovered, made the effort to ensure they followed Benford's law[1667]).

An excess of round numbers has been used to suggest that data has been fabricated.[1031]

If people are willing to invest some effort, it is possible to manipulate data such that some statistical tests meet expectations;[1219] see Github–communicating/warp-pts.R.

Chapter 15

Overview of R

This chapter gives a brief overview of R for developers who are fluent in at least one other computer language. The discussion pays attention to language features that are very different from languages the reader is likely to be familiar with; the focus is on a few language features that can be used to solve most problems.

The R language is defined by its one implementation; available from the R core team.[1548] A language definition,[1547] written in English prose, is gradually being written.

R programs tend to be very short, compared to programs in languages such as C++ and Java; 100 lines is a long R program. It is assumed that most readers will be casual users of R, whose programs generally follow the pattern:

```
d=read_data()
clean_d=clean_and_format(d)
d_result=applicable_statistical_routine(clean_d)
display_results(d_result)
```

If your problem cannot be solved using this algorithm, then the most efficient solution may be for you, dear reader, to use the languages and tools you are already familiar with, to preprocess the data so that it can be analysed and processed using R.

R is a domain specific language, whose designers have done an excellent job of creating a tool suited to the tasks frequently performed when analysing the kinds of datasets encountered in statistical analysis. Yes, R is Turing complete, so any algorithm that can be implemented in other programming languages can be implemented in R, but it has been designed to do certain things very well, with no regard to making it suitable for general programming tasks.

As a language the syntax and semantics of R is a lot smaller than many other languages. However, it has a very large base library, containing over 1,000 functions. Most of the investment needed to become a proficient user of R has to be targeted at learning to how to combine these functions to solve the problem at hand. There are over 10,000+ add-on packages available from the CRAN (Comprehensive R Archive Network).

Help on a specific identifier, if any is available, can be obtained using the `?` (question mark) unary operator, followed by the identifier. The `??` unary operator, followed by the identifier, returns a list of names associated with that identifier for which a help page is available.

The call `library(help=circular)`, lists the functions and objects provided by the package named in the argument.

15.1 Your first R program

Much like Python, Perl and many other interpreted implementations, R can be run in an interactive mode, where code can be typed and immediately executed (with `"Hello world"` producing the obvious output).

Your first R program ought to read some data and plot it, not just print `"Hello World"`. The following program reads a file containing a single column of values and plots them, to produce figure 15.1:

```
the_data=read.csv("hello_world.csv")

plot(the_data)
```

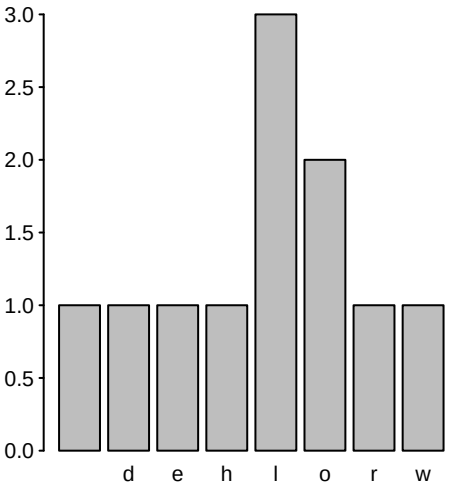


Figure 15.1: Plot produced by hello_world.R program. Github–Local

The `read.csv` function is included in the library that comes bundled with the base system (functions not included in this library have to be loaded using the `library` function, before they can be referenced; the package containing them may also need to be installed via the `install.packages` function) and has a variety of optional arguments (arguments can be omitted if the function definition includes default values for the corresponding parameter). Perhaps the most commonly used optional arguments are `sep` (the character used to separate, or delimit, values on a line, defaults to comma) and `header` (whether the contents of the first line should be treated as column names, default `TRUE`).

The value returned by `read.csv` has class `data.frame`, which might be thought of as a C `struct` type (it contains data only, there are no member functions as such).

The `plot` function attempts to produce a reasonable looking graphic of whatever data is passed, which for character data is a histogram of the number of occurrences. Users of R are not expected to be interested in manipulating low level details, and some effort is needed to get at the numeric values of characters.

There are a wide variety of options to change the appearance of `plot` output; these can be applied on each call to `plot`, or globally for every call (using the `par` function).

All objects in the current environment can be saved to a file using the `save` function, and a previously saved environment can be restored using the `load` function. When quiting an R session (by calling `q()`), the user is given the option of saving the current environment to a file named `.RData`; if a file of this name exists in the home directory, when R is started, its contents are automatically loaded.

15.2 Language overview

R is a language and an environment. Like Perl, it is defined by how its single implementation behaves, i.e., the software maintained by the R project.[1548]

R was designed, in the mid-1990s, to be largely compatible with S (a language, which like C, started life in the mid-1970s at Bell Labs). When S was created, Fortran was the dominant engineering language and the Fortran way of doing things had a strong impact on early design decisions, i.e., R does not have a C view of the world; for instance, it uses a row/column, rather than column/row ordering.

The designers of R have called it a functional language, and it does support a way of doing things that is most strongly associated with functional program languages (including making life cumbersome for developers wanting to assign to global variables).

The language also contains constructs that are said to make it an object-oriented language, and it certainly contains some features found in object-oriented languages. Object-oriented constructs were first added in the third iteration of the S language, and were more of an addition to the functional flavor of the language than a complete make-over. The primary OO feature usage is function overloading, when accessing functions from library packages.

Lateral thinking is often required to code a problem in R, using knowledge of functions contained in the base system, e.g., calling `order` to map a vector of strings to a unique vector of numbers.

Some data analysts write non-trivial programs in R, which means they have to deal with the testing and debugging issues experienced by users of other languages.

Base library support for debugging includes support for single stepping through a function, via the `debug` function, and setting breakpoints via the `setBreakpoint` function; package support includes: `RUnit` package for unit testing, and the `covr` package for measuring code coverage.

15.2.1 Differences between R and widely used languages

The following list describes language behaviors that are different from that encountered in other commonly used languages (Fortran developers will not consider some of these to be differences):

- there are no scalars, e.g., `2` is a vector containing one element and is equivalent to writing `c(2)`. Most unary and binary operators operate on all elements of a vector (among other things).

  Many operations that involve iterating over scalar values in other languages, e.g., adding two arrays, can be performed without explicit iteration in R, e.g., `c(1, 2) + c(3, 4)` has the value `c(4, 6)`,
- arrays start at one, not zero,
- matrices and data frames are indexed in row-column order (C-like languages use column-row order),
- case is significant in identifiers, e.g., `some_data` and `Some_data` are considered to refer to different objects,
- the period (dot, full stop, i.e., `.`) is a character than can occur in identifiers (e.g., `a.name`), it is not a separate token having the role of an operator,
- some language constructs, implemented via specific language syntax in other languages, are implemented as function calls in R, e.g., the functionality of `return` and `switch` is provided by function calls,
- assignment to a variable in an outer scope, from within a function, is specified using the **<<-** operator. The other assignment operators (e.g., **<-**, **->** and **=**) always assign to a local variable (creating one, if a variable of the given name does not already exist, in local scope),
- vectors/arrays/data.frames can be sliced to return a subset of the original,
- explicit support for `NA` (Not Available). This value denotes a number that may exist, but whose value is unknown. Operations involving `NA`, return `NA`, when the result value is not known because the value of `NA` is unknown, but will return a value when the result is independent of the value of `NA`, e.g., `NA || TRUE`,
- type conversion behavior may be driven by semantics rather than the underlying representation, e.g., `as.numeric("1") ==1` and `as.numeric("a")` returns `NA`.

The following R language features are found in commonly used languages:

- objects and functions come into existence, during program execution, when they are assigned a value, appear as a function parameter, or in more obscure ways (there is no mechanism for declaring any kind of identifier),
- the type of an object is the type of the value last assigned to it,
- decimal and hexadecimal literals have type numeric (literals starting with zero are not treated as octal literals; any leading zero is ignored) even if they look like integers, because they do not contain a decimal point. Some input functions, e.g., `read.csv`, consider a column to have integer type, if all its values can be represented as an integer,
- what most other languages consider to be a statement (i.e., something that does not return a value) R treats as an expression, e.g., if/for statements return a value.

15.2.2 Objects

Operations in R are performed on objects, sometimes known as variables. Objects are characterized by their names and their contents; with the contents in turn being characterized by attributes specifying the kind of data contained in the object.

The R type system has evolved over time, and includes the terms *mode* (a higher level view of the value representation, at least sometimes, than *typeof*, e.g., integer and double have mode numeric), *storage.mode* (a concept going back to the S language) and *typeof* (the underlying representation used by the C implementation of the language).

The `mode`, `storage.mode` and `typeof` functions return a string containing the respective information, e.g., `numeric`, `integer` or `function`.

The length of an object is the number of elements it contains, e.g., a two-dimensional array containing i rows and j columns, contains $i \times j$ elements. The `length` function returns the number of elements in its argument.

The assignment operator creates an object, with the object name being the left operand and its value and type being that of the right operand (left/right is reversed when the **->** assignment operator is used).

15.3 Operations on vectors

15.3.1 Creating a vector/array/matrix

An R vector can be thought of as a one-dimensional array. Vectors are indexed starting at 1 (not zero), and it is possible to added additional elements to a vector, but not remove an existing element.

```
x = 2                   # new vector containing one value
x = c(2, 4, 6, 8, 10)   # new vector containing five values
# new vector containing the contents of x and two values
x = c(x, 12, 14)
y = vector(length=5)    # new vector created by function call
y = 3:8                 # same as c(3, 4, 5, 6, 7, 8)
z = seq(from=3, to=13, by=3)      # create a sequence of values
# All elements converted to a common type
z = c(1, 2, "3")        # String has the greater conversion precedence
```

Multidimensional arrays can be created using the `array` function, with the common case of 2-dimensional arrays supported by a specific function, i.e., the `matrix` function.

```
> # create 3-dimensional array of 2 by 4 by 6, initialized to 0
> a3=array(0, c(2, 4, 6))
> matrix(c(1, 2, 3, 4, 5, 6), ncol=2) # default, populate in column order
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> # specify the number of rows and populate by row order
> matrix(c(1, 2, 3, 4, 5, 6), nrow=2, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> x = matrix(nrow=2, ncol=4) # create a new matrix
> y = c(1, 2, 3)
> z = as.matrix(y)  # convert a vector to a matrix
> str(y)
 num [1:3] 1 2 3
> str(z)
 num [1:3, 1] 1 2 3
```

15.3.2 Indexing

One or more elements of a vector/array/matrix can be accessed using indexing. Accesses to elements that do not exist return `NA`. Negative index values specify elements that are excluded from the returned value.

The zeroth element returns an empty vector.

```
> x = 10:19
> x[2]
[1] 11
> x[-1]                         # exclude element 1
[1] 11 12 13 14 15 16 17 18 19
> x[12]                         # there is no 12'th element
[1] NA
> x[12]=100                     # there is now
> x
 [1]  10  11  12  13  14  15  16  17  18  19  NA 100
```

Multiple elements can be returned by an indexing operation:

```
> x = 20:29
> x[c(2,5)]                     # elements 2 and 5
[1] 21 24
> y = x[x > 25]                 # all elements greater than 25
> y
[1] 26 27 28 29
```

```
> # The expression x > 25 returns a vector of boolean values
> i = x > 25
> i
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
> # an element of x is returned if the corresponding index is TRUE
> x[i]
[1] 26 27 28 29
```

Matrix indexing differs from vector indexing in that out-of-bounds accesses generate an error.

```
> x = matrix(c(1, 2, 3, 4, 5, 6), ncol=2)
> x[2, 1]
[1] 2
> # x[2, 3] need to be able to handle out-of-bounds subscripts in Sweave...
> x[, 1]
[1] 1 2 3
> x[1, ]
[1] 1 4
> x[3, ]=c(0, 9)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    0    9
> x=cbind(x, c(10, 11, 12))  # add a new column
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
> x=rbind(x, c(5, 10, 20))   # add a new row
> x
     [,1] [,2] [,3]
[1,]    1    4   10
[2,]    2    5   11
[3,]    0    9   12
[4,]    5   10   20
```

15.3.3 Lists

The difference between a list and a vector is that different elements in a list can have different modes (types) and existing elements can be removed.

```
> x = list(name="Bill", age=25, developer=TRUE)
> x
$name
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE
> x$name
[1] "Bill"
> x[[2]]
[1] 25
> x = list("Bill", 25, TRUE)
> x
[[1]]
[1] "Bill"

[[2]]
[1] 25

[[3]]
[1] TRUE
> y = unlist(x) # convert x to a vector, all elements are converted to strings
```

```
> y
[1] "Bill" "25"   "TRUE"
> x = list(name="Bill", age=25, developer=TRUE)
> x$sex="M"  # add a new element
> x
$name
[1] "Bill"

$age
[1] 25

$developer
[1] TRUE

$sex
[1] "M"
> x$age = NULL # remove an existing element
> x
$name
[1] "Bill"

$developer
[1] TRUE

$sex
[1] "M"
```

The **[[ ]]** operator returns a value, while **[ ]** returns a sublist (which has mode list).

15.3.4 Data frames

From the perspective of a programmer coming from another language, it may seem appropriate to think of a data frame as behaving like a matrix, and in some cases it can be treated in this way, e.g., when all columns have the same type, functions expecting a matrix argument may work. However, a better analogy is to think of it as an indexable structure type (where different members can have different types).

The `read.csv` functions reads a file containing columns, of potentially different types, and returns a data frame.

When indexing a data frame like a matrix, elements are accessed in row-column order (not the column-row order used in C-like languages). The following code selects all rows for which the `num` column is greater than 2.

```
> x = data.frame(num=c(1, 2, 3, 4), name=c("a", "b", "c", "d"))
> x
  num name
1   1    a
2   2    b
3   3    c
4   4    d
> # Middle two values of the column named num
> x$num[2:3]
[1] 2 3
> # Have to remember that rows are indexed first and also specify x twice
> x[x$num > 2, ]
  num name
3   3    c
4   4    d
> # Using the subset function removes the possibility of making common typo mistakes
> # No need to remember row/column order and only specify x once
> subset(x, num > 2)
  num name
3   3    c
4   4    d
```

If one or more columns contain character mode values (i.e., strings), `read.csv` will create, by default, a factor rather than a vector. The argument: `as.is=TRUE` causes strings to be represented as such.

Where ever possible, the code written for this book uses the `subset` function, rather than relying on correctly indexing a data frame.

15.3.5 Symbolic forms

An R expression can have a value which is its symbolic form.

```
exp = expression(x/(y+z))
eval(expr) # evaluate expression using the current values of x, y and z
```

Uses of expression values include: specifying which vectors in a table to plot in a graph, and including equations in graphs, for instance:

```
text(x, y, expression(p == over(1, 1+e^(alpha*x+beta))))
```

results in the following equation being displayed at the point (x, y): $p = \frac{1}{1+e^{\alpha x+\beta}}$

The `D` function takes an expression as its first argument, and based on the second argument, returns its derivative:

```
> D(expression(x/(y + z)^2), "z")
-(x * (2 * (y + z))/((y + z)^2)^2)
```

15.3.6 Factors and levels

When manipulating non-numeric values (e.g., names) statisticians sometimes find it convenient to map these values to integer values and manipulate them as integers. In programming terminology, a variable used to represent one or more of these integer values could be said to have a *factor* type, with the actual numeric values known as *levels* (a parallel can be drawn with the enumeration types found in C++ and C, except these assign names to integer values).

```
> factor(c("win", "win", "lose", "win", "lose", "lose"))
[1] win  win  lose win  lose lose
Levels: lose win
```

Some operations implicitly convert a sequence of values to a factor. For instance, `read.csv` will, by default, convert any column of string values to a factor; this conversion is a simplistic form of hashing, and (when a megabyte was considered a lot of memory) was once driven by the rationale of saving storage. These days the R implementation uses more sophisticated hashing, and we live with the consequences of historical baggage.

Operations of objects holding values represented as factors sometimes have surprising effects, for those unaware of how things used to be.

15.4 Operators

Operators in R have the same precedence rules as Fortran, which in some cases differ from the C precedence rules (which most commonly used languages now mimic). An example of this difference is: `!x ==y` which is equivalent to `!(x ==y)` in R, but in C-like languages is equivalent to `(!x) ==y` (if `x` and `y` have type boolean, there is no effective difference, but expressions such as: `!1 ==2` produce a different result).

A list of operators and their precedence can be obtained by typing `?Syntax`, at the R command line.

Within an expression operand evaluation is left to right, except assignment which evaluates the right operand and then the left.

In most cases, all elements of a vector are operated on by operators:

```
> c(5, 6) + 1
[1] 6 7
> c(1, 2) + c(3, 4)
[1] 4 6
```

```
> c(7, 8, 9, 10) + c(11, 12)
[1] 18 20 20 22
> c(0, 1) < c(1, 0)
[1]  TRUE FALSE
```

in the last two examples *recycling* occurs, that is the elements of the shorter vector are reused until all the elements of the longer vector have been operated on.

The **&&** and || operators differ from **&** and | in that they operate on just the first element of their operands, returning a vector containing one element, e.g., `c(0,1) && c(1,1)` returns the vector `FALSE`.

The base system includes a set of `bitw???` functions, that perform bitwise operations on their integer arguments; there is the `bitops` package.

Operators	Description
:: :::	access variables (right operand) in a name space (left operand)
$ @	component / slot extraction (member selection has lower precedence than subscripting in C-influenced languages)
[[[	array and list indexing
^ **	exponentiation (associates right to left)
- +	unary minus and plus
:	sequence operator
%any%	special operators (%% and %/% has the same precedence as * and / in C-influenced languages)
*** /**	multiply and divide
+ -	(binary) add and subtract
< > <= >= == !=	relational and equality (non-associative; equality has lower precedence in C-influenced languages)
!	negation (greater precedence than any binary operator in C-influenced languages)
& &&	and of all elements and the first element
| ||	or of all elements and the first element
~	as in formulae
-> ->>	local and global rightwards assignment
=	assignment (associates right to left)
<- <<-	local and global assignment (associates right to left)
?	help (unary and binary)

Table 15.1: R operators listed in precedence order.

The character used for exclusive-or in C-influenced languages, ^, is used for exponentiation in R; the `xor` function performs an exclusive-or of its operands. Like Fortran, R also supports the use of ** to denote exponentiation.

The `[` and `[[` operators differ by more than being array and list indexing. The result of the index `x[1]` has the same type as `x`, i.e., the operation preserves the type), while the result of `x[[1]]` is a simplified version of the type of `x` (if simplification is possible.[i]

```
> x = c(a = 1, b = 2)
> x[1]
a
1
> x[[1]]
[1] 1
> x = list(a = 1, b = 2)
> str(x[1])
List of 1
 $ a: num 1
> str(x[[1]])
 num 1
> x = matrix(1:4, nrow = 2)
> x[1, ]
[1] 1 3
> x[1, , drop = FALSE]
     [,1] [,2]
```

[i] Out-of-bounds handling is also different, but I'm sure readers' don't do that sort of thing.

```
[1,]    1    3
> # x[[1, ]] is not allowed
>
> df = data.frame(a = 1:2, b = 1:2)
> str(df[1])
'data.frame':   2 obs. of  1 variable:
 $ a: int  1 2
> str(df[[1]])
 int [1:2] 1 2
> str(df[, "a", drop = FALSE])
'data.frame':   2 obs. of  1 variable:
 $ a: int  1 2
> str(df[, "a"])
 int [1:2] 1 2
```

15.4.1 Testing for equality

In addition to the equality operators, the base system includes two equality related functions, `identical` and `all.equal`.

```
> x = 1:5  ;  y = 1:5
> x == y  # Return the result of equality test for each corresponding element
[1] TRUE TRUE TRUE TRUE TRUE
> identical(x, y) # Return a single value denoting exact equality
[1] TRUE
> 1L == 1   # 1L is stored internally as an integer, 1 is stored as a double
[1] TRUE
> identical(1L, 1)  # identical requires the stored type be the same
[1] FALSE
> 0.9 == (1.1 - 0.2)  # could be affected by lack of precision
[1] FALSE
> all.equal(0.9, 1.1 - 0.2)  # do a fuzzy compare
[1] TRUE
> all.equal(0.9, 1.1 - 0.2, tolerance=0)  # find out much how fuzz there is
[1] "Mean relative difference: 1.233581e-16"
```

The default tolerance used by the `all.equal` function is `.Machine$double.eps^0.5`.

Comparisons against `NA` always returns `NA`. The `is.na` function can be used to check for this quantity; the `anyNA` function returns `TRUE`, if its argument contains at least one `NA`.

15.4.2 Assignment

Four of the ways of assigning a value to a variable in R include:

```
x <- 3  # Operator used by people who follow the herd
x <<- 3 # Assigns to the x at global scope

3 -> x  # Rarely encountered outside descriptions of the language

x = 3   # Supported since R version 1.4
```

Many R books and articles use the two characters **<-**.[ii] Developers are used to seeing the **=** token, and with nothing other than conformity to existing R usage to recommend the alternative, the assignment token that developers are already very familiar with, is used in this book.

There is one context where **=** does not behave like normal assignment. R supports the use of parameter names in arguments to function calls, to explicitly specify that a named parameter is to be assigned a given value. In the context of a function argument list, the left operand of **=** is treated as the name of a parameter and the right operand as the value to be assigned. An error is flagged, if the function definition does not have a parameter having the specified name.

[ii] The developers of the S language used terminals that had a single key for this symbol sequence.

```
func = function (a, b, c) a + b * c

func(2, 3, 9)
func(c=9, b=3, a=2)

func(d=3, 4, 5)  # no parameter named d, an error is raised

# use <- if the intent is to assign to d and pass this value as an argument
func(d<-3, 4, 5)
```

15.5 The R type (mode) system

R supports values having the following basic types (R also has the concept of *mode*, which is based on semantics rather than underlying representation, e.g., the `mode` function returns `numeric` where `typeof` returns either `integer` or `double`):

- `NULL`:
- raw: essentially uninterpreted byte values,
- logical: holds one of the values: `TRUE`, `FALSE`, `T` or `F`. The conversion `as.logical(any_non_zero_value)` returns `TRUE`,
- character: what many other languages call a string type,
- integer: the only integer type, contains 32 bits (`NA` is represented using the most negative value, so this value is not available as an integer; trying to generate this, or any other value outside the representable value of a 32-bit integer, will result in a value having a double type),
- double: the only floating-point type, contains 64 bits. Can exactly represent all 32-bit integers,
- complex: contains a real and imaginary double type,

An object may be reported to have one of these basic types, but it may actually be a vector or array of this type.

More complicated types may be created, such as lists, data frames, etc.

15.5.1 Converting the type (mode) of a value

It is often possible to convert the mode (type) of a value by calling the `as.some_mode` function, where `some_mode` is the name of a mode, e.g., `integer`. If a conversion fails, `NA` is returned.

Conversion precedence

`NULL < raw < logical < integer < real < complex < character < list < expression`

15.6 Statements

R contains the usual language constructs that look like statements, but they can behave like expressions:

- **function**: defines a function, whose value has to be assigned to an object:

  `f=function(p1, p2) {return(p1+p2)}`
- blocks of code are bracketed using the punctuation pair: **{** and **}**,
- **;** (semicolon) is required to delimit multiple expressions on the same line, but is otherwise optional,
- **if**: which takes an optional **else** arm (there is no **then** keyword, but there is an `ifthenelse` function),
- **for**: which has the form `for (i in x)`, where `x` is a vector (such as `1:10`),
- **while** and **repeat** loops are available,

- loops may be terminated using the **break** keyword or the `break` function, and may be continued at the next iteration using the **next** keyword or `next` function,
- `return` is a function: `return(1+return(1))` returns the value `1`,
- `switch` is a function.

15.7 Defining a function

```
> g=1  # a global variable
> f = function(p1, p2) # define a function and assign it to f
+ {
+ l=g  # Value access, check lexical and dynamic scope for g
+ g=2  # Assignment: only check local scope, if no variable exists, create one
+
+ m=h  # h is dynamically in scope
+
+ return(return(1)+1) # return is a function call
+ }
> h=2  # another global variable
> f(1, 2)
[1] 1
> g
[1] 1
> h=3 # At global scope, so must be global variable
```

Argument evaluation is lazy, that is, they are evaluated the first time their value is required.

The `...` token (three dots) specifies that a variable number of unknown arguments may be passed.

```
unk_args=function(...)
{
a=list(...) # Convert any arguments passed to a list of values
# Access the list of values in a
}
```

15.8 Commonly used functions

Technically every operation is a function call (so `'+'(1, 2)` and `1+2` are equivalent), but not all function calls have equivalent operator tokens.

```
> x = 1:10
> if (any(x > 7)) print("At least one value greater than 7")
[1] "At least one value greater than 7"
> if (all(x > 0)) print("All values greater than zero")
[1] "All values greater than zero"
> rep(1:2, 3)
[1] 1 2 1 2 1 2
```

- `head`/`tail` mimics the behavior of the Unix `head`/`tail` programs,
- `length` returns the number of elements in its vector argument,
- `nrow`/`ncol` return the number of rows/columns in the data frame argument (`NROW`/`NCOL` gracefully handle vector arguments),
- `order` returns a vector containing an index in to the argument in the order needed to sort the argument values,
- `str` lists the columns in a variable, along with their type and the first few values in each row; it provides a quick way of verifying that columns have the expected type.
- `which` returns a vector of values containing the index of the argument values that are true,
- `methods`: list functions overloaded on the argument name
- `installed.packages`: list all installed packages
- `ls`: lists variables that exist in the current environment,
- `system.time`, `proc.time`: cpu time used, and the real, and cpu time of the currently running R process.

15.9 Input/Output

Functions are available for reading data having a variety of formats (e.g., comma separated values), from all the common data sources, e.g., files, databases, web pages. In some cases the contents of a compressed file will be automatically uncompressed before reading. In the case of files, all the data contained in the file is often read, and returned as a single object.

Many functions try to automatically deduce the datatype of the data read, e.g., whether it is integer, real, character sequence, etc. Sometimes the datatype selected is not correct, and work has to be done to ensure the data is treated as having the desired type; the `read.csv` function bases its decision on the type of each column, by analysing the first 6, or so, lines of the file.

Some functions in the base system, e.g., `read.csv`, convert columns containing string values to factors, by default; the original intent was, presumably, to reduce the storage needed to hold the data. A column of factors, as a type, does not always behave the same as a column of strings and this default conversion behavior is often a liability. Using the argument `as.is=TRUE` prevents values being converted to factors (it is used in all of this book's example code).

```
data=read.csv("measurements.csv.gz", as.is=TRUE) # file will be uncompressed

data=read.csv("measurements.csv", sep="|", as.is=TRUE) # change separator

data=read.csv("https://github.com/Derek-Jones/ESEUR-code-data/blob/master/benchmark/MST
```

The first line of the input file is assumed to denote the name of each column, specifying `header=FALSE` switches off this default behavior.

All characters on an input line after, and including, the comment character, #, are ignored (various options interact with this behavior, including the `comment.char` option which can be used to change the character used).

The `foreign` package supports the reading (and some writing) of data stored in some of the binary file formats used by other applications, e.g., `read.spss`.

If data is not already in a form that can be easily processed by R, it may be simpler to convert it using a language or tool that you are already familiar with, rather than using R.

The R environment includes a simple spreadsheet like editor for manual data entry and modifying existing data.

```
scores = edit(scores) # invoke built-in spreadsheet like editor
```

There are corresponding `write` functions for many of the `read` functions, e.g., `write.csv`.

The `print` function performs relatively simple formatted output (the `format` function can be used to create more sophisticated formatting, that can then be output); the `cat` function performs relatively little formatting, but is more flexible, and in particular does not terminate its output with a newline; the `sink` function can be used to specify an alternative location to write console output.

15.9.1 Graphical output

There are probably more functions supporting graphical output, in R, than textual output. Perhaps the most commonly used graphical output function is `plot`. This function often does a good job of producing a reasonable graphical representation of the data. Overloaded versions of this function are often provided by packages, to plot data having a particular class created by the package.

By default, graphical output is sent to the console device; this behavior can be overridden to produce a file having a particular format, e.g., `pdf`, `jpeg`, `png` and `pictex`. The list of supported output devices varies across the operating systems on which R runs.

The behavior of the `plot` function can be influenced by previous calls to the `par` function, which set configurable options.

Various packages providing graphical output are available, with the `ggplot` package probably being the most commonly used by frequent R users.

15.10 Non-statistical uses of R

While the target of R's domain specialised functionality is statistical data analysis, there are other application domains where this functionality could be useful (but may not warrant effort needed to learn R).

A variety of functions designed for manipulating the rows and columns of delimited data files are available; see Github–Rlang/Top500.R.

A technique for spotting whether a file contains compressed data (e.g., a virus hidden in a script by compressing it to look like a jumble of numbers) is to plot the fraction of distinct values appearing in successive, fixed size, blocks; see figure 15.2. Compressed data is likely to contain an approximately uniform distribution of byte values (compression is achieved by reducing apparent information content), your mileage may vary between compression methods.

The following code reads a pdf file, applies a sliding window to the data and then plots the fraction of distinct values in each window (at a given offset).

```
window_width=256 # if less than 256, divisor has to change in plot call

plot_unique=function(filename)
{
t=readBin(filename, what="raw", n=1e7)

# Sliding the window over every point is too much overhead
cnt_points=seq(1, length(t)-window_width, 5)

u=sapply(cnt_points, function(X) length(unique(t[X:(X+window_width)])))
plot(u/256, type="l", xlab="Offset", ylab="Fraction Unique", las=1)

return(u)
}

dummy=plot_unique("http://www.coding-guidelines.com/R_code/requirements.tgz")
```

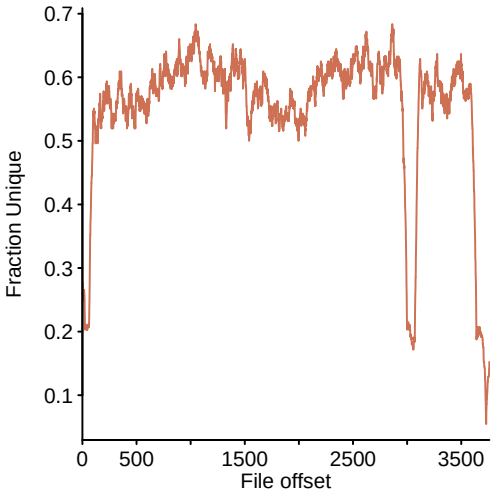


Figure 15.2: The unique bytes per window (256 bytes wide) of a pdf file. Github–Local

15.11 Very large datasets

While most existing software engineering datasets tend to be small, exceptions may occur from time to time. A variety of techniques are available for handling large datasets, including:

- the `bigmemory` package provides software defined memory management, e.g., swapping data between memory and main storage. The `bigtabulate` package, along with other `big???` packages contain functions that perform commonly used operations on this data.
- the `data.table` package extends data.frames to support up to 100G of storage,

References

1. 7Digital, Ltd. 7digital development team statistical analysis report april 2011-2012. blog article, July 2012. http://www.7digital.com. 137, 244, 329, 330, 380
2. J. T. Abbott, J. L. Austerweil, and T. L. Griffiths. Random walks on semantic networks can resemble optimal foraging. *Psychological Review*, 122(3):558–569, July 2015. 33
3. T. Abdel-Hamid and S. E. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice-Hall, Inc, 1991. 128, 356
4. D. Aboody and B. Lev. The value relevance of intangibles: The case of software capitalization. *Journal of Accounting Research*, 36:161–191, 1998. 84
5. ACAA Technical Agent. Ada conformity assessment test suite (ACATS). organization website, Jan. 2018. http://www.ada-auth.org/acats.html. 171
6. A. Adamatzky. A brief history of liquid computers. *Philosophical Transactions of The Royal Society B*, 374(1774), June 2019. 1
7. E. N. Adams. Optimizing preventive service of software products. *IBM Journal of Research and Development*, 28(1):2–14, Jan. 1984. 157, 159
8. J. Adams. *Risk and Freedom: The record of road safety regulation*. Transport Publishing Projects, 1985. 53
9. B. Adelson. Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9(4):422–433, July 1981. 34, 35
10. ADPE Selection Office. Federal COBOL compiler testing service compiler validation request information. Report No FCCTS/TR-77/05, Department of the Navy, USA, May 1977. 171
11. J. Agar. *The Government Machine A Revolutionary History of the Computer*. The MIT Press, 2003. 92
12. R. Agarwal and M. Gort. The evolution of markets and entry, exit and survival of firms. *The Review of Economics and Statistics*, 78(3):489–498, Aug. 1996. 99
13. P. J. Ågerfalk. Insufficient theoretical contribution: a conclusive rationale for rejection? *European Journal of Information Systems*, 23(6):593–599, Nov. 2014. 8
14. A. Aghayev and P. Desnoyers. Skylight-A window on shingled disk operation. In *13th USENIX Conference on File and Storage Technologies*, FAST'15, pages 135–149, Feb. 2015. 370
15. O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 spot instance pricing. *ACM Transactions on Economics and Computation*, 1(3), Sept. 2013. 62, 63
16. N. Agrawal. *Representative, reproducible, and practical benchmarking of file and storage systems*. PhD thesis, University of Wisconsin-Madison, 2009. 362
17. N. Agrawal, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Generating realistic impressions for file-system benchmarking. *ACM Transactions on Storage*, 5(4):125–138, Dec. 2009. 226
18. N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *ACM Transactions on Storage*, 3(3):31–45, Oct. 2007. 246
19. M. Ahasanuzzaman, S. Hassan, C.-P. Bezemer, and A. E. Hassan. A longitudinal study of popular ad libraries in the Google Play Store. *Empirical Software Engineering*, 25(1):824–858, Jan. 2020. 93
20. N. Ahmad. Measuring investment in software. OECD Science, Technology and Industry Working Papers 2003/06, OECD, May 2003. 82
21. N. Ahmad, C. Aspden, and OECD Task Force on R&D and Other Intellectual Property Products. *Handbook on Deriving Capital Measures of Intellectual Property Products*. OECD Publishing, 2010. 82
22. J. J. Ahonen and P. Savolainen. Software engineering projects may fail before they are started: Post-mortem analysis of five cancelled projects. *Journal of Systems and Software*, 83(11):2175–2187, Nov. 2010. 120
23. J. J. Ahonen, P. Savolainen, H. Merikoski, and J. Nevalainen. Reported project management effort, project size, and contract type. *The Journal of Systems and Software*, 109(C):205–213, Nov. 2015. 125
24. S. Ajami, Y. Woodbridge, and D. G. Feitelson. Syntax, predicates, idioms – What really affects code complexity? *Empirical Software Engineering*, 24(1):287–328, Feb. 2019. 181
25. F. Akdemir and F. A. Kirmani. Synergy: A synthetic study on teams. Thesis (m.s.), Umeå School of Business, July-Sept. 2008. 80, 81
26. G. A. Akerlof. The market for "Lemons": Quality uncertainty and the market mechanism. *The Quarterly Journal of Economics*, 84(3):488–500, Aug. 1970. 77
27. K. Akita, S. Itagaki, Y. Masawa, M. Nonaka, T. Hatani, K. Hattori, S. Morisaki, Y. Yanagida, T. Takaya, T. Furuyama, and O. Takashi. *Software Development Data White paper 2012-2013*. SEC BOOKS, 2012. 113, 119, 120
28. A. Akshintala, B. Jain, C.-C. Tsai, M. Ferdman, and D. E. Porter. x86-64 instruction usage among C/C++ applications. In *12th ACM International Systems and Storage Conference*, SYSTOR '19, pages 68–79, June 2019. 203
29. H. A. A. Al-Mutawa. On the classification of cyclic dependencies in Java programs. Thesis (m.s.), Massey University, New Zealand, 2013. 210
30. H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, and J. Raman. Analysis of safety-critical computer failures in medical devices. *IEEE Security & Privacy*, 11(4):14–26, July 2013. 294, 298
31. N. Ali, Z. Sharafi, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study on requirements traceability using eye-tracking. In *28th IEEE International Conference on Software Maintenance*, ICSM'12, pages 191–200, Sept. 2012. 29
32. T. Allee and M. Elsig. Are the contents of international treaties copied-and-pasted? Evidence from preferential trade agreements. Working Paper No. 8, World Trade Institute, Aug. 2016. 81
33. E. J. Allen, P. M. Dechow, D. G. Pope, and G. Wu. Reference-dependent preferences: Evidence from marathon runners. *Management Science*, 63(6):1657–1672, June 2017. 134, 135
34. R. Allen and K. Kennedy. *Optimizing Compilers for Modern Architecture*. Morgan Kaufmann Publishers, Mar. 2002. 205
35. R. C. Allen. The British industrial revolution in global perspective: How commerce created the industrial revolution and modern economic growth. Nuffield College, Oxford, 2006. 92
36. T. J. Allen and R. Katz. The dual ladder: Motivational solution or managerial delusion? Working Paper 1692-85, Massachusetts Institute of Technology, Sloan School of Management, Aug. 1985. 108
37. K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon. AndroZoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories*, MSR'16, pages 468–471, May 2016. 4
38. L. Allodi. Economic factors of vulnerability trade and exploitation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS'17, pages 1483–1499, Oct.-Nov. 2017. 154
39. L. Allodi and F. Massacci. A preliminary analysis of vulnerability scores for attacks in wild: The EKITS and SYM datasets. In *Proceedings of the 2012 ACM Workshop on Building analysis datasets and gathering experience returns for security*, DABGERS'12, pages 17–24, Oct. 2012. 154
40. D. A. Almeida, G. C. Murphy, G. Wilson, and M. Hoye. Do software developers understand open source licenses? In *25th IEEE International Conference on Program Comprehension*, ICPC'17, pages 1–11, May 2017. 69
41. M. G. Almiron, E. S. Almeida, and M. N. Miranda. The reliability of statistical functions in four software packages freely used in numerical computation. *Brazilian Journal of Probability and Statistics*, 23(2):107–119, 2009. 15
42. M. G. Almiron, B. Lopes, A. L. C. Oliveira, A. C. Medeiros, and A. C. Frery. On the numerical accuracy of spreadsheets. *Journal of Statistics*, 34(4):1–29, Apr. 2010. 15
43. A. Almossawi. How maintainable is the Firefox codebase? personal website, May 2013. http://almossawi.com/firefox/prose. 185
44. W. H. Alsup. ORACLE AMERICA, INC., plaintiff-appellant v. GOOGLE LLC, defendant-cross-appellant. Decision 3:10-cv-03561-WHA, United States District Court for the Northern District of California, Mar. 2018. 113

45. L. E. Alteneder. The learning curve in solving a jig-saw puzzle: A teaching device. *Journal of Educational Psychology*, 26(3):231–232, Mar. 1935. 36

46. E. M. Altmann. *Episodic Memory for External Information*. PhD thesis, Carnegie Mellon University, Aug. 1996. 30

47. E. M. Altmann. Functional decay of memory for tasks. *Psychological Research*, 66(4):287–297, 2002. 26

48. E. M. Altmann, J. G. Trafton, and D. Z. Hambrick. Effects of interruption length on procedural errors. *Journal of Experimental Psychology: Applied*, 23(2):216–229, June 2017. 33

49. H. Aman, S. Amasaki, T. Yokogawa, and M. Kawahara. A survival analysis of source files modified by new developers. In *International Conference on Product-Focused Software Process Improvement*, PROFES 2017, pages 80–88, Nov.-Dec. 2017. 163

50. Amazon, Inc. Amazon ec2 service level agreement. https://aws.amazon.com/ec2/sla, June 2013. 374

51. S. Ambler. IT project success survey results. http://www.ambysoft.com/surveys, 2017. 120

52. J. M. Amiri and V. V. K. Padmanabhuni. A comprehensive evaluation of conversion approaches for different function points. Thesis (m.s.), Blekinge Institute of Technology, Sweden, Sept. 2011. 294, 295

53. L. An, O. Mlouki, F. Khomh, and G. Antoniol. Stack Overflow: A code laundering platform? In *eprint arXiv:cs.SE/1703.03897*, Mar. 2017. 82

54. B. C. D. Anda, D. I. K. Sjøberg, and A. Mockus. Variability and reproducibility in software engineering: A study of four companies that developed the same system. *IEEE Transactions on Software Engineering*, 35(3):407–429, May 2009. 120, 130

55. C. Anderson, J. A. D. Hildreth, and L. Howland. Is the desire for status a fundamental human motive? A review of the empirical literature. *Psychological Bulletin*, 141(3):574–601, May 2015. 74

56. D. Anderson. Modeling and analysis of SQL queries in PHP systems. Thesis (m.s.), Department of Computer Science, East Carolina University, Apr. 2018. 202

57. J. R. Anderson. *Learning and Memory: An Integrated Approach*. John Wiley & Sons, Inc, second edition, 2000. 40

58. J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin. An integrated theory of the mind. *Psychological Review*, 111(4):1036–1060, Oct. 2007. 21

59. J. R. Anderson and R. Milson. Human memory: An adaptive perspective. *Psychological Review*, 96(4):703–719, Oct. 1989. 35

60. M. L. Anderson. Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, 33(4):245–313, Apr. 2010. 19

61. D. Andriesse, X. Chen, V. van der Veen, A. Slowinska, and H. Bos. An in-depth analysis of disassembly on full-scale x86/x64 binaries. In *Proceedings of the 25th USENIX Security Symposium*, SEC'16, pages 583–600, Aug. 2016. 172

62. J. Annett. Subjective rating scales: science or art? *Ergonomics*, 45(12):966–987, 2002. 376

63. A. Ansar. 'AppStore secrets' (What we've learned from 30,000,000 downloads). Presentation, pinch media, 2009. 109

64. F. J. Anscombe. Graphs in statistical analysis. *The American Statistician*, 27(1):17–21, Feb. 1973. 295, 296

65. ANSI X3.9. *American National Standard programming language FORTRAN*. American National Standards Institute, inc., Nov. 1978. 202

66. Ž. Antolić. Fault slip through measurement process implementation in CPP software verification. In *International Conference on Business Intelligence Systems*, miproBIS 2007, May 2007. 168

67. K. Aoki and M. W. Feldman. Evolution of learning strategies in temporally and spatially variable environments: A review of theory. *Theoretical Population Biology*, 91:3–19, Feb. 2014. 75

68. J. Aranda. Anchoring and adjustment in software estimation. Thesis (m.s.), Graduate Department of Computer Science, University of Toronto, 2005. 127

69. L. Argote, C. A. Insko, N. Yovetich, and A. A. Romero. Group learning curves: The effects of turnover and task complexity on group performance. *Journal of Applied Social Psychology*, 25(6):512–529, Mar. 1995. 77

70. H. R. Arkes, R. M. Dawes, and C. Christensen. Factors influencing the use of a decision rule in a probabilistic task. *Organizational Behavior and Human Decision Processes*, 37:93–110, 1986. 56

71. P. Armer. SHARE – A eulogy to cooperative effort. Technical Report P-969, The RAND Corporation, Oct. 1956. 68, 112

72. J. S. Armstrong. The seer-sucker theory: The value of experts in forecasting. *Technology Review*, pages 16–24, June-July 1980. 39

73. V. Arnaoudova, L. M. Eshkevari, M. Di Penta, R. Oliveto, G. Antoniol, and Y.-G. Guéhéneuc. REPENT: Analyzing the nature of identifier renamings. *IEEE Transactions on Software Engineering*, 40(5):502–532, May 2014. 197

74. J. Arndt. *Matters Computational: Ideas, Algorithms, Source Code*. Springer, 2010. 206

75. T. B. Arnold and J. W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, 3(2):34–39, Dec. 2011. 240

76. A. Arora, S. Belenzon, A. Patacconi, and J. Suh. The changing structure of American innovation: Some cautionary remarks for economic growth. Working Paper No. 25893, National Bureau of Economic Research, Aug. 2019. 10

77. A. Arora and A. Gambardella. *From Underdogs to Tigers: The Rise and Growth of the Software Industry in Brazil, China, India, Ireland, and Israel*. Oxford University Press, Mar. 2005. 62

78. A. Arora, R. Krishnan, R. Telang, and Y. Yang. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Information Systems Research*, 21(1):115–132, Mar. 2010. 154, 339, 341, 378

79. W. B. Arthur. Competing technologies, increasing returns, and lock-in by historical events. *The Economic Journal*, 99(394):116–131, Mar. 1989. 101

80. W. B. Arthur. *Increasing Returns and Path Dependency in the Economy*. The University of Michigan Press, 1994. 100, 101

81. S. E. Asch. Studies of independence and conformity: A minority of one against a unanimous majority. *Psychological Monographs: General and Applied*, 70(9):1–70, 1956. 54

82. A. H. Ashouri, W. Killian, J. Cavazos, G. Palermo, and C. Silvano. A survey on compiler autotuning using machine learning. In *eprint arXiv:cs.PL/1801.04405*, Mar. 2018. 178

83. T. A. Åstebro, S. A. Jeffrey, and G. K. Adomdza. Inventor perseverance after being told to quit: The role of cognitive biases. *Journal of Behavioral Decision Making*, 20(3):253–272, Apr. 2007. 55

84. S. Atkinson and G. Benefield. Software development: Why the traditional contract model is not fit for purpose. In *46th Hawaii International Conference on System Sciences*, HICSS, pages 4842–4851, Jan. 2013. 124

85. V. Atlidakis, J. Andrus, R. Geambasu, D. Mitropoulos, and J. Nieh. POSIX abstractions in modern operating systems: The old, the new, and the missing. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, page 19, Apr. 2016. 115, 116

86. Audit Scotland. i6: a review. Report, Audit Scotland, Mar. 2017. 120

87. Auerbach. Auerbach guide to time sharing. Computer technology report, Auerbach Publishers Inc., Jan. 1973. 105

88. N. R. Augustine. *Augustine's Laws*. American Institute of Aeronautics and Astronautics, Inc, sixth edition, 1997. 176

89. R. Auler and E. Borin. A LLVM just-in-time compilation cost analysis. Technical Report IC-13-13, Instituto de Computação Universidade Estadual de Campinas, May 2013. 180, 181

90. P. C. Austin. A tutorial on multilevel survival analysis: Methods, models and applications. *International Statistical Review*, 85(2):185–203, Aug. 2017. 340

91. R. D. Austin. The effects of time pressure on quality in software development: An agency model. *Information Systems Research*, 12(2):195–207, June 2001. 79

92. AUTOSAR. *Guidelines for the use of the C++14 language in critical and safety-related systems*. AUTOSAR, 839 edition, Mar. 2017. 153

93. J. L. Autran, D. Munteanu, P. Roche, and G. Gasiot. Real-time soft-error rate measurements: A review. *Microelectronics Reliability*, 54(8):1455–1476, Aug. 2014. 166

94. J.-L. Autran, S. Semikh, D. Munteanu, S. Serre, G. Gasiot, and P. Roche. Soft-error rate of advanced SRAM memories: Modeling and monte carlo simulation. In M. Andriychuk, editor, *Numerical Simulation – From Theory to Industry*, chapter 15, pages 309–336. InTech, Sept. 2012. 166

95. G. Avelino, L. Passos, A. Hora, and M. T. Valente. Measuring and analyzing code authorship in 1+118 open source projects. *Science of Computer Programming*, 176:14–32, May 2019. 141

96. E. Avidan. The significance of method parameters and local variables as beacons for comprehension: An empirical study. Thesis (m.s.), The Hebrew University of Jerusalem, Nov. 2016. 196

97. P. Azoulay, C. Fons-Rosen, and J. S. G. Zivin. Does science advance one funeral at a time? Working Paper No. 21788, National Bureau of Economic Research, USA, Dec. 2015. 10

98. R. H. Baayen, P. Milin, and M. Ramscar. Frequency in lexical processing. *Aphasiology*, 30(11):1174–1220, Mar. 2016. 196

99. C. Babbage, ESQ. *Reflections on the Decline of Science in England, and on Some of its Causes*. B. Fellows, Ludgate Street; and J. Booth, Duke Street, 1830. 11

100. C. Babbage, ESQ. *On the Economy of Machinery and Manufactures*. Charles Knight, Pall Mall East, 1832. 61

101. V. Babka. *Improving Accuracy of Software Performance Models on Multicore Platforms with Shared Caches*. PhD thesis, Faculty of Mathematics and Physics, Charles University in Prague, Oct. 2012. 372

102. V. Babka and P. Tůma. Investigating cache parameters of x86 family processors. In *Proceedings of the 2009 SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, pages 77–96, Jan. 2009. 372, 373

103. D. Baccarini, G. Salm, and P. E. D. Love. Management of risks in information technology projects. *Industrial Management & Data Systems*, 104(4):286–295, 2004. 130

104. A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE'13, pages 712–721, May 2013. 169

105. A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2010, pages 97–106, Nov. 2010. 152

106. A. Back and E. Westman. Comparing programming languages in Google code jam. Thesis (m.s.), Department of Computer Science and Engineering, Chalmers University of Technology, 2017. 197

107. J. Backus. The history of FORTRAN I, II, and III. *SIGPLAN Notices*, 13(8):165–180, 1978. 112

108. J. Backus. Programming in America in the 1950s– some personal impressions. In N. Metropolis, J. Howlett, and G.-C. Rota, editors, *A History of Computing in the Twentieth Century*, pages 125–135. Academic Press, Feb. 1981. 112, 113

109. J. W. Backus, R. J. Beeber, S. Best, R. Goldberg, H. L. Herrick, R. A. Hughes, L. B. Mitchell, R. A. Nelson, R. Nutt, D. Sayre, P. B. Sheridan, H. Stern, and I. Ziller. *The FORTRAN Automatic Coding System for the IBM 704 EDPM: Programmer's Reference Manual*. International Business Machines Corporation, 590 Madison Avenue, New York 22, N.Y., Oct. 1956. 113

110. A. Bacon, S. Handley, and S. Newstead. Individual differences in strategies for syllogistic reasoning. *Thinking & Reasoning*, 9(2):133–168, 2003. 45

111. A. Baddeley. Working memory. In A. Baddeley, M. W. Eysenck, and M. Anderson, editors, *Memory*, chapter 3, pages 41–69. Psychology Press, Feb. 2009. 31

112. A. Baddeley. Working memory: Theories, models, and controversies. *Annual Review of Psychology*, 63:1–29, Sept. 2012. 31

113. A. D. Baddeley, N. Thomson, and M. Buchanan. Word length and the structure of short-term memory. *Journal of Verbal Learning and Verbal Behavior*, 14(6):575–589, Dec. 1975. 31, 362

114. M. Bagherzadeh, N. Kahani, C.-P. Bezemer, A. E. Hassan, J. Dingel, and J. R. Cordy. Analyzing a decade of Linux system calls. *Empirical Software Engineering*, 23(3):1519–1551, June 2018. 116

115. J. N. Bailenson, M. S. Shum, S. Atran, D. L. Medin, and J. D. Coley. A bird's eye view: biological categorization and reasoning within and across cultures. *Cognition*, 84:1–53, 2002. 43

116. D. H. Bailey. Misleading performance reporting in the supercomputer field. Technical Report RNR-92-005, Numerical Aerodynamic Simulation Division, NASA Ames Research Center, Dec. 1992. 366

117. S. Baily, R. Gilbertson, and E. Straub. Modular multimode radar (CMMR) software acquisition study. Technical Report 2302-01-1-2291, ARINC Research Corporation, Mar. 1981. 107

118. E. Bainomugisha, A. L. Carreton, T. van Cutsem, S. Mostinckx, and W. de Meuter. A survey on reactive programming. *ACM Computing Surveys*, 45(4):52, Aug. 2013. 179

119. P. Bajari, S. Tadelis, and S. Houghton. Bidding for incomplete contracts: An empirical analysis of adaptation costs. *American Economic Review*, 104(4):1288–1319, Oct. 2011. 123

120. S. S. Bajwa, X. Wang, A. N. Duc, and P. Abrahamsson. Failures to be celebrated: an analysis of major pivots of software startups. In *eprint arXiv:cs.SE/1710.04037*, Oct. 2017. 130

121. A. H. Baker, D. M. Hammerling, M. N. Levy, H. Xu, J. M. Dennis, B. E. Eaton, J. Edwards, C. Hannay, S. A. Mickelson, R. B. Neale, D. Nychka, J. Shollenberger, J. Tribbia, M. Vertenstein, and D. Williamson. A new ensemble-based consistency test for the Community Earth System Model (pyCECT v1.0). *Geoscientific Model Development*, 8:2829–2840, Sept. 2015. 151

122. F. T. Baker. Chief programmer team management of production programming. *IBM Systems Journal*, 11(1):56–73, 1972. 73, 141

123. M. Bakkaloglu, J. J. Wylie, C. Wang, and G. R. Ganger. On correlated failures in survivable storage systems. Technical Report CMU-CS-02-129, Carnegie Mellon University, May 2002. 278

124. B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal. Accurate characterization of the variability in power consumption in modern mobile processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, Oct. 2012. 320, 321

125. M. Baldwin. Scientific autonomy, public accountability, and the rise of "peer review" in the Cold war United States. *Isis*, 109(3):538–558, Sept. 2018. 11

126. T. Ball and J. R. Larus. Branch prediction for free. Technical Report #1137, Computer Sciences Department, University of Wisconsin–Madison, Feb. 1993. 203

127. S. Baltes and S. Diehl. Usage and attribution of Stack Overflow code snippets in GitHub projects. In *eprint arXiv:cs.SE/1802.02938*, Feb. 2018. 82

128. S. Baltes and P. Ralph. Sampling in software engineering research: A critical review and guidelines. *ACM Transactions on Software Engineering and Methodology*, ???(???):???, Apr. 2020. 9

129. N. Banerjee, A. Rahmati, M. D. Corner, S. Rollins, and L. Zhong. Users and batteries: Interactions and adaptive energy management in mobile systems. In *International Conference on Ubiquitous Computing*, UbiComp 2007, pages 217–237, Sept. 2007. 95

130. P. Banyard and N. Hunt. Something missing? *The Psychologist*, 13(2):68–71, 2000. 21

131. L. Bao, Z. Xing, X. Xia, D. Lo, and S. Li. Who will leave the company?: A large-scale industry study of developer turnover by mining monthly work report. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR'17, pages 170–181, May 2017. 141

132. J. H. Barkow, L. Cosmides, and J. Tooby. *The Adapted Mind: Evolutionary Psychology and the Generation of Culture*. Oxford University Press, 1992. 20

133. W. P. Barnett. *The Red Queen among Organizations: How competitiveness evolves*. Princeton University Press, 2008. 5, 95

134. A. Baronchelli, V. Loreto, and A. Puglisi. Individual biases, cultural evolution, and the statistical nature of language universals: The case of colour naming systems. *PLoS ONE*, 10(5):e0125019, May 2015. 201

135. D. R. Barrett. *World Christian Encyclopedia: A Comparative Survey of Churches and Religions in the Modern World AD 1900-2000*. Oxford University Press, 1982. 96

136. E. Barrett, C. F. Bolz-Tereick, R. Killick, S. Mount, and L. Tratt. Virtual machine warmup blows hot and cold. In *eprint arXiv:cs.PL/1602.00602v4*, July 2017. 360

137. L. Barrett, R. Dunbar, and J. Lycett. *Human Evolutionary Psychology*. Palgrave Macmillan, 2002. 20

138. L. A. Barroso and U. Hölzle. The datacenter as a computer: An introduction to the design of warehouse-scale machines. Report, Morgan & Claypool, 2009. 95

139. V. R. Basili and J. Beane. Can the Parr curve help with manpower distribution and resource estimation problems? *The Journal of Systems and Software*, 2(1):59–69, Feb. 1981. 128

140. V. R. Basili, S. Green, O. Laitenberger, F. Lanubile, F. Shull, S. Sørumgård, and M. V. Zelkowitz. The empirical investigation of perspective-based reading. In *Proceedings of the Twentieth Annual Software Engineering Workshop*, pages 21–69, Dec. 1995. 9, 359

141. V. R. Basili, N. M. Panlilio-Yap, C. L. Ramsey, C. Shih, and E. E. Katz. A quantitative analysis of software developed in Ada. Technical Report TR-1403, Department of Computer Science, University of Maryland, May 1984. 49

142. V. R. Basili and A. J. Turner. Iterative enhancement: A practical technique for software development. *IEEE Transactions on Software Engineering*, SE-1(4):390–396, Dec. 1975. 132

143. F. M. Bass. A new product growth model for consumer durables. *Management Science*, 15(5):215–227, Jan. 1969. 87

144. P. I. Bass and F. M. Bass. Diffusion of technology generations: A model of adoption and repeat sales. company website, 2001. www.bassbasement.org/F/N/FMB/Pubs/Bass and Bass 2001.pdf. 87

145. H. A. Bastiaanse. *Very, Many, Small, Penguins: Vaguely Related Topics*. PhD thesis, Institute for Logic, Language and Computation, Universiteit van Amsterdam, Mar. 2014. 49

146. B. Baudry, S. Allier, and M. Monperrus. Tailored source code transformations to synthesize computationally diverse program variants. In *eprint arXiv:cs.SE/1401.7635v1*, Jan. 2014. 163, 200

147. F. L. Bauer and H. Wössner. The "Plankalkül" of Konrad Zuse: a forerunner of today's programming languages. *Communications of the ACM*, 15(7):678–685, July 1972. 113

148. J. Bauer, J. Siegmund, N. Peitek, J. C. Hofmeister, and S. Apel. Indentation: Simply a matter of style or support for program comprehension? In *Proceedings of the 27th International Conference on Program Comprehension*, ICPC'19, pages 154–164, May 2019. 192

149. A. Baumann. Hardware is the new software. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, HotOS'17, pages 132–137, May 2016. 118

150. R. F. Baumeister. *Is There Anything Good About Men?* Oxford University Press, 2010. 21

151. R. T. Baust. *Computer Characteristics Quarterly: Volume 7, Number 4-Volume 8, Number 1*. adams associates, 1968. 94

152. G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. The evolution of project inter-dependencies in a software ecosystem: the case of Apache. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance*, ICSM'13, pages 280–289, Sept. 2013. 102

153. G. Bavota, G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. How the Apache community upgrades dependencies: An evolutionary study? *Empirical Software Engineering*, 20(5):1275–1317, Oct. 2015. 102, 103

154. O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. The influence of non-technical factors on code review. In *20th Working Conference on Reverse Engineering*, WCRE'13, pages 122–131, Oct. 2013. 144, 145

155. B. L. Bayus, S. Jain, and A. G. Rao. Truth or consequences: An analysis of vaporware and new product announcements. *Journal of Marketing Research*, 38(1):3–13, Feb. 2001. 78

156. A. A. Beaujean. *Latent Variable Modeling Using R*. Routledge, 2014. 375

157. B. Beber and A. Scacco. What the numbers say: A digit-based test for election fraud. *Political Analysis*, 20(2):211–234, Apr. 2012. 386

158. C. Becker, F. Fagerholm, R. Mohanani, and A. Chatzigeorgiou. Temporal discounting in technical debt: How do software practitioners discount the future? In *eprint arXiv:cs.SE/1901.07024*, Jan. 2019. 56

159. G. S. Becker. Investment in human capital: A theoretical analysis. *Journal of Political Economy*, 70(5):9–49, Oct. 1962. 71

160. R. A. Becker and W. S. Cleveland. *Trellis Graphics User's Manual*. AT&T Bell Laboratories, Murray Hill, Dec. 1995. 225

161. J. Beckhusen. Occupations in information technology. American Community Survey Report ACS-35, U.S. Census Bureau, Aug. 2016. 108

162. M. Bekoff, C. Allen, and G. M. Burghardt. *The Cognitive Animal: Empirical and Theoretical Perspectives on Animal Cognition*. The MIT Press, 2002. 20

163. R. W. Belk and G. Tumbat. The cult of Macintosh. *Consumption, Markets and Culture*, 8(3):205–217, Sept. 2005. 74

164. C. G. Bell. Fundamentals of time shared computers. *Computer Design*, 7(2):44–59, Feb. 1968. 1

165. C. G. Bell. The mini and micro industries. *Computer*, 17(10):14–30, Oct. 1984. 94

166. G. Bell. Bell's law for the birth and death of computer classes: A theory of the computer's evolution. MSR-TR 2007-146, Microsoft Research, Silicon Valley, Nov. 2007. 94

167. G. Bell. Supercomputers: The amazing race (A history of supercomputing, 1960-2020). Technical Report MSR-TR-2015-2, Microsoft Research, Nov. 2014. 94

168. V. A. Bell and P. N. Johnson-Laird. A model theory of modal reasoning. *Cognitive Science*, 22(1):25–51, 1998. 44, 45

169. M. Beller, A. Zaidman, A. Karpov, and R. A. Zwaan. The last line effect explained. *Empirical Software Engineering*, 22(3):1508–1536, June 2017. 82, 163

170. D. J. Bem. Feeling the future: Experimental evidence for anomalous retroactive influences on cognition and affect. *Journal of Personality and Social Psychology*, 100(3):407–425, Mar. 2011. 265

171. R. W. Bemer. a view of the history of COBOL. *Honeywell Computer Journal*, 5(3):130–135, Nov. 1959. 112

172. G. Beniamini, S. Gingichashvili, A. Klein Orbach, and D. G. Feitelson. Meaningful identifier names: The case of single-letter variables. In *25th IEEE International Conference on Program Comprehension*, ICPC'17, pages 45–54, May 2017. 196

173. J. R. Beniger. *The Control Revolution: Technological and Economic Origins of the Information Society*. Hardvard University Press, 1986. 5

174. Y. Benkler. Coase's Penguin, or, Linux and the nature of the firm. *The Yale Law Journal*, 112(3), Dec. 2002. 61, 68

175. A. Benson, D. Li, and K. Shue. Promotions and the Peter principle. Working Paper n. 3047193, US universities, Feb. 2018. 108

176. R. A. Bentley, C. P. Lipo, H. A. Herzog, and M. W. Hahn. Regular rates of popular culture change reflect random copying. *Evolution and Human Behavior*, 28(3):151–158, May 2007. 76

177. F. C. Y. Benureau and N. P. Rougier. Re-run, repeat, reproduce, reuse, replicate: Transforming code into scientific contributions. In *eprint arXiv:cs.GL/1708.08205*, Aug. 2017. 113

178. E. D. Berger, C. Hollenbeck, P. Maj, O. Vitek, and J. Vitek. On the impact of programming languages on code quality: A reproduction study. *ACM Transactions on Programming Languages and Systems*, 41(4):21, Nov. 2019. 4

179. T. Berger, S. She, K. Czarnecki, and A. Wąsowski. Feature-to-code mapping in two large product lines. In J. Bosch and J. Lee, editors, *Software Product Lines: Going Beyond*, volume 6287 of *Lecture Notes in Computer Science*, pages 498–499. Springer Berlin Heidelberg, 2010. 241

180. T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. Variability modeling in the systems software domain. Technical Report GSDLAB-TR 2012-07-06, Generative Software Development Laboratory, University of Waterloo, July 2012. 139

181. E. Berghout, M. Nijland, and K. Grant. Seven ways to get your favoured IT project accepted – politics in IT evaluation. *The Electronic Journal of Information Systems Evaluation*, 8(1):31–40, 2005. 122

182. M. Berglund, W. Bester, and B. van der Merwe. Formalising Boost POSIX regular expression matching. In *International Colloquium on Theoretical Aspects of Computing*, ICTAC 2018, pages 99–115, Oct. 2018. 172

183. B. Berlin and P. Kay. *Basic Color Terms: Their Universality and Evolution*. Berkeley: University of California Press, 1969. 201, 202

184. R. Berman, L. Pekelis, A. Scott, and C. Van den Bulte. p-hacking and false discovery in A/B testing. Working Paper n. 3204791, US universities, Dec. 2018. 363

185. D. Bermbach and E. Wittern. Benchmarking web API quality. In *International Conference on Web Engineering*, ICWE'16, pages 188–206, June 2016. 167

186. A. Bernardo and I. Welch. On the evolution of overconfidence and entrepreneurs. *Journal of Economics & Management Strategy*, 10(3):301–330, 2001. 73

187. K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance CMOS variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, July 2006. 366

188. D. M. Berry, K. Daudjee, J. Dong, I. Fainchtein, M. A. Nelson, T. Nelson, and L. Ou. User's manual as a requirements specification: Case studies. *Requirements Engineering*, 9(1):67–82, Feb. 2004. 136

189. D. M. Berry, E. Kamsties, and M. M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity. Nov. 2003. 162

190. L. M. A. Bettencourt, A. Cintrón-Arias, D. I. Kaiserd, and C. Castillo-Chávez. The power of a good idea: Quantitative modeling of the spread of ideas from epidemiological models. *Physica A*, 364:513–536, May 2006. 76

191. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In C. Beeri and P. Buneman, editors, *Database Theory: 7th International Conference*, ICDT'99, pages 217–235. Springer-Verlag, Jan. 1999. 349

192. D. Biber, S. Johansson, G. Leech, S. Conrad, and E. Finegan. *Longman Grammar of Spoken and Written English*. Pearson Education, 1999. 45, 162, 199

193. C. Bicchieri. *The Grammar of Society: The Nature and Dynamics of Social Norms*. Cambridge University Press, Mar. 2006. 72, 78

194. B. Biddle, A. White, and S. Woods. How many standards in a laptop? (and other empirical questions). Working Paper n. 1619440, Arizona State University (ASU) - College of Law, Sept. 2010. 81

195. S. Biddle. Like everyone else, Twitter hides from U.S. taxes in Ireland. Gawker news site, Oct. 2013. http://valleywag.gawker.com/like-everyone-else-twitter-hides-from-u-s-taxes-in-ir-1447085830. 85

196. B. Biegel, F. Beck, W. Hornig, and S. Diehl. The order of things: How developers sort fields and methods. In *28th IEEE International Conference on Software Maintenance*, ICSM'12, pages 88–97, Sept. 2012. 209, 210, 355

197. S. Bikhchandani, D. Hirshleifer, and I. Welch. A theory of fads, fashion, custom, and cultural change as informational cascades. *Journal of Political Economy*, 100(5):992–1026, Oct. 1992. 54

198. P. Bilton, P. Dodimead, E. Livingstone, I. Rayner, G. Turner, M. Wynniatt, and S. Howes. Managing the risks of legacy ICT to public service delivery. HC 539 SESSION 2013-14, National Audit Office, UK, Sept. 2013. 95, 144

199. W. L. Bircher. *Predictive Power Management for Multi-Core Processors*. PhD thesis, The University of Texas at Austin, Dec. 2010. 368, 371

200. C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced? Bias in bug-fix datasets. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, FSE 2009, pages 121–130, Aug. 2009. 152

201. S. Bird. Software knows best: A case for hardware transparency and measurability. Thesis (m.s.), Department of Electrical Engineering and Computer Science, University of California at Berkeley, May 2010. 362

202. P. G. Bishop and R. E. Bloomfield. Worst case reliability prediction based on a prior estimate of residual defects. In *Proceedings 13th International Symposium on Software Reliability Engineering*, ISSRE'02, pages 295–303, Nov. 2002. 159

203. T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère. Popularity, interoperability, and impact of programming languages in 100,000 open source projects. In *37th Annual International Computer Software & Applications Conference*, COMPSAC 2013, pages 303–312, July 2013. 141, 222

204. Bitsavers' pdf document archive. organization website, July 2019. http:bitsavers.trailing-edge.com/pdf. 115

205. E. Biyalagorsky, W. Boulding, and R. Staelin. Stuck in the past: Why managers persist with new product failures. *Journal of Marketing*, 70(2):108–121, Apr. 2006. 59, 134

206. N. M. Blachman. A survey of automatic digital computers. Survey 111293, Office of Naval Research, Washington, D.C., 1953. 112, 365

207. S. M. Blackburn, R. Garner, C. Hoffmann, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo benchmarks: Java benchmarking development and analysis (extended version). Technical Report TR-CS-06-01, Department of Computer Science, Australian National University, Aug. 2006. 271

208. A.-R. Blais and E. U. Weber. A domain-specific risk-taking (DOSPERT) scale for adult populations. *Judgment and Decision Making*, 1(1):33–47, Apr. 2006. 53

209. D. M. Blank and G. J. Stigler. *The Demand and Supply of Scientific Personnel*. National Bureau of Economic Research, Inc., 1957. 108, 133

210. M. S. Blaubergs and M. D. S. Braine. Short-term memory limitations on decoding self-embedded sentences. *Journal of Experimental Psychology*, 102(4):745–748, 1974. 32

211. D. S. Blinder and D. M. Oppenheimer. Beliefs about what types of mechanisms produce random sequences. *Journal of Behavioral Decision Making*, 21(4):414–427, Oct. 2008. 386

212. N. Bloom, T. Kretschmer, and J. van Reenen. Are family-friendly workplace practices a valuable firm resource? *Strategic Management Journal*, 32(4):343–367, Apr. 2011. 109

213. B. I. Blum. Improving software maintenance by learning from the past: A case study. *Proceedings of the IEEE*, 77(4):596–606, Apr. 1989. 146

214. J. Boccara. Good news: strong types are (mostly) free in C++. blog: Fluent C++, May 2017. http://www.fluentcpp.com/2017/05/05/news-strong-types-are-free. 205

215. B. W. Boehm. *Software Engineering Economics*. Prentice-Hall, Inc, 1981. 294, 295

216. B. W. Boehm and P. N. Papaccio. A value-chain analysis or software productivity. Technical Report USC-CSE-86-500, Center for Systems and Software Engineering, University of Southern California, 1986. 85

217. G. D. Boetticher. Improving credibility of machine learner models in software engineering. In D. Zhang and J. J. P. Tsai, editors, *Advances in Machine Learning Applications in Software Engineering*, chapter 3, pages 52–73. Idea Group Publishing, Oct. 2006. 379

218. J. G. Bolten, R. S. Leonard, M. V. Arena, O. Younossi, and J. M. Sollinger. Sources of weapon system cost growth: Analysis of 35 major defense acquisition programs. Monograph series, RAND Corporation, 2008. 125

219. R. Bond and P. B. Smith. Culture and conformity: A meta-analysis of studies using Asch's (1952b, 1956) line judgment task. *Psychological Bulletin*, 119(1):111–137, Jan. 1996. 54

220. C. F. Bond, Jr. and L. J. Titus. Social facilitation: A meta-analysis of 241 studies. *Psychological Bulletin*, 94(2):265–292, Sept. 1983. 80

221. J. Bonvoisin, R. Mies, J.-F. Boujut, and R. Stark. What is the "source" of open source hardware? *Journal of open hardware*, 1(1):5, Sept. 2017. 70

222. C. F. Borges. An improved algorithm for `HYPOT(A,B)`. In *eprint arXiv:math.NA/1904.09481*, June 2019. 150

223. R. Bornat, S. Dehnadi, and Simon. Mental models, consistency and programming aptitude. In *Tenth Australasian Computing Education Conference*, ACE'08, pages 53–61, Jan. 2008. 179

224. L. Boroditsky. Metaphoric structuring: understanding time through spatial metaphors. *Cognition*, 75:1–28, 2000. 106

225. A. Börsch-Supan and M. Weiss. Productivity and age: Evidence from work teams at the assembly line. Technical Report 148-2007, Manheim Research Institute for the Economics of Aging, 2007. 59

226. L. Bossavit. *The Leprechauns of Software Engineering: How folklore turns into fact and what to do about it*. Leanpub, 2016. 83

227. N. Bostrom and A. Sandberg. The wisdom of nature: An evolutionary heuristic for human enhancement. In J. Savulescu and N. Bostrom, editors, *Human Enhancement*, chapter 18, pages 375–416. Oxford University Press, Jan. 2011. 20

228. A. Botchkarev. Estimating the accuracy of the return on investment (ROI) performance evaluations. *Interdisciplinary Journal of Information, Knowledge, and Management*, 10:217–233, 2015. 63

229. L. Boué. Real numbers, data science and chaos: How to fit any dataset with a single parameter. In *eprint arXiv:cs.LG/1904.12320*, Apr. 2019. 281

230. K. Boukhetala and A. Guidoum. Sim.DiffProc: A package for simulation of diffusion processes in R. HAL Id: hal-00629841, HAL archives-ouvertes.fr, Oct. 2011. 334

231. J. Bourn. New IT systems for Magistrates' courts: the Libra project. Report by the Comptroller and Auditor General HC 327 Session 2002-2003, National Audit Office, UK, Jan. 2003. 123, 124

232. E. M. Bowden and M. Jung-Beeman. Normative data for 144 compound remote associate problems. *Behavior Research Methods, Instruments, & Computers*, 35(4):634–639, Dec. 2003. 80

233. G. H. Bower, J. B. Black, and T. J. Turner. Scripts in memory for text. *Cognitive Psychology*, 11(2):177–220, Apr. 1979. 189

234. J. S. Bowers and C. J. Davis. Bayesian just-so stories in psychology and neuroscience. *Psychological Bulletin*, 138(3):389–414, 2012. 21, 254

235. R. Boyd and P. J. Richerson. Why does culture increase human adaptability. *Ethology and Sociobiology*, 16(2):125–143, Mar. 1995. 75

236. R. Boyd and P. J. Richerson. Why culture is common, but cultural evolution is rare. *Proceedings of the British Academy*, 88:77–93, Apr. 1996. 73

237. M. G. Bradac, D. E. Perry, and L. G. Votta. Prototyping a process monitoring experiment. *IEEE Transactions on Software Engineering*, 20(10):774–784, 1994. 133, 134

238. T. F. Brady, T. Konkle, G. A. Alvarez, and A. Oliva. Visual long-term memory has a massive storage capacity for object details. *PNAS*, 105(38):14325–14329, Sept. 2008. 57

239. D. Braha and Y. Bar-Yam. The statistical mechanics of complex product development: Empirical and analytical results. *Management Science*, 53(7):1127–1145, July 2007. 179

240. T. Brahe. *Tychonis Brahe Dani Scripta Astronomica*. Glydendaliana, 1915. Edited by I. L. E. Dreyer. 2

241. D. W. Braithwaite and R. L. Goldstone. Flexibility in data interpretation: effects of representational format. *frontiers in Psychology*, 4(980):1–16, Dec. 2013. 224

242. N. R. Bramley. *Constructing the world: Active causal learning in cognition*. PhD thesis, University College London, Feb. 2017. 47, 48

243. M. C. Branco, Y. Xiong, K. Czarnecki, J. Küster, and H. Völzer. An empirical study on consistency management of business and IT process models. Technical Report GSDLAB-TR 2012-03-02, Generative Software Development Laboratory, University of Waterloo, Mar. 2012. 102

244. S. Brand. *How buildings Learn: What happens after they're built*. Viking, 1994. 144

245. J. D. Bransford and J. J. Franks. The abstraction of linguistic ideas. *Cognitive Psychology*, 2(4):331–350, Oct. 1971. 188, 189

246. J. D. Bransford and M. K. Johnson. Contextual prerequisites for understanding: Some investigations of comprehension and recall. *Journal of Verbal Learning and Verbal Behavior*, 11(6):717–726, Dec. 1972. 187

247. G. Branwen. Laws of tech: Commoditize your complement. blog: Gwern, Mar. 2018. http://www.gwern.net/Complement. 89

248. S. Brass and C. Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, May 2006. 183

249. H. Braverman. *Labor and Monopoly Capital: The Degradation of Work in the Twentieth Century*. Monthly Review Press, Jan. 1974. 61, 73

250. R. A. Brealey, S. C. Myers, and F. Allen. *Principles of Corporate Finance*. McGraw-Hill Irwin, 10th edition, 2011. 64, 82

251. B. Brembs, K. Button, and M. Munafò. Deep impact: Unintended consequences of journal rank. *Frontiers in Human Neuroscience*, 7(291), June 2013. 11

252. S. Breu, R. Premraj, J. Sillito, and T. Zimmermann. Information needs in bug reports: Improving cooperation between developers and users. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, CSCW'10, pages 301–310, Feb. 2010. 221

253. C. A. Brewer. Color use guidelines for mapping and visualization. In A. M. Maceachren and D. R. F. Taylor, editors, *Visualization in Modern Cartography*, chapter 7, pages 123–147. Pergamon, Nov. 1994. 228

254. E. Brewer, L. Ying, L. Greenfield, R. Cypher, and T. Ts'o. Disks for data centers. Technical report, Google, Inc, Feb. 2016. 370

255. Brigham Young trace repository. No longer available: website, 201? Copy kindly supplied by Dror G. Feitelson. 156

256. P. Brinch Hansen and R. House. The COBOL compiler for the Siemens 3003. *BIT*, 6(1):1–23, Mar. 1966. 113

257. F. Brittman. The most common habits from more than 200 English papers written by graduate Chinese engineering students. Jan. 2007. 165

258. S. Broadbent. Font requirements for next generation air traffic management systems. Technical Report HRS/HSP-006-REP-01, European Organisation for the Safety of Air Navigation, 2000. 29

259. G. W. Brock. *The U.S. Computer Industry: A Study of Market Power*. Ballinger Publishing Company, 1975. 92

260. L. D. Brock and H. A. Goodman. Reliability analysis of the F-8 digital fly-by-wire system. NASA Contractor Report 163110, Dryden Flight Research Center, Oct. 1981. 147

261. A. D. Broido and A. Clauset. Scale-free networks are rare. In *eprint arXiv:physics.soc-ph/1801.03400*, Jan. 2018. 239

262. G. Bronevetsky and B. R. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the 22nd Annual International Conference on Supercomputing*, ICS'08, pages 155–164, June 2008. 167

263. J. Brooke. SUS: A 'quick' and 'dirty' usability scale. In P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, editors, *Usability Evaluation in Industry*, chapter 21, pages 189–194. Taylor and Francis, June 1996. 376

264. J. Brooke. SUS: A retrospective. *Journal of Usability Studies*, 8(2):29–40, Feb. 2013. 376

265. R. Brooks. A model of human cognitive behavior in writing code for computer programs, vol I. Report AFOSR-TR-75-1084, Carnegie Mellon University, May 1975. 37

266. F. P. Brooks, Jr. *The Mythical Man-Month*. Addison–Wesley, anniversary edition, 1995. 9, 142

267. G. D. A. Brown, I. Neath, and N. Chater. A temporal ratio model of memory. *Psychological Review*, 114(3):539–576, July 2007. 33, 34

268. N. C. C. Brown and A. Altadmri. Novice Java programming mistakes: Large-scale data vs. educator beliefs. *ACM Transactions on Computing Education*, 17(2):7, June 2017. 161

269. J. Brunner and P. C. Austin. Inflation of Type I error rate in multiple regression when independent variables are measured with error. *The Canadian Journal of Statistics*, 37(1):33–46, Mar. 2009. 288

270. M. Brysbaert, W. Fias, and M.-P. Noël. The Whorfian hypothesis and numerical cognition: is 'twenty-four' processed in the same way as 'four-and-twenty'? *Cognition*, 66(1):51–77, Apr. 1998. 201

271. I. Buchmann. *Batteries in a Portable World: A Handbook on rechargeable Batteries for Non-engineers*. Cadex Electronix Inc, third edition, 2011. 368

272. J. B. Buckheit and D. L. Donoho. WaveLab and reproducible research. In A. Antoniadis and G. Oppenheim, editors, *Wavelets and Statistics*, chapter 5, pages 55–81. Springer-Verlag, 1995. 10

273. M. Budden, P. Hadavas, L. Hoffman, and C. Pretz. Generating valid 4 × 4 correlation matrices. *Applied Mathematics E-Notes*, 7:53–59, 2007. 235

274. D. V. Budescu, H.-H. Por, S. B. Broomell, and M. Smithson. The interpretation of IPCC probabilistic statements around the world. *Nature Climate Change*, 4:508–512, Apr. 2014. 153

275. D. J. Buettner. *Designing an Optimal Software Intensive System Acquisition: A Game Theoretic Approach*. PhD thesis, University of Southern California, Sept. 2008. 128, 135, 142, 327, 356, 383

276. E. Bugnion, S. Devine, M. Rosenblum, J. Sugerman, and E. Y. Wang. Bringing virtualization to the x86 architecture with the original VMware workstation. *ACM Transactions on Computer Systems*, 30(4):12, Nov. 2012. 122

277. M. Bullynck. What is an operating system? A historical investigation (1954–1964). HAL Id: halshs-01541602, HAL archives-ouvertes.fr, Aug. 2017. 112

278. N. Bulnet and M. H. Halstead. Impurities found in algorithm implementations. Technical Report CSD-TR 111, Purdue University, Mar. 1974. 182, 183

279. J. S. Bunderson and K. M. Sutcliffe. Management team learning orientation and business unit performance. *Journal of Applied Psychology*, 88(3):552–560, June 2003. 76

280. Bureau of labor statistics. BLS website, July 2019. https://www.bls.gov/ces. 103

281. K. P. Burnham and D. R. Anderson. Multimodel inference: Understanding AIC and BIC in model selection. *Sociological Methods and Research*, 33(2):261–304, Nov. 2004. 290

282. J. C. Burns. The evolving market for word processing and typesetting systems. In *Proceedings of the National Computer Conference and Exposition*, AFIPS'76, pages 617–623, June 1976. 92

283. Q. L. Burrell. A note on ageing in a library circulation model. *Journal of Documentation*, 41(2):100–115, 1985. 35

284. R. P. L. Buse and W. R. Weimer. Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558, July 2010. 193

285. J. Businge. *Co-evolution of the Eclipse Framework and its Third-party Plug-ins*. PhD thesis, Eindhoven University of Technology, Sept. 2013. 336, 337, 338

286. R. W. Butler and G. B. Finelli. The infeasibility of quantifying the reliability of life-critical real-time software. *IEEE Transactions on Software Engineering*, 19(1):3–12, 1993. 157

287. G. Butts and K. Linton. The joint confidence level paradox: A history of denial. In *NASA 2009 Cost Estimating Symposium*. NASA Center for Aerospace Information, Apr. 2009. 126

288. D. Byrne and C. Corrado. ICT prices and ICT services: What do they tell us about productivity and technology? Finance and Economics Discussion series 2017-015, Federal Reserve Board, Washington, D.C., Feb. 2017. 5

289. B. Calder, D. Grunwald, and B. Zorn. Quantifying behavioral differences between C and C++ programs. *Journal of Programming Languages*, 2(4):313–351, 1995. 180

290. E. G. Cale, L. L. Gremillion, and J. L. McKenney. Price/performance patterns of U.S. computer systems. *Communications of the ACM*, 22(4):225–233, Apr. 1979. 106

291. J. Calhoun, C. Savoie, M. Randolph-Gips, and I. Bozkurt. Human reliability analysis in spaceflight applications. *Quality and Reliability Engineering International*, 29(6):869–882, Aug. 2013. 23

292. A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 255–270, Aug. 2015. 179, 192, 200

293. C. F. Camerer and E. F. Johnson. The process-performance paradox in expert judgment: How can the experts know so much and predict so badly? In K. A. Ericsson and J. Smith, editors, *Towards a general theory of expertise: Prospects and limits*. Cambridge University Press, 1991. 40

294. J. I. D. Campbell. On the relation between skilled performance of simple division and multiplication. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 23(5):1140–1159, 1997. 51

295. M. Campbell-Kelly. *Foundations of Computer Programming in Britain (1945 - 1955)*. PhD thesis, Department of Mathematics and Computer Studies, Sunderland Polytechnic, June 1980. 106

296. M. Campbell-Kelly. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*. The MIT Press, Apr. 2004. 62

297. M. Campbell-Kelly and D. D. Garcia-Swartz. Economic perspectives on the history of the computer time-sharing industry, 1965–1985. *IEEE Annals of the History of Computing*, 30(1):16–36, Jan.-Mar. 2008. 105

298. M. Campbell-Kelly and D. D. Garcia-Swartz. Pragmatism not ideology: IBM's love affair with open source software. Working Paper n. 1081613, UK universities, Jan. 2008. 68

299. M. Caneill and S. Zacchiroli. Debsources: Live and historical views on macro-level software evolution. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'14, pages 28:1–28:10, Sept. 2014. 116

300. G. Canfora, L. Cerulo, M. Cimitile, and M. Di Penta. Social interactions around cross-system bug fixings: the case of FreeBSD and OpenBSD. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR'11, pages 143–152, May 2011. 249

301. L. F. Capretz, P. Waychal, and J. Jia. Comparing popularity of testing careers among Canadian, Chinese, Indian students. In *IEEE/ACM 41st International Conference on Software Engineering*, ICSE-SEET, pages 258–259, May 2019. 108

302. B. Caprile and P. Tonella. Nomen est omen: Analyzing the language of function identifiers. In *Proceedings of the 6th Working Conference on Reverse Engineering*, WCRE'99, pages 112–122, Oct. 1999. 196

303. J. R. Carlberg. Scientific/engineering work stations: A market survey. Departmental Report DTNSRDC/CMLD-83/07, David W. Taylor Naval Ship Research and Development Center, May 1983. 115

304. S. Carter-Thomas and E. Rowley-Jolivet. *If*-conditionals in medical discourse: From theory to disciplinary practice. *Journal of English for Academic Purposes*, 7(3):191–205, July 2008. 45

305. E. Caspi. Empirical study of opportunities for bit-level specialization in word-based programs. Thesis (m.s.), University of California, Berkeley, 2000. 203

306. D. Castelvecchi. The biggest mystery in mathematics: Shinichi Mochizuki and the impenetrable proof. *Nature*, 526(7572):178–181, Oct. 2015. 148

307. M. P. Catherwood. Manpower impacts of electronic data processing. Publication B-171, New York State Department of Labor, Division of Research and Statistics, Sept. 1968. 92

308. A. Čaušević, R. Shukla, S. Punnekkat, and D. Sundmark. Effects of negative testing on TDD: An industrial experiment. In H. Baumeister and B. Weber, editors, *Agile Processes in Software Engineering and Extreme Programming*, volume 149 of *Lecture Notes in Business Information Processing*, pages 91–105. Springer Berlin Heidelberg, 2013. 175

309. J. P. Cavanagh. Relation between the immediate memory span and the memory search rate. *Psychological Review*, 79(6):525–530, Nov. 1972. 32

310. M. Ceccato, M. Di Penta, P. Falcarin, F. Ricca, M. Torchiano, and P. Tonella. A family of experiments to assess the effectiveness and efficiency of source code obfuscation techniques. *Empirical Software Engineering*, 19(4):1040–1074, 2014. 197

311. C. Cecot and W. K. Viscusi. Judicial review of agency benefit-cost analysis. *George Mason Law Review*, 22(3):575–617, Nov. 2015. 149

312. C. Cederström and P. Fleming. *Dead Man Working*. Zero books, 2012. 67

313. A. Celik, K. Palmskog, M. Parovic, E. J. G. Arias, and M. Gligoric. Mutation analysis for Coq. In *34th IEEE/ACM International Conference on Automated Software Engineering*, ASE'19, pages 539–551, Nov. 2019. 149

314. D. Centola and A. Baronchelli. The spontaneous emergence of conventions: An experimental study of cultural evolution. *PNAS*, 112(7):1989–1994, Feb. 2015. 76

315. D. Centola, J. Becker, D. Brackbill, and A. Baronchelli. Experimental evidence for tipping points in social convention. *Science*, 360(6393):1116–1119, June 2018. 75

316. P. E. Ceruzzi. The early computers of Konrad Zuse, 1935 to 1945. *Annals of the History of Computing*, 3(3):241–262, July 1981. 1

317. H. S. Cha. *Disrupting the Management Supply Chain: An Organizational Learning Model of IT Offshore Outsourcing*. PhD thesis, Faculty of the Committee on Business Administration, The University of Arizona, July 2007. 77

318. F. Chandler, I. A. Heard, M. Presley, A. Burg, E. Midden, and P. Mongan. NASA human error analysis. Technical report, NASA Office of Safety and Mission Assurance, Sept. 2010. 23

319. F. T. Chandler, Y. H. J. Chang, A. Mosleh, J. L. Marble, R. L. Boring, and D. I. Gertman. Human reliability analysis methods: Selection guidance for NASA. Nasa/osma technical report, NASA Headquarters Office of Safety and Mission Assurance, July 2006. 23

320. D. Chandlera and A. Kapelner. Breaking monotony with meaning: Motivation in crowdsourcing markets. In *eprint arXiv:stat.OT/1210.0962*, Oct. 2012. 74

321. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection–A survey. *ACM Computing Surveys*, 41(3):1–58, July 2009. 379, 380

322. K. Chandrasekar. *High-Level Power Estimation and Optimization of DRAMs*. PhD thesis, Technische Universiteit Delft, Oct. 2014. 371

323. A. C. Chang and P. Li. Is economics research replicable? Sixty published papers from thirteen journals say "usually not". Finance and Economics Discussion Series 2015-083, Washington: Board of Governors of the Federal Reserve System, Sept. 2015. 4

324. P. P. Chang, S. A. Mahlke, W. Y. Chen, and W. mei W. Hwu. Profile-guided automatic inline expansion for C programs. *Software–Practice and Experience*, 22(5):349–369, May 1992. 186

325. W. Chang. *R Graphics Cookbook*. O'Reilly, 2012. 225

326. A. Chao. Estimating population size for sparse data in capture–recapture experiments. *Biometrics*, 45(2):427–438, June 1989. 103

327. A. Chao, C.-H. Chiu, and L. Jost. Unifying species diversity, phylogenetic diversity, functional diversity, and related similarity and differentiation measures through Hill numbers. *Annual Review of Ecology, Evolution and Systematics*, 45(1):297–324, Nov. 2014. 96

328. A. Chao, R. K. Colwell, C.-W. Lin, and N. J. Gotelli. Sufficient sampling for asymptotic minimum species richness estimators. *Ecology*, 90(4):1125–1133, Apr. 2009. 104

329. A. Chao, S.-M. Lee, and S.-L. Jeng. Estimating population size for capture–recapture data when capture probabilities vary by time and individual animal. *Biometrics*, 48(1):201–216, Mar. 1992. 104

330. A. Chao and C.-W. Lin. Nonparametric lower bounds for species richness and shared species richness under sampling without replacement. *Biometrics*, 68(3):912–921, Sept. 2012. 103

331. A. Chao and M. C. K. Yang. Stopping rules and estimation for recapture debugging with unequal failure rates. *Biometrika*, 80(1):193–201, Mar. 1993. 175

332. C. Chapman, P. Wang, and K. T. Stolee. Exploring regular expression comprehension. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE'17, pages 405–416, Nov. 2017. 181

333. C. A. Chapman. Usage and refactoring studies of python regular expressions. Thesis (m.s.), Iowa State University, 2016. 172

334. M. R. Chapman. *In Search of Stupidity: Over 20 years of High-Tech Marketing Disasters*. Apress, second edition, 2006. 85, 92, 147

335. M. R. Chapman and R. J. Hujar. The softletter financial handbook 2011: Metrics and benchmarks mergers, IPOs, and venture finance compensation operations. company website, Sept. 2011. https://softletter.com/wp-content/uploads/2017/02/FINHANDBOOK_A0055E.pdf. 62

336. P. Charbachi, L. Eklund, and E. Enoiu. Can pairwise testing perform comparably to manually handcrafted testing carried out by industrial engineers? In *eprint arXiv:cs.SE/1706.01636*, June 2017. 175

337. G. Charness and U. Gneezy. Strong evidence for gender differences in risk taking. *Journal of Economic Behavior & Organization*, 83(1):50–58, June 2012. 53

338. W. G. Chase and K. A. Ericsson. Skill and working memory. In G. H. Bower, editor, *The Psychology of Learning and Motivation, Vol. 15*, pages 1–58. Academic Press, 1982. 40

339. N. Chater. *The Mind is Flat: The Illusion of Mental Depth and the Improvised Mind*. Allen Lane, Mar. 2018. 20

340. P. D. Chatzoglou and L. A. Macaulay. Requirements capture and analysis : A survey of current practice. *Requirements Engineering*, 1(2):75–87, June 1996. 135

341. M. Chekaf, N. Gauvrit, A. Guida, and F. Mathy. Compression in working memory and its relationship with fluid intelligence. *Cognitive Science*, 42(53):904–922, June 2018. 34

342. D. D. Chen and G.-J. Ahn. Security analysis of x86 processor microcode. Thesis (b.sc.), Arizona State University, Dec. 2014. 161

343. L. Chen, D. Wu, W. Ma, Y. Zhou, B. Xu, and H. Leung. How C++ templates are used for generic programming: An empirical study on 50 open source systems. *ACM Transactions on Software Engineering and Methodology*, 29(1):3, Feb. 2020. 185, 186

344. T. Chen, Y. Chen, Q. Guo, O. Temam, T. Wu, and W. Hu. Statistical performance comparisons of computers. In *18th International Symposium on High Performance Computer Architecture*, HPCA'12, pages 1–12, Feb. 2012. 257, 258, 260, 261

345. Y. Chen, A. Groce, X. Fern, C. Zhang, W.-K. Wong, E. Eide, and J. Regehr. Taming compiler fuzzers. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'13, pages 197–208, June 2013. 172, 318, 319

346. P. W. Cheng, K. J. Holyoak, R. E. Nisbett, and L. M. Oliver. Pragmatic versus syntactic approaches to training deductive reasoning. *Cognitive Psychology*, 18(3):293–328, July 1986. 40

347. A. Chesson and G. Chamberlin. Survey-based measures of software investment in the UK. Economic Trends 627, Office for National Statistics, UK, Feb. 2006. 62

348. R. N. Chesterman. Report of Queensland health payroll system commission of inquiry. Report, Queensland Government, Australia, July 2013. 120, 123

349. H. Cheung and S. Kemper. Competing complexity metrics and adults' production of complex sentences. *Applied Psycholinguistics*, 13:53–76, 1992. 188

350. R. C. Cheung. A user-oriented software reliability model. *IEEE Transactions on Software Engineering*, 6(2):118–125, 1980. 249

351. J. Y. Chiao, A. R. Bordeaux, and N. Ambady. Mental representations of social status. *Cognition*, 93(2):B49–B57, Sept. 2004. 50

352. J. J. Chilenski. An investigation of three forms of the modified condition decision coverage (MCDC) criterion. Final Report DOT/FAA/AR-01/18, U.S. Department of Transportation, Federal Aviation Administration, Apr. 2001. 174

353. J. J. Chilenski and S. P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, Sept. 1994. 174

354. S. Chilton, J. Covey, M. Jones-Lee, G. Loomes, and H. Metcalf. Valuation of health benefits associated with reductions in air pollution. Technical report, Department for Environment, Food and Rural Affairs, May 2004. 152

355. C.-H. Chiu, Y.-T. Wang, B. A. Walther, and A. Chao. An improved nonparametric lower bound of species richness via a modified Good–Turing frequency formula. *Biometrics*, 70(3):671–682, Sept. 2014. 104

356. H. Cho. *System-Level Effects of Soft Errors*. PhD thesis, Department of Electrical Engineering, Stanford University, Aug. 2015. 158

357. N. Chomsky. *Syntactic Structures*. Walter de Gruyter & Co, 13th edition, 1975. 177

358. K. R. Christensen. Negative and affirmative sentences increase activation in different areas in the brain. *Journal of Neurolinguistics*, 22(1):1–17, Jan. 2009. 162

359. S. Christey and B. Martin. Buying into the bias: Why vulnerability statistics suck. blackhat USA 2013, July-Aug. 2013. 152

360. T. Christie. The widespread and persistent myth that it is easier to multiply and divide with Hindu-Arabic numerals than with Roman ones. blog: Tony Christie, Feb. 2017. https://thonyc.wordpress.com/2017/02/10/the-widespread-and-persistent-myth-that-it-is-easier-to-multiply-and-divide-with-hindu-arabic-numerals-than-with-roman-ones. 107

361. R. A. Chubon and M. R. Hester. An enhanced standard computer keyboard system for single-finger and typing-stick typing. *Journal of Rehabilitation Research and Development*, 25(4):17–24, Oct.-Dec. 1988. 96

362. A. CIA. Analytic thinking and presentation for intelligence producers: Analysis training handbook. Technical report, Office of Training and Education, Central Intelligence Agency, Aug. 199? 224

363. Z. J. Ciechanowicz and A. C. De Weever. The 'completeness' of the Pascal test suite. *Software–Practice and Experience*, 14(5):463–471, 1984. 162

364. J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall. An empirical analysis of the Docker container ecosystem on GitHub. In *Proceedings of the 14th International Conference on Mining Software Repositories*, MSR'17, pages 323–333, May 2017. 114, 140

365. D. Citron. MisSPECulation: Partial and misleading use of SPEC CPU2000 in computer architecture conferences. In *Proceedings of the 30th annual International Symposium on Computer Architecture*, ISCA'03, pages 52–61, June 2003. 366

366. D. Citron and D. G. Feitelson. "look it up" or "do the math": An energy, area, and timing analysis of instruction resuse and memoization. Technical Report H-0196, International Business Machines Corporation, Oct. 2003. 363, 364, 365

367. I. Ciupa, A. Pretschner, M. Oriol, A. Leitner, and B. Meyer. On the number and nature of faults found by random testing. *Software Testing, Verification and Reliability*, 21(1):3–28, Mar. 2011. 171

368. Civil Service Department, UK. *Computers in Central Government Ten years ahead*. Her Majesty's Stationery Office, Jan. 1971. 105

369. H. H. Clark. *Understanding language*. Cambridge University Press, 1996. 178

370. H. H. Clark and D. Wilkes-Gibbs. Referring as a collaborative process. *Cognition*, 22:1–39, 1986. 76

371. A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. 319

372. W. S. Cleveland. *The Elements of Graphing Data*. Wadsworth Advanced Book Program, 1985. 225

373. W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, Sept. 1984. 225

374. J. Clune, J.-B. Mouret, and H. Lipson. The evolutionary origins of modularity. *Proceedings of the Royal Society B: Biological Sciences*, 280(1755):20122863, Jan. 2013. 184

375. A. Coad. Investigating the exponential age distribution of firms. *Economics: The Open-Access, Open-Assessment E-Journal*, 4(2010-17):1–30, Mar. 2010. 107

376. N. M. Coe. *The growth and locational dynamics of the UK computer services industry, 1981-1996*. PhD thesis, Department of Geography, University of Durham, 1996. 107

377. J. Coelho and M. T. Valente. Why modern open source projects fail. In *Proceedings of the 11th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'17, pages 186–196, Sept. 2017. 122

378. J. Cohen. *Statistical Power Analysis for the Behavioural Sciences*. Routledge, second edition, 1988. 256, 258

379. J. Cohen. The Earth is round ($p < 0.05$). *American Psychologist*, 49(12):997–1003, 1994. 266

380. J. Cohen, S. Teleki, and E. Brown. *Best Kept Secrets of Peer Code Review*. SmartBear Software, 2012. 377

381. Z. Coker, S. Hasan, J. Overbey, M. Hafiz, and C. Kästner. Integers in C: An open invitation to security attacks? Technical Report CSSE14-01, Auburn University, Feb. 2014. 207

382. M. Cokol, I. Iossifov, R. Rodriguez-Esteban, and A. Rzhetsky. How many scientific papers should be retracted? *European Molecular Biology Organization*, 8(5):422–423, Apr. 2007. 11

383. R. E. Cole. *Managing Quality Fads: How American Business Learned to Play the Quality Game*. Oxford University Press, Feb. 1999. 149

384. E. G. Coleman. *Coding Freedom: The Ethics and Aesthetics of Hacking*. Princeton University Press, Dec. 2012. 107

385. M. Collard, A. Ruttle, B. Buchanan, and M. J. O'Brien. Population size and cultural evolution in nonindustrial food-producing societies. *PLoS ONE*, 8(9):e72628, Sept. 2013. 76

386. C. Collberg, T. Proebsting, and A. M. Warren. Repeatability and benefaction in computer systems research-A study and a modest proposal. Technical Report TR 14-014, Department of Computer Science, University of Arizona, Feb. 2015. 4

387. D. Comin and B. Hobijn. Cross-country technology adoption: making the theories face the facts. *Journal of Monetary Economics*, 51(1):39–83, 2004. 7

388. C. Commeyne, A. Abran, and R. Djouab. Effort estimation with story points and cosmic function points - an industry case study. *Software Measurement News*, 21(1):25–36, 2016. 129, 130

389. Comptroller General of the United States. Multiyear leasing and government-wide purchasing of automatic data processing equipment should result in significant savings. Technical Report B-115369, U.S. General Accounting Office, Apr. 1971. 105

390. Comptroller General of the United States. Federal agencies' maintenance of computer programs: Expensive and undermanaged. Technical Report AFMD-81-25, U.S. General Accounting Office, Feb. 1981. 114

391. Computing Technology Industry Association, The. Cyberstates 2019: The definitive guide to the U.S. tech industry and tech workforce. Research report, The Computing Technology Industry Association, Mar. 2019. 71

392. S. Condon, M. Regardie, M. Stark, and S. Waligora. Cost and schedule estimation study report. Technical Report SEL-93-002, Goddard Space Flight Center, Nov. 1993. 82, 133

393. Foundations for evidence-based policymaking Act of 2018 H.R.4175, Jan. 2018. 115th Congress of the United States of America, 2nd session. 2

394. M. Conoscenti, V. Besner, A. Vetrò, and D. M. Fernández. Combining data analytics and developers feedback for identifying reasons of inaccurate estimations in agile software development. *The Journal of Systems and Software*, 156:126–135, Oct. 2019. 129

395. B. Conrad and M. Mitzenmacher. Power laws for monkeys typing randomly: The case of unequal probabilities. *IEEE Transactions on Information Theory*, 50(7):1403–1414, July 2004. 247

396. J. J. Cook and C. Zilles. A characterization of instruction-level error derating and its implications for error detection. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, DSN 2008, pages 482–491, June 2008. 286

397. K. Cook. Ubuntu security hardening statistics (amd64). personal website, July 2019. 102

398. P. Coombs. *IT Project Estimation: A Practical Guide to the Costing of Software*. Cambridge University Press, 2003. 121

399. T. Copeland and V. Antikarov. *Real Options A Practitioner's Guide*. Texere Publishing Limited, Apr. 2001. 66, 67

400. A. Corazza, V. Maggio, and G. Scanniello. Coherence of comments and method implementations: a dataset and an empirical investigation. *Software Quality Journal*, 26(2):751–777, June 2018. 193

401. J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it? Technical report, The Linux Foundation, Dec. 2010. 121

402. J. Corbet, G. Kroah-Hartman, and A. McPherson. Linux kernel development: How fast it is going, who is doing it, what they are doing, and who is sponsoring it. Technical report, The Linux Foundation, Mar. 2012. 286

403. M. Correll and M. Gleicher. Error bars considered harmful: Exploring alternate encodings for mean and error. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2142–2151, Dec. 2014. 221

404. J. W. Cortada. *The Digital Flood: The Diffusion of Information Technology Across the U.S., Europe and Asia*. Oxford University Press, Sept. 2012. 7, 102

405. M. J. Cortese and M. M. Khanna. Age of acquisition predicts naming and lexical-decision performance above and beyond 22 other predictor variables: An analysis of 2,342 words. *The Quarterly Journal of Experimental Psychology*, 60(8):1072–1082, Aug. 2007. 196

406. L. Cosmides and J. Tooby. Evolutionary psychology: A primer. Technical report, Center for Evolutionary Psychology, University of California, Santa Barbara, 1998. 20, 44

407. D. E. Costa, S. Mujahid, R. Abdalkareem, and E. Shihab. Breaking type-safety in Go: An empirical study on the usage of the `unsafe` package. In *eprint arXiv:cs.SE/2006.09973*, June 2020. 197

408. D. L. Costa and M. E. Kahn. Changes in the value of life, 1940-1980. Working Paper No. 9396, National Bureau of Economic Research, USA, Dec. 2002. 153

409. V. Costan and S. Devadas. Intel SGX explained. In *Cryptology ePrint Archive: Report 2016/086*, Jan. 2016. 95

410. D. Cotroneo, A. K. Iannillo, R. Natella, and R. Pietrantuono. A comprehensive study on software aging across Android versions and vendors. In *eprint arXiv:cs.SE/2005.11523*, May 2020. 358

411. D. Cotroneo, R. Pietrantuono, S. Russo, and K. Trivedi. How do bugs surface? A comprehensive study on the characteristics of software bugs manifestation. *The Journal of Systems and Software*, 113(C):27–43, Mar. 2016. 150

412. J. D. Couger and M. A. Colter. *Maintenance Programming: Improving Productivity Through Motivation*. Prentice-Hall, Inc, 1985. 71

413. N. Cowan. The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and Brain Sciences*, 24(1):87–185, 2001. 31

414. M. F. Cowlishaw. Decimal floating-point: Algorism for computers. In *Proceedings of the 16th IEEE Symposium on Computer Arithmetic*, pages 104–111, June 2003. 150

415. J. F. Coyle and G. D. Polsky. Acqui-hiring. *Duke Law Journal*, 63(2):281–346, Nov. 2013. 93

416. Cray Research. *M Series Site Planning Reference Manual*. Cray Research, Inc, Apr. 1983. 94

417. W. Crooymans, P. Pradhan, and S. Jansen. Exploring network modelling and strategy in the Dutch software business ecosystem. In *Proceedings of the International Conference on Software Business*, ICSOB 2015, pages 45–59, June 2015. 108

418. F. E. Croxton and R. E. Stryker. Bar charts versus circle diagrams. *Journal of the American Statistical Association*, 22(160):473–482, Dec. 1927. 224

419. J. Culver. The life cycle of a CPU. blog: The cpushack, 2010. `http://www.cpushack.com/life-cycle-of-cpu.html`. 254, 255

420. G. Cumming and R. Maillardet. Confidence intervals and replication: Where will the next mean fall? *Psychological Methods*, 11(3):217–227, 2006. 268

421. C. R. Cummins. *The interpretation and use of numerically-quantified expressions*. PhD thesis, Research Centre for English and Applied Linguistics, University of Cambridge, Nov. 2011. 50

422. P. G. Curran and K. A. Hauser. I'm paid biweekly, just not by leprechauns: Evaluating valid-but-incorrect response rates to attention check items. *Journal of Research in Personality*, 82(103849), Oct. 2019. 375

423. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, Nov. 1988. 130

424. B. Curtis, S. B. Sheppard, and E. Kruesi. Evaluation of software life cycle data from the PAVE PAWS project. Technical Report RADC-TR-80-28, Rome Air Development Center, Griffiss Air Force Base, Mar. 1980. 133, 134, 151, 161, 162

425. M. A. Cusumano. Factory concepts and practices in software development: An historical overview. Working Paper #3095-89 BPS, Alfred P. Sloan School of Management, Dec. 1989. 73, 141

426. M. A. Cusumano. Shifting economies: From craft production to flexible systems and software factories. Working Paper #3325-91/BPS, Alfred P. Sloan School of Management, Aug. 1991. 141

427. M. A. Cusumano, A. Gawer, and D. B. Yoffie. *The Business of Platforms: Strategy in the Age of Digital Competition, Innovation, and Power*. Harper Business, June 2019. 109

428. K. Cwalina and B. Abrams. *Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries*. Addison–Wesley, 2006. 184

429. J. Czerwonka. On use of coverage metrics in assessing effectiveness of combinatorial test designs. In *Sixth International Conference on Software Testing, Verification and Validation, Workshops Proceedings*, ICST 2013, pages 257–266, Mar. 2013. 173

430. J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, Feb. 2006. 167

431. A. Damasio. *Self Comes to Mind: Constructing the Conscious Brain*. Vintage books, 2012. 20

432. M. Daneman and P. A. Carpenter. Individual differences in working memory and reading. *Journal of Verbal Learning and Verbal Behavior*, 19(4):450–466, Aug. 1980. 190, 191

433. M. Daneman and P. A. Carpenter. Individual differences in integrating information between and within sentences. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 9(4):561–584, 1983. 191

434. C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts. No country for old members: User lifecycle and linguistic change in online communities. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW 2013, pages 307–318, May 2013. 195

435. B. Danglot, P. Preux, B. Baudry, and M. Monperrus. Correctness attraction: A study of stability of software behavior under runtime perturbation. *Empirical Software Engineering*, 23(4):2086–2119, Aug. 2018. 158, 159

436. A. Danowitz, K. Kelley, J. Mao, J. P. Stevenson, and M. Horowitz. CPU DB: Recording microprocessor history. *Communications of the ACM*, 55(4):55–63, Apr. 2012. 96, 218, 366

437. J. Darley and C. D. Batson. "from Jerusalem to Jericho": A study of situational and dispositional variables in helping behavior. *Journal of Personality and Social Psychology*, 27(1):100–108, 1973. 23

438. P. A. David. Clio and the economics of QWERTY. *The American Economic Review*, 75(2):332–337, May 1985. 96

439. P. A. David. Computer and dynamo: The modern productivity paradox in a not-too distant mirror. No. 339, Department of Economics, Stanford University, July 1989. 6

440. J. W. Davidson, J. R. Rabung, and D. B. Whalley. Relating static and dynamic machine code measurements. Technical Report CS-89-03, Department of Computer Science, University of Virginia, July 1989. 180

441. C. J. Davis. The spatial coding model of visual word identification. *Psychological Review*, 117(3):713–758, July 2010. 32, 177

442. J. C. Davis, C. A. Coghlan, F. Servant, and D. Lee. The impact of regular expression denial of service (ReDoS) in practice: An empirical study at the ecosystem scale. In *Proceedings of the 26th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'18, pages 246–256, Nov. 2018. 148

443. J. C. Davis, L. G. Michael, F. Servant, C. A. Coghlan, and D. Lee. Why aren't regular expressions a lingua franca? An empirical study on the re-use and portability of regular expressions. In *Proceedings of the 27th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'19, pages 443–454, Aug. 2019. 172

444. J. C. Davis, D. Moyer, A. M. Kazerouni, and D. Lee. Testing regex generalizability and its implications A large-scale many-language measurement study. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering*, ASE'19, pages 427–439, Nov. 2019. 269, 270

445. S. J. Davis and B. S. de la Parra. Application flows. Working paper, University of Chicago Booth School of Business, Mar. 2017. 108, 114

446. S. J. Davis, J. MacCrisken, and K. M. Murphy. Economic perspectives on software design: PC operating systems and platforms. Working Paper No. 8411, National Bureau of Economic Research, USA, Aug. 2001. 1

447. S. Dayal. Characterizing HEC storage systems at rest. Technical Report CMU-PDL-08-109, Parallel Data Laboratory, Carnegie Mellon University, July 2008. 246

448. R. de Bliek. *Empirical studies on the economic impact of trust*. PhD thesis, Erasmus Research Institute of Management, Rotterdam, May 2015. 72

449. S. De Deyne, S. Verheyen, E. Ameel, W. Vanpaemel, M. J. Dry, W. Voorspoels, and G. Storms. Exemplar by feature applicability matrices and other Dutch normative data for semantic concepts. *Behavior Research Methods*, 40(4):1030–1048, Nov. 2008. 43

450. A. D. de Groot. *Thought and Choice in Chess*. Amsterdam University Press, 2008. 39

451. J. L. de la Vara, M. Borg, K. Wnuk, and L. Moonen. An industrial survey of safety evidence change impact analysis practice. *IEEE Transactions on Software Engineering*, 42(12):1095–1117, Dec. 2016. 112, 144

452. B. B. de Mesquita, A. Smith, R. M. Siverson, and J. D. Morrow. *The Logic of Political Survival*. The MIT Press, 2005. 130

453. R. A. De Millo, R. J. Lipton, and A. J. Perlis. Social processes and proofs of theorems and programs. *Communications of the ACM*, 22(5):271–280, May 1979. 148, 266

454. A. B. de Oliveira, J.-C. Petkovich, T. Reidemeister, and S. Fischmeister. DataMill: Rigorous performance evaluation made easy. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ICPE'13, pages 137–148, Apr. 2013. 374, 375

455. F. G. de Oliveira Neto, R. Torkar, R. Feldt, L. Gren, C. A. Furia, and Z. Huang. Evolution of statistical analysis in empirical software engineering research: Current state and steps forward. In *eprint arXiv:cs.SE/1706.00933*, June 2017. 9

456. G. B. de Pádua and W. Shang. Revisiting exception handling practices with exception flow analysis. In *International Conference on Source Code Analysis and Manipulation*, SCAM'17, pages 11–20, Sept. 2017. 204, 205

457. C. B. De Soto, M. London, and S. Handel. Social reasoning and spatial paralogic. *Journal of Personality and Social Psychologs*, 2(4):513–521, 1965. 46

458. K. De Vogeleer. *La loi de convexité énergie-fréquence de la consommation des programmes : modélisation, thermosensibilité et applications*. PhD thesis, Informatique [cs] Telecom ParisTech, Sept. 2015. 369

459. K. De Vogeleer, G. Memmi, and P. Jouvelot. Parameter sensitivity analysis of the energy/frequency convexity rule for nanometer-scale application processors. In *eprint arXiv:cs.DS/1508.07740*, Aug. 2015. 368

460. I. De Voldere, J.-F. Romainville, S. Knotter, E. Durinck, E. Engin, A. Le Gall, P. Kern, E. Airaghi, T. Pletosu, H. Ranaivoson, and K. Hoelck. Mapping the creative value chains: A study on the economy of culture in the digital age. Final report, Directorate-General for Education and Culture Directorate D, European Commission, 2017. 85

461. G. de Wit. Firm size distributions: An overview of steady-state distributions resulting from firm dynamics models. Technical Report N200418, EIM Business and Policy Research, Jan. 2005. 107

462. J. Dearden, B. W. Ickes, and L. Samuelson. To innovate or not to innovate: Incentives and innovation in hierarchies. *The American Economic Review*, 80(5):1105–1124, Dec. 1990. 5, 74

463. I. J. Deary. *Intelligence: A Very Short Introduction*. Oxford University Press, 2001. 52

464. B. K. Debnath, M. F. Mokbel, and D. J. Lilja. Exploiting the impact of database system configuration parameters: A design of experiments approach. *IEEE Data Engineering Bulletin*, 31(1):3–10, Mar. 2008. 365

465. Debsources developers, The. Statistics | Debian sources. organization website, June 2019. https://sources.debian.org/stats. 113

466. A. Decan and T. Mens. What do package dependencies tell us about semantic versioning? *IEEE Transactions on Software Engineering*, ???(???):???, Nov. 2019. 117

467. A. Decan, T. Mens, and M. Claes. An empirical comparison of dependency issues in OSS packaging ecosystems. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*, SANER 2017, pages 2–12, Feb. 2017. 117

468. A. Decan, T. Mens, M. Claes, and P. Grosjean. On the development and distribution of R packages: An empirical analysis of the R ecosystem. In *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW'15, page 41, Sept. 2015. 116

469. A. Decan, T. Mens, M. Claes, and P. Grosjean. When *GitHub* meets *CRAN*: An analysis of inter-repository package dependency problems. In *23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER'16, pages 493–504, Mar. 2016. 116

470. A. Decan, T. Mens, and E. Constantinou. On the impact of security vulnerabilities in the npm package dependency network. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR'18, pages 181–191, May 2018. 165

471. L. A. DeChurch and J. R. Mesmer-Magnus. Maintaining shared mental models over long-duration exploration missions: Literature review & operational assessment. Technical Memorandum TM-2015-218590, National Aeronautics and Space Administration, Sept. 2015. 140

472. Defence technical information center. Search page for DTIC reports, July 2016. http://dsearch.dtic.mil. 8

473. Defense, Department of. Military standard DOD-STD-2167 defense system software development. Standard DOD-STD-2167, U.S. Department of Defense, 1985. 131

474. Defense, Department of. Standard practice system safety. Standard MIL-STD-882E, U.S. Department of Defense, May 2012. 153

475. S. Dehaene. Symbols and quantities in parietal cortex: elements of a mathematical theory of number representation and manipulation. In P. Haggard, Y. Rossetti, and M. Kawato, editors, *Sensorimotor Foundations of Higher Cognition (Attention and Performance) XXII*, chapter 24, pages 527–574. Oxford University Press, Nov. 2007. 48

476. S. Dehaene. *Reading in the Brain: The Science and evolution of a human invention*. Viking, 2009. 19

477. S. Dehaene. *The Number Sense*. Oxford University Press, revised and updated edition, 2011. 46, 48

478. S. Dehaene, S. Bossini, and P. Giraux. The mental representation of parity and number magnitude. *Journal of Experimental Psychology: General*, 122(3):371–396, Sept. 1993. 22

479. S. Dehaene, E. Dupoux, and J. Mehler. Is numerical comparison digits? Analogical and symbolic effects in two-digit number comparisons. *Journal of Experimental Psychology: Human Perception and Performance*, 16(3):626–641, 1990. 49

480. S. Dehaene, V. Izard, E. Spelke, and P. Pica. Log or linear? Distinct intuitions of the number scale in Western and Amazonian indigene cultures. *Science*, 320(5880):1217–1220, May 2008. 21, 48

481. S. M. Dekleva. The influence of the information systems development approach on maintenance. *MIS Quarterly*, 16(3):355–372, Sept. 1992. 143

482. R. T. DeLamarter. *Big Blue: IBM's Use and Abuse of Power*. Pan Books, 1988. 77, 105, 130, 131

483. S. DellaVigna. Psychology and economics: Evidence from the field. Working Paper No. 13420, National Bureau of Economic Research, USA, Sept. 2007. 56

484. J. Demmel and Y. Hilda. Accurate floating point summation. Technical Report UCB/CSD-02-1180, University of California, Berkeley, May 2002. 147

485. M. Derex, J.-F. Bonnefon, R. Boyd, and A. Mesoudi. Causal understanding is not necessary for the improvement of culturally evolving technology. *Nature Human Behaviour*, 3(5):446–452, May 2019. 75

486. G. Destefanis. Which programming language should a company use? A Twitter-based analysis. Technical Report CRIM-14/10-23-MODL, Computer Research Institute of Montréal, Oct. 2014. 114

487. S. Deutsch and M. H. Jørgensen. Studying the hidden costs of offshoring – the effect of psychic distance. Thesis (m.s.), Copenhagen Business School, Aug. 2014. 127

488. J. P. DeVale. *High Performance Robust Computer Systems*. PhD thesis, Electrical and Computer Engineering, Pittsburgh, Oct. 2001. 156

489. T. Dey and A. Mockus. Deriving a usage-independent software quality metric. In *eprint arXiv:cs.SE/2002.09989*, Feb. 2020. 156

490. A. Di Franco, H. Guo, and C. Rubio-González. A comprehensive study of real-world numerical bug characteristics. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE'17, pages 509–519, Nov. 2017. 161

491. C. Di Martino, Z. Kalbarczyk, R. K. Iyer, F. Baccanico, J. Fullop, and W. Kramer. Lessons learned from the analysis of system failures at petascale: The case of Blue Waters. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN 2014, pages 610–621, June 2014. 166

492. M. Di Penta, L. Cerulo, and L. Aversano. The life and death of statically detected vulnerabilities: an empirical study. *Information and Software Technology*, 51(10):1469–1484, Oct. 2009. 153, 154, 343

493. A. Di Sorbo, J. Spillner, G. Canfora, and S. Panichella. "won't we fix this issue?" Qualitative characterization and automated identification of wontfix issues on GitHub. In *eprint arXiv:cs.SE/1904.02414*, Apr. 2019. 4, 149

494. T. F. Dickey. Programmer variability. *Proceedings of the IEEE*, 69(7):844–845, July 1981. 10

495. L. S. Dickstein. The effect of figure on syllogistic reasoning. *Memory & Cognition*, 6(1):76–83, 1978. 45

496. A. Diekmann. Not the first digit! Using Benford's law to detect fraudulent scientific data. *Journal of Applied Statistics*, 34(3):321–329, Oct. 2007. 386

497. J. Dietrich, K. Jezek, and P. Brada. What Java developers know about compatibility, and why this matters. In *eprint arXiv:cs.SE/1408.2607v1*, Aug. 2014. 376

498. S. Dietrich, I. Hertrich, and H. Ackermann. Training of ultra-fast speech comprehension induces functional reorganization of the central-visual system in late-blind humans. *Frontiers in Human Neuroscience*, 7(701), Oct. 2013. 19

499. E. W. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, Mar. 1968. 203

500. C. DiMarco, G. Hirst, and M. Stede. The semantic and stylistic differentiation of synonyms and near-synonyms. In *AAAI Spring Symposium on Building Lexicons for Machine Translation*, pages 114–121, Mar. 1993. 236

501. A. Dinaburg. Bitsquatting: DNS hijacking without exploitation. Reference 2011-307, Raytheon Company, July 2011. 167

502. D. K. Dirlam. Most efficient chunk sizes. *Cognitive Psychology*, 3(2):355–359, Apr. 1972. 34

503. A. K. Dixit and R. S. Pindyck. *Investment under Uncertainty*. Princeton University Press, 1994. 65, 66, 334

504. H. Do, S. Mirarab, L. Tahvildari, and G. Rothermel. An empirical study of the effect of time constraints on the cost-benefits of regression testing. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2008, pages 71–82, Nov. 2008. 175

505. C. Domas. Breaking the x86 ISA. blackhat USA 2017, July 2017. 161

506. D. J. Dooling and R. E. Christiaansen. Episodic and semantic aspects of memory for prose. *Journal of Experimental Psychology: Human Learning and Memory*, 3(4):428–436, 1977. 190

507. J. R. Douceur and W. J. Bolosky. A large-scale study of file-system contents. In *Proceedings of the 1999 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'99, pages 59–70, July 1999. 246

508. J. Downer. Watching the watchmaker: On regulating the social in lieu of the technical. Discussion Paper 54, London School of Economics and Political Science, June 2009. 149

509. J. R. Doyle. Survey of time preference, delay discounting models. *Judgment and Decision Making*, 8(2):116–135, Mar. 2013. 56

510. G. Dréan. *The Computer Industry: Structure, economics, perspectives*. Gérard Dréan, english edition, 2012. 94

511. S. Drobisz, T. Mens, and R. Di Cosmo. A historical analysis of Debian package conflicts. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR'15, pages 212–223, June 2015. 117

512. S. Duffy, J. Huttenlocher, L. V. Hedges, and L. E. Crawford. Category effects on stimulus estimation: Shifting and skewed frequency distributions. *Psychonomic Bulletin & Review*, 17(2):224–230, Apr. 2010. 49

513. J. Duggan. Implementing a metapopulation Bass diffusion model using the R package deSolve. *The R Journal*, 9(1):153–163, June 2017. 356

514. R. I. M. Dunbar and R. Sosis. Optimising human community sizes. *Evolution and Human Behavior*, 39(1):106–111, Jan. 2018. 99, 100

515. J. R. Dunham and L. A. Lauterbach. An experiment in software reliability additional analyses using data from automated replications. NASA Contractor Report 178395, Research Triangle Institute, North Carolina, Jan. 1988. 157

516. J. R. Dunham and J. L. Pierce. An experiment in software reliability. NASA Contractor Report 172553, NASA Langley Research Center, Mar. 1986. 157, 158, 229

517. L. M. Dunn. *An Investigation of the Factors Affecting the Lifecycle Costs of COTS-Based Systems*. PhD thesis, School of Computing, University of Portsmouth, June 2011. 111

518. D. Dunning, C. Heath, and J. M. Suls. Flawed self-assessment: Implications for health, education, and the workplace. *Psychological Science in the Public Interest*, 5(3):69–106, Apr. 2004. 375

519. S. Duplichan. Intel overstates FPU accuracy. personal website, June 2013. http://notabs.org/fpuaccuracy. 150

520. V. H. S. Durelli, J. Offutt, N. Li, M. E. Delamaro, J. Guo, Z. Shi, and X. Ai. What to expect of predicates: An empirical analysis of predicates in real world programs. *The Journal of Systems and Software*, 113:324–336, Mar. 2016. 204

521. C. Dutang. CRAN task view: Probability distributions. website, June 2016. http://CRAN.R-project.org/view=Distributions. 236, 242

522. G. Dutilh, J. Annis, S. D. Brown, P. Cassey, N. J. Evans, R. P. P. P. Grasman, G. E. Hawkins, A. Heathcote, W. R. Holmes, A.-M. Krypotos, C. N. Kupitz, F. P. Leite, V. Lerche, Y.-S. Lin, G. D. Logan, T. J. Palmeri, J. J. Starns, J. S. Trueblood, L. van Maanen, D. van Ravenzwaaij, J. Vandekerckhove, I. Visser, A. Voss, C. N. White, T. V. Wiecki, J. Rieskamp, and C. Donkin. The quality of response time data inference: A blinded, collaborative assessment of the validity of cognitive models. *Psychonomic Bulletin & Review*, 26(4):1051–1069, Aug. 2019. 22

523. T. Dybå, V. B. Kampenes, and D. I. K. Sjøberg. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, 48(8):745–755, Aug. 2006. 8, 358

524. R. Dyer, H. Rajan, H. A. Nguyen, and T. N. Nguyen. Mining billions of AST nodes to study actual and potential usage of Java language features. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE'14, pages 779–790, May-June 2014. 202

525. P. C. Earley. Social loafing and collectivism: A comparison of the United States and the People's Republic of China. *Administrative Science Quarterly*, 34(4):565–581, Dec. 1989. 72

526. H. Ebbinghaus. *Über das Gedächtnis. Untersuchungen zur experimentellen Psychologie*. Teachers College, Columbia University, 1885. English translation by Henry A. Ruger and Clara E. Bussenius as Memory: A Contribution to Experimental Psychology (Teachers College, Columbia University, 1913). 35

527. A. Eckbreth, C. Saff, K. Connolly, N. Crawford, C. Eick, M. Goorsky, N. Kacena, D. Miller, R. Schafrik, D. Schmidt, D. Stein, M. Stroscio, G. Washington, and J. Zolper. Sustaining Air Force aging aircraft into the 21st century. Technical Report SAB-TR-11-01, United States Air Force Scientific Advisory Board, Aug. 2011. 98

528. Economist Data team. The changing US technology sector: Daily chart for april 21 2015. The Economist website, Apr. 2015. As of Q1 2015, Sources: Thomson Reuters; awk scripts+R converted the data embedded in Javascript. 5

529. EDB. Offensive security's exploit database archive. https://www.exploit-db.com, Mar. 2018. 151

530. S. Eder, M. Junker, E. Jürgens, B. Hauptmann, R. Vaas, and K.-H. Prommer. How much does unused code matter for maintenance? In *34th International Conference on Software Engineering*, ICSE'12, pages 1102–1111, June 2012. 65

531. A. Edmundson, B. Holtkamp, E. Rivera, M. Finifter, A. Mettler, and D. Wagner. An empirical study on the effectiveness of security code review. In *Proceedings of the 5th International Conference on Engineering Secure Software and Systems*, ESSoS'13, pages 197–212, Feb. 2013. 268, 292, 301

532. M. A. Edwards and S. Roy. Academic research in the 21st century: Maintaining scientific integrity in a climate of perverse incentives and hypercompetition. *Environmental Engineering Science*, 34(1):51–61, Jan. 2017. 10

533. K. Ehrlich and P. N. Johnson-Laird. Spatial descriptions and referential continuity. *Journal of Verbal Learning and Verbal Behavior*, 21(3):296–306, June 1982. 191

534. S. G. Eick, C. R. Loader, M. D. Long, L. G. Votta, and S. V. Wiel. Estimating software fault content before coding. In *Proceedings of the 14th International Conference on Software engineering*, ICSE'92, pages 59–65, May 1992. 169

535. P. Ein-Dor. Grosch's law re-revisited: CPU power and the cost of computation. *Communications of the ACM*, 28(2):142–151, Feb. 1985. 94

536. T. Eisensee and D. Strömberg. News droughts, news floods, and U.S. disaster relief. *The Quarterly Journal of Economics*, 122(2):693–728, May 2007. 152

537. K. El Emam, S. Benlarbi, N. Goel, W. Melo, H. Lounis, and S. N. Rai. The optimal class size for object-oriented software. *IEEE Transactions on Software Engineering*, 28(5):494–509, Mar. 2002. 229

538. K. El Emam and A. G. Koru. A replicated survey of IT software project failures. *IEEE Software*, 25(5):84–90, Apr. 2008. 122

539. A. Elci. The dependence of operating system size upon allocatable resources. Technical Report 75-172, Department of Computer Science, Purdue University, Dec. 1975. 110

540. I. R. Elliott. Life cycle planning for a large mix of commercial systems. In B. Elkins and L. Hunt, editors, *Software Phenomenology – Working Papers of the Software Life Cycle Management Workshop*, chapter 10, pages 203–215. Computer Systems Command, United States Army, Aug. 1977. 143

541. J. Elliott, M. Hoemmen, and F. Mueller. Exploiting data representation for fault tolerance. In *eprint arXiv:cs.NA/1312.2333v1*, Dec. 2013. 147

542. N. C. Ellis and R. A. Hennelly. A bilingual word-length effect: Implications for intelligence testing and the relative ease of mental calculation in Welsh and English. *British Journal of Psychology*, 71:43–51, 1980. 31, 362

543. P. D. Ellis. *The Essential Guide to Effect Sizes: Statistical Power, Meta-Analysis, and the Interpretation of Research Results*. Cambridge University Press, 2010. 256

544. R. Engbert, A. Nuthmann, E. M. Richter, and R. Kliegl. SWIFT: A dynamical model of saccade generation during reading. *Psychological Review*, 112(4):777–813, Oct. 2005. 28

545. J. Engblom. Why SpecInt95 should not be used to benchmark embedded systems tools. *ACM SIGPLAN Notices*, 34(7):96–103, July 1999. 193, 194

546. B. Enke and F. Zimmermann. Correlation neglect in belief formation. *The Review of Economic Studies*, 86(1):313–332, Jan. 2019. 39

547. J. Ensign and D. K. Akaka. Defense acquisitions: DOD has paid billions in award and incentive fees regardless of acquisition outcomes. Technical Report GAO-06-66, United States Government Accountability Office, Dec. 2005. 124

548. N. L. Ensmenger. Letting the "computer boys" take over: Technology and the politics of organizational transformation. *International Review of Social History*, 48(S11):153–180, Dec. 2003. 131

549. Y.-H. Eom and H.-H. Jo. Generalized friendship paradox in complex networks: The case of scientific collaboration. In *eprint arXiv:cs.SI/1401.1458*, Apr. 2014. 100

550. D. M. Erceg-Hurn and V. M. Mirosevich. Modern robust statistical methods. *American Psychologist*, 63(7):591–601, Oct. 2008. 253

551. K. A. Ericsson and N. Charness. Expert performance. *American Psychologist*, 49(8):725–747, Aug. 1994. 39

552. K. A. Ericsson and K. W. Harwell. Deliberate practice and proposed limits on the effects of practice on the acquisition of expert performance: Why the original definition matters and recommendations for future research. *frontiers in Psychology*, 10:2396, Oct. 2019. 39

553. K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer. The role of deliberate practice in the acquisition of expert performance. *Psychological Review*, 100(3):363–406, July 1993. also University of Colorado, Technical Report #91-06. 40

554. K. A. Ericsson and A. C. Lehmann. Expert and exceptional performance: Evidence of maximal adaption to task constraints. *Annual Review of Psychology*, 47:273–305, 1996. 40

555. K. Eriksson, D. H. Bailey, and D. C. Geary. The grammar of approximating number pairs. *Memory & Cognition*, 38(3):333–343, Apr. 2010. 49

556. K. Eriksson, F. Jansson, and J. Sjöstrand. Bentley's conjecture on popularity toplist turnover under random copying. *The Ramanujan Journal*, 23(1-3):371–396, Dec. 2010. 76

557. N. A. Ernst, J. C. Carver, D. Mendez, and M. Torchiano. Understanding peer review of software engineering papers. In *eprint arXiv:cs.SE/2009.01209*, Sept. 2020. 10

558. L. Eshkevari, F. D. Santos, J. R. Cordy, and G. Antoniol. Are PHP applications ready for Hack? In *IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering*, SANER 2015, pages 63–72, Mar. 2015. 208

559. L. M. Eshkevari, V. Arnaoudova, M. Di Penta, R. Oliveto, Y.-G. Guéhéneuc, and G. Antoniol. An exploratory study of identifier renamings. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR'11, pages 33–42, May 2011. 139

560. H. Esmaeilzadeh, T. Cao, X. Yang, S. M. Blackburn, and K. S. McKinley. Looking back on the language and hardware revolutions: Measured power, performance, and scaling. In *Proceedings of the sixteenth International Conference on Architectural support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 319–332, Mar. 2011. 262

561. W. K. Estes. *Classification and Cognition*. Oxford University Press, 1994. 42

562. J. A. Etzel, J. M. Zacks, and T. S. Braver. Searchlight analysis: promise, pitfalls, and potential. *NeuroImage*, 78:261–269, Sept. 2013. 177

563. A. N. Evans, B. Campbell, and M. L. Soffa. Is Rust used safely by software developers? In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE'20, pages 246–257, July 2020. 197

564. J. S. B. T. Evans, J. L. Barston, and P. Pollard. On the conflict between logic and belief in syllogistic reasoning. *Memory & Cognition*, 11(3):295–306, 1983. 45, 46

565. J. L. Eveleens and C. Verhoef. The rise and fall of the Chaos report figures. *IEEE Software*, 27(1):30–36, Jan. 2010. 122

566. J. Eyolfson, L. Tan, and P. Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR'11, pages 153–162, May 2011. 121, 326, 344

567. J. Eyolfson, L. Tan, and P. Lam. Correlations between bugginess and time-based commit characteristics. *Empirical Software Engineering*, 19(4):1009–1039, Aug. 2014. 346, 347

568. Facebook. Facebook Inc. 2013 Form 10-K. SEC website, 2014. https://www.sec.gov/Archives/edgar/data/1326801/000132680114000007/fb-12312013x10k.htm. 88

569. Facebook. Facebook Inc. 2015 Form 10-K. SEC website, 2016. https://www.sec.gov/Archives/edgar/data/1326801/000132680116000043/fb-12312015x10k.htm. 88

570. R. Falk and C. Konold. Making sense of randomness: Implicit encoding as a basis for judgment. *Psychological Review*, 104(2):301–318, Apr. 1997. 51

571. D. Fanelli. How many scientists fabricate and falsify research? A systematic review and meta-analysis of survey data. *PLoS ONE*, 4(5):e5738, May 2009. 11

572. D. Fanelli. "Positive" results increase down the hierarchy of the sciences. *PLoS ONE*, 5(4):e10068, Apr. 2010. 4

573. F. C. Fang, R. G. Steen, and A. Casadevall. Misconduct accounts for the majority of retracted scientific papers. *PNAS*, 109(42):17028–17033, Oct. 2012. 11

574. M. Fang and M. Hafiz. Discovering buffer overflow vulnerabilities in the wild: An empirical study. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'14, pages 23:1–23:10, Sept. 2014. 152

575. L. Farr and B. Nanus. Factors that affect the cost of computer programming, volume I. Technical Documentary Report ESD-TDR-64-448, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, July 1964. 127

576. L. Farr and H. J. Zagorski. Factors that affect the cost of computer programming, volume II: A quantitative analysis. Technical Documentary Report ESD-TDR-64-448, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Sept. 1964. 127

577. J. Farrell and P. Klemperer. Coordination and lock-in: Competition with switching costs and network effects. In M. Armstrong and R. H. Porter, editors, *Handbook of Industrial Organization, Volume 3*, chapter 31, pages 1967–2072. North-Holland, Oct. 2007. 100

578. J. Farrell and C. Shapiro. Dynamic competition with switching costs. *RAND Journal of Economics*, 19(1):123–137, 1988. 100

579. S. Farrell, M. J. Hurlstone, and S. Lewandowsky. Sequential dependencies in recall of sequences: Filling in the blanks. *Memory & Cognition*, 41(6):938–52, Aug. 2013. 34

580. S. Farrell, K. Oberauer, M. Greaves, K. Pasiecznik, S. Lewandowsky, and C. Jarrold. A test of interference versus decay in working memory: Varying distraction within lists in a complex span task. *Journal of Memory and Language*, 90:66–87, Oct. 2016. 33

581. FDA. General principles of software validation. Final guidance for industry and FDA staff, U.S. Food and Drug Administration, Jan. 2002. 153

582. Federal Food and Drug Administration. PMA approvals. FDA website, July 2019. https://www.fda.gov/medical-devices/device-approvals-denials-and-clearances/pma-approvals. 154

583. Federal Register. *United States v. Adobe Systems, Inc., et al.; Proposed Final Judgment and Competitive Impact Statement*, 2010. 75 (No. 190; October 1), 24624. 109

584. Federal Trade Commission. DELL COMPUTER CORPORATION CONSENT ORDER, ETC., IN REGARD TO ALLEGED VIOLATION OF SEC. 5 OF THE FEDERAL TRADE COMMISSION ACT, Docket C-3658. In P. C. Epperson, editor, *Federal Trade Commission decisions: Findings, opinions and orders volume 121*, pages 616–643. U.S. Government Printing Office, May 1996. 81

585. D. G. Feitelson. *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, 2014. 112, 366, 386

586. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, chapter 19, pages 337–360. Springer-Verlag, June 1995. 377

587. S. L. Feld. Why your friends have more friends than you do. *The American Journal of Sociology*, 96(6):1464–1477, May 1991. 100

588. J. Feldman. Minimization of boolean complexity in human concept learning. *Nature*, 407:630–633, Oct. 2000. 42, 43

589. J. Feldman. An algebra of human concept learning. *Journal of Mathematical Psychology*, 50(4):339–368, Aug. 2006. 42

590. A. Feldstein and P. Turner. Overflow, underflow, and severe loss of significance in floating-point addition and subtraction. *IMA Journal of Numerical Analysis*, 6(2):241–251, Apr. 1986. 156

591. M. Felici. *Observational Models of Requirements Evolution*. PhD thesis, School of Informatics, University of Edinburgh, 2004. 144

592. S. Feng, S. Gupta, A. Ansari, and S. Mahlke. Shoestring: Probabilistic soft error reliability on the cheap. In *Proceedings of the fifteenth edition of ASPLOS on Architectural support for programming languages and operating systems*, ASPLOS'10, pages 385–396, Mar. 2010. 167

593. N. Fenton, M. Neil, W. Marsh, P. Hearty, Ł. Radliński, and P. Krause. On the effectiveness of early life cycle defect prediction with Bayesian nets. *Empirical Software Engineering*, 13(5):499–537, Oct. 2008. 293, 294

594. D. V. Ferens and D. S. Christensen. Calibrating software cost models to Department of Defense databases-A review of ten studies. *ISPA Journal of Parametrics*, XVIII(2):55–74, Nov. 1998. 128

595. C. J. Ferguson and M. Heene. A vast graveyard of undead theories: Publication bias and psychological science's aversion to the Null. *Perspectives on Psychological Science*, 7(6):555–561, Nov. 2012. 4, 265

596. P. Fernández. Valuing real options: Frequently made errors. Working Paper n. 274855, Instituto de Estudios Superiores de la Empresa, Madrid, June 2001. 67

597. L. Ferrand, M. Brysbaert, E. Keuleers, B. New, P. Bonin, A. Méot, M. Augustinova, and C. Pallier. Comparing word processing times in naming, lexical decision, and progressive demasking: evidence from Chronolex. *frontiers in Psychology*, 2(306), Nov. 2011. 196

598. S. Ferson, J. O'Rawe, A. Antonenko, J. Siegrist, J. Mickley, C. C. Luhmann, K. Sentz, and A. M. Finkel. Natural language of uncertainty: numeric hedge words. *International Journal of Approximate Reasoning*, 57:19–39, Feb. 2015. 50

599. R. G. Fichman and C. F. Kemerer. Incentive compatibility and systematic software reuse. *Journal of Systems and Software*, 57(1):45–60, Apr. 2001. 82

600. A. Filippin and P. Crosetto. A reconsideration of gender differences in risk attitudes. IZA DP No. 8184, The Institute for the Study of Labor, Bonn, May 2014. 53

601. C. J. Fillmore. Topics in lexical semantics. In R. W. Cole, editor, *Current Issues in Linguistic Theory*, pages 76–138. Indiana University Press, 1977. 44

602. Financial Accounting Standards Board. Statement of financial accounting standards no. 86. Technical report, Financial Accounting Foundation, Aug. 1985. 84

603. M. Finifter. Towards evidence-based assessment of factors contributing to the introduction and detection of software vulnerabilities. Technical Report UCB/EECS-2013-49, Electrical Engineering and Computer Sciences, University of California at Berkeley, May 2013. 169, 170

604. E. Fischer. The evolution of character codes, 1874-1968. Nov. 2002. 106

605. D. A. Fisher. A common programming language for the Department of Defense – background and technical requirements. Paper P-1191, Institute for Defense Analyses, Science and Technology Division, June 1976. 113

606. J. Fisher and R. A. Hinde. The opening of milk bottles by birds. *British Birds*, 42(11):347–357, 1949. 75

607. J. C. Fisher and R. H. Pry. A simple substitution model of technological change. *Technological Forecasting & Social Change*, 3:75–88, Apr. 1971-1972. 86

608. P. Flajolet, P. Dumas, and V. Puyhaubert. Some exactly solvable models of urn process theory. In P. Chassaing, editor, *Proceedings of Fourth Colloquium on Mathematics and Computer Science Algorithms, Trees, Combinatorics and Probabilities*, pages 59–118, 2006. 102

609. K. Flamm. *Targeting the Computer*. The Brookings Institution, Washington, D.C., 1987. 8, 104

610. K. Flamm. *Creating the Computer*. The Brookings Institution, Washington, D.C., 1988. 1

611. K. Flamm. Measuring Moore's law: Evidence from price, cost, and quality indexes. Working Paper No. 24553, National Bureau of Economic Research, USA, Apr. 2018. 7

612. D. Flater. Estimation of uncertainty in application profiles. NIST TN.1826, National Institute of Standards and Technology, Apr. 2014. 374

613. D. Flater. Screening for factors affecting application performance in profiling measurements. NIST Technical Note 1855, National Institute of Standards and Technology, Oct. 2014. 372

614. D. Flater and W. F. Guthrie. A case study of performance degradation attributable to run-time bounds checks on C++ vector access. *Journal of Research of the National Institute of Standards and Technology*, 118(012):260–279, May 2013. 202, 295, 297

615. P. J. Fleming and J. J. Wallace. How not to lie with statistics: The correct way to summarize benchmark results. *Communications of the ACM*, 29(3):218–221, Mar. 1986. 367

616. J. I. Flombaum, J. A. Junge, and M. D. Hauser. Rhesus monkeys (*Macaca mulatta*) spontaneously compute addition operations over large numbers. *Cognition*, 97(3):315–325, Oct. 2005. 49

617. B. Floyd, T. Santander, and W. Weimer. Decoding the representation of code in the brain: An fMRI study of code review and expertise. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE'17, pages 175–186, May 2017. 177

618. B. Flyvbjerg. How planners deal with uncomfortable knowledge: The dubious ethics of the American Planning Association. *Cities*, 32:157–163, June 2013. 127

619. B. Flyvbjerg, M. S. Holm, and S. L. Buhl. Underestimating costs in public works projects: Error or lie? *Journal of the American Planning Association*, 68(3):279–295, June 2002. 125

620. J. Fodor. *The Modularity of Mind: An Essay on Faculty Psychology*. The MIT Press, 1983. 20

621. R. A. Foley. An evolutionary and chronological framework for human social behaviour. *Proceedings of the British Academy*, 88:95–117, 1996. 19

622. P. Fonseca, K. Zhang, X. Wang, and A. Krishnamurthy. An empirical study on the correctness of formally verified distributed systems. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys'17, pages 328–343, Apr. 2017. 168

623. R. E. Fontana Jr. and G. M. Decad. Moore's law realities for recording systems and memory storage components: HDD, tape, NAND, and optical. *AIP Advances*, 8(5):056506, May 2018. 98

624. C. E. Ford and S. A. Thompson. Conditionals in discourse: A text-based study from English. In E. C. Traugott, A. T. Meulen, J. S. Reilly, and C. A. Furguson, editors, *On Conditionals*, chapter 18, pages 353–372. Cambridge University Press, 1986. 45

625. C. Foroughi and A. D. Stern. Digital innovation with high costs of entry: Evidence from software-driven medical devices. HBS Working Paper #18-094, Harvard Business School, Mar. 2018. 141

626. J. Förster, E. T. Higgins, and A. T. Bianco. Speed/accuracy decisions in task performance: Built-in trade-off or separate strategic concerns? *Organizational Behavior and Human Decision Processes*, 90(1):148–164, Jan. 2003. 24

627. J. Fowkes and C. Sutton. Parameter-free probabilistic API mining across GitHub. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 254–265, Nov. 2016. 352, 353

628. M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison–Wesley, 1999. 9

629. W. B. Frakes, C. J. Fox, and B. A. Nejmeh. *Software Engineering in the Unix/C Environment*. Prentice-Hall, Inc, 1991. 184

630. S. Frederick, G. Loewenstein, and T. O'Donoghue. Time discounting: A critical review. *Journal of Economic Literature*, 40(2):351–401, June 2002. 178

631. D. P. Freedman and G. M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews*. Dorset House Publishing, 1990. 169

632. P. A. Freund and N. Kasten. How smart do you think you are? A meta-analysis on the validity of self-estimates of cognitive ability. *Psychological Bulletin*, 138(2):296–321, Mar. 2011. 21

633. A. Frumusanu. The Samsung Exynos 7420 deep dive - Inside a modern 14nm SoC. Anantech news site, June 2015. http://www.anandtech.com/show/9330/exynos-7420-deep-dive/5. 368, 369

634. W.-T. Fu and W. D. Gray. Memory versus perceptual-motor tradeoffs in a blocks world task. In *Proceedings of the Twenty-second Annual Conference of the Cognitive Science Society*, pages 154–159. Erlbaum, 2000. 25

635. Y. Funami and M. H. Halstead. A software physics analysis of akiyama's debugging data. Technical Report CSD-TR 144, Purdue University, May 1975. 182

636. B. M. Fung, K. Wang, R. Chen, and P. S. Yu. Privacy-preserving data publishing: A survey of recent developments. *ACM Computing Surveys*, 42(4):1–53, June 2010. 378

637. C. A. Furia. Bayesian statistics in software engineering: Practical guide and case studies. In *eprint arXiv:cs.SE/1608.06865*, Aug. 2016. 254

638. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. Statistical semantics: Analysis of the potential performance of key-word information systems. *The Bell System Technical Journal*, 62(6):1753–1805, July-Aug. 1983. 106

639. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11):964–971, Nov. 1987. 106

640. T. Futagami, M. Itoh, Y. Mihara, F. Mitsuhashi, H. Nishiyama, M. Shukuguchi, N. Tachi, K. Toyama, H. Obata, Y. Ooizumi, T. Shimizu, and S. Takeichi. *ESCR Embedded System development Coding Reference guide [C Language Edition]*. Information-technology Promotion Agency, Japan, 2.0 edition, 2017. 184

641. R. Futrell, K. Mahowald, and E. Gibson. Large-scale evidence of dependency length minimization in 37 languages. *PNAS*, 112(33):10336–10341, Aug. 2015. 187, 188

642. M. T. Gailliot and R. F. Baumeister. The physiology of willpower: Linking blood glucose to self-control. *Personality and Social Psychology Review*, 11(4):303–327, Nov. 2007. 57

643. W. A. Gale. Good-Turing smoothing without tears. Technical Report 94.5, AT&T Bell Laboratories, Aug. 1994. 383

644. K. Gallaba, C. Macho, M. Pinzger, and S. McIntosh. Noise and heterogeneity in historical build data: An empirical study of Travis CI. In *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering*, ASE'18, pages 87–97, Sept. 2018. 140

645. K. Gallaba, A. Mesbah, and I. Beschastnikh. Don't call us, we'll call you: Characterizing callbacks in JavaScript. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM'15, pages 247–256, Oct. 2015. 208, 209

646. C. R. Gallistel, S. Fairhurst, and P. Balsam. The learning curve: Implications of a quantitative analysis. *PNAS*, 101(36):13124–13131, Sept. 2004. 36

647. C. R. Gallistel, M. Krishan, Y. Liu, R. Miller, and P. E. Latham. The perception of probability. *Psychological Review*, 121(1):96–123, Jan. 2014. 51

648. T. J. Gandomani, K. T. Wei, and A. K. Binhamid. A case study research on software cost estimation using experts' estimates, Wideband Delphi, and Planning Poker technique. *International Journal of Software Engineering and Its Applications*, 8(11):173–182, Apr. 2014. 274

649. A. Gandy. *The entry of established electronics companies into the early computer industry in the UK and USA*. PhD thesis, London School of Economics and Political Science, 1992. 1, 112

650. J. D. Gannon. An experimental evaluation of data type conversions. *Communications of the ACM*, 20(8):584–595, Aug. 1977. 205

651. Z. Gao, Y. Liang, M. B. Cohen, A. M. Memon, and Z. Wang. Making system user interactive tests repeatable: When and what should we control? In *Proceedings of the 37th International Conference on Software Engineering*, ICSE'15, pages 55–65, May 2015. 174

652. M. K. Gardner, E. Z. Rothkopf, R. Lapan, and T. Laferty. The word frequency effect in lexical decision: Finding a frequency-based component. *Memory & Cognition*, 15(1):24–28, 1987. 195

653. M. R. Garman. The generalizability of private sector research on software project management in two USAF organizations: An exploratory study. Thesis (m.s.), Air Force Institute of Technology, USA, Mar. 2003. 139

654. R. Garner and F. R. Dill. The legendary IBM 1401 data processing system. *IEEE Solid-State Circuits Magazine*, 2(1):28–39, Jan. 2010. 105

655. V. Garousi, M. Borg, and M. Oivo. Cut to the chase: Revisiting the relevance of software engineering research. In *eprint arXiv:cs.SE/1812.01395*, Dec. 2018. 9

656. Gartner. Worldwide smartphone sales. https://en.wikipedia.org/wiki/Mobile_operating_system, July 2017. 5, 93

657. J. Gascoigne. Introducing open salaries at buffer our transparent formula and all individual salaries buffer. company website, Dec. 2013. https://open.buffer.com/introducing-open-salaries-at-buffer-including-our-transparent-formula-and-all-individual-salaries. 71

658. B. Gates. Shell plans - iShellBrowser. Plaintiff's Exhibit 2151, JOE COMES, RILEY PAINT, INC., SKEFFINGON'S FORMAL WEAR, INC., PATRICIA ANNE LARSEN vs. MICROSOFT CORPORATION; IOWA District Court for Polk County, Oct. 1994. 117

659. D. C. Gause and G. M. Weinberg. *Exploring Requirements: Quality before design*. Dorset House Publishing, 1989. 135

660. G. Gay, A. Rajan, M. Staats, M. Whalen, and M. P. E. Heimdahl. The effect of program and model structure on the effectiveness of MC/DC test adequacy coverage. *ACM Transactions on Software Engineering and Methodology*, 25(3):25, Aug. 2016. 174

661. J. E. Gayek, L. G. Long, K. D. Bell, R. M. Hsu, and R. K. Larson. Software cost and productivity model. Technical Report ATR-2004(8311)-1, Aerospace Corporation, Feb. 2004. 84

662. Gcc releases. organization website, July 2019. https://gcc.gnu.org/releases.html. 95, 118

663. Y. Ge and B. Xu. Dynamic staffing and rescheduling in software project management: A hybrid approach. *PLoS ONE*, 11(6):e0157104, June 2016. 130, 132

664. Y. Geffen and S. Maoz. On method ordering. In *IEEE 24th International Conference on Program Comprehension*, ICPC'16, pages 1–10, May 2016. 209

665. W. Gellerich, M. Kosiol, and E. Ploedereder. Where does GOTO go to? In *Reliable Software Technology – Ada-Europe 1996*, volume 1088 of *LNCS*, pages 385–395. Springer, 1996. 204

666. S. A. Gelman and E. M. Markman. Categories and induction in young children. *Cognition*, 23:183–209, 1986. 41

667. D. Gentner and S. Goldin-Meadow. *Language In Mind: Advances in the Study of Language and Thought*. The MIT Press, 2003. 201

668. S. L. Gerhart and L. Yelowitz. Observations of fallibility in applications of modern programming methodologies. *IEEE Transactions on Software Engineering*, SE-2(3):195–207, Sept. 1976. 168

669. M. Gerlach, B. Farb, W. Revelle, and L. A. N. Amaral. A robust data-driven approach identifies four personality types across four large data sets. *Nature Human Behaviour*, 2(10):735–742, Sept. 2018. 52

670. D. M. German, B. Adams, and A. E. Hassan. Continuously mining distributed version control systems: An empirical study of how Linux uses git. *Empirical Software Engineering*, 21(1):260–299, Feb. 2016. 4, 378

671. D. M. German and J. M. González-Barahona. An empirical study of the reuse of software licensed under the GNU general public license. In *The 5th International Conference on Open Source Systems*, OSS 2009, pages 185–198, June 2009. 68

672. D. M. German, Y. Manabe, and K. Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE'10, pages 437–446, Apr. 2010. 69

673. E. H. Gibbs, G. A. Munroe, A. M. Zeman, and C. T. Cottingham. JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated and the general public, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, case no. 1-05-CV-039231, filing #g-43414. Opinion, Superior court of the state of California for the county of Santa Clara, 2012. 366

674. E. Gibson. Linguistic complexity: locality of syntactic dependencies. *Cognition*, 68(1):1–76, Aug. 1998. 187

675. E. Gibson and J. Thomas. Memory limitations and structured forgetting: The perception of complex ungrammatical sentences as grammatical. *Language and Cognitive Processes*, 14(3):225–248, 1999. 191

676. G. Gigerenzer. Striking a blow for sanity in theories of rationality. In M. Augier and J. G. March, editors, *Models of a man: Essays in memory of Herbert A. Simon*, pages 389–409. The MIT Press, May 2004. 53

677. G. Gigerenzer. *Rationality for Mortals-How People cope with Uncertainty*. Oxford University Press, 2008. 43, 266

678. G. Gigerenzer, W. Gaissmaier, E. Kurz-Milcke, L. M. Schwartz, and S. Woloshin. Helping doctors and patients make sense of health statistics. *Psychological Science in the Public Interest*, 8(2):53–96, Apr. 2008. 224

679. G. Gigerenzer, S. Krauss, and O. Vitouch. The null ritual: What you always wanted to know about significance testing but were afraid to ask. In D. Kaplan, editor, *The Sage handbook of quantitative methodology for the social sciences*, chapter 21, pages 391–408. Sage Publications, Inc, 2004. 267

680. G. Gigerenzer, P. M. Todd, and The ABC Research Group. *Simple Heuristics That Make Us Smart*. Oxford University Press, 1999. 20, 43, 53

681. B. Gilchrist and R. E. Weber. Employment of trained computer personnel–A quantitative survey. In *Proceedings of the Spring Joint Computer Conference*, AFIPS'72, pages 641–648, May 1972. 108

682. J. Gimpel. Software that checks software: The impact of PC-lint. *IEEE Software*, 31(1):15–19, Jan.-Feb. 2014. 144

683. V. Girotto, A. Mazzocco, and A. Tasso. The effect of premise order on conditional reasoning: a test of the mental model theory. *Cognition*, 63:1–28, 1997. 45

684. M. Givon, V. Mahajan, and E. Muller. Software piracy: Estimation of lost sales and the impact on software diffusion. *Journal of Marketing*, 59(1):29–37, Jan. 1995. 88, 331, 332

685. T. J. Glauthier. Computer time sharing: Its origins and development. *Computers and Automation*, 16(10):23–27, Oct. 1967. 1

686. A. Glenberg. Few believe the world is flat: How embodiment is changing the scientific understanding of cognition. *Canadian Journal of Experimental Psychology*, 69(2):165–171, June 2015. 22

687. F. Gobet. *Understanding Expertise: A Multi-disciplinary Approach*. Palgrave, 2016. 39

688. D. R. Godden and A. D. Baddeley. Context-dependent memory in two natural environments: On land and underwater. *British Journal of Psychology*, 66(3):325–331, 1975. 33

689. M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In 16^{th} *International Conference on Software Maintenance*, ICSM'00, pages 131–142, Oct. 2000. 291

690. M. W. Godfrey and L. Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering*, 31(2):166–181, Feb. 2005. 210

691. A. L. Goel. An experimental investigation into software reliability. Final Technical Report RADC-TR-88-213, CASE Center, Syracuse University, Oct. 1988. 163

692. M. Goeminne and T. Mens. Towards a survival analysis of database framework usage in Java projects. In *31st International Conference on Software Maintenance and Evolution*, ICSME 2015, pages 551–556, Sept.-Oct. 2015. 344

693. S. S. Gokhale and R. E. Mullen. The marginal value of increased testing: An empirical analysis using four code coverage measures. *Journal of the Brazilian Computer Society*, 12(3):13–30, Dec. 2006. 175

694. M. M. Gold. A methodology for evaluating time-shared computer system usage. Technical report, Carnegie Mellon University, Aug. 1967. 118

695. K. Goldberg, T. Roeder, D. Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. In *Information Retrieval*, 4, pages 133–151, July 2001. 234

696. L. R. Goldberg. An alternative "description of personality": The big-five factor structure. *Journal of Personality and Social Psychologs*, 59(6):1216–12297, 1990. 52

697. L. R. Goldberg, J. A. Johnson, H. W. Eber, R. Hogan, M. C. Ashton, C. R. Cloninger, and H. G. Gough. The international personality item pool and the future of public-domain personality measures. *Journal of Research in Personality*, 40(1):84–96, 2006. 52

698. M. S. Goldberg and A. Touw. Statistical methods for learning curves and cost analysis. Technical Report CIMD0006870.A3/1Rev, The CNA Corporation, Mar. 2003. 76

699. H. H. Goldstine and J. von Neumann. Planning and coding of problems for an electronic computing instrument. Technical Report Part II, Volume 1-3, Institute for Advanced Study, Princeton, Apr. 1947. 131

700. P. Golle. Revisiting the uniqueness of simple demographics in the US population. In *Proceedings of the 5th ACM workshop on Privacy in electronic society*, WPES'06, pages 77–80, Oct. 2006. 378

701. R. W. Gomulkiewicz. Enforcement of open source software licenses: The MDY trio's inconvenient compliations. *Yale Journal of Law & Technology*, 14:106–137, 2011. 70

702. I. R. Gonzaga, Jr. Empirical studies on fine-grained feature dependencies. Thesis (m.s.), Universidade Federal de Alagoas, Instituto de Computação, Aug. 2015. 207

703. R. Gonzalez and G. Wu. On the shape of the probability weighting function. *Cognitive Psychology*, 38(1):129–166, Feb. 1999. 51

704. J. M. González-Barahona, G. Robles, I. Herraiz, and F. Ortega. Studying the laws of software evolution in a long-lived FLOSS project. *Journal of Software: Evolution and Process*, 26(7):589–612, July 2014. 220, 257, 258, 318, 333

705. B. H. Good, Y.-A. de Montjoye, and A. Clauset. The performance of modularity maximization in practical contexts. In *eprint arXiv:physics.data-an/0910.0165v2*, Apr. 2010. 249

706. J. Goodman. Lessons learned from seven Space Shuttle missions. NASA Contractor Report CR-2007-213697, Lyndon B. Johnson Space Center, Jan. 2007. 148

707. P. Goodridge, J. Haskel, and G. Wallis. Estimating UK investment in intangible assets and intellectual property rights. Technical Report No. 2014/36, Intellectual Property Office, UK government, Sept. 2014. 6

708. Google books ngram dataset. corporate website, 2015. http://storage.googleapis.com/books/ngrams/books/datasetsv2.html. 385

709. A. Gopal and B. R. Koka. The role of contracts on quality and returns to quality in offshore software development outsourcing. *Decision Sciences*, 41(3):491–516, Aug. 2010. 124

710. R. Gopinath. *On the Limits of Mutation Analysis*. PhD thesis, Oregon State University, June 2017. 175

711. R. Gopinath, A. Alipour, I. Ahmed, C. Jensen, and A. Groce. How hard does mutation analysis have to be, anyway? In *IEEE 26th International Symposium on Software Reliability Engineering*, ISSRE'15, pages 216–227, Nov. 2015. 175, 235

712. R. Gopinath, C. Jensen, and A. Groce. Code coverage for suite evaluation by developers. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE'14, pages 72–82, June 2014. 174, 175, 306

713. R. Gopinath, C. Jensen, and A. Groce. Mutations: How close are they to real faults? In *IEEE 25th International Symposium on Software Reliability Engineering*, ISSRE'14, pages 189–200, Nov. 2014. 164, 175

714. R. D. Gordon and M. H. Halstead. An experiment comparing Fortran programming times with the software physics hypothesis. Technical Report TR 167, Purdue University, Oct. 1975. 182

715. R. J. Gordon. The postwar evolution of computer prices. Working Paper No. 2227, National Bureau of Economic Research, USA, Apr. 1987. 5

716. K. Goševa-Popstojanova, M. Hamill, and R. Perugupalli. Large empirical case study of architecture-based software reliability. In *Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering*, ISSRE'05, pages 43–52, Nov. 2005. 248

717. M. Gottscho. ViPZonE: Exploiting DRAM power variability for energy savings in Linux x86-64. Thesis (m.s.), Electrical Engineering, UCLA, Mar. 2014. 371

718. M. Gottscho, A. A. Kagalwalla, and P. Gupta. Power variability in contemporary DRAMs. *IEEE Embedded Systems Letters*, 4(12):37–40, June 2012. 371

719. S. Götz, T. Ilsche, J. Cardoso, J. Spillner, U. Aßmann, W. Nagel, and A. Schill. Energy-efficient data processing at sweet spot frequencies. In *OTM Workshops*, 2014, pages 154–171, Apr. 2014. 368

720. G. Gousios and A. Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR'14, pages 368–371, May 2014. 217

721. G. Gousios, A. Zaidman, M.-A. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE'15, pages 358–368, May 2015. 222, 223

722. E. M. Grabbe, S. Ramo, and D. E. Wooldridge. *Handbook of Automation, Computation, and Control, Volume 2: Computers and Data Processing*. John Wiley & Sons, Inc, 1959. 113

723. P. Grady. *Termination of the SIREN ICT project*. Grant Thornton UK LLP, June 2014. 132

724. R. B. Grady and D. L. Caswell. *Software Metrics: Establishing a company-wide program*. Prentice-Hall, Inc, 1987. 151

725. A. C. Graesser, S. B. Woll, D. J. Kowalski, and D. A. Smith. Memory for typical and atypical actions in scripted activities. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):503–515, June 1980. 189, 190

726. S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière. Autotuning for floating-point precision with discrete stochastic arithmetic. HAL Id: hal-01331917, HAL archives-ouvertes.fr, June 2016. 150

727. E. E. Grant and H. Sackman. An exploratory investigation of programmer performance under on-line and off-line conditions. *IEEE Transactions on Human Factors in Electronics*, 8(1):33–48, Mar. 1967. 10, 57, 275

728. Graphviz-graph visualization software. organization website, 2015. http://www.graphviz.org. 222

729. C. A. Graver, W. M. Carriere, E. E. Balkovich, and R. Thibodeau. Cost reporting elements and activity cost tradeoffs for defense system software (study results). Technical Report ESD-TR-77-262, Vol. 1, General Research Corporation, May 1977. 133

730. D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. The misuse of the NASA metrics data program data sets for automated software defect prediction. In *15th Annual Conference on Evaluation & Assessment in Software Engineering 2011*, EASE 2011, pages 96–103, Apr. 2011. 378

731. J. Gray, C. Nyberg, M. Shah, and N. Govindaraju. Sort benchmark. http://sortbenchmark.org, July 2014. 366

732. K. Gray, D. G. Rand, E. Ert, K. Lewis, S. Hershman, and M. I. Norton. The emergence of "us and them" in 80 lines of code: Modeling group genesis in homogeneous populations. *Association for Psychological Science*, 25(4):982–990, Apr. 2014. 78

733. W. D. Gray, C. R. Sims, W.-T. Fu, and M. J. Schoelles. The soft constraints hypothesis: A rational analysis approach to resource allocation for interactive behavior. *Psychological Review*, 113(3):461–482, July 2006. 25

734. M. Grechanik, C. McMillan, L. DeFerrari, M. Comi, S. Crespi, D. Poshyvanyk, C. Fu, Q. Xie, and C. Ghezzi. An empirical investigation into a large-scale Java open source code repository. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'10, pages 11:1–11:10, Sept. 2010. 200

735. J. H. Greenberg. Some universals of grammar with particular reference to the order of meaningful elements. In J. H. Greenberg, editor, *Universals of Language*, chapter 5, pages 58–90. The MIT Press, 1963. 45

736. H. I. Greenfield. An economist looks at data processing. *Computers and Automation*, 6(10):18–23, Oct. 1957. 105

737. S. Greenstein. Did computer technology diffuse quickly?: Best and average practice in mainframe computers, 1968-1983. Working Paper No. 4647, National Bureau of Economic Research, USA, Feb. 1994. 99

738. C. Gregg and K. Hazelwood. Where is the data? Why you cannot debate CPU vs. GPU performance without the answer. In *IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS, pages 134–144, Apr. 2011. 361

739. P. Grice. *Studies in the Way of Words*. Harvard University Press, 1989. 178

740. D. A. Grier. The ENIAC, the verb "to program" and the emergence of digital computers. *IEEE Annals of the History of Computing*, 18(I):51–55, 1996. 1

741. D. A. Grier. *When Computers were Human*. Princeton University Press, 2005. 1, 92

742. S. Grimstad and M. Jørgensen. Inconsistency of expert judgement-based estimates of software development effort. *Journal of Systems and Software*, 80(11):1770–1777, Nov. 2007. 126

743. R. E. Griswold, J. F. Poage, and I. P. Polonsky. *The SNOBOL 4 Programming Language*. Prentice-Hall, Inc, second edition, 1968. 172

744. E. Grochowski and R. E. Fontana, Jr. Future technology challenges for NAND flash and HDD products. Flash Memory Summit 2012, Santa Clara, CA, July 2012. 318

745. U. Grömping. Relative importance for linear regression in R: The package `relaimpo`. *Journal of Statistical Software*, 17(1):1–27, Sept. 2006. 309

746. E. H. B. M. Gronenschild, P. Habets, H. I. L. Jacobs, R. Mengelers, N. Rozendaal, J. van Os, and M. Marcelis. The effects of FreeSurfer version, workstation type, and Macintosh operating system version on anatomical volume and cortical thickness measurements. *PLoS ONE*, 7(6):e38234, June 2012. 150

747. H. R. J. Grosch. High speed arithmetic: The digital computer as a research tool. *Journal of the Optical Society of America*, 43(4):306–310, Apr. 1953. 1

748. A. S. Grove. *Only the Paranoid Survive: How to Exploit the Crisis Points That Challenge Every Company and Career*. HarperCollins-Busines, Apr. 1988. 92

749. A. Grübler and N. Nakićenović. Long waves, technology diffusion, and substitution. Technical Report RP-91-17, International Institute for Applied Systems Analysis Laxenburg, Austria, Oct. 1991. 2

750. W. Gruhl. Lessons learned cost/schedule assessment guide. Slides of talk, July 199? 218

751. M. Gubler. *Protean and boundaryless career orientations - an empirical study of IT professionals in Europe*. PhD thesis, Loughborough University, July 2011. 71

752. T. Gue. Triggering infection: Distribution and derivative works under the GNU general public license. *Journal of Law, Technology & Policy*, 2012(1):95–140, 2012. 68

753. L. Guerrouj, M. Di Penta, Y.-G. Guéhéneuc, and G. Antoniol. An experimental investigation on the effects of context on source code identifiers splitting and expansion. *Empirical Software Engineering*, 19(6):1706–1753, Dec. 2014. 362

754. A. Gunasekaran, E. W. T. Ngai, and R. E. McGaughey. Information technology and systems justification: A review for research and applications. *European Journal of Operational Research*, 173(3):957–983, Sept. 2006. 119

755. H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What bugs live in the cloud? A study of 3000+ issues in cloud systems. In *Proceedings of the 5th ACM Symposium on Cloud Computing*, SOCC'14, pages 1–14, Nov. 2014. 279

756. H. S. Gunawi, C. Rubio-González, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and B. L. Liblit. EIO: Error handling is Occasionally correct. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies*, FAST'08, pages 207–222, Feb. 2008. 163

757. N. J. Gunther. A simple capacity model of massively parallel transaction systems. In *Proceedings of the 19th International CMG Conference*, pages 1035–1044, Dec. 1993. 227

758. N. J. Gunther. *Analysing Computer System Performance with Perl::PDQ*. Springer-Verlag, 2005. 227, 366

759. O. Gurevich, M. A. Johnson, and A. E. Goldberg. Incidental verbatim memory for language. *Language and Cognition*, 2(1):45–78, May 2010. 188

760. R. K. Guy. The strong law of small numbers. *American Mathematical Monthly*, 95(8):697–712, Oct. 1988. 255

761. E. A. E. Habib. Geometric mean for negative and zero values. *International Journal of Research & Reviews in Applied Sciences*, 11(3):419–432, June 2012. 262

762. Hackerone. The 2018 hacker report. Technical report, hackerone, Dec. 2017. 68

763. J. Haidt. *The Righteous Mind*. Vintage books, 2012. 44

764. S. Haine. As low as reasonably practicable (ALARP) risk-informed decision framework applied to public utility safety. Staff white paper, California Public Utilities Commission, Dec. 2015. 147

765. A. G. Haldane and R. Davies. The short long. Speech, May 2011. 29th Société Universitaire Européene de Recherches Financiéres Colloquium. 107

766. A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, and B. Baudry. Test them all, is it worth it? A ground truth comparison of configuration sampling strategies. In *eprint arXiv:cs.SE/1710.07980*, Oct. 2017. 140, 169

767. T. Halkjelsvik and M. Jørgensen. *Time Predictions: Understanding and Avoiding Unrealism in Project Planning and Everyday Life*. Springer International Publishing AG, Apr. 2018. 129

768. B. H. Hall and M. MacGarvie. The private value of software patents. *Research Policy*, 39(7):994–1009, Sept. 2010. 68

769. T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, Nov. 2012. 378

770. M. H. Halstead. A theoretical relationship between mental work and machine language programming. Technical Report CSD-TR 67, Purdue University, Feb. 1972. 182

771. M. H. Halstead and P. M. Zislis. Experimental verification of two theorems of software physics. Technical Report TR 97, Purdue University, June 1973. 182

772. D. Z. Hambrick, F. L. Oswald, E. M. Altmann, E. J. Meinz, F. Gobet, and G. Campitelli. Deliberate practice: Is that all it takes to become an expert? *Intelligence*, 45(1):34–45, July-Aug. 2014. 39

773. M. Hamill and K. Goševa-Popstojanova. Exploring the missing link: an empirical study of software fixes. *Software Testing, Verification and Reliability*, 24(8):684–705, Dec. 2014. 223, 224

774. M. T. Hannan and G. R. Carroll. *Dynamics of Organizational Populations: Density, Legitimation, and Competition*. Oxford University Press, Jan. 1992. 99

775. J. E. Hannay, D. I. K. Sjøberg, and T. Dybå. A systematic review of theory use in software engineering experiments. *IEEE Transactions on Software Engineering*, 33(2):87–107, Feb. 2007. 8

776. M. Harchol-Balter and A. B. Downey. Exploiting process lifetime distributions for dynamic load balancing. Report No. UCB/CSD-95-887, Computer Science Division, University of California Berkeley, Nov. 1995. 112

777. D. Harhoff, B. H. Hall, G. von Graevenitz, K. Hoisl, S. Wagner, A. Gambardella, and P. Giuri. The strategic use of patents and its implications for enterprise and competition policies. Final Report ENTR/05/82, DG Enterprise, European Commission, July 2007. 68

778. B. R. Harmon and N. I. Om. Schedule assessment methods for ballistic missile defense ground-based software development. IDA Paper P-3600, Institute for Defense Analyses, Aug. 2003. 128

779. N. Harrand, S. Allier, M. Rodriguez-Cancio, M. Monperrus, and B. Baudry. A journey among Java neutral program variants. In *eprint arXiv:cs.SE/1901.02533*, Jan. 2019. 179

780. A. Hart, L. Maxim, M. Siegrist, N. Von Goetz, C. da Cruz, C. Merten, O. Mosbach-Schulz, M. Lahaniatis, A. Smith, and A. Hardy. Guidance on communication of uncertainty in scientific assessments. *EFSA Journal*, 17(1):5520, Jan. 2019. 230

781. T. Harter, C. Dragga, M. Vaughn, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. A file is not a file: Understanding the I/O behavior of Apple desktop applications. *ACM Transactions on Computer Systems*, 30(3):10, Aug. 2012. 373, 375

782. J. Haskel and S. Westlake. *Capitalism without Capital: The Rise of the Intangible Economy*. Princeton University Press, 2018. 6, 61

783. H. Hata, C. Treude, R. G. Kula, and T. Ishio. 9.6 million links in source code comments: Purpose, evolution, and decay. In *eprint arXiv:cs.SE/1901.07440*, Jan. 2019. 118

784. L. Hatton. *Safer C : Developing Software for High-integrity and Safety-critical Systems*. McGraw-Hill, 1995. 184

785. L. Hatton. Reexamining the fault density-component size connection. *IEEE Software*, 14(2):89–97, Mar. 1997. 229

786. L. Hatton. How accurately do engineers predict software maintenance tasks? *Computer*, 40(2):64–69, Feb. 2007. 144, 145, 220

787. M. D. Hauser, S. Carey, and L. B. Hauser. Spontaneous number representation in semi-free-ranging rhesus monkeys. *Proceedings of the Royal Society B: Biological Sciences*, 267(1445):829–833, Apr. 2000. 20

788. J. P. Haverty and R. L. Patrick. Programming languages and standardization in command and control. Research Memorandum RM-3447-PR, The RAND Corporation, Jan. 1963. 113

789. D. M. Hawkins. *Identification of Outliers*. Springer, 1980. 379

790. G. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. Context effects in multi-alternative decision making: Empirical data and a Bayesian model. *Cognitive Science*, 36(3):498–516, Apr. 2012. 59

791. G. E. Hawkins, S. D. Brown, M. Steyvers, and E.-J. Wagenmakers. An optimal adjustment procedure to minimize experiment time in decisions with multiple alternatives. *Psychonomic Bulletin & Review*, 19(2):339–348, Apr. 2012. 362

792. J. A. Hawkins. *Efficiency and Complexity in Grammars*. Oxford University Press, 2007. 187

793. B. Hayes. Third base. *American Scientist*, 89(6):490–494, 2001. 8

794. S. Hazelhurst. Truth in advertising: Reporting performance of computer programs, algorithms and the impact of architecture and systems environment. *South African Computer Journal*, 46:24–37, Dec. 2010. 316

795. M. L. Head, L. Holman, R. Lanfear, A. T. Kahn, and M. D. Jennions. The extent and consequences of p-hacking in science. *PLoS Biology*, 13(3):e1002106, Mar. 2015. 268

796. S. Head and J. Nelson. Data rights valuation in software acquisitions. Technical Report DRM-2012-001825-Final, CNA Analysis & Solutions, Sept. 2012. 70

797. A. Heathcote, S. Brown, and D. J. K. Mewhort. The power law repealed: The case for an exponential law of practice. *Psychonomic Bulletin & Review*, 7(2):185–207, Apr. 2000. 36

798. R. Heeks. The uneven profile of Indian software exports. Working Paper No. 3, University of Manchester, Oct. 1998. 62

799. J. Heer and M. Bostock. Crowdsourcing graphical perception: Using Mechanical Turk to assess visualization design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 2010, pages 203–212, Apr. 2010. 225

800. K. Heinze, N. Claussen, and V. LaBolle. Management of computer programming for command and control systems: A survey. Technical Memorandum TM-903/000/02, System Development Corporation, Santa Monica, May 1963. 57

801. D. H. Helmer, S. Mackay, K. Selvey-Clinton, R. Yoon, and H. Furukawa. Worldwide capital and fixed assets guide 2016. Technical Report EYG no. DL1528, EYGM Limited, 2016. 84

802. D. R. Helsel. *Statistics for Censored Environmental Data using Minitab and R*. John Wiley & Sons, second edition, 2012. 336

803. A. Hemel and R. Koschke. Reverse engineering variability in source code using clone detection-A case study for Linux variants of consumer electronic devices. In *19th Working Conference on Reverse Engineering*, WCRE'12, pages 357–366, Oct. 2012. 97

804. M. Hendrickson. 2010 state of the computer book market. company website, Feb. 2011. http://radar.oreilly.com/2011/02/2010-book-market-1.html. 114

805. A. Henik and J. Tzelgov. Is three greater than five: The relation between physical and semantic size in comparison tasks. *Memory & Cognition*, 10(4):389–395, 1982. 50

806. J. Henrich. How adaptive cultural processes can produce maladaptive losses–The Tasmanian case. *American Antiquity*, 69(2):197–214, Apr. 2004. 77

807. J. Henrich. The evolution of costly displays, cooperation and religion: credibility enhancing displays and their implications for cultural evolution. *Evolution and Human Behavior*, 30(4):244–260, July 2009. 74

808. J. Henrich. *The Weirdest People in the World: How the West Became Psychologically Peculiar and Particularly Prosperous*. Allen Lane, Sept. 2020. 21

809. J. Henrich, M. Chudek, and R. Boyd. The big man mechanism: how prestige fosters cooperation and creates prosocial leaders. *Philosophical Transactions of The Royal Society B*, 370(1683), Dec. 2015. 119

810. J. Henrich and F. J. Gil-White. The evolution of prestige: Freely conferred deference as a mechanism for enhancing the benefits of cultural transmission. *Evolution and Human Behavior*, 22(3):165–196, May 2001. 75

811. J. Henrich, S. J. Heine, and A. Norenzayan. The weirdest people in the world? Working Paper No. 139, German Data Forum (RatSWD), Apr. 2010. 21

812. J. Henrich and R. McElreath. Are peasants risk-averse decision makers? *Current Anthropology*, 43(1):172–181, Feb. 2002. 53

813. I. Heras-Saizarbitoria and O. Boiral. Symbolic adoption of ISO 9000 in small and medium-sized enterprises: The role of internal contingencies. *International Small Business Journal*, 33(3):299–320, May 2015. 149

814. S. Herculano-Houzel. The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost. *PNAS*, 109(1):10661–10668, June 2012. 19

815. F. Hermans and E. Murphy-Hill. Enron's spreadsheets and related emails: A dataset and analysis. In *Proceedings of the 37th International Conference on Software Engineering-Volume 2*, ICSE'15, pages 7–16, May 2015. 179

816. T. Herr, B. Schneier, and C. Morris. Taking stock: Estimating vulnerability rediscovery. Paper, Belfer Center for Science and International Affairs, Harvard Kennedy School, Oct. 2017. 160

817. I. Herraiz Tabernero. *A statistical examination of the properties and evolution of libre software*. PhD thesis, Universidad Rey Juan Carlos, Oct. 2008. 180, 283, 325, 334

818. E. Herrmann, J. Call, M. V. Hernández-Lloreda, B. Hare, and M. Tomasello. Humans have evolved specialized skills of social cognition: The cultural intelligence hypothesis. *Science*, 317(5843):1360–1366, Sept. 2007. 52, 75

819. R. Hersh. *18 Unconventional Essays on the Nature of Mathematics*. Springer, 2006. 149

820. K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE'13, pages 392–401, May 2013. 4, 149, 152, 379

821. K. Herzig and A. Zeller. The impact of tangled code changes. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pages 121–130, May 2013. 379

822. T. Hesterberg. What teachers should know about the bootstrap: Resampling in the undergraduate statistics curriculum. In *eprint arXiv:stat.OT/1411.5279*, Nov. 2014. 268

823. R. J. Heuer, Jr. *Psychology of Intelligence Analysis*. Central Intelligence Agency, 1999. 213

824. M. Hicks, C. O'Malley, S. Nichols, and B. Anderson. Comparison of 2D and 3D representations for visualising telecommunication usage. *Behaviour & Information Technology*, 22(3):185–201, May 2003. 225

825. E. T. Higgins. Value from regulatory fit. *Current Directions in Psychological Science*, 14(4):209–213, 2005. 24

826. N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 1996. 147

827. M. Hilbert and P. López. Supporting online material for: The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, Apr. 2011. 94

828. B. M. Hill and A. Monroy-Hernández. A longitudinal dataset of five years of public activity in the Scratch online community. *Scientific Data*, 4(170002), Jan. 2017. 179

829. T. P. Hill. An evolutionary theory for the variability hypothesis. In *eprint arXiv:q-bio.PE/1703.04184*, Aug. 2018. 21

830. T. T. Hills, P. M. Todd, and M. N. Jones. Foraging in semantic fields: How we search through memory. *Topics in Cognitive Science*, 7(3):513–534, July 2015. 33

831. D. J. Hilton. The social context of reasoning: Conversational inference and rational judgment. *Psychological Bulletin*, 118(2):248–271, 1995. 44

832. A. Hindle, M. W. Godfrey, and R. C. Holt. Reading beside the lines: Indentation as a proxy for complexity metrics. In *The 16th IEEE International Conference on Program Comprehension*, ICPC 2008, pages 133–142, June 2008. 328, 329

833. T. Hirao, A. Ihara, Y. Ueda, P. Phannachitta, and K. ichi Matsumoto. The impact of a low level of agreement among reviewers in a code review process. In *IFIP International Conference on Open Source Systems*, OSS 2016, pages 97–110, May-June 2016. 169

834. S. C. Hirtle and J. Jonides. Evidence for hierarchies in cognitive maps. *Memory & Cognition*, 13(3):208–217, 1985. 50

835. C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–583, Oct. 1969. 148

836. M. Hocko and T. Kalibera. Reducing performance non-determinism via cache-aware page allocation strategies. In *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, WOSP/SIPEW'10, pages 223–234, Jan. 2010. 372

837. A. Höfer. Exploratory comparison of expert and novice pair programmers. *Computing and Informatics*, 29(1):73–91, 2010. 304

838. E. Hoffer. *The True Believer: Thoughts on the Nature of Mass Movements*. HarperPerennial, 1951. 99

839. D. D. Hoffman. *Visual Intelligence: How We Create What We See*. W. W. Norton, 2000. 19, 26

840. R. Hofman. *Behavioral Products Quality Assessment Model on the Software Market*. PhD thesis, Poznan University of Economics, Oct. 2011. 139

841. J. Hofmeister, J. Siegmund, and D. V. Holt. Shorter identifier names take longer to comprehend. In *24th International Conference on Software Analysis, Evolution and Reengineering*, SANER 2017, pages 217–227, Feb. 2017. 197

842. G. Hofstede. *Culture's Consequences: International Differences in Work-Related Values*. Sage Publications, abridged edition, 1984. 73

843. R. M. Hogarth and H. J. Einhorn. Order effects in belief updating: The belief-adjustment model. *Cognitive Psychology*, 24(1):1–55, Jan. 1992. 38, 39

844. R. M. Hogarth, C. R. M. McKenzie, B. J. Gibbs, and M. A. Marquis. Learning from feedback: Exactness and incentives. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 17(4):734–752, 1991. 39

845. B. D. Holbrook and W. S. Brown. A history of computing research at Bell Laboratories (1937-1975). Computing Science Technical Report No. 99, AT&T Bell Laboratories, 1982. 92

846. M. Holdway. An alternative methodology: Valuing quality change for microprocessors in the PPI. In *Issues in Measuring Price Change and Consumption*. Bureau of Labor Statistics, June 2000. 7

847. W. B. Holland. Soviet cybernetics technology: viii. Report on the algorithmic language ALGEC (final version). Research Memorandum RM-5136-PR, The RAND Corporation, Dec. 1966. 107

848. J. K. Hollmann. Estimate accuracy: Dealing with reality. *Cost Engineering Journal*, 54(6):17–27, Nov.-Dec. 2012. 125

849. Hood & Strong. Mozilla foundation and subsidiary december 31, 2015 and 2014. Independent auditors' report and consolidated financial statements, Hood & Strong LLC, Nov. 2016. 121

850. A. A. Hook, B. Brykczynski, C. W. McDonald, S. H. Nash, and C. Youngblut. A survey of computer programming languages currently used in the Department of Defense. IDA Paper P-3054, Institute for Defense Analyses, Jan. 1995. 114

851. R. Hoosain. Correlation between pronunciation speed and digit span size. *Perception and Motor Skills*, 55:1128–1128, 1982. 362

852. R. Hoosain and F. Salili. Language differences, working memory, and mathematical ability. In M. M. Grunberg, P. E. Morris, and R. N. Sykes, editors, *Practical aspects of memory: Current research and issues*, volume 2, pages 512–517. John Wiley & Sons, Inc, 1988. 31

853. M. Hoppe and S. Hanenberg. Do developers benefit from generic types? An empirical comparison of generic and raw types in Java. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA'13, pages 457–474, Oct. 2013. 206

854. W. Hoppitt and K. N. Laland. *Social Learning: An Introduction to Mechanisms, Methods, and Models*. Princeton University Press, July 2013. 75

855. W. Hordijk, M. L. Ponisio, and R. Wieringa. Harmfulness of code duplication a structured review of the evidence. In 13th *International Conference on Evaluation and Assessment in Software Engineering*, EASE'09, pages 88–97, Apr. 2009. 82

856. V. Horký. *Performance Awareness in Agile Software Development*. PhD thesis, Charles University, Faculty of Mathematics and Physics, Mar. 2018. 138

857. Z. Horne, M. Muradoglu, and A. Cimpian. Explanation as a cognitive process. *Trends in Cognitive Sciences*, 23(3):187–199, Mar. 2019. 186

858. M. R. Horton. *Portable C Software*. Prentice-Hall, Inc, Upper Saddle River, NJ 07458, USA, 1990. 184

859. S. Hossenfelder. *Lost in Math: How Beauty Leads Physics Astray*. Basic Books, June 2018. 12

860. D. A. Hounshell. *From the American System to Mass Production 1800-1932: The Development of Manufacturing Technology in the United States*. The Johns Hopkins University Press, 1984. 101

861. A. D. Householder and J. M. Foote. Probability-based parameter selection for black-box fuzz testing. Technical Note CMU/SEI-2012-TN-019, Software Engineering Institute, Carnegie Mellon University, Aug. 2012. 173

862. M. W. Howard and M. J. Kahana. Context variability and serial position effects in free recall. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 25(4):923–941, 1999. 34

863. J. Howison and J. B. Herbsleb. Incentives and integration in scientific software production. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW'13, pages 459–470, Mar. 2013. 74

864. L. Hribar, S. Bogovac, and Z. Marinčić. Implementation of fault slip through in design phase of the project. In *miproBIS 2008: International Conference on Business Intelligence Systems*, May 2008. 168

865. H. Hsu. *The Appsmiths: Community, Identity, Affect and Ideology Among Cocoa Developers From NeXT to Iphone*. PhD thesis, Graduate School of Cornell University, May 2015. 74

866. http archive. https:httparchive.org, July 2018. 98

867. X. Huang, J. Xie, N. O. Otecko, and M. Peng. Accessibility and update status of published software: Benefits and missed opportunities. *Frontiers in Research Metrics and Analytics*, 2(doi.org/10.3389/frma.2017.00001), Feb. 2017. 144

868. B. A. Huberman. The dynamics of organizational learning. *Computational & Mathematical Organization Theory*, 7(2):145–153, Aug. 2001. 76

869. H. Huijgens and R. van Solingen. Measuring best-in-class software releases. In *Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*, pages 137–146, Oct. 2013. 129

870. H. Huijgens and F. Vogelezang. Do estimators learn? On the effect of a positively skewed distribution of effort data on software portfolio productivity. Technical Report TUD-SERG-2016-004, Delft University of Technology, 2016. 77

871. J. C. Hull. *Options, Futures, and other Derivatives*. Pearson, seventh edition, Oct. 2010. 65

872. C. Hulme, S. Maughan, and G. D. A. Brown. Memory for familiar and unfamiliar words: Evidence for a long-term memory contribution to short-term memory span. *Journal of Memory and Language*, 30(6):685–701, 1991. 32

873. C. R. Hulten. Decoding Microsoft: Intangible capital as a source of company growth. Working Paper 15799, National Bureau of Economic Research, USA, Mar. 2010. 83

874. R. Hundt, E. Raman, M. Thuresson, and N. Vachharajani. MAO-an extensible micro-architectural optimizer. In *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO'11, pages 1–10, Apr. 2011. 369

875. S. Hunold and A. Carpen-Amarie. MPI benchmarking revisited: Experimental design and reproducibility. In *eprint arXiv:cs.DC/1505.07734v3*, Sept. 2015. 368

876. S. Hunold, A. Carpen-Amarie, and J. L. Träff. Reproducible MPI microbenchmarking isn't as easy as you think. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA'14, pages 69–76, Sept. 2014. 245

877. E. Hunt. The Whorfian hypothesis: A cognitive psychology perspective. *Psychological Review*, 98(3):377–389, July 1991. 201

878. J. E. Hunter, F. L. Schmidt, and M. K. Judiesch. Individual differences in output variability as a function of job complexity. *Journal of Applied Psychology*, 75(1):28–42, Feb. 1990. 57

879. M. J. Hurlstone, G. J. Hitch, and A. D. Baddeley. Memory for serial order across domains: An overview of the literature and directions for future research. *Psychonomic Bulletin & Review*, 140(2):229–373, Mar. 2014. 34

880. A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: Understanding the nature of DRAM errors and the implications for system design. In *Proceedings of the seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 111–122, Mar. 2012. 166

881. J. Hwang. *The Social Shaping of ICTs Standards: A Case of National Coded Character Set Standards Controversy in Korea*. PhD thesis, The University of Edinburgh, 2005. 106

882. S. Ibba, F. E. Pani, J. G. Stockton, G. Barabino, M. Marchesi, and D. Tigano. Incidence of predatory journals in computer science literature. *Library Review*, 66(6-7):505–522, Sept. 2017. 11

883. IBM. Specifications for the IBM mathematical FORmula TRANSlating system, FORTRAN. Programming Research Group, Applied Science Division, International Business Machines Corporation, Nov. 1954. 113

884. R. Ierusalimschy, L. H. de Figueiredo, and W. Celes. The evolution of Lua. In *Proceedings of the Third ACM SIGPLAN conference on History of Programming Languages*, HOPL III, pages 1–26, June 2007. 93

885. J. Iivonen. Identifying and characterizing highly performing testers-A case study in three software product companies. Thesis (m.s.), Helsinki University of Technology, Department of Computer Science and Engineering, Oct. 2009. 58

886. S. Ikeda, A. Ihara, R. G. Kula, and K. Matsumoto. An empirical study of README contents for JavaScript packages. *IEICE Transactions on Information & Systems*, E102-D(2):280–288, Feb. 2019. 178

887. Y. Ikutani, T. Kubo, S. Nishida, H. Hata, K. Matsumoto, K. Ikeda, and S. Nishimoto. Expert programmers have fine-tuned cortical representations of source code. In *bioRxiv doi: 10.1101/2020.01.28.923953*, Jan. 2020. 177

888. I. Imbo and J.-A. LeFevre. Cultural differences in complex addition: Efficient Chinese versus adaptive Belgians and Canadians. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 35(6):1465–1476, Nov. 2009. 49

889. I. Imbo, A. Vandierendonck, and E. Vergauwe. The role of working memory in carrying and borrowing. *Psychological Research*, 71(4):467–483, July 2007. 49

890. Information Technology Laboratory. National vulnerability database. https://nvd.nist.gov, Dec. 2014. 151, 380

891. L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE'14, pages 435–445, June 2014. 174

892. Intel. 6th generation Intel processor family. Specification update 332689-010EN, Intel Corporation, Apr. 2017. 161

893. J. P. A. Ioannidis. Contradicted and initially stronger effects in highly cited clinical research. *JAMA*, 294(2):218–228, July 2005. 4

894. J. P. A. Ioannidis. Why most published research findings are false. *PLoS Medicine*, 2(8):e124, Aug. 2005. 268

895. A. Iosup, M. Jan, O. Sonmez, and D. H. J. Epema. On the dynamic resource availability in grids. Rapport de recherche no 6172, Institut National de Recherche en Informatique et en Automatique, Apr. 2007. 167

896. G. Irlam. Unix file size survey-1993. http://www.base.com/gordoni/ufs93.html, Sept. 1993. 246, 247

897. F. Irving. Github users since service started. https://classic.scraperwiki.com/scrapers/github_users_each_year, Mar. 2016. 87

898. ISO. *ISO/IEC Guide 25:1990 General requirements for the competence of calibration and testing laboratories*. International Organization for Standardization, 1990. 172

899. ISO. *ISO/IEC 9945:2008 Information technology – Portable Operating System Interface (POSIX®)*. International Organization for Standardization, 2008. 115, 162

900. ISO SC22. *ISO/IEC 18009:1999 Information technology – Programming languages – Ada: Conformity assessment of a language processor*. International Organization for Standardization, 1990. Last reviewed and confirmed in 2015. 172

901. ISO SC22. *ISO/IEC 13210:1999 Information technology – Requirements and guidelines for test methods specifications and test method implementation for measuring conformance to POSIX standards*. International Organization for Standardization, 1999. 172

902. A. Israeli and D. G. Feitelson. The Linux kernel as a case study in software evolution. *Journal of Systems and Software*, 83(3):485–501, Mar. 2010. 182, 183, 290, 291, 296, 317

903. R. K. Iyer, S. E. Butner, and E. J. McCluskey. An exponential failure/load relationship: Results of a multi-computer statistical study. Technical Report #CRC-81-6, Computer Systems Laboratory, Stanford University, Aug. 1981. 151

904. J. L. C. Izquierdo and J. Cabot. A survey of software foundations in Open Source. In *eprint arXiv:cs.SE/2005.10063.pdf*, May 2020. 93

905. M. Y. Jaber. Learning and forgetting models and their applications. In A. B. Badiru, editor, *Handbook of Industrial and Systems Engineering*, chapter 30. CRC Press-Taylor & Francis Group, Dec. 2005. 76

906. A. N. Jackson. Formats over time: Exploring UK web history. In *eprint arXiv:cs.DL/1210.1714v1*, Oct. 2012. 111

907. P. Jackson. Opus Development Postmortem: JOE COMES, RILEY PAINT, INC., SKEFFINGON'S FORMAL WEAR, INC., PATRICIA ANNE LARSEN vs. MICROSOFT CORPORATION. Plaintiff's Exhibit 8875, IOWA District Court for Polk County, Dec. 1989. 141, 142

908. J. Jacobs and B. Rudis. *Data-Driven Security: Analysis, Visualization and Dashboards*. John Wiley & Sons, Inc, 2014. 213, 385

909. R. Jaeschke. *Portability and the C Language*. Hayden Books, 4300 West 62nd Street, Indianapolis, IN 46268, USA, 1989. 184

910. L. R. Jager and J. T. Leek. Empirical estimates suggest most published research is true. *Biostatistics*, 15(1):1–12, 2014. 268

911. B. Jamtveit, E. Jettestuen, and J. Mathiesen. Scaling properties of European research units. *PNAS*, 106(32):13160–13163, Aug. 2009. 108

912. A. R. Jansen. *Encoding and Parsing of Algebraic Expressions by Experienced Users of Mathematics*. PhD thesis, School of Computer Science and Software Engineering, Monash University, Jan. 2002. 29

913. A. R. Jansen, K. Marriott, and G. W. Yelland. Parsing of algebraic expressions by experienced users of mathematics. *European Journal of Cognitive Psychology*, 19(2):286–320, 2007. 29

914. C. J. M. Jansen and M. M. W. Pollmann. On round numbers: Pragmatic aspects of numerical expressions. *Journal of Quantitative Linguistics*, 8(3):187–201, 2001. 49

915. Y. Jansen and K. Hornbæk. A psychophysical investigation of size as a physical variable. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):479–488, Jan. 2016. 225

916. J. J. Jenkins. Remember that old theory of memory? Well, forget it! *American Psychologist*, 29(11):785–795, 1974. 188

917. J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. Why and how developers fork what from whom in GitHub. *Empirical Software Engineering*, 22(1):547–578, Feb. 2017. 96

918. Y. Jiang, B. Adams, and D. M. German. Will my patch make it? And how fast? Case study on the Linux kernel. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR'13, pages 101–110, May 2013. 145

919. D. D. P. Johnson, N. B. Weidmann, and L.-E. Cederman. Fortune favours the bold: An agent-based model reveals adaptive advantages of overconfidence in war. *PLoS ONE*, 6(6):e20851, Apr. 2011. 55

920. J. A. Johnson. Measuring thirty facets of the Five Factor model with a 120-item public domain inventory: Development of the IPIP-NEO-120. *Journal of Research in Personality*, 51:78–89, Aug. 2014. 52

921. L. Johnson. Applied data research inc. (ADR). Technical report, Computer History Museum, Feb. 2010. 107

922. P. M. Johnson and A. M. Disney. A critical analysis of PSP data quality: Results from a case study. *Empirical Software Engineering*, 4(4):317–349, Dec. 1999. 378

923. W. L. Johnson. *Intention-Based Diagnosis of Novice Programming Errors*. Morgan Kaufmann Publishers, Inc, 1986. 199

924. C. I. Jones. The facts of economic growth. In J. B. Taylor and H. Uhlig, editors, *Handbook of Macroeconomics, Volume 2A*, chapter 1, pages 3–69. Elsevier B. V., Nov. 2016. 7

925. D. Jones. Why userspace sucks–or 101 really dumb things your app shouldn't do. In *Proceedings of the Linux Symposium*, Volume One, pages 441–450, July 2006. 372

926. D. M. Jones. Who guards the guardians? www.knosof.co.uk/whoguard.html, 1992. 162

927. D. M. Jones. *The Open Systems Portability Checker Reference Manual*. Knowledge Software Ltd, ??? edition, May 1999. 162

928. D. M. Jones. The 7±2 urban legend. MISRA C 2002 conference http://www.knosof.co.uk/cbook/misart.pdf, Oct. 2002. 30, 362

929. D. M. Jones. Memory for a short sequence of assignment statements (part 2 of 2). *C Vu*, 17(1):34–37, Feb. 2005. 182

930. D. M. Jones. The new C Standard: An economic and cultural commentary. Knowledge Software, Ltd, 2005. 94, 115, 136, 163, 174, 193, 194, 195, 200, 204, 206, 207, 208, 210, 215, 226, 250, 301, 362, 385

931. D. M. Jones. Developer beliefs about binary operator precedence. *C Vu*, 18(4):14–21, Aug. 2006. 37, 38, 53, 59, 206, 354, 355, 362, 375, 376

932. D. M. Jones. Operand names influence operator precedence decisions. *C Vu*, 20(1):5–11, Feb. 2008. 53, 195, 375, 376

933. D. M. Jones. Deciding between `if` and `switch` when writing code. *C Vu*, 21(5):14–20, Nov. 2009. 204

934. D. M. Jones. Developer characterization of data structure fields decisions. *C Vu*, 20(6):14–18, Jan. 2009. 43, 59, 250, 251, 353, 354, 375, 376

935. D. M. Jones. Birth month of compiler writers. blog: The Shape of Code, Feb. 2012. http://shape-of-code.coding-guidelines.com. 346

936. D. M. Jones. Effects of risk attitude on recall of assignment statements. *C Vu*, 23(6):19–22, Jan. 2012. 53

937. D. M. Jones. Amount of end-user usage of code in Firefox. blog: The Shape of Code, July 2013. http://shape-of-code.coding-guidelines.com/2013/07/26/amount-of-end-user-usage-of-code-in-firefox. 161, 162

938. D. M. Jones. Tag data extracted from stack overflow website. url-https://stackoverflow.com, July 2019. 114

939. D. M. Jones. Code & data used in: Evidence-based software engineering: based on the publicly available data. http://www.github.com/Derek-Jones/ESEUR, 2020. 1, 2

940. D. M. Jones and R. Borgatti. The Renzo Pomodoro dataset: a conversation. http://www.github.com/Derek-Jones/renzo-pomodoro, Dec. 2019. 143

941. D. M. Jones and S. Cullum. A conversation around the analysis of the SiP effort estimation dataset. In *eprint arXiv:cs.SE/1901.01621*, Jan. 2019. 126, 127, 137, 138, 141

942. M. N. Jones and D. J. K. Mewhort. Case-sensitive letter and bigram frequency counts from large-scale English corpora. *Behavior Research Methods, Instruments, & Computers*, 36(3):388–396, 2004. 199

943. R. Jongeling, P. Sarkar, S. Datta, and A. Serebrenik. On negative results when using sentiment analysis tools for software engineering research. *Empirical Software Engineering*, 22(5):2543–2584, Oct. 2017. 350

944. M. R. Jongerden. *Model-based energy analysis of battery powered systems*. PhD thesis, Centre for Telematics and Information Technology, University of Twente, Dec. 2010. 368

945. M. Jørgensen. An empirical study of software maintenance tasks. *Software Maintenance: Research and Practice*, 7(1):27–48, Jan. 1995. 304

946. M. Jørgensen. Regression models of software development effort estimation accuracy and bias. *Empirical Software Engineering*, 9(4):297–394, Dec. 2004. 126

947. M. Jørgensen. Better selection of software providers through trialsourcing. *IEEE Software*, 33(5):48–53, Sept.-Oct. 2016. 126, 130

948. M. Jørgensen. A survey on the characteristics of projects with success in delivering client benefits. *Information and Software Technology*, 78:83–94, Oct. 2016. 135

949. M. Jørgensen. Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays. *Journal of Systems and Software*, 117:274–281, July 2016. 51

950. M. Jørgensen and G. J. Carelius. An empirical study of software project bidding. *IEEE Transactions on Software Engineering*, 30(12):953–969, Dec. 2004. 123, 275

951. M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg. Incorrect results in software engineering experiments: How to improve research practices. *Journal of Systems and Software*, 116:133–145, June 2016. 268

952. M. Jørgensen and S. Grimstad. Software development estimation biases: The role of interdependence. *IEEE Transactions on Software Engineering*, 38(3):677–693, May 2012. 21, 25

953. M. Jørgensen, U. Indahl, and D. I. K. Sjøberg. Software effort estimation by analogy and "regression toward the mean". *Journal of Systems and Software*, 68(3):253–262, Dec. 2003. 289

954. M. Jørgensen and K. Moløkken. Eliminating over-confidence in software development effort estimates. In F. Bomarius and H. Iida, editors, *Product Focused Software Process Improvement*, volume 3009 of *Lecture Notes in Computer Science*, pages 174–184. Springer Berlin Heidelberg, Apr. 2004. 276

955. M. Jørgensen and K. Moløkken. Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method. *IEEE Transactions on Software Engineering*, 30(12):993–1007, Dec. 2004. 23, 129

956. M. Jørgensen and K. Moløkken. How large are software cost overruns? A review of the 1994 CHAOS report. *Journal Information and Software Technology*, 48(4):297–301, Apr. 2006. 122

957. M. Jørgensen and D. I. K. Sjøberg. The impact of customer expectation on software development effort estimates. *International Journal of Project Management*, 22(4):317–325, May 2004. 24, 127

958. M. Jørgensen and D. I. K. Sjøberg. Learning from experience in a software maintenance environment. *Journal of Computer Science*, 1(4):538–542, Apr. 2005. 297

959. D. Joseph, W. F. Boh, S. Ang, and S. A. Slaughter. The career paths less (or more) travelled: A sequence analysis of IT career histories, mobility patterns, and career success. *MIS Quarterly*, 36(2):427–452, June 2012. 109

960. J. Jung, H. Hu, D. Solodukhin, D. Pagan, K. H. Lee, and T. Kim. Fuzzification: Anti-fuzzing techniques. In *28th USENIX Security Symposium*, SEC'19, pages 1913–1930, Aug. 2019. 173

961. S. Kahrs. Mistakes and ambiguities in the definition of standard ML. LFCS report ECS-LFCS-93-257, University of Edinburgh, Scotland, Apr. 1993. 165

962. J. W. Kalat. *Biological Psychology*. Wadsworth, seventh edition, 2001. 20

963. T. Kalibera, L. Bulej, and P. Tma. Benchmark precision and random initial state. In *International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SPECTS'05, pages 853–862. Society for Modeling and Simulation (SCS), July 2005. 372

964. A. Kaltenbrunner, V. Gómez, A. Moghnieh, R. Meza, J. Blat, and V. López. Homogeneous temporal activity patterns in a large online communication space. In *eprint arXiv:cs.NI/0708.1579*, Aug. 2007. 246

965. C. Kaltenecker, A. Grebhahn, N. Siegmund, J. Guo, and S. Apel. Distance-based sampling of software configuration spaces. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE'19, pages 1084–1094, May 2019. 172

966. D. Kaminsky, M. Eddington, and A. Cecchetti. Showing how security has (and hasn't) improved, after ten years of trying. CanSecWest Applied Security Conference, Dec. 2011. 159, 160

967. T. Kamiya. How code skips over revisions. In *Proceedings of the 5th International Workshop on Software Clones*, IWSC 2011, pages 69–70, May 2011. 201

968. V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg. A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, Apr. 2007. 8

969. P. Kampstra and C. Verhoef. Benchmarking the expected loss of a federal IT portfolio. July 2009. 285

970. P. Kampstra and C. Verhoef. Reliability of function point counts. http://www.cs.vu.nl/~x/rofpc/rofpc.pdf, 2009. 129

971. T. Kanda, T. Ishio, and K. Inoue. Approximating the evolution history of software from source code. *IEICE Transactions on Information & Systems*, E98-D(6):1185–1193, June 2015. 352

972. C. Kaner. Liability for defective documentation. In *Proceedings of the 21st annual International Conference on Documentation*, SIGDOC'03, pages 192–197, Oct. 2003. 139, 165

973. C. Kaner and D. Pels. *Bad Software: What To Do When Software Fails*. John Wiley & Sons, Inc, 1998. 154

974. Y. Kang, B. Ray, and S. Jana. APEx: Automated inference of error specifications for C APIs. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, ASE'16, pages 472–482, Sept. 2016. 174

975. S. J. Karau and K. D. Williams. Social loafing: A meta-analytic review and theoretical integration. *Journal of Personality and Social Psychologs*, 65(4):681–706, Oct. 1993. 72, 80

976. D. Karlis and E. Xekalaki. Mixed Poisson distributions. *International Statistical Review*, 73(1):35–58, Apr. 2005. 239

977. J. Karlsson and K. Ryan. A cost-value approach for prioritizing requirements. *IEEE Software*, 14(5):67–74, Sept. 1997. 136

978. D. S. Katz, K. McHenry, C. Reinking, and R. Haines. Research software development & management in universities: Case studies from Manchester's RSDS group, Illinois' NCSA, and Notre Dame's CRC. In *eprint arXiv:cs.SE/1903.00732*, Mar. 2019. 109

979. G. Kawasaki. *Selling the Dream: How to Promote Your Product, Company, or Ideas–and Make a Difference–Using Everyday Evangelism*. HarperBusiness, Jan. 1991. 74, 100

980. D. Kawrykow and M. P. Robillard. Non-essential changes in version histories. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE'11, pages 351–360, May 2011. 139

981. M. Kazandjieva, B. Heller, O. Gnawali, P. Levis, and C. Kozyrakis. Green enterprise computing data: Assumptions and realities. In *Proceedings of the 2012 International Green Computing Conference*, IGCC'12, pages 1–10, June 2012. 95

982. F. C. Keil. Explanation and understanding. *Annual Review of Psychology*, 57:227–254, Jan. 2006. 187

983. M. Keil and D. Robey. Blowing the whistle on troubled software projects. *Communications of the ACM*, 44(4):87–93, Apr. 2001. 134

984. P. Keil, J. M. Bennett, B. Bourgeois, G. E. Garcá-Peña, A. A. M. MacDonald, C. Meyer, K. S. Ramirez, and B. Yguel. From computer operating systems to biodiversity: co-emergence of ecological and evolutionary patterns. *PNAS*, 4:e2367, Aug. 2016. 97

985. C. F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987. 128

986. Z. Kenessey. The primary, secondary, tertiary and quaternary sectors of the economy. *The Review of Income and Wealth*, 33(4):359–385, Dec. 1987. 61

987. D. O. Kennedy and A. B. Scholey. Glucose administration, heart rate and cognitive performance: effects of increasing mental effort. *Psychopharmacology*, 149(1):63–71, May 2000. 57

988. E. Keogh and A. Mueen. Time series data mining using the matrix profile: A unifying view of motif discovery, anomaly detection, segmentation, classification, clustering and similarity joins. Tutorial at KDD 2017, Aug. 2017. 334

989. B. W. Kernighan and R. Pike. *The Practice of Programming*. Addison–Wesley, 1999. 184

990. N. L. Kerr. HARKing: Hypothesizing After the Results are Known. *Personality and Social Psychology Review*, 2(3):196–217, Aug. 1998. 3

991. E. Keuleers, P. Lacey, K. Rastle, and M. Brysbaert. The British lexicon project: Lexical decision data for 28,730 monosyllabic and disyllabic English words. *Behavior and Research Methods*, 44(1):287–304, Mar. 2012. 35

992. H. Khalid, M. Nagappan, E. Shihab, and A. E. Hassan. Prioritizing the devices to test your app on: A case study of Android game apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 610–620, Nov. 2014. 84

993. L. M. Khan. Amazon's antitrust paradox. *The Yale Law Journal*, 126(3):710–805, Jan. 2017. 102

994. M. W. Khaw, L. Stevens, and M. Woodford. Discrete adjustment to a changing environment: Experimental evidence. *Journal of Monetary Economics*, 91(C):88–103, 2017. 51, 52

995. P.-V. Khuong and P. Morin. Array layouts for comparison-based searching. In *eprint arXiv:cs.DS/1509.05053*, Mar. 2017. 371

996. P. D. Killworth and H. R. Bernard. Informant accuracy in social network data. *Human Organization*, 35(3):269–286, Sept.-Nov. 1976. 358

997. D. Kim, E. Murphy-Hill, C. Parnin, C. Bird, and R. Garciad. The reaction of open-source projects to new language features: An empirical study of C# generics. *The Journal of Object Technology*, 12(4):1–26, Nov. 2013. 186

998. H. Kim. *Informed Storage Management for Mobile Platforms*. PhD thesis, College of Computing, Georgia Institute of Technology, Dec. 2012. 370

999. J. D. Kim. Startup acquisitions as a hiring strategy: Worker choice and turnover. SSRN Working Paper 3252784, Wharton School, University of Pennsylvania, Mar. 2020. 109

1000. J. Y. Kim, S. Shepherd, T. H. Campbell, and A. C. Kay. Understanding contemporary forms of exploitation: Attributions of passion serve to legitimize the poor treatment of workers. *Journal of Personality and Social Psychology: Interpersonal Relations and Group Processes*, 118(1):121–148, Jan. 2020. 71

1001. S. Kim. The classification of information and communication technology investment in financial accounting. Thesis (m.s.), School of Information Technologies, University of Sydney, 2013. 84

1002. K. Kina, M. Tsunoda, H. Hata, H. Tamada, and H. Igaki. Analyzing the decision criteria of software developers based on prospect theory. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER'16, pages 644–648, Mar. 2016. 55

1003. D. King and C. Janiszewski. The sources and consequences of the fluent processing of numbers. *Journal of Marketing Research*, XLVIII(2):327–341, 2011. 50

1004. J. King and M. A. Just. Individual differences in syntactic processing: The role of working memory. *Journal of Memory and Language*, 30:580–602, 1991. 32

1005. W. Kintsch. *Comprehension: A paradigm for cognition*. Cambridge University Press, 1998. 190

1006. W. Kintsch and J. Keenan. Reading rate and retention as a function of the number of propositions in the base structure of sentences. *Cognitive Psychology*, 5(3):257–274, Nov. 1973. 188

1007. W. Kintsch, E. Kozminsky, W. J. Streby, G. McKoon, and J. M. Keenan. Comprehension and recall of text as a function of content variables. *Journal of Verbal Learning and Verbal Behavior*, 14(2):196–214, Apr. 1975. 188

1008. W. Kintsch, T. S. Mandel, and E. Kozminsky. Summarizing scrambled stories. *Memory & Cognition*, 5(5):547–552, 1977. 191

1009. K. N. Kirby and R. J. Herrnstein. Preference reversals due to myopic discounting of delayed reward. *Psychological Science*, 6(2):83–89, Mar. 1995. 56

1010. D. Kirsh and P. Maglio. On distinguishing epistemic from pragmatic action. *Cognitive Science*, 18(4):513–549, Oct. 1994. 22

1011. L. B. Kish. Moore's law and the energy requirement of computing versus performance. *IEE Proceedings-Circuits, Devices and Systems*, 151(2):190–194, Apr. 2004. 166

1012. J. V. Kistowski, H. Block, J. Beckett, K.-D. Lange, J. A. Arnold, and S. Kounev. Analysis of the influences on server power consumption and energy efficiency for CPU-intensive workloads. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE'15, pages 223–234, Jan. 2015. 255

1013. S. Kitayama and M. Karasawa. Implicit self-esteem in Japan: Name-letters and birthday numbers. *Personality & Social Psychology Bulletin*, 23(7):736–742, 1997. 196

1014. B. Kitchenham, S. L. Pfleeger, B. McColl, and S. Eagan. An empirical study of maintenance and development estimation accuracy. *The Journal of Systems and Software*, 64(1):57–77, Oct. 2002. 126

1015. B. A. Kitchenham and N. R. Taylor. Software project development cost estimation. *The Journal of Systems and Software*, 5(4):267–278, Nov. 1985. 133

1016. A. Kivi, T. Smura, and J. Töyli. Technology product evolution and the diffusion of new product features. *Technological Forecasting & Social Change*, 79(1):107–126, Jan. 2012. 87

1017. D. Klahr, W. G. Chase, and E. A. Lovelace. Structure and process in alphabetic retrieval. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 9(3):462–477, 1983. 34

1018. K. C. Klauer, J. Musch, and B. Naumer. On belief bias in syllogistic reasoning. *Psychological Review*, 107(4):852–884, Oct. 2000. 45

1019. J. Klayman and Y.-W. Ha. Confirmation, disconfirmation, and information in hypothesis testing. *Psychological Review*, 94(2):211–228, Apr. 1987. 25

1020. B. Klein. The decision making problem in development. In Universities-National Bureau Committee for Economic Research, Committee on Economic Growth of the Social Science Research Council, editor, *The Rate and Direction of Inventive Activity: Economic and Social Factors*, chapter 19, pages 477–508. Princeton University Press, 1962. 130

1021. S. B. Klein, L. Cosmides, J. Tooby, and S. Chance. Decisions and the evolution of memory: Multiple systems, multiple functions. *Psychological Review*, 109(2):306–329, Apr. 2002. 29

1022. J. Kleinberg and M. Raghu. Team performance with test scores. In *eprint arXiv:cs.DS/1506.00147v2*, Mar. 2018. 141

1023. S. Kleinschmager, R. Robbes, A. Stefik, S. Hanenberg, and É. Tanter. Do static type systems improve the maintainability of software systems? An empirical study. In *20th International Conference on Program Comprehension*, ICPC'12, pages 153–162, June 2012. 206

1024. S. Klepper and K. L. Simons. Technological extinctions of industrial firms: An inquiry into their nature and causes. *Industrial and Corporate Change*, 6(2):379–460, Mar. 1997. 107

1025. P. Klint, D. Landman, and J. Vinju. Exploring the limits of domain model recovery. In *29th IEEE International Conference on Software Maintenance*, ICSM'13, pages 120–129, Sept. 2013. 136

1026. K. E. Knight. Changes in computer performance. *Datamation*, 12(9):40–54, Sept. 1966. 365

1027. K. E. Knight. Evolving computer performance 1963-1967. *Datamation*, 14(1):31–35, Jan. 1968. 8, 365

1028. D. E. Knuth. *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison–Wesley, first edition, 1973. 94

1029. D. E. Knuth. Structure programming with `go to` statements. *Computing Surveys*, 6(4):261–301, Dec. 1974. 203

1030. D. E. Knuth. The errors of TEX. *Software–Practice and Experience*, 19(7):607–685, 1989. 151

1031. D. Kobak, S. Shpilkin, and M. S. Pshenichnikov. Integer percentages as electoral falsification fingerprints. In *eprint arXiv:stat.AP/1410.6059v4*, June 2016. 386

1032. A. Koenig. *C Traps and Pitfalls*. Addison–Wesley, 1989. 184

1033. P. A. Kolers. Reading A year later. *Journal of Experimental Psychology: Human Learning and Memory*, 2(3):554–565, 1976. 192, 193

1034. P. A. Kolers and D. N. Perkins. Spatial and ordinal components of form perception and literacy. *Cognitive Psychology*, 7(2):228–267, Apr. 1975. 192

1035. J. G. Koomey, S. Berard, M. Sanchez, and H. Wong. Implications of historical trends in the electrical efficiency of computing. *IEEE Annals of the History of Computing*, 33(3):46–54, July-Sept. 2011. 6

1036. A. Koriat. How do we know that we know? The accessibility model of the feeling of knowing. *Psychological Review*, 100(4):609–639, Oct. 1993. 33

1037. A. G. Koru, K. El Emam, D. Zhang, H. Liu, and D. Mathew. Theory of relative defect proneness: Replicated studies on the functional form of the size-defect relationship. *Empirical Software Engineering*, 13(5):473–498, Oct. 2008. 164

1038. J. Kossik. Clark's sector model for US economy 1850-2009. personal website, 2011. http://www.63alfred.com/whomakesit/clarksmodel.htm. 61

1039. S. M. Kosslyn. *Graph Design for the Eye and Mind*. Oxford University Press, 2006. 228

1040. S. M. Kosslyn and S. P. Shwartz. Empirical constraints on theories of visual imagery. In J. Long and A. D. Baddeley, editors, *Attention and Performance IX*, pages 241–260. Lawrence Erlbaum Associates, 1981. 22

1041. KPMG. Project Tesla due diligence assistance. submitted as evidence in Autonomy/HP court case, Aug. 2011. 83

1042. P. Kraft. *Programmers and Managers: The Routinization of Computer Programming in the United States*. Springer-Verlag, July 1977. 73, 141

1043. M. Kremer. The O-ring theory of economic development. *The Quarterly Journal of Economics*, 108(3):551–575, Aug. 1993. 141

1044. E. Krevat, J. Tucek, and G. R. Ganger. Disks are like snowflakes: No two are alike. In *Proceedings of the 13th USENIX conference on Hot topics in operating systems*, HotOS'13, May 2013. 370

1045. F. Křikava, H. Miller, and J. Vitek. Scala implicits are everywhere: A large-scale study of the use of implicits in the wild. In *eprint arXiv:cs.PL/1908.07883*, Aug. 2019. 198

1046. G. Kroah-Hartman. Linux kernel statistics. Github account, June 2016. https://www.github.com/gregkh/kernel-history. 287, 312, 313

1047. J. K. Kruschke. Human category learning: Implications for backpropagation models. *Connection Science*, 5(1):3–36, 1993. 36, 37

1048. J. K. Kruschke. Dimensional relevance shifts in category learning. *Connection Science*, 8(2):225–247, June 1996. 36, 37

1049. E. C. Kubie. Recollections of the first software company. *IEEE Annals of the History of Computing*, 16(2):65–71, June 1994. 107

1050. M. Kubovy and M. van den Berg. The whole is equal to the sum of its parts: A probabilistic model of grouping by proximity and similarity in regular patterns. *Psychological Review*, 115(1):131–154, Jan. 2008. 27, 28

1051. T. Kuchta, T. Lutellier, E. Wong, L. Tan, and C. Cadar. On the correctness of electronic documents: studying, finding, and localizing inconsistency bugs in PDF readers and files. *Empirical Software Engineering*, 23(6):3187–3220, Dec. 2018. 111

1052. B. M. Kuhn, Free Software Foundation, Inc., Software Freedom Law Center, A. K. Sebro, Jr., D. Gingerich, and C. Legal. *Copyleft and the GNU General Public License: A Comprehensive Tutorial and Guide*. copyleft.org, 2018. 68

1053. D. R. Kuhn, R. N. Kacker, and Y. Lei. A model for t-way fault profile evolution during testing. In *IEEE International Conference on Software Testing, Verification and Validation Workshops*, ICSTW 2017, pages 162–170, Mar. 2017. 173

1054. M. Kuhrmann, C. Konopka, P. Nellemann, P. Diebold, and J. Münch. Software process improvement: Where is the evidence? In *Proceedings of the 2015 International Conference on Software and System Process*, pages 107–116, Aug. 2015. 9

1055. R. Kumar. The business of scaling. *IEEE Solid-State Circuits Society Newsletter*, 12(1):22–26, 2007. 7

1056. S. Kumar. Enforcing the GNU GPL. *Journal of Law, Technology & Policy*, 2006(1), 2006. 70

1057. G. Kunda. *Engineering Culture: Control and Commitment in a High-Tech Corporation*. Temple University Press, 1992. 107

1058. P. Küngas, S. Vakulenko, M. Dumas, C. Parra, and F. Casati. Reverse-engineering conference rankings: What does it take to make a reputable conference? *Scientometrics*, 96(2):651–665, Aug. 2013. 11

1059. G. Kunst. Language popularity. http://langpop.corger.nl/results, 2013. 292

1060. R. Kurzban, A. Duckworth, J. W. Kable, and J. Myers. An opportunity cost model of subjective effort and task performance. *Behavioral and Brain Sciences*, 36(6):661–679, Dec. 2013. 26

1061. D. S. Kusumo, M. Staples, L. Zhu, and R. Jeffery. Analyzing differences in risk perceptions between developers and acquirers in OTS-based custom software projects using stakeholder analysis. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'12, pages 69–78, Sept. 2012. 124

1062. A. Kvarven, E. Strømland, and M. Johannesson. Comparing meta-analyses and preregistered multiple-laboratory replication projects. *Nature Human Behaviour*, 4(4):423–434, Apr. 2020. 265

1063. C. Labbé and D. Labbé. Duplicate and fake publications in the scientific literature: how many SCIgen papers in computer science? HAL Id: hal-00641906, HAL archives-ouvertes.fr, July 2012. 11

1064. T. Labiner. A big decision: Lease or buy? *Computers and Automation*, 6(10):6–8, Oct. 1957. 105

1065. W. Labov. The boundaries of words and their meaning. In C.-J. N. Bailey and R. W. Shuy, editors, *New ways of analyzing variation of English*, pages 340–373. Georgetown Press, 1973. 43

1066. W. Labov. *Principles of Linguistic Change, volume 3: Cognitive and Cultural Factors*. Wiley-Blackwell, 2010. 195

1067. E. Labro and L. Stice-Lawrence. Updating accounting systems: Longitudinal evidence from the health care sector. *Management Science*, ???(???):???, Apr. 2019. 92

1068. J. C. Lagarias. *The Kepler Conjecture: The Hales-Ferguson Proof by Thomas C. Hales Samuel P. Ferguson*. Springer, 2010. 148

1069. K. Laitinen. *Natural naming in software development and maintenance*. PhD thesis, University of Oulu, Finland, Oct. 1995. 194

1070. G. Lakoff and M. Johnson. *Metaphors We Live By*. The University of Chicago Press, 1980. 47, 106

1071. A. LaMarca and R. E. Ladner. The influence of caches on the performance of sorting. *Journal of Algorithms*, 31(1):66–104, Apr. 1999. 94

1072. B. L. Lambert, K.-Y. Chang, and P. Gupta. Effects of frequency and similarity neighborhoods on pharmacists' visual perception of drug names. *Social Science and Medicine*, 57(10):1939–1955, Nov. 2003. 196

1073. R. Lämmel, E. Pek, and J. Starek. Large-scale, AST-based API-usage analysis of open-source Java projects. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC'11, pages 1317–1324, Mar. 2011. 208

1074. B. W. Lampson. A critique of "an exploratory investigation of programmer performance under on-line and off-line conditions". *IEEE Transactions on Human Factors in Electronics*, 8(1):33–48, Mar. 1967. 10

1075. T. K. Landauer. How much do people remember? Some estimates of the quantity of learned information in long-term memory. *Cognitive Science*, 10:477–493, 1986. 57

1076. R. M. Landers, J. B. Rebitzer, and L. J. Taylor. Rat race reduce: Adverse selection in the determination of work hours in law firms. *The American Economic Review*, 86(3):329–348, June 1996. 78

1077. D. Landman, A. Serebrenik, E. Bouwers, and J. J. Vinju. Empirical analysis of the relationship between CC and SLOC in a large corpus of Java methods and C functions. *Journal of Software: Evolution and Process*, 28(7):589–618, July 2016. 163, 179, 180, 182, 183, 186, 192

1078. D. Landy, D. Brookes, and R. Smout. Abstract numeric relations and the visual structure of algebra. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 40(5):1404–1418, Sept. 2014. 27

1079. D. Landy, A. Charlesworth, and E. Ottmar. Categories of large numbers in line estimation. *Cognitive Science*, 41(2):326–353, Mar. 2017. 48, 49

1080. D. Landy and R. L. Goldstone. Proximity and precedence in arithmetic. *The Quarterly Journal of Experimental Psychology*, 63(10):1953–1968, Oct. 2010. 214

1081. D. Landy, B. Guay, and T. Marghetis. Bias and ignorance in demographic perception. *Psychonomic Bulletin & Review*, 25(5):1606–1618, Oct. 2018. 51

1082. E. J. Langer. The illusion of control. *Journal of Personality and Social Psychologs*, 32(2):311–328, 1975. 56

1083. R. N. Langlois. External economies and economic progress: The case of the microcomputer industry. *The Business History Review*, 66(1):1–50, 1992. 94

1084. LANL. LANL failure data. http://institute.lanl.gov/data/lanldata.shtml, 2006. 167

1085. L. Lapointe and S. Rivard. A multilevel model of resistance to information technology implementation. *MIS Quarterly*, 29(3):461–492, Sept. 2005. 120

1086. I. Larkin. The cost of high-powered incentives: Employee gaming in enterprise software sales. Technical Report 13-073, Harvard Business School, Feb. 2013. 88

1087. C. Larman and V. R. Basili. Iterative and incremental development: A brief history. *Computer*, 36(6):47–56, June 2003. 131

1088. J. Larres. Performance variance evaluation on Mozilla Firefox. Thesis (m.s.), Victoria University of Wellington, May 2012. 373

1089. R. H. Larson, J. K. Salmon, R. O. Dror, M. M. Deneroff, C. Young, J. Grossman, Y. Shan, J. L. Klepeis, and D. E. Shaw. High-throughput pairwise point interactions in Anton, a specialized machine for molecular dynamics simulation. In *IEEE 14th International Symposium on High Performance Computer Architecture*, HPCA 2008, pages 331–342, Feb. 2008. 112

1090. B. Latané and J. M. Darley. Bystander "apathy". *American Scientist*, 57(2):244–269, June-Sept. 1969. 80

1091. B. Latané, K. Williams, and S. Harkins. Many hands make light the work: The causes and consequences of social loafing. *Journal of Personality and Social Psychology*, 37(6):822–832, 1979. 80

1092. R. Latorre. Effects of developer experience on learning and applying unit test-driven development. *IEEE Transactions on Software Engineering*, 40(4):381–395, Apr. 2014. 37, 38

1093. P. R. Laughlin. *Group Problem Solving*. Princeton University Press, Apr. 2015. 80

1094. J. Laukemann, J. Hammer, J. Hofmann, G. Hager, and G. Wellein. Automated instruction stream throughput prediction for Intel and AMD microarchitectures. In *IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, PMBS 2018, pages 121–131, Nov. 2018. 205

1095. E. Laukkanen, M. Paasivaara, J. Itkonen, C. Lassenius, and T. Arvonen. Towards continuous delivery by reducing the feature freeze period: A case study. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, ICSE-SEIP'17, pages 23–32, May 2017. 140

1096. S. Laumer, C. Maier, A. Eckhardt, and T. Weitzel. Work routines as an object of resistance during information systems implementations: theoretical foundation and empirical evidence. *European Journal of Information Systems*, 25(4):317–343, July 2016. 120

1097. L. Lauterbach. Development of N-version software samples for an experiment in software fault tolerance. NASA Contractor Report 178363, Software Research and Development Center for Digital Systems Research, Sept. 1987. 130, 162

1098. C. W. Lazar. Lease/buy decisions for computer acquisition under conditions of uncertain technological change. In *Proceedings–1968 ACM National Conference*, pages 685–690, Jan. 1968. 105

1099. E. Lazear and M. Gibbs. *Personnel Economics for Managers*. John Wiley & Sons, Inc, second edition, 2007. 72, 108, 141

1100. C. Lebiere. *The Dynamics of Cognition: An ACT-R Model of Cognitive Arithmetic*. PhD thesis, Carnegie Mellon University, Nov. 1998. 49

1101. P. L'Ecuyer and R. Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 33(4):1–22, Aug. 2007. 165

1102. A. L. Lederer and J. Prasad. Causes of inaccurate software development cost estimates. *Journal of Systems and Software*, 31(2):125–134, Nov. 1995. 125

1103. B. C. Lee and D. M. Brooks. Regression modeling strategies for microarchitecture performance and power prediction. Technical Report TR-08-06, Division of Engineering and Applied Sciences, Harvard University, Mar. 2006. 323, 363

1104. D. Lee, Y. Kim, G. Pekhimenko, S. Khan, V. Seshadri, K. Chang, and O. Mutlu. Adaptive-latency DRAM: Optimizing DRAM timing for the common-case. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*, HPCA'15, pages 489–501, Feb. 2015. 371

1105. M. D. Lee, K. A. Gluck, and M. M. Walsh. Understanding the complexity of simple decisions: Modeling multiple behaviors and switching strategies. *Decision*, 6(4):335–368, Oct. 2019. 51

1106. G. Leech, P. Rayson, and A. Wilson. *Word Frequencies in Written and Spoken English*. Pearson Education, 2001. 46, 156, 199

1107. L. Lefebvre and N. J. Boogert. Avian social learning. In M. D. Breed and J. Moore, editors, *Encyclopedia of Animal Behavior: volume 1*, pages 124–130. Oxford: Academic Press, July 2010. 75

1108. J.-A. LeFevre and J. Liu. The role of experience in numerical skill: Multiplication performance in adults from Canada and China. *Mathematical Cognition*, 3(1):31–62, 1997. 51

1109. G. E. Legge, T. A. Hooven, T. S. Klitz, J. S. Mansfield, and B. S. Tjan. Mr. Chips 2002: New insights from an ideal-observer model of reading. *Vision Research*, 42(18):2219–2234, Aug. 2002. 28

1110. C. Leggett. The Ford Pinto case: The valuation of life as it applies to the negligence-efficiency argument. https://users.wfu.edu/palmitar/Law&Valuation/Papers/1999/Leggett-pinto.html, 1999. 155

1111. D. R. Lehman, R. O. Lempert, and R. E. Nisbett. The effects of graduate training on reasoning. *American Psychologist*, 43(6):431–442, 1988. 40

1112. L. Lehmann, K. Aoki, and M. W. Feldman. On the number of independent cultural traits carried by individuals and populations. *Philosophical Transactions of The Royal Society B*, 366(1563):424–435, Feb. 2011. 76, 103

1113. P. Lemaire and M. Fayol. When plausibility judgments supersede fact retrieval: The example of the odd-even effect on product verification. *Memory & Cognition*, 23(1):34–48, Feb. 1995. 49

1114. P. Lennie. The cost of cortical computation. *Current Biology*, 13(6):493–497, Mar. 2003. 57

1115. F. Lequiller, N. Ahmad, S. Varjonen, W. Cave, and K.-H. Ahn. Report of the OECD task force on software measurement in the national accounts. STD/NA (2002)2, Organisation for Economic Co-operation and Development, Sept. 2002. 82

1116. K. Lerman, X. Yan, and X.-Z. Wu. The "majority illusion" in social networks. *PLoS ONE*, 11(2):e0147617, Feb. 2016. 100

1117. K. Letrud and S. Hernes. Affirmative citation bias in scientific myth debunking: A three-in-one case study. *PLoS ONE*, 14(9):e0222213, Sept. 2019. 9

1118. D. E. Levari, D. T. Gilbert, T. D. Wilson, B. Sievers, D. M. Amodio, and T. Wheatley. Prevalence-induced concept change in human judgment. *Science*, 360(6396):1465–1467, June 2018. 52

1119. B. W. Leverett, R. G. G. Cattell, S. O. Hobbs, J. M. Newcomer, A. H. Reiner, B. R. Schatz, and W. A. Wulf. An overview of the production-quality compiler-compiler project. Technical Report CMU-CS-79-105, Carnegie Mellon University, Feb. 1979. 178

1120. P. Lewicki, T. Hill, and E. Bizot. Acquisition of procedural knowledge about a pattern of stimuli that cannot be articulated. *Cognitive Psychology*, 20(1):24–37, Jan. 1988. 193

1121. A. C. Lewis. A study of idea generation over time. Thesis (m.s.), Georgia Institute of Technology, June 1972. 81

1122. G. Lewis and P. Bajari. Incentives and adaptation: Evidence from highway procurement in Minnesota. Working Paper 17647, National Bureau of Economic Research, USA, Dec. 2011. 129

1123. J. R. Lewis. Evaluation of procedures for adjusting problem-discovery rates estimated from small samples. *International Journal of Human-Computer Interaction*, 13(4):445–479, Dec. 2001. 170

1124. K. Li, E. Yan, and Y. Feng. How is R cited in research outputs? Structure, impacts, and citation standard. *Journal of Informetrics*, 11(4):989–1002, Nov. 2017. 11

1125. L. Li, T. F. Bissyandé, and J. Klein. MoonlightBox: Mining Android API histories for uncovering release-time inconsistencies. In *IEEE 29th International Symposium on Software Reliability Engineering*, ISSRE'18, pages 212–223, Oct. 2018. 84

1126. L. Li, T. F. Bissyandé, Y. L. Traon, and J. Klein. Accessing inaccessible Android APIs: An empirical study. In *International Conference on Software Maintenance and Evolution*, ICSME 2016, pages 411–422, Oct. 2016. 117

1127. Q. Li and H. Pham. A testing-coverage software reliability model considering fault removal efficiency and error generation. *PLoS ONE*, 12(7):e0181524, July 2017. 158

1128. X. Li. *Soft Error Modeling and Analysis for Microprocessors*. PhD thesis, University of Illinois at Urbana-Champaign, May 2008. 167

1129. X. Li, L. Molleman, and D. van Dolder. Conditional punishment: descriptive social norms drive negative reciprocity. Working Paper 3571220, Centre for Decision Research and Experimental Economics, University of Nottingham, May 2020. 79

1130. Y. Li. Empirical study of Python call graph. In *34th IEEE/ACM International Conference on Automated Software Engineering*, ASE'19, pages 1274–1276, Nov. 2019. 358

1131. Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, 32(3):176–192, Mar. 2006. 82

1132. Y. Liang, Y. Zhang, A. Sivasubramaniam, R. K. Sahoo, J. Moreira, and M. Gupta. Filtering failure logs for a BlueGene/L prototype. In *Proceedings of the International Conference on Dependable Systems and Networks*, DSN'05, pages 476–485, June 2005. 385

1133. S. Lichtenstein and B. Fischhoff. Do those who know more also know more about how much they know? *Organizational Behavior and Human Performance*, 20:159–183, 1977. 55

1134. S. Lichtenstein, P. Slovic, B. Fischhoff, M. Layman, and B. Combs. Judged frequency of lethal events. *Journal of Experimental Psychology: Human Learning and Memory*, 4(6):551–578, Nov. 1978. 152

1135. Y. Lichtenstein and A. McDonnell. Pricing software development services. In *European Conference on Information Systems*, ECIS 2003, 2003. 124

1136. C. Lidbury, A. Lascu, N. Chong, and A. F. Donaldson. Many-core compiler fuzzing. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'15, pages 65–76, June 2015. 155

1137. G. A. Liebchen and M. Shepperd. Data sets and data quality in software engineering. In *Proceedings of the 4th International workshop on Predictor Models in Software Engineering*, PROMISE'08, pages 39–44, May 2008. 377

1138. J. Liebig, S. Apel, C. Lengauer, C. Kästner, and M. Schulze. An analysis of the variability in forty preprocessor-based software product lines. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, ICSE'10, pages 105–114, May 2010. 199

1139. J. Liebig, A. von Rhein, C. Kästner, S. Apel, J. Dörre, and C. Lengauer. Scalable analysis of variable software. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'13, pages 81–91, Aug. 2013. 172

1140. F. Lieder, T. L. Griffiths, Q. J. M. Huys, and N. D. Goodman. The anchoring bias reflects rational use of cognitive resources. *Psychonomic Bulletin & Review*, 25(1):322–349, Feb. 2018. 24

1141. J. H. Lienhard. *How Invention Begins: Echoes of Old Voices in the Rise of New Machines*. Oxford University Press, 2006. 96

1142. J. S. Light. When computers were women. *Technology and Culture*, 40(3):455–483, July 1999. 108

1143. S. L. Lim. *Social Networks and Collaborative Filtering for Large-Scale Requirements Elicitation*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, Aug. 2010. 135, 136, 144

1144. S. L. Lim, P. J. Bentley, N. Kanakam, F. Ishikawa, and S. Honiden. Investigating country differences in mobile app user behavior and challenges for software engineering. *IEEE Transactions on Software Engineering*, 41(1):40–64, Jan. 2015. 105

1145. B. Lin, L. Ponzanelli, A. Mocci, G. Bavota, and M. Lanza. On the uniqueness of code redundancies. In *IEEE/ACM 25th International Conference on Program Comprehension*, ICPC'17, pages 121–131, May 2017. 200

1146. B. Lin, F. Zampetti, G. Bavota, M. Di Penta, M. Lanza, and R. Oliveto. Sentiment analysis for software engineering: How far can we go? In *Proceedings of the 40th International Conference on Software Engineering*, ICSE'18, pages 94–104, May-June 2018. 350

1147. D.-Y. Lin and I. Neamtiu. Collateral evolution of applications and databases. In *Proceedings of the joint International and annual ERCIM workshops on Principles of software evolution and Software Evolution workshops*, IWPSE-Evol'09, pages 31–40, Aug. 2009. 202

1148. L. C. H. Lin and N. Shen. GPL-3.0 in the Chinese intellectual property court in Beijing. *International Free and Open Source Software Law Review*, 10(1):1–7, 2018. 70

1149. M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. Di Penta, R. Oliveto, and D. Poshyvanyk. API change and fault proneness: A threat to the success of Android apps. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'13, pages 477–487, Aug. 2013. 155

1150. K. R. Linberg. Software developer perceptions about software project failure: a case study. *The Journal of Systems and Software*, 49(2–3):177–192, Dec. 1999. 120

1151. K. Lind and R. Heldal. A practical approach to size estimation of embedded software components. *IEEE Transactions on Software Engineering*, 38(5):993–1007, Sept.-Oct. 2012. 130, 131

1152. R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, 36(4):119–150, Dec. 2004. 361

1153. T. Little. Schedule estimation and uncertainty surrounding the cone of uncertainty. *IEEE Software*, 23(3):48–54, May 2006. 134, 135

1154. D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway. Mental models and software maintenance. In E. Soloway and S. Iyengar, editors, *Empirical Studies of Programmers*, chapter 6, pages 80–98. Ablex Publishing Corporation, 1986. 187

1155. S. Livieri, Y. Higo, M. Matsushita, and K. Inoue. Analysis of the Linux kernel evolution using code clone coverage. In *Fourth International Workshop on Mining Software Repositories*, MSR'07, pages 22–25, May 2007. 211

1156. G. D. Logan. Shapes of reaction-time distributions and shapes of learning curves: A test of the instance theory of automaticity. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 18(5):883–914, 1992. 36

1157. C. V. Lopes, P. Maj, P. Martins, V. Saini, D. Yang, J. Zitny, H. Sajnani, and J. Vitek. DéjàVu: A map of code duplicates on GitHub. In *Conference on Object-Oriented Programming Systems, Languages, and Applications*, OOPSLA'17, page 84, Oct. 2017. 200, 201

1158. C. V. Lopes and J. Ossher. How scale affects structure in Java programs. In *eprint arXiv:cs.SE/1508.00628*, Aug. 2015. 180

1159. I. Lorge and H. Solomon. Two models of group behavior in the solution of Eureka-type problems. *Psychometrika*, 20(2):139–148, June 1955. 80

1160. M. Lorko, M. Servátka, and L. Zhang. Anchoring in project duration estimation. *Journal of Economic Behavior & Organization*, 162:49–65, June 2019. 40

1161. D. D. Loschelder, M. Friese, M. Schaerer, and A. D. Galinsky. The too-much-precision effect: When and why precise anchors backfire with experts. *Psychological Science*, 27(12):1573–1587, Oct. 2016. 123

1162. R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wąsowski. Evolution of the Linux kernel variability model. In *Proceedings of the 14th International Conference on Software Product Lines: going beyond*, SPLC'10, pages 136–150, Sept. 2010. 332, 333

1163. P. Louridas, D. Spinellis, and V. Vlachos. Power laws in software. *ACM Transactions on Software Engineering and Methodology*, 18(1):1–26, Sept. 2008. 319

1164. L. Lu, A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, and S. Lu. A study of Linux file system evolution. In *11th USENIX Conference on File and Storage Technologies*, FAST'13, pages 31–44, Feb. 2013. 186, 219

1165. J. D. Lucente. *On the Viability of Quantitative Assessment Methods in Software Engineering and Software Services*. PhD thesis, School of Engineering and Computer Science, University of Denver, Jan. 2015. 155

1166. L. Lucia. *Ranking-Based Approaches for Localizing Faults*. PhD thesis, Singapore Management University, June 2014. 164

1167. L. Lucia, D. Lo, L. Jiang, F. Thung, and A. Budi. Extended comprehensive study of association measures for fault localization. *Journal of Software: Evolution and Process*, 26(2):172–219, Feb. 2014. 163

1168. K. M. Lui and K. C. C. Chan. Pair programming productivity: Novice-novice vs. expert-expert. *International Journal of Human-Computer Studies*, 64(9):915–925, Sept. 2006. 37

1169. A. W. Lukaszewski, M. Gurven, C. R. von Rueden, and D. P. Schmitt. What explains personality covariation? A test of the socioecological complexity hypothesis. *Social Psychological and Personality Science*, 8(8):943–952, Nov. 2017. 52

1170. P. Lukowicz, E. A. Heinz, L. Prechelt, and W. F. Tichy. Experimental evaluation in computer science: A quantitative study. Technical Report 17/94, University of Karlsruhe, Germany, Aug. 1994. 8

1171. A. Lundqvist and D. Rodic. GNU/Linux distribution timeline. personal website, Oct. 2012. `http://futurist.se/gldt`. 99

1172. M. I. Lunesu. *Process Software Simulation Model of Lean-Kanban Approach*. PhD thesis, Department of Electrical and Electronic Engineering, University of Cagliari, Apr. 2013. 342, 343

1173. Y. Luo, S. Govindan, B. Sharma, M. Santaniello, J. Meza, A. Kansal, J. Liu, B. Khessib, K. Vaid, and O. Mutlu. Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory. In *Proceedings of the 2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 467–478, June 2014. 166

1174. A. K. Luria. Towards the problem of the historical nature of psychological processes. *International Journal of Psychology*, 6(4):259–272, 1971. 21, 44

1175. A. R. Luria. *The Mind of a Mnemonist*. Penguin Education, 1975. 35

1176. B. Luthiger and C. Jungwirth. Pervasive fun. *First Monday*, 12(1), Jan. 2007. 305, 306

1177. W. J. Lynn III. A new approach for delivering information technology capabilities in the Department of Defense. Report to Congress, Office of the Secretary of Defense, Nov. 2010. 131

1178. W. Ma, L. Chen, X. Zhang, Y. Zhou, and B. Xu. How do developers fix cross-project correlated bugs? A case study on the GitHub scientific Python ecosystem. In *IEEE/ACM 39th International Conference on Software Engineering*, ICSE'17, pages 381–392, May 2017. 152

1179. W. Ma, J.-C. S. Liu, and A. Forin. Design and testing of a CPU emulator. Technical Report MSR-TR-2009-155, Microsoft Research, Aug. 2009. 165

1180. F. MacCrory, V. Choudhary, and A. Pinsonneault. Designing promotion ladders to mitigate turnover of IT professionals. *Information Systems Research*, 27(3):648–660, Sept. 2016. 71

1181. N. Macdonald. Computing services survey. *Computers and Automation*, 7(7):9–12, July 1958. 105

1182. N. Macdonald. *Computer Census 1962-74*. Computers and People, 1974. 105

1183. J. N. MacGregor. Short-term memory capacity: Limitation or optimization? *Psychological Review*, 94(1):107–108, Jan. 1987. 34

1184. A. Machiry, R. Tahiliani, and M. Naik. Dynodroid: An input generation system for Android apps. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'13, pages 224–234, Aug. 2013. 347, 348

1185. C. E. Mackenzie. *Coded Character Sets, History and Development*. Addison–Wesley, 1980. 106

1186. D. MacKenzie. Computer-related accidental death: an empirical exploration. *Science and Public Policy*, 21(4):233–248, Aug. 1994. 153

1187. I. S. MacKenzie. Fitts' law as a research and design tool in human-computer interaction. *International Journal of Human-Computer Interaction*, 7(1):91–139, Mar. 1992. 58

1188. R. J. Madachy. *Software Process Dynamics*. John Wiley & Sons, Inc, 2008. 128, 356

1189. A. Maddison. Business cycles, long waves and phases of capital development. In A. Maddison, editor, *Dynamic Forces in Capitalist Development: A Long-run Comparative View*, chapter 4, page 85. Oxford University Press, Oct. 1991. 7

1190. W. T. Maddox and C. J. Bohil. Costs and benefits in perceptual categorization. *Memory & Cognition*, 28:597–615, 2000. 41

1191. M. Magidin and E. Viso. On the experiments in algorithm dynamics. Technical Report UAMR0853, Universidad Autónoma Metropolitana-Iztapalapa, México, Oct. 1976. 182

1192. T. Maillart, M. Zhao, J. Grossklags, and J. Chuang. Given enough eyeballs, all bugs are shallow? Revisiting Eric Raymond with bug bounty programs. *Journal of Cybersecurity*, 3(2):81–90, June 2017. 109

1193. S. Majd and R. S. Pindyck. Time to build, option value, and investment decisions. *Journal of Financial Economics*, 18(1):7–27, Mar. 1987. 62

1194. V. Makarov. 2016 Export of Russian software development industry. Annual Survey 13-th, RUSSOFT Association, Aug. 2016. 62, 113

1195. V. N. Makarow. SPEC benchmark page. http://vmakarov.fedorapeople.org/spec/index.html, July 2014. 373, 374

1196. B. A. Malloy and J. F. Power. Quantifying the transition from Python 2 to 3: An empirical study of Python applications. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM'17, pages 314–323, Dec. 2017. 202

1197. S. Mandal, R. Gandhi, and H. Siy. Modular norm models: practical representation and analysis of contractual rights and obligations. *Requirements Engineering*, 25(3):383–412, Sept. 2020. 125

1198. M. Mangel and F. J. Samaniego. Abraham Wald's work on aircraft survivability. *Journal of the American Statistical Association*, 79(386):259–267, June 1984. 255

1199. Manpower. Computers in offices. Studies No. 4, Manpower Research Unit, Ministry of Labour, G.B., 1965. 92

1200. M. M. Mantei and T. J. Teorey. Cost/benefit analysis for incorporating human factors in the software lifecycle. *Communications of the ACM*, 31(4):428–438, Apr. 1988. 130

1201. M. V. Mäntylä and J. Itkonen. How are software defects found? The role of implicit defect detection, individual responsibility, documents, and knowledge. *Information and Software Technology*, 56(12):1597–1612, Dec. 2014. 174

1202. A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree. An empirical survey of performance and energy efficiency variation on Intel processors. In *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, E2SC'17, pages 1–8, Nov. 2017. 369

1203. A. Marchand. The power of an installed base to combat lifecycle decline: The case of video games. *International Journal of Research in Marketing*, 33(1):140–154, Mar. 2016. 87

1204. M. Marcozzi, Q. Tang, A. F. Donaldson, and C. Cadar. Compiler fuzzing: How much does it matter? *ACM Transactions on Programming Languages and Systems*, 3:155, Oct. 2019. 171

1205. J. N. Marewski and L. J. Schooler. Cognitive niches: An ecological model of strategy selection. *Psychological Review*, 118(3):292–437, July 2011. 53

1206. B. H. Margolin, R. P. Parmelee, and M. Schatzoff. Analysis of free-storage algorithms. *IBM Systems Journal*, 10(4):283–304, 1971. 203

1207. P. Marinescu, P. Hosek, and C. Cadar. COVRIG: A framework for the analysis of code, test, and coverage evolution in real software. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ISSTA'14, pages 93–104, July 2014. 172

1208. F. Marotta-Wurgler. What's in a standard form contract? An empirical analysis of software license agreements. *Journal of Empirical Legal Studies*, 7(4):677–713, Dec. 2007. 124

1209. F. Marotta-Wurgler. Will increased disclosure help? Evaluating the recommendations of the ALI's "principles of the law of software contracts". *University of Chicago Law Review*, 78(1), 2011. 70

1210. D. Martin. *An Empirical Analysis of GNU Make Feature Use in Open Source Projects*. PhD thesis, School of Computing, Queen's University, Ontario, Apr. 2017. 198, 199

1211. F. Martineau. PNFG: A framework for computer game narrative analysis. Thesis (m.s.), School of Computer Science, McGill University, June 2006. 185

1212. A. G. Martínez. *Chaos Monkeys: Inside The Silicon Valley Money Machine*. Ebury Press, 2016. 93

1213. M. Martinez and M. Monperrus. Mining software repair models for reasoning on the search space of automated program fixing. In *eprint arXiv:cs.SE/1311.3414v1*, Nov. 2013. 193

1214. M. Maruyama, C. Pallier, A. Jobert, M. Sigman, and S. Dehaene. The cortical representation of simple mathematical expressions. *NeuroImage*, 61(4):1444–1460, July 2012. 29

1215. E. Masanet, A. Shehabi, J. Liang, L. Ramakrishnan, X. Ma, V. Hendrix, B. Walker, and P. Mantha. The energy efficiency potential of cloud-based software: A U.S. case study. LBNL Paper LBNL-6298E, Lawrence Berkeley National Laboratory, June 2013. 95

1216. E. J. Masicampo and D. R. Lalande. A peculiar prevalence of *p* values just below .05. *The Quarterly Journal of Experimental Psychology*, 65(11):2271–2279, Aug. 2012. 268

1217. F. Massacci, S. Neuhaus, and V. H. Nguyen. After-life vulnerabilities: A study on Firefox evolution, its vulnerabilities, and fixes. In *Proceedings of the Third International Conference on Engineering secure software and systems*, ESSoS'11, pages 195–208, Feb. 2011. 160, 161, 162

1218. R. C. Masse, C. Liu, Y. Li, L. Mai, and G. Cao. Energy storage through intercalation reactions: electrodes for rechargeable batteries. *National Science Review*, 4(1):26–53, 2017. 368

1219. J. Matejka and G. Fitzmaurice. Same stats, different graphs: Generating datasets with varied appearance and identical statistics through simulated annealing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI'17, pages 1290–1294, May 2017. 386

1220. B. G. Mateus and M. Martinez. On the adoption, usage and evolution of Kotlin features on Android development. In *eprint arXiv:cs.CS/1907.09003*, July 2019. 113

1221. F. Mathy, M. Chekaf, and N. Cowan. Simple and complex working memory tasks allow similar benefits of information compression. *Journal of Cognition*, 1(31):1–12, May 2018. 32, 33

1222. E. Matias, I. S. MacKenzie, and W. Buxton. One-handed touch-typing on a QWERTY keyboard. *International Journal of Human-Computer Interaction*, 11:1–27, 1996. 23

1223. C. Mayer, S. Hanenberg, R. Robbes, É. Tanter, and A. Stefik. An empirical study of the influence of static type systems on the usability of undocumented software. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA'12, pages 683–702, Oct. 2012. 206

1224. D. Mazinanian, A. Ketkar, N. Tsantalis, and D. Dig. Understanding the use of lambda expressions in Java. In *Proceedings of the ACM on Programming Languages*, OOPSLA'17, page 85, Oct. 2017. 202

1225. A. Mazouz. *An Empirical Study of Program Performance of OpenMP Applications on Multicore Platforms*. PhD thesis, Université de Versailles-Saint Quentin en Yvelines, Dec. 2013. 372, 373

1226. A. Mazouz, A. Laurent, B. Pradelle, and W. Jalby. Evaluation of CPU frequency transition latency. *Computer Science - Research and Development*, 29(3-4):187–195, Aug. 2014. 368

1227. M. Mazzucato. Risk, variety and volatility in the PC industry: *New* economy or *Early* life-cycle? In *NY Federal Reserve Bank conference on "Productivity Growth: A New Era?"*, Nov. 2001. 100

1228. D. F. McAllister and M. A. Vouk. Experiments in fault tolerant software reliability. Technical Report 5–NAG-1-667, North Carolina State University, Apr. 1989. 130, 175

1229. T. J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, Dec. 1976. 183

1230. J. C. McCallum. Historical cost of computer memory and storage. http://www.jcmit.com, July 2016. 1

1231. S. McCloud. *Understanding Comics*. HarperPerennial, 1993. 228

1232. S. McConnell. *Code Complete*. Microsoft Press, 1993. 184

1233. M. H. McCormack. *The Terrible Truth about Lawyers*. Beech Tree books, William Morrow, 1987. 124

1234. M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin*, 33(4):125–180, June 2001. 361

1235. R. R. McCrae and P. T. Costa, Jr. Reinterpreting the Myers-Briggs type indicator from the perspective of the five-factor model of personality. *Journal of Personality*, 57(1):17–40, Mar. 1989. 52

1236. B. D. McCullough. Microsoft Excel's 'Not The Wichmann-Hill' random number generator. *Computational Statistics & Data Analysis*, 52:4587–4593, 2008. 165

1237. B. D. McCullough and D. A. Heiser. On the accuracy of statistical procedures in Microsoft Excel 2007. *Computational Statistics & Data Analysis*, 52:4570–4578, 2008. 15

1238. J. Mcdonald. The impact of project planning team experience on software project cost estimates. *Empirical Software Engineering*, 10(2):219–234, Apr. 2005. 125, 126

1239. R. McElreath and R. Boyd. *Mathematical Models of Social Evolution: A Guide for the Perplexed*. The University of Chicago Press, 2008. 75, 100

1240. D. McFadden. Rationality for economists? *Journal of Risk and Uncertainty*, 19:73–105, 1999. 53

1241. R. W. McGee. *Accounting for Software in the United States*. PhD thesis, School of Industrial and Business Studies, University of Warwick, Apr. 1986. 82

1242. B. McGonigle, M. Chalmers, and A. Dickinson. Concurrent disjoint and reciprocal classification by Cebus apella in seriation tasks: evidence for hierarchical organization. *Animal Cognition*, 6(3):185–197, Sept. 2003. 41

1243. S. McJohn. The GPL meets the UCC: Does free software come with a warranty of no infringement? *Journal of High Technology Law*, XV(1):1–62, 2014. 70

1244. K. B. McKeithen, J. S. Reitman, H. H. Rueter, and S. C. Hirtle. Knowledge organization and skill differences in computer programmers. *Cognitive Psychology*, 13(3):307–325, July 1981. 39, 40

1245. J. McManus and T. Wood-Harper. Understanding the sources of information systems project failure: A study in IS project failure. *Management Services*, 51(3):38–43, Aug. 2007. 122

1246. D. S. McNamara and J. Magliano. Toward a comprehensive model of comprehension. In B. Ross, editor, *The Psychology of Learning and Motivation, Vol. 51*, chapter 9, pages 297–384. Academic Press, Nov. 2009. 190

1247. T. P. McNamara, J. K. Hardy, and S. C. Hirtle. Subjective hierarchies in spatial memory. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 15(2):211–227, 1989. 30

1248. J. McNerney, J. D. Farmer, S. Redner, and J. E. Trancik. Role of design complexity in technology improvement. *Proceedings of the National Academy of Sciences*, 108(22):9008–9013, May 2011. 76

1249. D. L. McNicol. Influences on the timing and frequency of cancellations and truncations of major defense acquisition programs. IDA Paper P-8280, Institute for Defense Analyses, Mar. 2017. 122

1250. I. McPhee. Customs' cargo management re-engineering project: Australian customs service. Audit Report No.24 2006-07, Australian National Audit Office, Aug. 2007. 122

1251. J. H. McWhorter. The world's simplest grammars are creole grammars. *Linguistic Typology*, 5(2-3):125–166, 2001. 201

1252. A. D. Meacham. *Data Processing Equipment Encyclopedia: Electronic Devices*, volume 2. Gille Associates, Inc., 1962. 112, 365

1253. F. Mechner. Probability relations within response sequences under ratio reinforcement. *Journal of the Experimental Analysis of Behavior*, 1(2):109–121, Apr. 1958. 20

1254. F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, and S. Apel. A comparison of 10 sampling algorithms for configurable systems. In *eprint arXiv:cs.SE/1602.02052*, Feb. 2016. 140, 169, 255

1255. F. Medeiros, C. Kästner, M. Ribeiro, S. Nadi, and R. Gheyi. The love/hate relationship with the C preprocessor: An interview study. In *Proceedings of the 29th. European Conference on Object-Oriented Programming*, ECOOP'15, pages 495–518, July 2015. 184

1256. M. Meeks. German comments in LibreOffice. personal webpage, Apr. 2017. https://people.gnome.org/~michael/data/2015-08-01-5.5-data.ods. 107

1257. M. Meeter, J. M. J. Murre, and S. M. J. Janssen. Remembering the news: Modeling retention data from a study with 14,000 participants. *Memory & Cognition*, 33(5):793–810, July 2004. 35, 36

1258. M. Mehrara and T. Austin. Exploiting selective placement for low-cost memory protection. *ACM Transactions on Architecture and Code Optimization*, 5(3):1–24, Nov. 2008. 166

1259. L. K. Melhus and R. E. Jensen. Measurement bias from address aliasing. In *Eleventh International Workshop on Automatic Performance Tuning*, iWAPT 2016, pages 1506–1515, May 2016. 370

1260. R. Meloca, G. Pinto, L. Baiser, M. Mattos, I. Polato, I. S. Wiese, and D. M. German. Understanding the usage, impact, and adoption of non-OSI approved licenses. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR'18, pages 270–280, May 2018. 69, 70

1261. M. Mencher. *Get in the Game! Careers in the Game Industry*. New Riders Publishing, Oct. 2002. 108

1262. D. Mendez, B. Baudry, and M. Monperrus. Empirical evidence of large-scale diversity in API usage of object-oriented software. In *International Conference on Source Code Analysis and Manipulation*, SCAM'13, pages 43–52, Apr. 2013. 207, 208

1263. H. Mengistu, J. Huizinga, J.-B. Mouret, and J. Clune. The evolutionary origins of hierarchy. *PLoS Computational Biology*, 12(6):e1004829, June 2016. 184

1264. H. Mercier and D. Sperber. Why do humans reason? Arguments for an argumentative theory. *Behavioral and Brain Sciences*, 34(2):57–111, Apr. 2011. 44

1265. H. Mercier and D. Sperber. Why do humans reason? Arguments for an argumentative theory. HAL Id: hal-00904097, HAL archives-ouvertes.fr, Nov. 2013. 44

1266. R. C. Merkle. Energy limits to the computational power of the human brain. *Foresight Update*, 6, Aug. 1989. 56

1267. E. W. Merrow, L. McDonnel, and R. W. Argüden. Understanding the outcomes of megaprojects: A quantitative analysis of very large civilian projects. Report R-3560-PSSP, The RAND Corporation, Mar. 1988. 125

1268. S. Mertens and C. Baethge. The virtues of correct citation. *Deutsches Ärzteblatt International*, 108(33):550–552, Apr. 2011. 148

1269. A. Mesoudi. Cultural evolution: A review of theory, findings and controversies. *Evolutionary Biology*, 43(4):481–497, Dec. 2016. 73

1270. P. W. Metzger. *Managing a Programming Project*. Prentice-Hall, Inc, second edition, Mar. 1981. 131

1271. A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering*, 43(12):1178–1193, Dec. 2017. 137

1272. L. A. Meyerovich and A. Rabkin. How not to survey developers and repositories: Experiences analyzing language adoption. In *Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU'12, pages 7–16, Oct. 2012. 115

1273. X. Mi, Y. Zhang, F. Qian, and X. Wang. An empirical characterization of IFTTT: Ecosystem, usage, and performance. In *Proceedings of the 2017 Internet Measurement Conference*, IMC'17, pages 398–404, Nov. 2017. 179

1274. T. Micceri. The unicorn, the normal curve, and other improbable creatures. *Psychological Bulletin*, 105(1):156–166, Apr. 1989. 253

1275. S. E. Michalak, A. J. DuBois, C. B. Storlie, H. M. Quinn, W. N. Rust, D. H. DuBois, D. G. Modl, A. Manuzzato, and S. P. Blanchard. Assessment of the impact of cosmic-ray-induced neutrons on hardware in the Roadrunner supercomputer. *IEEE Transactions on Device and Materials Reliability*, 12(2):445–454, May 2012. 167

1276. J.-B. Michel, Y. K. Shen, A. P. Aiden, A. Veres, M. K. Gray, The Google Books Team, J. P. Pickett, D. Hoiberg, D. Clancy, P. Norvig, J. Orwant, S. Pinker, M. A. Nowak, and E. L. Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 14(6014):176–182, Jan. 2011. 114, 385

1277. Microsoft server protocol documentation. corporate website, 2015. http://www.microsoft.com. 81, 118, 165, 222

1278. S. Milgram. *Obedience to Authority*. McGraw-Hill, 1974. 54

1279. S. Milgram, L. Bickman, and L. Berkowitz. Note on the drawing power of crowds of different size. *Journal of Personality and Social Psychologs*, 13(2):79–82, 1969. 75

1280. A. Mili, S. F. Chmiel, R. Gottumukkala, and L. Zhang. Managing software reuse economics: An integrated ROI-based model. *Annals of Software Engineering*, 11(1):175–218, Nov. 2001. 81

1281. K. Milis. *Success factors for ICT projects: Empirical research, utilising qualitative and quantitative approaches (incl. Bayesian networks, Probabilistic feature models)*. PhD thesis, Toegepaste Economische Wetenschappen, Limburgs Universitair Centrum, Dec. 2002. 120

1282. D. M. Miller. Application of Halstead's timing model to predict the compilation time of Ada compilers. Thesis (m.s.), School of Engineering of the Air Force Institute of Technology, USA, Dec. 1986. 180

1283. D. R. Miller. Exponential order statistic models of software reliability growth. NASA Contractor Report 3909, NASA Langley Research Center, July 1985. 103, 152, 159

1284. G. A. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information. *The Psychological Review*, 63(2):81–97, Mar. 1956. 30, 362

1285. G. A. Miller and S. Isard. Free recall of self-embedded English sentences. *Information and Control*, 7:292–303, 1964. 32

1286. L. A. Miller. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal*, 29(2):184–215, 1981. 198

1287. S. J. Miller and M. J. Nigrini. Order statistics and Benford's law. *International Journal of Mathematics and Mathematical Sciences*, 2008, 2008. 386

1288. W. R. Miller and M. Sanchez-Craig. How to have a high success rate in treatment: advice for evaluators of alcoholism programs. *Addiction*, 91(6):779–785, Apr. 1996. 359

1289. S. Minakawa, T. Hirata, K. Masame, H. Okada, and K. Maruyama. A psychological analysis on the meaning of "reliance". *Tohoku Psychologica Folia*, 46(1-4):111–117, Apr. 1987. 152

1290. C. H. Mireles. *Marketing Modeling for New Products*. PhD thesis, Erasmus University, Rotterdam, June 2010. 87

1291. MISRA. *MISRA-C:2004 Guidelines for the Use of the C Language in Vehicle Based Software*. Motor Industry Research Association, 2004. 153, 184, 203

1292. MISRA. *MISRA-C++:2008 Guidelines for the Use of the C++ Language in Critical Systems*. Motor Industry Research Association, June 2008. 153

1293. K. E. Mitchell. The copyleft bust up: loopholes, licenses, and realpolitik in open source. blog: /dev/lawyer blog, Nov. 2018. https://writing.kemitchell.com/2018/11/04/Copyleft-Bust-Up.html. 68

1294. R. K. Mitchell, B. R. Agle, and D. J. Wood. Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. *The Academy of Management Review*, 22(4):853–886, Oct. 1997. 136

1295. K. Mitropoulou. *Performance Optimizations for Compiler-based Error Detection*. PhD thesis, School of Informatics, University of Edinburgh, Oct. 2014. 167

1296. S. Mittal. A survey of architectural techniques for managing process variation. *ACM Computing Surveys*, 48(4), May 2016. 367

1297. S. Mittal. A survey of value prediction techniques for leveraging value locality. *Concurrency and Computation: Practice and Experience*, 29(21):e4250, Nov. 2017. 203

1298. S. Mittal. A survey of techniques for dynamic branch prediction. In *eprint arXiv:cs.AR/1804.00261*, Apr. 2018. 203

1299. M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2003. 239

1300. M. Mitzenmacher. Dynamic models for file sizes and double Pareto distributions. *Internet Mathematics*, 1(3):305–333, Apr. 2004. 227, 246

1301. O. Mlouki. On the detection of licenses violations in the Android ecosystem. Thesis (m.s.), Université de Montréal, Apr. 2016. 69

1302. A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance*, ICSM'00, pages 120–130, Oct. 2000. 143

1303. A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2), Apr.-June 2000. 37

1304. S. N. Mohanty. Software cost estimation: Present and future. *Software–Practice and Experience*, 11(2):103–121, Feb. 1981. 128

1305. T. Moher and G. M. Schneider. Methods for improving controlled experimentation in software engineering. In *Proceedings of the 5th International Conference on Software Engineering*, ICSE'81, pages 224–233, Mar. 1981. 361

1306. K. Moløkken-Østvold and K. M. Furulund. The relationship between customer collaboration and software project overruns. In *2007 Agile Conference*, AGILE'07, pages 72–83, Aug. 2007. 310

1307. K. Moløkken-Østvold, M. Jørgensen, S. S. Tanilkan, H. Gallis, A. C. Lien, and S. E. Hove. A survey on software estimation in the Norwegian industry. In *Proceedings 10th International Symposium on Software Metrics*, pages 208–219, Sept. 2004. 122, 123

1308. C. Montalvo, D. Peck, and E. Rietveld. A longer lifetime for products: Benefits for consumers and companies. Study IP/A/IMCO/2015-11, Policy Department A: Economic and Scientific Policy, European Parliament, June 2016. 98

1309. G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8):114–117, Apr. 1965. 7

1310. S. K. Moore. The node is nonsense: There are better ways to measure progress than the old Moore's law. *IEEE Spectrum*, 57(8):25–30, Aug. 2020. 7

1311. A. C. Morgan, D. J. Economou, S. F. Way, and A. Clauset. Prestige drives epistemic inequality in the diffusion of scientific ideas. In *eprint arXiv:cs.SI/1805.09966*, Oct. 2018. 74

1312. T. J. H. Morgan, L. E. Rendell, M. Ehn, W. Hoppitt, and K. N. Laland. The evolutionary basis of human social learning. *Proceedings of the Royal Society B: Biological Sciences*, 279(1729):653–662, Jan. 2012. 54

1313. F. L. Morris and C. B. Jones. An early program proof by Alan Turing. *Annals of the History of Computing*, 6(2):139–143, Apr. 1984. 147

1314. R. J. Morrison, R. E. Nolan, and J. S. Devlin. *Work Measurement in Machine Accounting: Controls, Incentives, Scheduling, and Costing Procedures*. The Ronald Press Company, Dec. 1963. 73

1315. S. P. Morse, B. W. Ravenel, S. Mazor, and W. B. Pohlman. Intel microprocessors: 8008 to 8086. *IEEE Computer*, 13(10):42–60, Oct. 1980. 94

1316. T. Moscibroda and R. Oshman. Resilience of mutual exclusion algorithms to transient memory faults. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, PODC'11, pages 69–78, June 2011. 167

1317. F. Mosteller and C. Youtz. Quantifying probabilistic expressions. *Statistical Science*, 5(1):2–12, Feb. 1990. 153

1318. C. Motta. Analysing the evolution of system requirements – A case study of AUTOSAR at Volvo car group. Thesis (m.s.), Department of Computer Science and Engineering, Gothenburg, June 2016. 105, 106

1319. J. F. Motz. IN RE MICROSOFT CORPORATION ANTITRUST LITIGATION * SUN MICROSYSTEMS, INC. v. MICROSOFT CORPORATION. Opinion MDL 1332 * Civil No. JFM-02-2739, UNITED STATES DISTRICT COURT FOR THE DISTRICT OF MARYLAND, 2002. 110, 172

1320. Y. Moy and A. Wallenburg. Tokeneer: Beyond formal program verification. In *Proceedings of the 5th International Congress on Embedded Real Time Software and Systems*, *ERTS*^{2} 2010, May 2010. 168

1321. S. T. Mueller and A. Krawitz. Reconsidering the two-second decay hypothesis in verbal working memory. *Journal of Mathematical Psychology*, 53(1):14–25, Feb. 2009. 31

1322. S. T. Mueller and C. T. Weidemann. Alphabetic letter identification: Effects of perceivability, similarity, and bias. *Acta Psychologica*, 139(1):19–37, Jan. 2012. 23, 196

1323. D. Mulcahy, B. Weeks, and H. S. Bradley. "we have met the enemy... and he is us" Lessons from twenty years of the Kauffman Foundation's investments in venture capital funds and the triumph of hope over experience. Report, Ewing Marion Kauffman Foundation, May 2012. 93

1324. C. W. Mulford and S. Misra. Capitalization of software development costs: Accounting practices in the software industry, 2014 and 2015. Technical report, Georgia Tech College of Management, Jan. 2016. 62, 83, 84

1325. C. W. Mulford and J. Roberts. Capitalization of software development costs: A survey of accounting practices in the software industry. Technical report, Georgia Tech College of Management, May 2006. 84

1326. M. M. Müller and A. Höfer. The effect of experience on the test-driven development process. *Empirical Software Engineering*, 12(6):593–615, 2007. 216, 360, 379

1327. J. Mulligan and B. Patrovsky. *Developing Online Games: An Insider's Guide*. New Riders Publishing, 2003. 130

1328. E. Mumford. *Job Satisfaction: A study of computer specialists*. Longman Group Limited, Oct. 1972. 68, 71

1329. A. M. Munñiz Jr. and H. J. Schau. Religiosity in the abandoned Apple Newton brand community. *Journal of Consumer Research*, 31(4):737–747, Mar. 2005. 74

1330. D. Muna, M. Alexander, A. Allen, R. Ashley, D. Asmus, R. Azzollini, M. Bannister, R. Beaton, A. Benson, G. B. Berriman, M. Bilicki, P. Boyce, J. Bridge, J. Cami, E. Cangi, X. Chen, N. Christiny, C. Clark, M. Collins, J. Comparat, N. Cook, D. Croton, I. D. Davids, É. Depagne, J. Donor, L. A. dos Santos, S. Douglas, A. Du, M. Durbin, D. Erb, D. Faes, J. G. Fernández-Trincado, A. Foley, S. Fotopoulou, S. Frimann, P. Frinchaboy, R. Garcia-Dias, A. Gawryszczak, E. George, S. Gonzalez, K. Gordon, N. Gorgone, C. Gosmeyer, K. Grasha, P. Greenfield, R. Grellmann, J. Guillochon, M. Gurwell, M. Haas, A. Hagen, D. Haggard, T. Haines, P. Hall, W. Hellwing, E. C. Herenz, S. Hinton, R. Hlozek, J. Hoffman, D. Holman, B. W. Holwerda, A. Horton, C. Hummels, D. Jacobs, J. J. Jensen, D. Jones, A. Karick, L. Kelley, M. Kenworthy, B. Kitchener, D. Klaes, S. Kohn, P. Konorski, C. Krawczyk, K. Kuehn, T. Kuutma, M. T. Lam, R. Lane, J. Liske, D. Lopez-Camara, K. Mack, S. Mangham, Q. Mao, D. J. E. Marsh, C. Mateu, L. Maurin, J. McCormac, I. Momcheva, H. Monteiro, M. Mueller, R. Munoz, R. Naidu, N. Nelson, C. Nitschelm, C. North, J. Nunez-Iglesias, S. Ogaz, R. Owen, J. Parejko, V. Patrício, J. Pepper, M. Perrin, T. Pickering, J. Piscionere, R. Pogge, R. Poleski, A. Pourtsidou, A. M. Price-Whelan, M. L. Rawls, S. Read, G. Rees, H. Rein, T. Rice, S. Riemer-Sørensen, N. Rusomarov, S. F. Sanchez, M. Santander-García, G. Sarid, W. Schoenell, A. Scholz, R. L. Schuhmann, W. Schuster, P. Scicluna, M. Seidel, L. Shao, P. Sharma, A. Shulevski, D. Shupe, C. Sifón, B. Simmons, M. Sinha, I. Skillen, B. Soergel, T. Spriggs, S. Srinivasan, A. Stevens, O. Streicher, E. Suchyta, J. Tan, O. G. Telford, R. Thomas, C. Tonini, G. Tremblay, S. Tuttle, T. Urrutia, S. Vaughan, M. Verdugo, A. Wagner, J. Walawender, A. Wetzel, K. Willett, P. K. G. Williams, G. Yang, G. Zhu, and A. Zonca. The Astropy problem. In *eprint arXiv:astro-ph.IM/1610.03159*, Oct. 2016. 74, 121

1331. B. B. Murdoch, Jr. The serial position effect of free recall. *Journal of Experimental Psychology*, 64(5):482–488, 1962. 34

1332. E. M. Murphy. In the matter of Knight Capital Americas LLC respondent. Order instituting administrative and cease-and-desist proceedings, pursuant to sections 15(b) and 21c of the securities exchange Act of 1934, making findings, and imposing remedial sanctions and a cease-and-desist order. Administrative Proceeding File No. 3-15570, Securities and Exchange Commission, Oct. 2013. 154

1333. E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE'09, pages 287–297, Apr. 2009. 139

1334. J. M. J. Murre and A. G. Chessa. Power laws from individual differences in learning and forgetting: mathematical analyses. *Psychonomic Bulletin & Review*, 18(3):592–597, June 2011. 247

1335. J. M. J. Murre and J. Dros. Replication and analysis of Ebbinghaus' forgetting curve. *PLoS ONE*, 10(7):e0120644, July 2015. 35

1336. P. Murrell. *R Graphics*. Chapman & Hall/CRC, 1st edition, 2006. 225

1337. M. Muthukrishna, J. Henrich, W. Toyokawa, T. Hamamura, T. Kameda, and S. J. Heine. Overconfidence is universal? Elicitation of genuine overconfidence (EGO) procedure reveals systematic differences across domain, task knowledge, and incentives in four populations. *PLoS ONE*, 13(8):e0202288, Aug. 2018. 56

1338. M. Muthukrishna, B. W. Shulman, V. Vasilescu, and J. Henrich. Sociality influences cultural complexity. *Proceedings of the Royal Society B: Biological Sciences*, 281(1774), Nov. 2013. 77

1339. L. H. Mutuel. Single event effects mitigation techniques report. Final Report DOT/FAA/TC-15/62, U.S. Department of Transportation, Federal Aviation Administration, Feb. 2016. 166

1340. G. J. Myers. A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM*, 21(9):760–768, Sept. 1978. 169

1341. T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. We have it easy, but do we have it right? In *International Symposium on Parallel and Distributed Processing*, IPDPS 2008, pages 1–7, Apr. 2008. 372

1342. T. Mytkowicz, P. F. Sweeney, M. Hauswirth, and A. Diwan. Observer effect and measurement bias in performance analysis. Technical Report CU-CS 1042-08, University of Colorado at Boulder, June 2008. 370

1343. M. Nagappan, R. Robbes, Y. Kamei, É. Tanter, S. McIntosh, A. Mockus, and A. E. Hassan. An empirical study of goto in C code from GitHub repositories. In *Proceedings of the 10th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'15, pages 404–414, Aug.-Sept. 2015. 204

1344. M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'13, pages 466–476, Aug. 2013. 255

1345. N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *IEEE 21st International Symposium on Software Reliability Engineering*, ISSRE'10, pages 309–318, Nov. 2010. 314

1346. P. M. Nagel, F. W. Scholz, and J. A. Skrivan. Software reliability: Additional investigations into modeling with replicated experiments. NASA Contractor Report 172378, Boeing Computer Services Company, Space and Military Applications Division, June 1984. 157

1347. P. M. Nagel and J. A. Skrivan. Software reliability: Repetitive run experimentation and modeling. NASA Contractor Report 165836, Boeing Computer Services Company, Space and Military Applications Division, Feb. 1982. 156, 157

1348. T. Nagle, J. Hogan, and J. Zale. *The Strategy and Tactics of Pricing*. Pearson, fifth edition, 2015. 85

1349. J. S. Nairne. The loss of positional certainty in long-term memory. *Psychological Science*, 3(2):199–202, May 1992. 34

1350. J. S. Nairne. Adaptive memory: Evolutionary constraints on remembering. In B. H. Ross, editor, *Psychology of Learning and Motivation, Volume 53*, chapter 1, pages 1–32. Academic Press, June 2010. 29

1351. J. Nandhakumar and D. E. Avison. The fiction of methodological development: a field study of information systems development. *Information Technology & People*, 12(2):176–191, Feb. 1999. 131

1352. S. Nanz and C. A. Furia. A comparative study of programming languages in Rosetta Code. In *eprint arXiv:cs.SE/1409.0252v1*, Aug. 2014. 197

1353. E. Nasseri. *An Empirical Investigation of Inheritance Trends in Java OSS Evolution*. PhD thesis, Department of Information Systems, Computing and Mathematics, Brunel University, June 2009. 208

1354. M. B. Nathanson. Desperately seeking mathematical truth. *Notices of the AMS*, 55(7):773–773, Aug. 2008. 148

1355. National Research Council. *Setting Priorities for Large Research Facility Projects Supported by the National Science Foundation (2004)*. The National Academies Press, Jan. 2004. 2

1356. National Statistics, UK Office for. GFCF estimates for computer software purchases, own account computer software. ONS website, June 2017. http://www.ons.gov.uk/ons/rel/bus-invest/business-investment/index.html. 6, 105

1357. P. Naur and B. Randell. Software engineering report on a conference sponsored by the NATO science committee. Technical report, NATO, Jan. 1969. 8, 112

1358. A. D. Navarro and E. Fantino. The sunk cost effect in pigeons and humans. *Journal of the Experimental Analysis of Behavior*, 83(1):1–13, Jan. 2005. 59

1359. D. J. Navarro and A. F. Perfors. Hypothesis generation, sparse categories, and the positive test strategy. *Psychological Review*, 118(1):120–134, Jan. 2011. 25

1360. I. Neamtiu, J. S. Foster, and M. Hicks. Understanding source code evolution using abstract syntax tree matching. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR'05, pages 1–5, May 2005. 208, 209

1361. I. G. Neamtiu. *Practical Dynamic Software Updating*. PhD thesis, University of Maryland, College Park, Aug. 2008. 208

1362. S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig. Is it dangerous to use version control histories to study source code evolution? In *Proceedings of the 26th European conference on Object-Oriented Programming*, ECOOP'12, pages 79–103, June 2012. 200, 381

1363. D. L. Nelson, C. L. McEvoy, and T. A. Schreiber. The university of South Florida word association, rhyme and word fragment norms. w3.usf.edu/FreeAssociation, 1998. 195

1364. E. A. Nelson. Management handbook for the estimation of computer programming costs. Technical Documentary Report ESD-TDR-67-66, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Oct. 1966. 127

1365. D. A. Nembhard and N. Osothsilp. An empirical comparison of forgetting models. *IEEE Transactions on Engineering Management*, 48(3):283–291, Aug. 2001. 76

1366. R. E. NeSmith II. A study of software maintenance costs of Air Force large scale computer systems. Thesis (m.s.), School of Systems and Logistics, Air Force Institute of Technology, USA, Sept. 1986. 120, 303

1367. D. Nettle. Explaining global patterns of language diversity. *Journal of Anthropological Archaeology*, 17(4):354–374, Dec. 1998. 113

1368. B. New, L. Ferrand, C. Pallier, and M. Brysbaert. Reexamining the word length effect in visual word recognition: New evidence from the English lexicon project. *Psychonomic Bulletin & Review*, 13(1):45–52, Feb. 2006. 196

1369. A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1991. 20

1370. A. Newell and P. S. Rosenbloom. Mechanisms of skill acquisition and the power law of practice. Technical report, Carnegie Mellon University, Aug. 1982. 35

1371. S. E. Newstead and K. R. Coventry. The role of context and functionality in the interpretation of quantifiers. *European Journal of Cognitive Psychology*, 12(2):243–259, June 2000. 50

1372. G. Nezlek and G. DeHondt. An empirical investigation of gender wage differences in information systems occupations: 1991-2008. In *Proceedings of the 43rd Hawaii International Conference on System Sciences–2010*, HICSS, pages 4059–4068, Jan. 2010. 71

1373. T. H. D. Nguyen, B. Adams, and A. E. Hassan. A case study of bias in bug-fix datasets. In *17th Working Conference on Reverse Engineering*, WCRE'10, pages 259–268, Oct. 2010. 152

1374. V. H. Nguyen and F. Massacci. The (un)reliability of NVD vulnerable versions data: an empirical experiment on Google Chrome vulnerabilities. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*, ASIA CCS'13, pages 493–498, May 2013. 152

1375. T. Nicholas. *VC: An American History*. Harvard University Press, June 2019. 93

1376. T. Nichols. A penny saved: Psychological pricing. blog: Gumroad, Oct. 2013. http://blog.gumroad.com/post/64417917582/a-penny-saved-psychological-pricing. 86

1377. W. Nichols. The end to the myth of "Individual Programmer Productivity". *IEEE Software*, 36(5):71–75, Sept.-Oct. 2019. 57

1378. W. R. Nichols, J. D. McHale, D. Sweeney, W. Snavely, and A. Volkman. Composing effective software security assurance workflows. Technical Report CMU/SEI-2018-TR-004, Software Engineering Institute, Carnegie Mellon University, Oct. 2018. 73, 141, 161, 168, 169

1379. A. Nieder. The adaptive value of numerical competence. *Trends in Ecology & Evolution*, 35(7):605–617, July 2020. 20

1380. J. Nielsen and T. K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the conference on Human factors in computing systems*, INTERCHI'93, pages 206–213, Apr. 1993. 170

1381. E. B. Nightingale, J. R. Douceur, and V. Orgovan. Cycles, cells and platters: An empirical analysis of hardware failures on a million consumer PCs. In *Proceedings of the sixth conference on Computer systems*, EuroSys'11, pages 343–356, Apr. 2011. 278

1382. M. J. Nigrini and S. J. Miller. Data diagnostics using second order tests of Benford's law. *Auditing: A Journal of Practice and Theory*, 28(2):305–324, June 2009. 386

1383. D. E. Nikonov and I. A. Young. Overview of beyond-CMOS devices and a uniform methodology for their benchmarking. In *eprint arXiv:cond-mat.mes-hall/1302.0244*, Feb. 2013. 369

1384. J. Ninio and K. A. Stevens. Variations on the Hermann grid: an extinction illusion. *Perception*, 29(10):1209–1217, Oct. 2000. 192

1385. R. E. Nisbett, D. H. Krantz, C. Jepson, and Z. Kunda. The use of statistical heuristics in everyday inductive reasoning. *Psychological Review*, 90(4):339–363, Oct. 1983. 41

1386. S. Nørby. Why forget? On the adaptive value of memory loss. *Perspectives on Psychological Science*, 10(5):551–578, Sept. 2015. 35

1387. P. V. Norden. Resource usage and network planning techniques. In B. V. Dean, editor, *Operations Research in Research and Development*, chapter 5, pages 149–169. John Wiley & Sons, Inc, 1963. 128

1388. W. D. Nordhaus. The progress of computing. Cowles Foundation Discussion Paper No. 1324, Yale University, Sept. 2001. 1

1389. J. A. Norton and F. M. Bass. A diffusion theory model of adoption and substitution for successive generations of high-technology products. *Management Science*, 33(9):1069–1086, Sept. 1987. 87

1390. M. A. Nowak. *Evolutionary Dynamics: Exploring the Equations of Life*. The Belknap press of Harvard University press, 2006. 100

1391. M. A. Nowak and K. Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393(6685):573–577, June 1998. 78

1392. D. Nowroth, I. Polian, and B. Becker. A study of cognitive resilience in a JPEG compressor. In *IEEE International Dependable Systems and Networks With FTCS and DCC*, DSN 2008, pages 32–41, June 2008. 166

1393. H.-C. Nuerk, G. Wood, and K. Willmes. The universal SNARC effect: The association between number magnitude and space is amodal. *Experimental Psychology*, 52(3):187–194, 2005. 22, 23

1394. Y. S. Nugroho, H. Hata, and K. Matsumoto. How different are different diff algorithms in Git? Use `-histogram` for code changes. In *eprint arXiv:cs.SE/1902.02467*, July 2019. 358

1395. R. E. Núñez. No innate number line in the human brain. *Journal of Cross-Cultural Psychology*, 42(4):651–668, 2011. 48

1396. J. M. Nuttin, Jr. Affective consequence of mere ownership: The name letter effect in twelve European languages. *European Journal of Social Psychology*, 17:381–402, 1987. 196

1397. NVIDIA. *CUDA CUBLAS Library*. NVIDIA Corporation, CA, USA, 3.1 edition, Aug. 2010. 361

1398. A. Nyman and M. E. Stamer. How to attract talented software developers: Developing a culturally differentiated employee value proposition. Thesis (m.s.), Department of Management and Engineering Industrial Economics, Linköpings universitet, 2013. 74

1399. M. Oaksford and N. Chater. A rational analysis of the selection task as optimal data selection. *Psychological Review*, 101(4):608–631, Oct. 1994. 44

1400. K. Oberauer, S. Lewandowsky, E. Awh, G. D. A. Brown, A. Conway, N. Cowan, C. Donkin, S. Farrell, G. J. Hitch, M. Hurlstone, W. J. Ma, C. C. Morey, D. E. Nee, J. Schweppe, E. Vergauwe, and G. Ward. Benchmarks for models of short term and working memory. *Psychological Bulletin*, 144(9):885–958, Sept. 2018. 30

1401. K. Oberauer, H.-M. Süß, O. Wilhelm, and W. W. Wittmann. The multiple faces of working memory: Storage, processing, supervision, and coordination. *Intelligence*, 31(2):167–193, Mar.-Apr. 2003. 31

1402. M. Ochodek, J. Nawrocki, and K. Kwarciak. Simplifying effort estimation based on use case points. *Information and Software Technology*, 53(3):200–213, Mar. 2011. 129

1403. O. O. Odeh, A. M. Featherstone, and J. S. Bergtold. Reliability of statistical software. *American Journal of Agricultural Economics*, 92(5):1472–1489, Sept. 2010. 15

1404. OECD. *OECD Digital Economy Outlook 2015*. OECD Publishing, 2015. 62

1405. C. Ogden. Killed by Google. https://killedbygoogle.com, Oct. 2020. 65, 99, 100

1406. S. Ogilvie. *The European Guilds: An Economic Analysis*. Princeton University Press, Feb. 2019. 79, 99

1407. J. Oh, D. Batory, M. Heule, M. Myers, and P. Gazzillo. Uniform sampling from Kconfig feature models. Technical Report 19-02, The University of Texas at Austin, Department of Computer Science, 2019. 255

1408. S. Ohlsson. The learning curve for writing books: Evidence from Professor Asimov. *Psychological Science*, 3(6):380–382, Nov. 1992. 39

1409. M. Ohm, H. Plate, A. Sykosch, and M. Meier. Backstabber's knife collection: A review of open source software supply chain attacks. In *eprint arXiv:cs.CR/2005.09535*, May 2020. 161

1410. H. Ohtsuki, C. Hauert, E. Lieberman, and M. A. Nowak. A simple rule for the evolution of cooperation on graphs and social networks. *Nature*, 441(7092):502–505, May 2006. 78

1411. H. Ohtsuki and Y. Iwasa. How should we define goodness?–reputation dynamics in indirect reciprocity. *Journal of Theoretical Biology*, 231(1):107–120, Nov. 2004. 78

1412. P. Oladimeji. Devices, errors and improving interaction design-A case study using an infusion pump. Thesis (m.res.), Department of Computer Science, Swansea University, Oct. 2008. 247

1413. A. Oliner and J. Stearley. What supercomputers say: A study of five system logs. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, DSN'07, pages 575–584, June 2007. 168

1414. P. Oliver. Experiences in building and using compiler validation systems. In R. Merwin and J. Zanca, editors, *AFIPS Conference Proceedings*, Volume 48, pages 1051–1057, June 1979. 171, 172

1415. S. O'Mahony. *Can Medicine be Cured? The Corruption of a Profession*. Head of Zeus Ltd, Feb. 2019. 12

1416. O*NET OnLine. organization website, July 2019. https://www.onetonline.org. 108

1417. Open Group, The. The Austin common standards revision group. http://austingroupbugs.net, July 2017. 163

1418. Open Science Collaboration. Estimating the reproducibility of psychological science. *Science*, 349(6251):aac4716, Aug. 2015. 4

1419. OpenCorporates. UK registered company data. https://opencorporates.com, Mar. 2015. 107

1420. OpenRefine. organization website, Oct. 2014. http://openrefine.org. 378

1421. OpenSignal. Android fragmentation visualized (august 2015). Technical report, OpenSignal, Aug. 2015. 225, 226, 379

1422. A. Orlitsky, A. T. Suresh, and Y. Wu. Optimal prediction of the number of unseen species. *PNAS*, 113(47):13283–13288, Nov. 2016. 104

1423. N. Osaka. Eye fixation and saccade during kana and kanji text reading: Comparison of English and Japanese text processing. *Bulletin of the Psychonomic Society*, 27(6):548–550, 1989. 28

1424. A. T. Oskarsson, L. V. Boven, G. H. McClelland, and R. Hastie. What's next? Judging sequences of binary events. *Psychological Bulletin*, 135(2):262–285, 2009. 21, 51

1425. H. Osman, M. Leuenberger, M. Lungu, and O. Nierstrasz. Tracking null checks in open-source Java systems. In *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, SANER'16, pages 304–313, Mar. 2016. 204

1426. E. Ostrom, J. Walker, and R. Gardner. Covenants with and without a sword: Self-Governance is possible. *The American Political Science Review*, 86(2):404–417, June 1992. 79

1427. L. M. Ottenstein, V. B. Schneider, and M. H. Halstead. Predicting the number of bugs expected in a program module. Technical Report CSD-TR 205, Purdue University, Oct. 1976. 182

1428. M. A. Oumaziz, A. Charpentier, J.-R. Falleri, and X. Blanc. Documentation reuse: Hot or not? An empirical study. In *16th International Conference on Software Reuse*, ICSR 2017, pages 12–27, May 2017. 163

1429. G. L. Ourada. Software cost estimating models: A calibration, validation, and comparison. Thesis (m.s.), Air Force Institute of Technology, USA, Dec. 1991. 8, 128

1430. C. Overney, J. Meinicke, C. Kästner, and B. Vasilescu. How to not get rich: An empirical study of donations in Op€n $our¢e. In *42nd International Conference on Software Maintenance*, ICSE'20, page ???, July 2020. 93

1431. S. Owsowitz and A. Sweetland. Factors affecting coding errors. Research Memorandum RM-4346-PR, The RAND Corporation, Apr. 1965. 23

1432. D. Oyserman, H. M. Coon, and M. Kemmelmeier. Rethinking individualism and collectivism: Evaluation of theoretical assumptions and meta-analyses. *Psychological Bulletin*, 128(1):3–72, Jan. 2002. 54

1433. S. C. Özbek. *Introducing Innovations into Open Source Projects*. PhD thesis, Freie Universität Berlin, Aug. 2010. 132

1434. A. Ozment and S. E. Schechter. Milk or wine: Does software security improve with age? In *USENIX Security Symposium*, SEC'06, pages 93–104, July-Aug. 2006. 144

1435. P. Padfield. *Battleship*. Thistle Publishing, 2015. 5

1436. R. Paleari, L. Martignoni, G. F. Roglia, and D. Bruschi. A fistful of red-pills: How to automatically generate procedures to detect CPU emulators. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, WOOT'09, pages 2–2, Aug. 2009. 148

1437. N. Palix, J. Lawall, and G. Muller. Tracking code patterns over multiple software versions with Herodotus. In *Proceedings of the 9th International Conference on Aspect-Oriented Software Development*, AOSD'10, pages 169–180, Mar. 2010. 154

1438. N. Palix, S. Saha, G. Thomas, C. Calvès, J. Lawall, and G. Muller. Faults in Linux: Ten years later. Technical Report RR-7357, Institut National de Recherche en Informatique et en Automatique, Aug. 2010. 147

1439. J. Pallister, S. Hollis, and J. Bennett. Identifying compiler options to minimise energy consumption for embedded platforms. In *eprint arXiv:cs.PF/1303.6485*, Aug. 2013. 365

1440. E. M. Palmer, T. S. Horowitz, A. Torralba, and J. M. Wolfe. What are the shapes of response time distributions in visual search? *Journal of Experimental Psychology: Human Perception and Performance*, 37(1):58–71, Feb. 2011. 27, 28

1441. S. E. Palmer. *Vision Science: Photons to Phenomenology*. The MIT Press, 1999. 27

1442. H.-Y. Pan, A. Chao, and W. Foissner. A nonparametric lower bound for the number of species shared by multiple communities. *Journal of Agricultural, Biological, and Environmental Statistics*, 14(4):452–468, Dec. 2009. 104

1443. K. Pan. *Using Evolution Patterns to Find Duplicated Bugs*. PhD thesis, Department of Computer Science, University of California at Santa Cruz, Oct. 2006. 163

1444. T. Pani. Loop patterns in C programs. Thesis (m.s.), Fakultät für Informatik der Technischen Universität Wien, Dec. 2013. 182, 205

1445. R. Parker and B. Grimm. Recognition of business and government expenditures for software as investment: Methodology and quantitative impacts, 1959-98. In *BEA Advisory Committee meeting*, May 2000. 61

1446. A. Parkhomenko, A. Redkina, and O. Maslivets. Estimating hedonic price indexes for personal computers in Russia. MPRA Paper No. 5019, Higher School of Economics, Jan. 2007. 86

1447. C. Parnin, C. Bird, and E. Murphy-Hill. Adoption and use of Java generics. *Empirical Software Engineering*, 18(6):1047–1089, Dec. 2013. 186

1448. F. N. Parr. An alternative to the Rayleigh curve model for software development effort. *IEEE Transactions on Software Engineering*, SE-6(3):291–296, May 1980. 128

1449. H. E. Pashler. *The Psychology of Attention*. The MIT Press, 1999. 26

1450. L. Passos, J. Guo, L. Teixeira, K. Czarnecki, A. Wąsowski, and P. Borba. Coevolution of variability models and related artefacts: A case study of the Linux kernel. In *Proceedings of the 17th International Software Product Line Conference*, SPLC'13, pages 91–100, Apr. 2013. 97

1451. A. Patel. Auditors' belief revision: Recency effects of contrary and supporting audit evidence and source reliability. Technical Report 2001-1, Department of AFM/SSE, University of South Pacific, June 2001. 39

1452. M. R. Patterson. *Antitrust Law in the New Economy :Google, Yelp, LIBOR, and the Control of Information*. Harvard University Press, 2017. 94, 102

1453. F. M. Paulus, L. Rademacher, T. A. J. Schäfer, L. Müller-Pinzler, and S. Krach. Journal impact factor shapes scientists' reward signal in the prospect of publication. *PLoS ONE*, 10(11):e0142537, Nov. 2015. 10

1454. A. Pavese and C. Umiltà. Symbolic distance between numerosity and identity moulates Stroop-like interference. *Journal of Experimental Psychology: Human Perception and Performance*, 24(5):1535–1545, 1998. 30

1455. E. Pavese, E. Soremekun, N. Havrikov, L. Grunske, and A. Zeller. Inputs from hell: Generating uncommon inputs from common samples. In *eprint arXiv:cs.SE/1812.07525*, Dec. 2018. 173

1456. J. W. Payne, J. R. Bettman, and E. J. Bettman. *The Adaptive Decision Maker*. Cambridge University Press, 1993. 53, 54

1457. G. Paz-y-Miño C, A. B. Bond, A. C. Kamil, and R. P. Balda. Pinyon jays use transitive inference to predict social dominance. *Nature*, 430:778–781, Aug. 2004. 46

1458. R. D. Pea. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research*, 2(1):25–36, Feb. 1986. 178

1459. J. Pearl. *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2000. 47

1460. R. K. Pearson. The problem of disguised missing data. *ACM SIGKDD Explorations Newsletter*, 8(1):83–92, June 2006. 381

1461. Y. Peers. *Econometric Advances in Diffusion Models*. PhD thesis, Erasmus University, Rotterdam, Dec. 2011. 87

1462. E. Pek. *Corpus-based Empirical Research in Software Engineering*. PhD thesis, Department of Computer Science, Universität Koblenz-Landau, Oct. 2013. 367

1463. D. G. Pelli, C. W. Burns, B. Farell, and D. C. Moore-Page. Feature detection and letter identification. *Vision Research*, 46(28):4646–4674, 2006. 29

1464. J. Peltokorpi and E. Niemi. Effects of group size and learning on manual assembly performance: an experimental study. *International Journal of Production Research*, 57(2):452–469, 2019. 142

1465. E. Peltonen, E. Lagerspetz, P. Nurmi, and S. Tarkoma. Energy modeling of system settings: A crowdsourced approach. In *IEEE International Conference on Pervasive Computing and Communications*, PerCom'15, pages 37–45, Mar. 2015. 368

1466. N. Pennington. Comprehension strategies in programming. In G. Olson, S. Shepard, and E. Soloway, editors, *Empirical Studies of Programmers: Second Workshop*, chapter 7, pages 100–113. Ablex Publishing Corporation, 1987. 187

1467. N. Pennington. Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, 19(3):295–341, July 1987. 34

1468. B. T. Pentland and H. H. Rueter. Organizational routines as grammars of action. *Administrative Science Quarterly*, 39(3):484–510, Sept. 1994. 106

1469. C. Perez. *Technological Revolutions and Financial Capital: The Dynamics of Bubbles and Golden Ages*. Edward Elgar Publishing, 2003. 5, 7

1470. D. Perez and B. Livshits. Smart contract vulnerabilities: Does anyone care? In *eprint arXiv:cs.CR/1902.06710*, Feb. 2019. 151

1471. D. E. Perry and W. M. Evangelist. An empirical study of software interface faults – An update. In *Proceedings of the Twentieth Annual Hawaii International Conference on Systems Sciences, Vol II*, HICSS, pages 113–126, Jan. 1987. 9, 151

1472. D. E. Perry, N. A. Staudenmayer, and L. G. Votta, Jr. Understanding and improving time usage in software development. In A. Fuggetta and A. L. Wolf, editors, *Trends in Software Process*, chapter 5, pages 111–135. John Wiley & Sons, Mar. 1995. 358

1473. D. E. Perry and C. S. Stieg. Software faults in evolving a large, real-time system: a case study. In *Proceedings of the 1993 European Software Engineering Conference*, pages 48–67, Sept. 1993. 151

1474. R. Perugupalli. Empirical assessment of architecture-based reliability of open-source software. Thesis (m.s.), Department of Computer Science and Electrical Engineering, West Virginia University, May 2004. 248

1475. H. Petroski. *Design Paradigms: Case Histories of Error and Judgment in Engineering*. Cambridge University Press, 1994. 148

1476. C. Peukert. Switching costs and information technology: The case of IT outsourcing. In 1st *ICT Conference Munich on ICT and Economic Growth*, Nov. 2010. 140

1477. A. Pewsey, M. Neuhäuser, and G. D. Ruxton. *Circular Statistics in R*. Oxford University Press, 2013. 344, 345, 346

1478. P. M. Pexman and M. J. Yap. Individual differences in semantic processing: Insights from the Calgary semantic decision project. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 44(7):1091–1112, Feb. 2018. 195

1479. R.-H. Pfeiffer. What constitutes software? An empirical, descriptive study of artifacts. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR'20, page ???, June 2020. 178

1480. S. A. Phatak, A. Lovitt, and J. B. Allen. Consonant confusions in white noise. *Journal of the Acoustic Society of America*, 124(2):1220–1233, Aug. 2008. 196

1481. A. Phillips. *Technology and Market Structure: A Study of the Aircraft Industry*. Heath Lexington Books, 1972. 99

1482. C. Phillips. *Order and Structure*. PhD thesis, M.I.T., Aug. 1996. 32

1483. M. Phister, Jr. *Data Processing Technology and Economics*. Santa Monica Publishing Company and Digital Press, second edition, 1979. 8, 105

1484. S. T. Piantadosi. Zipf's word frequency law in natural language: a critical review and future directions. *Psychonomic Bulletin & Review*, 21(5):1112–1130, Oct. 2014. 206

1485. S. T. Piantadosi. A rational analysis of the approximate number system. *Psychonomic Bulletin & Review*, 23(3):877–886, June 2016. 48

1486. R. Pieters and L. Warlop. Visual attention during brand choice: The impact of time pressure and task motivation. *International Journal of Research in Marketing*, 16:1–16, 1999. 28

1487. D. J. Pigott and B. M. Axtens. Online historical encyclopedia of programming languages. http://hopl.info, 2015. 113

1488. R. S. Pindyck. Investments of uncertain cost. *Journal of Financial Economics*, 34(1):53–76, Aug. 1993. 66

1489. J. Pipitone. Software quality in climate modelling. Thesis (m.s.), Department of Computer Science, University of Toronto, 2010. 156

1490. A. M. Pires and C. Amado. Interval estimators for a binomial proportion: Comparison of twenty methods. *REVSTAT-Statistical Journal*, 6(2):165–197, June 2008. 268

1491. P. Pirolli. *Information Foraging Theory: Adaptive Interaction with Information*. Oxford University Press, May 2007. 183

1492. D. J. Pittenger. Measuring the MBTI . . . And coming up short. *Journal of Career Planning and Employment*, 54(1):48–52, Nov. 1993. 52

1493. PK. How many developers are there in America, and where do they live? company website, Apr. 2019. https://dqydj.com/number-of-developers-in-america-and-per-state. 103

1494. J. Plamondon. Effective Evangelism: JOE COMES, RILEY PAINT, INC., SKEFFINGON'S FORMAL WEAR, INC., PATRICIA ANNE LARSEN vs. MICROSOFT CORPORATION. Plaintiff's Exhibit 3096, IOWA District Court for Polk County, Jan. 2000. 86

1495. A. Pluchino, A. Rapisarda, and C. Garofalo. The Peter principle revisited: A computational study. In *eprint arXiv:physics.soc-ph/0907.0455v3*, Oct. 2009. 71

1496. T. Plum. *Reliable data structures in C*. Plum Hall, 1985. 184

1497. T. Plum. *C Programming guidelines*. Plum Hall, 1989. 184

1498. I. P. L. Png. On the reliability of software piracy statistics. *Electronic Commerce Research and Applications*, 9(5):365–373, Sept.-Oct. 2010. 88

1499. A. Pogačnik and A. Črnič. iReligion: Religious elements of the Apple phenomenon. *The Journal of Religion and Popular Culture*, 26(3):353–364, Sept.-Nov. 2014. 74

1500. C. Poivey, J. L. Barth, K. A. LaBel, G. Gee, and H. Safren. In-flight observations of long-term single-event effect (SEE) performance on Orbview-2 solid state recorders (SSR). In *2003 IEEE Radiation Effects Data Workshop*, pages 102–107, July 2003. 225

1501. C. Politowski, F. Petrillo, G. C. Ullmann, J. de Andrade Werly, and Y.-G. Guéhéneuc. Dataset of video game development problems. In *eprint arXiv:cs.SE/2001.00491*, Jan. 2020. 125

1502. R. Pollack. How to believe a machine-checked proof. In G. Sambin and J. M. Smith, editors, *Twenty Five Years of Constructive Type Theory*, chapter 11, pages 205–220. Oxford University Press, Oct. 1998. 149

1503. A. Pollatsek, E. D. Reichle, and K. Rayner. Tests of the E-Z reader model: Exploring the interface between cognition and eye-movement control. *Cognitive Psychology*, 52(1):1–56, Feb. 2006. 29

1504. G. Poo-Caamaño and D. M. German. Software patents: A replication study. In *Proceedings of the 11th International Symposium on Open Collaboration*, OpenSym'15, pages 5:1–5:4, Aug. 2015. 68

1505. D. Pope and U. Simonsohn. Round numbers as goals: Evidence from baseball, SAT takers, and the lab. *Psychological Science*, 22(1):71–79, Jan. 2011. 49

1506. K. R. Popper. *Conjectures and Refutations*. Routledge, 1969. 25

1507. A. Porter, H. Siy, A. Mockus, and L. Votta. Understanding the sources of variation in software inspections. *ACM Transactions on Software Engineering Methodology*, 7(1):41–79, Jan. 1998. 169, 303, 360

1508. M. E. Porter. *Competitive Advantage: Creating and Sustaining Superior Performance*. First Free Press, 1985. 85

1509. M. E. Porter. The five competitive forces that shape strategy. *Harvard Business Review*, 86(1):78–93, Jan. 2008. 63, 100

1510. R. D. Portugal and B. F. Svaiter. Weber-Fechner law and the optimality of the logarithmic scale. *Minds & Machines*, 21(1):73–81, Feb. 2011. 58

1511. A. S. Posamentier and I. Lehmann. *Magnificent mistakes in mathematics*. Prometheus books, 2013. 148

1512. D. E. Post and R. P. Kendall. Software project management and quality engineering practices for complex, coupled multi-physics, massively parallel computational simulations: Lessons learned from ASCI. Report LA-UR-03-1274 Rev. 2, Los Alamos National Laboratory, Mar. 2004. 111

1513. A. Potanin, M. Damitio, and J. Noble. Are your incoming aliases really necessary? Counting the cost of object ownership. In *Proceedings of the 2013 International Conference on Software*, ICSE'13, pages 742–751, May 2013. 271

1514. E. M. Pothos and N. Chater. Rational categories. In *Proceedings of the Twentieth Annual Conference of the Cognitive Science Society*, pages 848–853, 1998. 41

1515. M. C. Potter, A. Moryadas, I. Abrams, and A. Noel. Word perception and misperception in context. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 19(1):3–22, 1993. 196

1516. J. Potts, J. Hartley, L. Montgomery, C. Neylon, and E. Rennie. A journal is a club: A new economic model for scholarly publishing. Working Paper n. 2763975, Australian universities, Apr. 2016. 11

1517. A. L. Powell. *Right on Time: Measuring, Modelling and Managing Time-Constrained Software Development*. PhD thesis, Department of Computer Science, University of York, Aug. 2001. 121, 334

1518. D. A. Powner and K. A. Rhodes. Business systems modernization: IRS needs to complete recent efforts to develop policies and procedures to guide requirements development and management. Technical Report GAO-06-310, United States Government Accountability Office, Mar. 2006. 135

1519. M. Pradel. *Program Analyses for Automatic and Precise Error Detection*. PhD thesis, ETH Zurich, 2012. 157, 158

1520. M. Pradel and T. Sen. DeepBugs: A learning approach to name-based bug detection. In *Proceedings of the ACM on Programming Languages*, OOPSLA'18, page 147, Nov. 2018. 197

1521. V. Prasad, A. Vandross, C. Toomey, M. Cheung, J. Rho, S. Quinn, S. J. Chacko, D. Borkar, V. Gall, S. Selvaraj, N. Ho, and A. Cifu. A decade of reversal: An analysis of 146 contradicted medical practices. *Mayo Clinical Proceedings*, 88(8):790–798, Aug. 2013. 4

1522. L. Prechelt. The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really? Technical Report iratr-1999-18, Universität Karlsruhe, 1999. 10, 57, 222

1523. L. Prechelt. Plat_Forms 2007: The web development platform comparison – evaluation and results. Technical Report B-07-10, Institut für Informatik, Freie Universität Berlin, June 2007. 130, 131, 135, 137, 214

1524. L. Prechelt, D. Graziotin, and D. M. Fernández. On the status and future of peer review in software engineering. In *eprint arXiv:cs.SE/1706.07196*, June 2017. 11

1525. L. Prechelt and W. F. Tichy. A controlled experiment measuring the effect of procedure argument type checking on programmer productivity. *IEEE Transactions on Software Engineering*, 24(4):302–312, Apr. 1998. 198

1526. L. Prechelt, F. Zieris, and H. Schmeisky. Difficulty factors of obtaining access for empirical studies in industry. In *Proceedings of the Third International Workshop on Conducting Empirical Studies in Industry*, CESI'15, pages 19–25, May 2015. 8

1527. L. S. Premo and S. L. Kuhn. Modeling effects of local population extinctions on cultural change and diversity in the paleolithic. *PLoS ONE*, 5(12):e15582, Dec. 2010. 76, 227

1528. R. A. Prentice and J. H. Langmore. Beware of vaporware: Product hype and the securities fraud liability of high-tech companies. *Harvard Journal of Law & Technology*, 8(1):1–74, Oct.-Dec. 1994. 77

1529. C. C. Presson and D. R. Montello. Updating after rotational and translational body movements: coordinate structure of perspective space. *Perception*, 23:1447–1455, 1994. 22

1530. T. Preston-Werner. Semantic versioning 2.0.0. organization website, July 2019. https://semver.org. 117

1531. D. Pritchard. Frequency distribution of error messages. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU'15, pages 1–8, Oct. 2015. 163

1532. V. Propp. *Morphology of the Folktale*. University of Texas Press, second edition, 1968. English translation by Laurence Scott. 185

1533. J. Prümper, D. Zapf, F. C. Brodbeck, and M. Frese. Some surprising differences between novice and expert errors in computerized office work. *Behaviour & Information Technology*, 11(6):319–328, 1992. 147

1534. Public Accounts, Committee of. HM revenue and customers: ASPIRE–re-competition of outsourced IT services. Technical Report Twenty-eighth Report of Session 2006-07, UK Parliament, June 2007. 100, 124

1535. D. Pukhkaiev. Energy-efficient benchmarking for energy-efficient software. Thesis (m.s.), Technische Universität, Dresden, Dec. 2015. 363

1536. R. Purushothaman and D. E. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31(6):511–526, June 2005. 164, 165

1537. L. H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Transactions on Software Engineering*, SE-4(4):345–361, July 1978. 128

1538. L. H. Putnam and W. Myers. *Measures for Excellence: Reliable software on time, within budget*. Prentice-Hall, Inc, 1992. 230

1539. PwC. Converging forces are building that could re-shape the entire industry. Global 100 software leaders, PwC Technology Institute, May 2013. 88

1540. PwC. The growing importance of apps and services. Global 100 software leaders, PwC Technology Institute, Mar. 2014. 88

1541. PwC. Digital intelligence conquers the world below and the cloud above. Global 100 software leaders, PwC Technology Institute, 2016. 88

1542. PwC. IPO review full-year and Q4 2015. Global technology, PwC Technology Institute, Feb. 2016. 93

1543. Z. Pylyshyn. Is vision continuous with cognition? The case for cognitive impenetrability of visual perception. *Behavioral and Brain Sciences*, 22(3):341–423, 1999. 27

1544. X. Qu. *Configuration aware prioritization for regression testing*. PhD thesis, The Graduate College at the University of Nebraska, Apr. 2010. 174

1545. S. Qualline. *C Elements of Style*. M&T Books, 1992. 184

1546. R. Queiroz, L. Passos, M. T. Valente, C. Hunsen, S. Apel, and K. Czarnecki. The shape of feature code: an analysis of twenty C-preprocessor-based systems. *Journal on Software and Systems Modeling*, 16(1):77–96, Feb. 2017. 319

1547. R Core Team. R language definition. Technical Report 3.3.1, R Foundation for Statistical Computing, June 2016. 387

1548. R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2019. ISBN 3-900051-07-0. 387, 388

1549. H. Rabinowitz and C. Schaap. *Portable C*. Prentice-Hall, Inc, 1990. 184

1550. J. J. Rachlinski, A. J. Wistrich, and C. Guthrie. Can judges make reliable numeric judgments? Distorted damages and skewed sentences. *Indiana Law Journal*, 90(2):695–739, Apr. 2015. 49

1551. J. W. Radatz. Analysis of IV & V data. Technical Report RADC-TR-81-145, Rome Air Development Center, Griffiss Air Force Base, June 1981. 162

1552. G. Radden and R. Dirven. *Cognitive English Grammar*. John Benjamins Publishing Company, 2007. 178

1553. D. Raffo, J. Settle, and W. Harrison. Investigating financial measures for planning software IV&V. Technical Report TR-99-05, Portland State University, 1999. 64

1554. C. Ragkhitwetsagul, J. Krinke, and D. Clark. A comparison of code similarity analysers. *Empirical Software Engineering*, 23(4):2464–2519, Aug. 2018. 200

1555. F. Rahman, C. Bird, and P. Devanbu. Clones: What is that smell? In *Proceedings of the 7th International Workshop on Mining Software Repositories*, MSR'10, pages 72–81, May 2010. 9

1556. M. T. Rahman, E. Shihab, and P. C. Rigby. The modular and feature toggle architectures of Google Chrome. *Empirical Software Engineering*, 24(2):826–853, July 2019. 199

1557. J. Ranade and A. Nash. *The Elements of C Programming Style*. McGraw-Hill, Inc, 1992. 184

1558. A. Rashid, H. Chivers, G. Danezis, E. Lupu, and A. Martin. CyBOK: The cyber security body of knowledge. Technical Report 1.0, The National Cyber Security Centre, UK, Oct. 2019. 153

1559. R. Ratcliff, G. McKoon, and P. Gomez. A diffusion model account of the lexical decision task. *Psychological Review*, 111(1):159–182, Jan. 2004. 22

1560. R. Ratcliff, P. L. Smith, S. D. Brown, and G. McKoon. Diffusion decision model: Current issues and history. *Trends in Cognitive Sciences*, 20(4):260–281, Apr. 2016. 22

1561. B. Ray. *Analysis of Cross-System Porting and Porting Errors in Software Projects*. PhD thesis, University of Texas at Austin, Aug. 2013. 97, 98

1562. B. Ray, M. Wilcox, and C. Voskoglou. Developer economics | State of the developer nation q3 2015. State of the Nation 9th edition, VisionMobile, July-Sept. 2015. 114

1563. K. Rayner. Eye movements and attention in reading, scene perception, and visual search. *The Quarterly Journal of Experimental Psychology*, 62(8):1457–1506, 2009. 28

1564. K. Rayner. Eye movements in reading: Models and data. *Journal of Eye Movement Research*, 2(5):1–10, Apr. 2009. 28

1565. N. M. Razali and Y. B. Wah. Power comparisons of Shapiro-Wilk, Kolmogorov-Smirnov, Lilliefors and Anderson-Darling tests. *Journal of Statistical Modeling and Analytics*, 2(1):21–33, 2011. 263

1566. J. Reason. *Human Error*. Cambridge University Press, 1990. 147, 161

1567. A. S. Reber and S. M. Kassin. On the relationship between implicit and explicit modes in the learning of a complex rule structure. *Journal of Experimental Psychology: Human Learning and Memory*, 6(5):492–502, 1980. 36

1568. J. L. Recón. Building skyscrapers, and spending on major projects. Github account, Oct. 2018. https://nintil.com/2018/10/07/building-skyscrapers-and-spending-on-major-projects. 126

1569. B. Regnell, M. Höst, J. N. och Dag, P. Beremark, and T. Hjelm. An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software. *Requirements Engineering*, 6(1):51–62, Apr. 2001. 54, 136

1570. P. Reibel, H. Yousaf, and S. Meiklejohn. Why is a Ravencoin like a TokenDesk? An exploration of code diversity in the cryptocurrency landscape. In *eprint arXiv:cs.CR/1810.08420*, Oct. 2018. 117

1571. E. D. Reichle, T. Warren, and K. McConnell. Using E-Z reader to model the effects of higher-level language processing on eye movements during reading. *Psychonomic Bulletin & Review*, 16(1):1–21, Feb. 2009. 29

1572. R. J. Reid. The Reid list of the first course language for computer science majors. http://www.csee.wvu.edu/~vanscoy/reid.htm, Aug. 2002. 115

1573. J. Reimer. Computer smartphone and tablet marketshare: 1975-2012. personal website, Dec. 2012. http://jeremyreimer.com/m-item.lsp?i=137. 5, 93

1574. G. A. Reis III. *Software Modulated Fault Tolerance*. PhD thesis, Department of Electrical Engineering, Princeton University, June 2008. 167

1575. G. Remillard. Implicit learning of second-, third-, and fourth-order adjacent and nonadjacent sequential dependencies. *The Quarterly Journal of Experimental Psychology*, 61(3):400–424, Apr. 2008. 30

1576. R. W. Remington, H. W. H. Yuen, and H. Pashler. With practice, keyboard shortcuts become faster than menu selection: A crossover interaction. *Journal of Experimental Psychology: Applied*, 22(1):95–106, 2016. 41

1577. Research Councils, UK. RCUK policy on open access and supporting guidance. Technical report, RCUK, Mar. 2013. 11

1578. A. Rice, E. Aftandilian, C. Jaspan, E. Johnston, M. Pradel, and Y. Arroyo-Paredes. Detecting argument selection defects. *Proceedings of the ACM on Programming Languages*, 1(1):104, Oct. 2017. 195

1579. G. Richards, C. Hammer, B. Burg, and J. Vitek. The eval that men do A large-scale study of the use of eval in JavaScript applications. In *Proceedings of the 25th European conference on Object-oriented programming*, ECOOP'11, pages 52–78, July 2011. 199

1580. G. Richards, S. Lebresne, B. Burg, and J. Vitek. An analysis of the dynamic behavior of JavaScript programs. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'10, pages 1–12, June 2010. 199

1581. R. Richardson. 2008 CSI computer crime & security survey. Technical report, Computer Security Institute, Aug. 2008. 154

1582. D. F. Rico. Short history of software methods. http://davidfrico.com/rico04e.pdf, July 2004. 131

1583. R. K. Ridgway. Compiling routines. In *Proceedings of the 1952 ACM national meeting (Toronto)*, ACM'52, pages 1–5, Sept. 1952. 113

1584. R. Riesen, K. Ferreira, J. Stearley, R. Oldfield, J. H. Laros III, K. Pedretti, and R. Brightwell. Redundant computing for exascale systems. Technical Report SAND2010-8709, Sandia National Laboratories, Dec. 2010. 166

1585. M. Rigger, S. Marr, B. Adams, and H. Mössenböck. Understanding GCC builtins to develop better tools. In *eprint arXiv:cs.PL/1907.00863*, July 2019. 115

1586. M. Rigger, S. Marr, S. Kell, D. Leopoldseder, and H. Mössenböck. An analysis of x86-64 inline assembly in C programs. In *Proceedings of the 14th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE'18, pages 84–99, Mar. 2018. 202

1587. M. Rinard, C. Cadar, and H. H. Nguyen. Exploring the acceptability envelope. In *Companion to the 20th annual ACM SIGPLAN conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA'05, pages 21–30, Oct. 2005. 147

1588. M. Ringelmann. Recherches sur les moteurs animés: Travail de l'homme. *Annales de l'Institut National Agronomique*, 2(XII):1–40, 1913. 80

1589. J. S. Riordon. An evolution dynamics model of software systems development. In B. Elkins and L. Hunt, editors, *Software Phenomenology Working Papers of the Software Life Cycle Management Workshop*, pages 339–360. US Army Institute for Research in Management Information and Computer SCience, Aug. 1997. 138

1590. R. Robbes, D. Róthlisberger, and É. Tanter. Object-oriented software extensions in practice. *Empirical Software Engineering*, 20(3):745–782, June 2015. 208, 209

1591. M. J. Roberts, D. J. Gilmore, and D. J. Wood. Individual differences and strategy selection in reasoning. *British Journal of Psychology*, 88:473–492, 1997. 44

1592. S. Roberts and J. Winters. Linguistic diversity and traffic accidents: Lessons from statistical studies of cultural traits. *PLoS ONE*, 8(8):e70902, Aug. 2013. 267

1593. D. E. Robinson. Fashions in shaving and trimming of the beard: The men of the Illustrated London News, 1842-1972. *American Journal of Sociology*, 81(5):1133–1141, Mar. 1976. 10

1594. G. Robles and J. M. González-Barahona. A comprehensive study of software forks: Dates, reasons and outcomes. In *The 8th International Conference on Open Source Systems*, OSS 2012, pages 1–14, Sept. 2012. 96

1595. G. Robles, I. Herraiz, D. M. Germán, and D. Izquierdo-Cortázar. Modification and developer metrics at the function level: Metrics for the study of the evolution of a software project. In *3rd International Workshop on Emerging Trends in Software Metrics*, WETSoM'12, pages 49–55, June 2012. 211, 212

1596. G. Robles, L. A. Reina, A. Serebrenik, B. Vasilescu, and J. M. González-Barahona. FLOSS 2013: A survey dataset about free software contributors: Challenges for curating, sharing, and combining. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR'14, pages 396–399, May 2014. 218, 375

1597. E. Rodrigues, Jr. and R. Terra. How do developers use dynamic features? The case of Ruby. *Computer Languages, Systems & Structures*, 53:73–89, Sept. 2018. 203

1598. W. H. Roetzheim. When the software becomes a nightmare: Dealing with failed projects. *Business Law Today*, 13(6):42–48, July-Aug. 2004. 134

1599. R. D. Rogers and S. Monsell. Costs of a predictable switch between simple cognitive tasks. *Journal of Experimental Psychology: General*, 124(2):207–231, 1995. 26

1600. M. Rönkkö, O.-P. Mutanen, N. Koivisto, J. Ylitalo, J. Peltonen, A.-M. Touru, S. Hyrynsalmi, P. Poikonen, O. Junna, J. Ali-Yrkkö, A. Valtakoski, Y. Huang, and J. Kantola. The Finnish software industry in 2007. National Software Industry Survey 2008, Software Business Lab, 2008. 62

1601. E. Rosch, C. B. Mervis, W. D. Gray, D. M. Johnson, and P. Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8(3):382–439, July 1976. 41

1602. L. Rosen. *Open Source Licensing: Software Freedom and Intellectual Property Law*. Prentice Hall PTR, July 2004. 68

1603. M. Rosenfelder. *The Language Construction Kit*. Yonagu Books, 2010. 113

1604. A. Ross. *No-Collar: The Humane Workplace and its Hidden Costs*. Temple University Press, 2003. 107

1605. B. H. Ross and G. L. Murphy. Food for thought: Cross-classification and category organization in a complex real-world domain. *Cognitive Psychology*, 38(4):495–552, June 1999. 354

1606. L. Ross, M. R. Lepper, and M. Hubbard. Perseverance in self-perception and social perception: Biased attributional processes in the debeliefing paradigm. *Journal of Personality and Social Psychologs*, 32(5):880–892, 1975. 39

1607. B. Rossi, B. Russo, and G. Succi. Path dependent stochastic models to detect planned and actual technology use: A case study of OpenOffice. *Information & Software Technology*, 53(11):1209–1226, Nov. 2011. 102

1608. J. Rost and R. L. Glass. *The Dark Side of Software Engineering: Evil on Computing Projects*. John Wiley & Sons, Inc, 2011. 122, 135

1609. V. Rothberg, N. Dintzner, A. Ziegler, and D. Lohmann. Feature models in Linux-from symbols to semantics. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS'16, pages 65–72, Jan. 2016. 97

1610. B. F. Roukema. A first-digit anomaly in the 2009 Iranian presidential election. In *eprint arXiv:stat.AP/0906.2789*, June 2013. 386

1611. G. Rousseau, R. Di Cosmo, and S. Zacchiroli. Software provenance tracking at the scale of public source code. HAL Id: hal-02543794, HAL archives-ouvertes.fr, Apr. 2020. 9

1612. E. G. Roy, D. C. Quintero, J. R. Hurley, A. Cierny, and D. Norcia. Plaintiffs, vs. SAMSUNG TELECOMMUNICATIONS AMERICA, LLC, a New York Corporation, and SAMSUNG ELECTRONICS AMERICA, INC., a New Jersey Corporation, Defendants. PLAINTIFF'S NOTICE OF UNOPPOSED MOTION AND UNOPPOSED MOTION FOR PRELIMINARY APPROVAL OF CLASS ACTION SETTLEMENT; MEMORANDUM OF POINTS AND AUTHORITIES CASE NO. 3:14-cv-582-JD, UNITED STATES DISTRICT COURT NORTHERN DISTRICT OF CALIFORNIA, Oct. 2019. 366

1613. M. M. Roy, N. J. S. Christenfeld, and C. R. M. McKenzie. Underestimating the duration of future events: Memory incorrectly used or memory bias? *Psychological Bulletin*, 131(5):738–756, 2005. 56

1614. W. W. Royce. Managing the development of large software systems. In *Technical Papers of Western Electronic Show and Convention*, WesCon, pages 1–9, Aug. 1970. 131

1615. P. Royston, D. G. Altman, and W. Sauerbrei. Dichotomizing continuous predictors in multiple regression: a bad idea. *Statistics in Medicine*, 25(1):127–141, Jan. 2006. 304

1616. D. C. Rubin, S. Hinton, and A. Wenzel. The precise time course of retention. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 25(5):1161–1176, Sept. 1999. 35, 36

1617. D. C. Rubin and A. E. Wenzel. One hundred years of forgetting: A quantitative description of retention. *Psychological Review*, 103(4):734–760, Oct. 1996. 35

1618. C. Rubio-González and B. Libit. Expect the unexpected: Error code mismatches between documentation and the real world. In *Proceedings of the 9th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, PASTE'10, pages 73–80, June 2010. 165

1619. C. Rubio-González, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough. Precimonious: Tuning assistant for floating-point precision. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC'13, Nov. 2013. 150

1620. J. Ruohonen and V. Leppänen. How PHP releases are adopted in the wild? In *eprint arXiv:cs.SE/1710.05570*, Oct. 2017. 98

1621. A. L. Russell. 'rough consensus and running code' and the internet-OSI standards war. *IEEE Annals of the History of Computing*, 28(3):48–61, July-Sept. 2006. 132

1622. R. Sabherwal, A. Jeyaraj, and C. Chowa. Information systems success: Individual and organizational determinants. *Management Science*, 52(12):1849–1864, Dec. 2006. 265

1623. R. Saborido, V. Arnaoudova, G. Beltrame, F. Khomh, and G. Antoniol. On the impact of sampling frequency on software energy measurements. *Peerj PrePrints*, 3:e1219, July 2015. 369, 370

1624. H. Sackman, W. J. Erikson, and E. E. Grant. Exploratory experimental studies comparing online and offline programming performance. *Communications of the ACM*, 11(1):3–11, Jan. 1968. 10

1625. M. Sadat, A. B. Bener, and A. V. Miranskyy. Rediscovery datasets: Connecting duplicate reports. In *eprint arXiv:cs.SE/1703.06337v1*, Mar. 2017. 151, 160, 161

1626. M. Sadinle. On the performance of dual system estimators of population size: A simulation study. Documentos de CERAC No. 13, Centro de Recursos parael Análisis de Conflictos, Bogotá, Columbia, Dec. 2008. 103

1627. D. Sahal. *Patterns of Technological Innovation*. Addison–Wesley, Dec. 1981. 5

1628. S. K. Sahoo, J. Criswell, and V. Adve. An empirical study of reported bugs in server software with implications for automated bug diagnosis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, ICSE'10, pages 485–494, May 2010. 152

1629. J. Sajaniemi and R. N. Prieto. Roles of variables in experts' programming knowledge. In *17th Workshop of the Psychology of Programming Interest Group*, PPIG'05, pages 145–159, June 2005. 207

1630. A. Salahirad, H. Almulla, and G. Gay. Choosing the fitness function for the job: Automated generation of test suites that detect real faults. *Software Testing, Verification and Reliability*, 29(4-5):e1701, June-Aug. 2019. 173

1631. P. H. Salus. *A Quarter Century of UNIX*. Addison–Wesley, 1994. 115

1632. P. H. Salus. Duelling UNIXes and the UNIX wars. *;login:*, 40(2):66–68, Apr. 2015. 115

1633. A. Sampson, W. Dietl, E. Fortuna, and D. Gnanapragasam. EnerJ: Approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, PLDI'11, pages 164–174, June 2011. 368

1634. D. M. Sanbonmatsu, S. S. Posavac, A. A. Behrends, S. M. Moore, and B. N. Uchino. Why a confirmation strategy dominates psychological science. *PLoS ONE*, page e0138197, Sept. 2015. 25

1635. D. Sarkar. *Lattice Multivariate Data Visualization with R*. Springer Science+Business Media, 2008. 225

1636. M. Savić, M. Ivanović, Z. Budimac, and M. Radovanović. Do students' programming skills depend on programming language? In *American Institute of Physics Conference Proceedings*, page 240006, June 2016. 198

1637. SC22/WG14. *Implementation of ISO/IEC 9899:1990 (E) Programming languages – C*. British Standards Institution, Dec. 1990. 162

1638. W. Scacchi. Understanding software productivity. In W. D. Hurley, editor, *Software Engineering and Knowledge Engineering: Trends for the Next Decade Vol. 4*, chapter 10, pages 273–316. World Scientific Press, June 1995. 57

1639. S. R. Schach, T. O. S. Adeshiyan, D. Balasubramanian, G. Madl, E. P. Osses, S. Singh, K. Suwanmongkol, M. Xie, and D. G. Feitelson. Common coupling and pointer variables, with application to a Linux case study. *Software Quality Journal*, 15(1):99–113, Mar. 2006. 170

1640. S. R. Schach, B. Jin, L. Yu, G. Z. Heller, and J. Offutt. Determining the distribution of maintenance categories: Survey versus measurement. *Empirical Software Engineering*, 8(4):351–363, Dec. 2003. 355, 356

1641. J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. In *Proceedings of the VLDB Endowment*, pages 460–471, Sept. 2010. 374, 375

1642. K. W. Schaie. *Developmental Influences on Adult Intelligence: The Seattle Longitudinal Study*. Oxford University Press, second edition, 2013. 59

1643. R. R. Schaller. *Technological Innovation in the Semiconductor Industry: A Case Study of the International Technology Roadmap for Semiconductors (ITRS)*. PhD thesis, George Mason University, 2004. 95

1644. M. Schief. *Business Models in the Software Industry: The Impact on Firm and M&A Performance*. Springer Gabler, Apr. 2014. 85, 89

1645. F. L. Schmidt, I.-S. Oh, and J. A. Shaffer. The validity and utility of selection methods in personnel psychology: Practical and theoretical implications of 100 years of research findings. Working paper, Tippie College of Business, University of Iowa, Oct. 2016. 21

1646. E. Schneider, M. Maruyama, S. Dehaene, and M. Sigman. Eye gaze reveals a fast, parallel extraction of the syntax of arithmetic formulas. *Cognition*, 125(3):475–490, Dec. 2012. 29

1647. S. Schneider. The dirty little secret of software pricing. website, 2012. http://www.rti.com/whitepapers/Dirty_Little_Secret.pdf. 85

1648. J. Schock, M. J. Cortese, M. M. Khanna, and S. Toppi. Age of acquisition estimates for 3,000 disyllabic words. *Behavior and Research Methods*, 44(4):971–977, Dec. 2012. 196

1649. A. Scholey and L. Owen. Effects of chocolate on cognitive function and mood: a systematic review. *Nutrition Reviews*, 71(10):665–681, Apr. 2013. 57

1650. R. Schöne, D. Hackenberg, and D. Molka. Memory performance at reduced CPU clock speeds: An analysis of current x86_64 processors. In *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*, HotPower'12, Oct. 2012. 223, 371

1651. M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical Science*, 16(1):58–74, 2001. 72

1652. L. J. Schooler and R. Hertwig. How forgetting aids heuristic inference. *Psychological Review*, 112(3):610–628, July 2005. 35

1653. E. R. Schotter, B. Angele, and K. Rayner. Parafoveal processing in reading. *Attention, Perception & Psychophysics*, 74(1):5–35, Jan. 2012. 29

1654. J.-P. Schraepler and G. G. Wagner. Identification of faked interviews in surveys by means of Benford's law?: An analysis by means of genuine fakes in the raw data of SOEP. Technical report, Technische Universiät Berlin, Aug. 2004. 386

1655. A. Schulman, M. Pietrek, and D. Maxey. *Undocumented Windows: A Programmers Guide to Reserved Microsoft Windows API Functions*. Addison–Wesley, July 1992. 117

1656. J. F. Schulz, D. Bahrami-Rad, J. P. Beauchamp, and J. Henrich. The church, intensive kinship, and global psychological variation. *Science*, 366(6466):eaau5141, Nov. 2019. 21

1657. M.-A. Schulz, B. Schmalbach, P. Brugger, and K. Witt. Analysing humanly generated random number sequences: A pattern-based approach. *PLoS ONE*, 7(7):e41531, July 2012. 386

1658. P. Schuurman, E. Berghout, and P. Powell. Benefits are from Venus, costs are from Mars. CITER WP/010/PSEBPP, University of Groningen Centre for IT Economics Research, June 2008. 119

1659. E. S. Schwartz and C. Zozaya-Gorostiza. Investment under uncertainty in information technology: Acquisition and development projects. *Management Science*, 49(1):57–70, Jan. 2003. 66

1660. C. Scott. Numbers every programmer should know. Github account, Oct. 2016. https://github.com/colin-scott/interactive_latencies. 231

1661. C. F. Scott, P. Cole, R. B. Hesse, and P. R. Malone. UNITED STATES OF AMERICA, et al., v. ORACLE CORPORATION. Plaintiff's post-trial brief CASE NO. C 04-0807 VRW, UNITED STATES DISTRICT COURT NORTHERN DISTRICT OF CALIFORNIA SAN FRANCISCO DIVISION, July 2004. 102

1662. M. D. Scott. Tort liability for vendors of insecure software: Has the time finally come? *Maryland Law Review*, 67(2):425–484, 2008. 154

1663. P. D. Scott and M. Fasli. Benford's law: An empirical investigation and a novel explanation. CSM Technical Report 349, Department of Computer Science, University of Essex, Aug. 2001. 386

1664. S. Scribner. Modes of thinking and ways of speaking: culture and logic reconsidered. In P. N. Johnson-Laird and P. C. Wason, editors, *Thinking: Readings in Cognitive Science*, chapter 29, pages 483–500. Cambridge University Press, 1977. 44

1665. R. C. Seacord. *The CERT C Secure Coding Standard*. Addison–Wesley, 2009. 184

1666. R. C. Seamans, Jr. *Aiming at Targets: The Autobiography of Robert C. Seamans, Jr.* NASA History Office, 1996. 120

1667. SEC. The world's largest hedge fund is a fraud. SEC MADOFF EXHIBITS-04451, Nov. 2005. November 7, 2005 Submission to the SEC, Madoff Investment Securities, LLC. 386

1668. P. Sehgal, V. Tarasov, and E. Zadok. Evaluating performance and energy in file system server workloads. In *Proceedings of the 8th USENIX conference on File and storage technologies*, FAST'10, Feb. 2010. 373

1669. J. Selby and K. Mayer. Startup firm acquisitions as a human resource strategy for innovation: The acqhire phenomenon. *Academy of Management Annual Meeting Proceedings*, 1:17109–17109, Nov. 2013. 109

1670. R. W. Selby, Jr., V. R. Basili, and F. T. Baker. CLEANROOM software development: An empirical evaluation. Technical Report TR-1415, Department of Computer Science, University of Maryland, Feb. 1985. 130

1671. L. L. Selwyn. *Economies of Scale in Computer Use: Initial Tests and Implications for the Computer Utility*. PhD thesis, Alfred P. Sloan School of Management, June 1969. 105

1672. J. A. Sexton. Detecting errors in software using a parameter checker: An analysis. Thesis (m.s.), Rochester Institute of Technology, Apr. 1989. 163

1673. N. Shadbolt. Shadbolt review of computer sciences degree accreditation and graduate employability. Technical Report IND/16/5, Department for Business, Innovation & Skills, UK, Apr. 2016. 72, 361

1674. T. M. Shaft and I. Vessey. The relevance of application domain knowledge: Characterizing the computer program comprehension process. *Journal of Management Information Systems*, 15(1):51–78, 1998. 187

1675. C. R. Shalizi. g, a statistical myth. blog: Three-Toed Sloth, Oct. 2007. http://bactra.org/weblog/523.html. 52

1676. J. Shallit. Randomized algorithms in "primitive" cultures or what is the oracle complexity of a dead chicken? *ACM SIGACT News*, 23(4):77–80, Sept.-Nov. 1992. 187

1677. C. Shaoul, R. H. Baayen, and C. F. Westbury. N-gram probability effects in a cloze task. *The Mental Lexicon*, 9(3):437–472, 2014. 195

1678. C. Shapiro and H. R. Varian. The art of standards wars. *California Management Review*, 41(2):8–32, Jan. 1999. 81

1679. R. Sharp, M. Paul, A. Nagesh, D. Bell, and M. Surdeanu. Grounding gradable adjectives through crowdsourcing. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation*, LREC 2018, May 2018. 153

1680. W. F. Sharpe. *The Economics of Computers*. Columbia University Press, 1969. 105

1681. O. Shatnawi. Measuring commercial software operational reliability: an interdisciplinary modelling approach. *Eksploatacja i Niezawodnosc – Maintenance and Reliability*, 16(4):585–594, 2014. 155

1682. D. E. Shaw, R. O. Dror, J. K. Salmon, J. P. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles. Millisecond-scale molecular dynamics simulations on Anton. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC'09, page 39, Nov. 2009. 112

1683. B. R. Shear and B. D. Zumbo. False positives in multiple regression: Unanticipated consequences of measurement error in the predictor variables. *Educational and Psychological Measurement*, 73(5):733–756, Oct. 2013. 288

1684. D. Shefer. Pricing for software product managers. 2005. 86

1685. B. A. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, Mar. 1981. 9

1686. S. Shekhar, M. Dietz, and D. S. Wallach. AdSplit: Separating smartphone advertising from applications. In *Proceedings of the 21st USENIX conference on Security symposium*, Security'12, page 28, Aug. 2012. 89

1687. T.-J. Shen, A. Chao, and C.-F. Lin. Predicting the number of new species in further taxonomic sampling. *Ecology*, 84(3):798–804, Mar. 2003. 104

1688. V. Y. Shen, S. D. Conte, and H. E. Dunsmore. Software science revisited: A critical analysis of the theory and its empirical support. Technical Report CSD-TR 375, Purdue University, Jan. 1981. 182

1689. A. Shenhav, S. Musslick, F. Lieder, W. Kool, T. L. Griffiths, J. D. Cohen, and M. M. Botvinick. Toward a rational and mechanistic account of mental effort. *Annual Review of Neuroscience*, 40:99–124, July 2017. 26

1690. R. N. Shepard, C. I. Hovland, and H. M. Jenkins. Learning and memorization of classifications. *Psychological Monographs: General and Applied*, 75(15):1–39, 1961. 42

1691. R. N. Shepard and J. Metzler. Mental rotation of three-dimensional objects. *Science*, 171:701–703, Feb. 1971. 22

1692. S. B. Sheppard and E. Kruesi. The effects of the symbology and spatial arrangement of software specifications in a coding task. Technical Report TR-81-388200-3, Information Systems Programs, General Electric, Feb. 1981. 162

1693. M. Shepperd, C. Mair, and M. Jørgensen. An experimental evaluation of a de-biasing intervention for professional software developers. In *eprint arXiv:cs.SE/1804.03919*, Apr. 2018. 127

1694. L. Shi, H. Zhong, T. Xie, and M. Li. An empirical study on evolution of API documentation. In *Proceedings of the 14th International Conference on Fundamental approaches to software engineering*, FASE'11/ETAPS'11, pages 416–431, Apr. 2011. 118

1695. E. Shihab, A. Ihara, Y. Kamei, W. M. Ibrahim, M. Ohira, B. Adams, A. E. Hassan, and K. ichi Matsumoto. Predicting re-opened bugs: A case study on the Eclipse project. In *17th Working Conference on Reverse Engineering*, WCRE'10, pages 249–258, Oct. 2010. 350, 351

1696. E. Shihab, Z. M. Jiang, W. M. Ibrahim, B. Adams, and A. E. Hassan. Understanding the impact of code and process metrics on post-release defects: A case study on the Eclipse project. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'10, pages 1–4, Sept. 2010. 164

1697. M. Shimasaki, S. Fukaya, K. Ikeda, and T. Kiyono. An analysis of Pascal programs in compiler writing. *Software–Practice and Experience*, 10(2):149–157, Feb. 1980. 129, 130

1698. A. L. Shimpi and B. Klug. They're (almost) all dirty: The state of cheating in Android benchmarks. Anantech news site, Oct. 2013. http://www.anandtech.com/show/7384/state-of-cheating-in-android-benchmarks. 366

1699. T. C. Shrum. Calibration and validation of the checkpoint model to the Air Force electronic systems center software database. Thesis (m.s.), Graduate School of Logistics and Acquisition Management or the Air Force Institute of Technology, USA, Sept. 1997. 8

1700. A. Shterenlikht. On quality of implementation of Fortran 2008 complex intrinsic functions on branch cuts. In *eprint arXiv:cs.MS/1712.10230*, Dec. 2017. 165

1701. O. Shy. *How to Price: A Guide to Pricing Techniques and Yield Management*. Cambridge University Press, 2008. 85

1702. R. M. Siegfried, J. P. Siegfried, and G. Alexandro. A longitudinal analysis of the Reid list of first programming languages. *Information Systems Education Journal*, 14(6):47–54, Nov. 2016. 115

1703. N. Siegmund, M. Rosenmüller, C. Kästner, P. G. Giarrusso, S. Apel, and S. S. Kolesnikov. Scalable prediction of non-functional properties in software product lines: Footprint and memory consumption. *Information and Software Technology*, 55(3):491–507, Mar. 2013. 138

1704. I. Siket, Á. Beszédes, and J. Taylor. Differences in the definition and calculation of the LOC metric in free tools. Technical Report TR2014-001, Department of Software Engineering, University of Szeged, 2014. 257

1705. R. Silberzahn, E. L. Uhlmann, D. P. Martin, P. Anselmi, F. Aust, E. Awtrey, Š. Bahník, F. Bai, C. Bannard, E. Bonnier, R. Carlsson, F. Cheung, G. Christensen, R. Clay, M. A. Craig, A. D. Rosa, L. Dam, M. H. Evans, I. F. Cervantes, N. Fong, M. Gamez-Djokic, A. Glenz, S. Gordon-McKeon, T. J. Heaton, K. Hederos, M. Heene, A. J. H. Mohr, F. Högden, K. Hui, M. Johannesson, J. Kalodimos, E. Kaszubowski, D. M. Kennedy, R. Lei, T. A. Lindsay, S. Liverani, C. R. Madan, D. Molden, E. Molleman, R. D. Morey, L. B. Mulder, B. R. Nijstad, N. G. Pope, B. Pope, J. M. Prenoveau, F. Rink, E. Robusto, H. Roderique, A. Sandberg, E. Schlüter, F. D. Schönbrodt, M. F. Sherman, S. A. Sommer, K. Sotak, S. Spain, C. Spörlein, T. Stafford, L. Stefanutti, S. Tauber, J. Ullrich, M. Vianello, E.-J. Wagenmakers, M. Witkowiak, S. Yoon, and B. A. Nosek. Many analysts, one data set: Making transparent how variations in analytic choices affect results. *Advances in Methods and Practices in Psychological Science*, 1(3):337–356, Apr. 2018. 253

1706. T. Simcoe. Standard setting committees: Consensus governance for shared technology platforms. *American Economic Review*, 102(1):305–336, Feb. 2013. 81

1707. T. Simcoe. Modularity and the evolution of the internet. In A. Goldfarb, S. M. Greenstein, and C. E. Tucker, editors, *Economic Analysis of the Digital Economy*, chapter 1, pages 21–47. University of Chicago Press, May 2015. 184, 185

1708. T. S. Simcoe and D. M. Waguespack. Status, quality and attention: What's in a (missing) name? *Management Science*, 57(2):274–290, Sept. 2011. 74, 75

1709. K. M. Simmons and D. Sutter. False alarms, tornado warnings, and tornado casualties. *Weather, Climate, and Society*, 1(1):38–53, Oct. 2009. 233

1710. H. A. Simon. *Models of Bounded Rationality: Behavioral Economics and Business Organization*. The MIT Press, 1982. 53

1711. H. A. Simon. Making management decisions: the role of intuition and emotion. *The Academy of Management Executive (1987-1989)*, 1(1):57–64, Feb. 1987. 38

1712. I. Simonson. Choice based on reasons: The case of attraction and compromise effects. *Journal of Consumer Research*, 16:158–173, Sept. 1989. 54

1713. I. C. Simpson, P. Mousikou, J. M. Montoya, and S. Defior. A letter visual-similarity matrix for Latin-based alphabets. *Behavior and Research Methods*, 45(2):431–439, June 2013. 23

1714. J. Singer, M. Luján, and I. Watson. Meaningful type names as a basis for object lifetime prediction. In *Proceedings of the 2008 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*, OOPSLA'08, page ???, Apr. 2008. 203

1715. P. V. Singh and C. Phelps. Networks, social influence, and the choice among competing innovations: Insights from open source software licenses. *Information Systems Research*, 24(3):539–560, Nov. 2013. 69

1716. D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanović, E. F. Koren, and M. Vokác. Conducting realistic experiments in software engineering. In *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, ISESE'02, pages 17–26, Oct. 2002. 360

1717. D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanović, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. Technical Report 2004-4, SIMULA Research Laboratory, 2004. 358

1718. D. Skau and R. Kosara. Arcs, angles, or areas: Individual data encodings in pie and donut charts. In *Eurographics Conference on Visualization*, EuroVis'16, pages 121–130, June 2016. 225

1719. J. Skelley. Open source tactics: Bargaining power for strategic litigation. *Chicago-Kent Journal of Intellectual Property*, 16(1), 2016. 70

1720. I. Skoulis. Analysis of schema evolution for databases in open-source software. Thesis (m.s.), University of Ioannina, Greece, Sept. 2013. 145, 146

1721. G. Slade. *Made to Break: Technology and Obsolescence in America*. Harvard University Press, 2007. 6, 167

1722. S. A. Slaughter, S. Ang, and W. F. Boh. Firm-specific human capital and compensation-organizational tenure profiles: An archival analysis of salary data for IT professionals. *Human Resource Management*, 46(3):373–394, 2007. 71

1723. S. A. Sloman. The empirical case for two systems of reasoning. *Psychological Bulletin*, 119(1):3–22, 1996. 43

1724. S. A. Sloman. Categorical inference is not a tree: The myth of inheritance hierarchies. *Cognitive Psychology*, 35(1):1–33, Feb. 1998. 41

1725. S. A. Sloman, M. C. Harrison, and B. C. Malt. Recent exposure affects artifact naming. *Memory & Cognition*, 30(5):687–695, 2002. 196

1726. S. A. Sloman and D. Lagnado. Causality in thought. *Annual Review of Psychology*, 66:223–247, 2015. 47

1727. S. A. Sloman and D. A. Lagnado. Do we "do"? *Cognitive Science*, 29(1):5–39, Jan.-Feb. 2005. 47

1728. P. Slovic. *The Perception of Risk*. Earthscan Publications Ltd, 2000. 152

1729. P. E. Smaldino and R. McElreath. The natural selection of bad science. *Royal Society Open Science*, 3:160384, Aug. 2016. 11

1730. D. Šmite, R. Britto, and R. van Solingen. Calculating the extra costs and the bottom-line hourly cost of offshoring. In *IEEE 12th International Conference on Global Software Engineering*, ICGSE'17, pages 96–105, May 2017. 127

1731. G. K. Smith, A. A. Barbour, T. L. McNaugher, M. D. Rich, and W. L. Stanley. The use of prototypes in weapon system development. Report R-2345-AF, The RAND Corporation, Mar. 1981. 135

1732. C. M. So. An analysis of mathematical expressions used in practice. Thesis (m.s.), The University of Western Ontario, 2005. 199

1733. F. Söhnchen and S. Albers. Pipeline management for the acquisition of industrial projects. *Industrial Marketing Management*, 39(8):1356–1364, Nov. 2010. 123

1734. M. Sojer, O. Alexy, S. Kleinknecht, and J. Henkel. Understanding the drivers of unethical programming behavior: The inappropriate reuse of internet-accessible code. *Journal of Management Information Systems*, 31(3):287–325, 2014. 124

1735. Solganick & Co. Software M&A update. http://www.solganickco.com/wp-content/uploads/2017/02/Solganick-Software-Q4-2016-final.pdf, Apr. 2016. 93

1736. M. B. Solomon, Jr. Economies of scale and the IBM System/360. *Communications of the ACM*, 9(6):435–440, June 1966. 94

1737. G. S. Sommer. *Astronomical Odds A Policy Framework for the Cosmic Impact Hazard*. PhD thesis, Pardee RAND Graduate School, USA, June 2004. 152

1738. J. Sonnemans. Price clustering and natural resistance points in the Dutch stock market: A natural experiment. *European Economic Review*, 50(8):1937–1950, Nov. 2006. 49

1739. C. Soto-Valero, M. Monperrus, N. Harrand, and B. Baudry. A comprehensive study of bloated dependencies in the Maven ecosystem. In *eprint arXiv:cs.SE/2001.07808*, Jan. 2020. 199

1740. R. W. Soukoreff. *Quantifying Text Entry Performance*. PhD thesis, York University, Toronto, Canada, Apr. 2010. 23

1741. SPEC. SPEC power_ssj 2008. http://spec.org/power_ssj2008, June 2016. 314

1742. SPEC. Standard performance evaluation corporation. http://spec.org, Sept. 2020. 91, 217, 219, 269, 307, 366

1743. SPECpower Committee. Power and performance benchmark methodology. V 2.2, Standard Performance Evaluation Corporation (SPEC), Dec. 2014. 369

1744. I. Spence. Visual psychophysics of simple graphical elements. *Journal of Experimental Psychology: Human Perception and Performance*, 16(4):683–692, Nov. 1990. 225

1745. I. Spence and S. Lewandowsky. Displaying proportions and percentages. *Applied Cognitive Psychology*, 5(1):61–77, Apr. 1991. 224, 225

1746. M. Spence. Job market signalling. *The Quarterly Journal of Economics*, 87(3):355–374, Aug. 1973. 72

1747. D. Sperber and D. Wilson. *Relevance: Communication and Cognition*. Blackwell Publishers, second edition, 1995. 44, 178

1748. D. Spinellis. *Code Reading: The Open Source Perspective*. Addison–Wesley, 2003. 184

1749. D. Spinellis, V. Karakoidas, and P. Louridas. Comparative language fuzz testing: Programming languages vs. fat fingers. In *Proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools*, PLATEAU'12, pages 25–34, Oct. 2012. 163

1750. D. Spinellis, Z. Kotti, K. Kravvaritis, G. Theodorou, and P. Louridas. A dataset of enterprise-driven open source software. In *eprint arXiv:cs.SE/2002.03927*, Feb. 2020. 3

1751. J. Spolsky. Fog Creek professional ladder. https://www.joelonsoftware.com/2009/02/13/fog-creek-professional-ladder, Feb. 2009. 71

1752. J. Sprouse and D. Almeida. Assessing the reliability of textbook data in syntax: Adger's core syntax. *Journal of Linguistics*, 48(3):609–652, Nov. 2012. 162

1753. D. Spuler. *C++ and C debugging, testing and reliability*. Prentice-Hall, Inc, 1994. 184

1754. L. R. Squire and A. J. O. Dede. Conscious and unconscious memory systems. *Perspectives in Biology*, 7(3):a021667, Mar. 2015. 29

1755. J. Srinivasan. *Lifetime Reliability Aware Microprocessor*. PhD thesis, University of Illinois at Urbana-Champaign, Oct. 2006. 166

1756. E. B. Staats. Millions in savings possible in converting programs from one computer to another. Technical Report FGMSD-77-34, Office of Management and Budget, Nationai Bureau of Standards, Sept. 1977. 113

1757. C. B. Stabell and Ø. D. Fjeldstad. Configuring value for competitive advantage: On chains, shops, and networks. *Strategic Management Journal*, 19(5):413–437, May 1998. 85

1758. Standish Group. The CHAOS report. Technical report, The Standish Group International, Inc, Aug. 1994. 122

1759. P. Stanley-Marbell, V. Estellers, and M. Rinard. Crayon: Saving power through shape and color approximation on next-generation displays. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, page 11, Apr. 2016. 368

1760. K. E. Stanovich. *Who Is Rational? Studies of Individual Differences in Reasoning*. Lawrence Erlbaum Associates, 1999. 43, 44

1761. K. E. Stanovich and R. F. West. Individual differences in reasoning: Implications for the rationality debate? *Behavioral and Brain Sciences*, 23(5):645–726, Oct. 2000. 43

1762. M. Staples, R. Kolanski, G. Klein, C. Lewis, J. Andronick, T. Murray, R. Jeffery, and L. Bass. Formal specifications better than function points for code sizing. In *International Conference on Software Engineering*, ICSE'13, pages 1257–1260, May 2013. 216

1763. J. Starek. A large-scale analysis of Java API usage. Thesis (m.s.), Institut für Informatik, Universität Koblenz-Landau, Mar. 2010. 302

1764. E. Starr. The use, abuse, and enforceability of non-compete and no-poach agreements: A brief review of the theory, evidence, and recent reform efforts. Issue brief, Economic Innovation Group, Washington D.C., Feb. 2019. 109

1765. T. N. Starr, L. K. Picton, and J. W. Thornton. Alternative evolutionary histories in the sequence space of an ancient protein. *Nature*, 549:409–413, Sept. 2017. 96

1766. M. Stasinopoulos, B. Rigby, V. Voudouris, G. Heller, and F. D. Bastiani. Flexible regression and smoothing: The GAMLSS packages in R. draft book, July 2015. 302

1767. G. Stasser and W. Titus. Effects of information load and percentage of shared information on the dissemination of unshared information during group discussion. *Journal of Personality and Social Psychologs*, 53(1):81–93, July 1987. 80

1768. M. Steele and J. Chaseling. Powers of discrete goodness-of-fit test statistics for a uniform null against a selection of alternative distributions. *Communications in Statistics-Simulation and Computation*, 35(4):1067–1075, Apr. 2006. 242

1769. G. L. Steele, Jr. and R. P. Gabriel. The evolution of Lisp. In *The second ACM SIGPLAN conference on History of Programming Languages*, HOPL II, pages 231–270, Apr. 1993. 113

1770. R. G. Steen, A. Casadevall, and F. C. Fang. Why has the number of scientific retractions increased? *PLoS ONE*, 8(7), Apr. 2013. 11

1771. J. Steffens. *Newgames: Strategic Competition in the PC revolution*. Pergamon Press, 1994. 94

1772. T. Stengos and E. Zacharias. Intertemporal pricing and price discrimination: A semiparametric hedonic analysis of the personal computer market. Discussion Paper 2002-11, Department of Economics, University of Cyprus, June 2002. 88

1773. K. Stenning and M. van Lambalgen. Semantics as a foundation for psychology: A case study of Wason's selection task. *Journal of Logic, Language and Information*, 10(3):273–317, June 2001. 43

1774. K. Stenning and M. van Lambalgen. A little logic goes a long way: basing experiment on semantic theory in the cognitive science of conditional reasoning. *Cognitive Science*, 28(4):481–530, July-Aug. 2004. 44

1775. K. Stenning and M. van Lambalgen. *Human Reasoning and Cognitive Science*. The MIT Press, 2008. 44

1776. M. A. Stephens. EDF statistics for goodness of fit and some comparisons. *Journal of the American Statistical Association*, 69(347):730–737, Sept. 1974. 240

1777. R. J. Sternberg and E. M. Weil. An aptitude-strategy interaction in linear syllogistic reasoning. Technical Report 15, Department of Psychology, Yale University, Apr. 1979. 44

1778. S. Sternberg. Memory-scanning: Mental processes revealed by reaction-time experiments. *American Scientist*, 57(4):421–457, 1969. 31

1779. A. Stevens and P. Coupe. Distortions in judged spatial relations. *Cognitive Psychology*, 10(4):422–437, Oct. 1978. 41

1780. N. Stewart, N. Chater, and G. D. A. Brown. Decision by sampling. *Cognitive Psychology*, 53(1):1–26, Jan. 2006. 54

1781. N. Stewart, C. Ungemach, A. J. L. Harris, D. M. Bartels, B. R. Newell, G. Paolacci, and J. Chandler. The average laboratory samples a population of 7,300 Amazon Mechanical Turk workers. *Judgment and Decision Making*, 10(5):479–491, Sept. 2015. 361

1782. G. Stikkel. Dynamic model for the system testing process. *Information and Software Technology*, 48(7):578–585, July 2006. 170, 171

1783. V. Stodden, P. Guo, and Z. Ma. Toward reproducible computational research: An empirical analysis of data and code policy adoption by journals. *PLoS ONE*, 8(6):e13636, June 2013. 11

1784. Z. Stojanova, D. Dobrilovic, and J. Stojanova. Analyzing trends for maintenance request process assessment: Empirical investigation in a very small software company. *Theory and Applications of Mathematics & Computer Science*, 3(2):59–74, Nov. 2013. 144

1785. G. P. Stone, D. B. Levin, H. Hwang, M. Kim, and C. Mckay. JANET SKOLD and DAVID DOSSANTOS, on behalf of themselves and all others similarly situated, v. INTEL CORPORATION, HEWLETT PACKARD COMPANY and DOES 1-50, case no. 1-05-CV-039231, filing #g-64475. Opinion, Superior court of the state of California for the county of Santa Clara, 2014. 366

1786. H. S. Stone. Life-cycle cost analysis of instruction-set architecture standardization for military computer systems. *Computer*, 12(4):35–47, Apr. 1979. 96

1787. J. Stone, M. Greenwald, C. Partridge, and J. Hughes. Performance of checksums and CRCs over real data. *IEEE/ACM Transactions on Networking*, 6(5):529–543, Oct. 1998. 151

1788. P. Stoneman. *Technological Diffusion and the Computer Revolution: The UK experience*. Cambridge University Press, Jan. 1976. 1, 87, 92

1789. D. Straker. *C-Style standards and guidelines*. Prentice-Hall, Inc, 1992. 184

1790. S. Strand, I. J. Deary, and P. Smith. Sex differences in cognitive abilities test scores: A UK national picture. *British Journal of Educational Psychology*, 76(3):463–480, Apr. 2006. 20, 21

1791. W. Stroebe, B. A. Nijstad, and E. F. Rietzschel. Beyond productivity loss in brainstorming groups: The evolution of a question. Working Paper No. 2014-05, Center for Research in Economics, Management and the Arts, CREMA Südstrasse 11 CH, 2014. 80

1792. R. Sudan, S. Ayers, P. Dongier, A. Muente-Kunigami, and C. Z.-W. Qiang. The global opportunity in IT-based services: Assessing and enhancing country competitiveness. Report, The World Bank, 2010. 62

1793. C. Sun, V. Le, Q. Zhang, and Z. Su. Toward understanding compiler bugs in GCC and LLVM. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*, ISSTA'16, pages 294–305, July 2016. 160

1794. L. Sun. What we are paying for: A quality adjusted price index for laptop microprocessors. Honors thesis, Wellesley College, Apr. 2014. 85, 86

1795. Sun Microsystems, Inc. Java code conventions. Technical report, Sun Microsystems, Inc, Sept. 1997. 209, 355

1796. T. Sunada, A. Monden, and K. Matsumoto. On estimating source lines of code from a binary program. In *Joint Conference of International Workshop on Software Measurement and International Conference on Software Process and Product Measurement*, IWSM/Mensura 2011, pages 3–6, Nov. 2011. 180

1797. C. R. Sunstein. *The Cost-Benefit Revolution*. The MIT Press, Aug. 2018. 149

1798. A. Suresh, B. N. Swamy, E. Rohou, and A. Seznec. Intercepting functions for memoization: A case study using transcendental functions. *Transactions on Architecture and Code Optimization*, 12(2):18, July 2015. 203

1799. P. Suthipornopas, P. Leelaprute, A. Monden, H. Uwano, Y. Kamei, N. Ubayashi, K. Araki, K. Yamada, and K. ichi Matsumoto. Industry application of software development task measurement system: TaskPit. *IEICE Transactions on Information & Systems*, E100(3):462–472, Mar. 2017. 137

1800. K. Suzuki and S. Swanson. A survey of trends in non-volatile memory technologies: 2000-2014. In *IEEE International Memory Workshop*, IMW, pages 1–4, May 2015. 370

1801. T. N. Suzuki, D. Wheatcroft, and M. Griesser. Experimental evidence for compositional syntax in bird calls. *Nature Communications*, 7(10986), Mar. 2016. 20

1802. M. Swan and B. Smith. *Learner English: A teacher's guide to interference and other problems*. Cambridge University Press, second edition, 2001. 196

1803. E. B. Swanson and C. M. Beath. *Maintaining Information Systems in Organizations*. John Wiley & Sons, Inc, 1989. 108, 142

1804. G. M. Swift and S. M. Guertin. In-flight observations of multiple-bit upset in DRAMs. *IEEE Transactions on Nuclear Science*, 47(6):2386–2391, Dec. 2000. 166

1805. R. A. Syed, B. Robinson, and L. Williams. Does hardware configuration and processor load impact software fault observability? In *Third International Conference on Software Testing, Verification and Validation*, ICST'10, pages 285–294, Apr. 2010. 258, 259

1806. N. Taerat, N. Naksinehaboon, C. Chandler, J. Elliott, C. B. Leangsuksun, G. Ostrouchov, S. L. Scott, and C. Englemann. Blue Gene/L log analysis and time to interrupt estimation. In *International Conference on Availability, Reliability and Security*, ARES'09, pages 173–180, Oct. 2009. 385

1807. L. Takeyama. The shareware industry: Some stylized facts and estimates of rates of return. *Economics of Innovation and New Technology*, 3(2):161–174, Jan. 1994. 85

1808. P. P. Tallon, R. J. Kauffman, H. C. Lucas, A. B. Whinston, and K. Zhu. Using real options analysis for evaluating uncertain investments in information technology: Insights from the ICIS 2001 debate. *Communications of the Association for Information Systems*, 9:136–167, Sept. 2002. 119

1809. K. Y. Tam. Capital budgeting in information systems development. *Information & Management*, 23(6):345–357, Dec. 1992. 62

1810. T. Tamai. Experiment on coordination within software development teams. *Information and Software Technology*, 34(7):437–442, July 1992. 137

1811. T. Tamai and Y. Torimitsu. Software lifetime and its evolution process over generations. In *Proceedings of 1992 Conference on Software Maintenance*, pages 63–69, Nov. 1992. 65, 99, 100

1812. P.-N. Tan and V. K. J. Srivastava. Selecting the right objective measure for association analysis. *Information Systems*, 29(4):293–313, June 2004. 279

1813. E. Tang, E. Barr, X. Li, and Z. Su. Perturbing numerical calculations for statistical analysis of floating-point program (in)stability. In *Proceedings of the 19th International symposium on Software testing and analysis*, ISSTA'10, pages 131–142, July 2010. 150

1814. V. Tarasov, A. Mudrankit, W. Buik, P. Shilane, G. Kuenning, and E. Zadok. Generating realistic datasets for deduplication analysis. In *Proceedings of the 2012 USENIX Annual Technical Conference*, ATC'12, June 2012. 248, 362

1815. R. C. Tausworthe. Staffing implications of software productivity models. In E. C. Posner, editor, *The Telecommunication and Data Acquisition Report 42-72*, pages 70–77. Jet Propulsion Laboratory, California Institute of Technology, Oct.-Dec. 1982. 142

1816. F. W. Taylor. *The Principles of Scientific Management*. Harper & Brothers Publishers, 1919. 73

1817. Q. C. Taylor. Analysis and characterization of author contribution patterns in open source software development. Thesis (m.s.), Brigham Young University, Apr. 2012. 187

1818. M. Tedre. Computing as a science: A survey of competing viewpoints. *Minds & Machines*, 21(3):361–387, Aug. 2011. 8

1819. J. J. Tehrani. The phylogeny of little red riding hood. *PLoS ONE*, 8(11):e78871, Nov. 2013. 185

1820. J. Teixeira, G. Robles, and J. M. González-Barahona. Lessons learned from applying social network analysis on an industrial free/libre/open source software ecosystem. *Journal of Internet Services and Applications*, 6(1):1–27, 2015. 81, 82

1821. M. Templ, B. Meindl, and A. Kowarik. Introduction to statistical disclosure control (SDC). Technical report, International Household Survey Network, Oct. 2015. 378

1822. K. Tentori, D. Osherson, L. Hasher, and C. May. Wisdom and ageing: Irrational preferences in college students but not older adults. *Cognition*, 81(3):B87–B99, 2001. 53

1823. P. E. Tetlock. Accountability: The neglected social context of judgment and choice. *Research in Organizational Behavior*, 7:297–332, 1985. 54

1824. P. E. Tetlock. An alternative metaphor in the study of judgment and choice: People as politicians. *Theory and Psychology*, 1(4):451–475, 1991. 54

1825. Tezzaron Semiconductor. Soft errors in electronic memory. Technical Report 1.1, Tezzaron Semiconductor, Naperville, IL, Jan. 2004. 166

1826. T. A. Thayer, M. Lipow, and E. C. Nelson. *Software Reliability*. North-Holland Publishing Company, 1978. 8, 151

1827. The Commission. Report of investigation pursuant to section 21(a) of the securities exchange Act of 1934: The DAO. Release No. 81207, Securities and Exchange Commission, July 2017. 148

1828. E. Thereska, B. Doebel, A. X. Zheng, and P. Nobel. Practical performance models for complex, popular applications. In *Performance Evaluation Review*, SIGMETRICS'10, pages 1–12, June 2010. 223

1829. D. R. Thomas. *Security metrics for computer systems*. PhD thesis, Cambridge Computer Laboratory, University of Cambridge, Sept. 2015. 149

1830. M. Thomas and V. Morwitz. Penny wise and pound foolish: The left-digit effect in price cognition. *Journal of Consumer Research*, 32(1):54–64, June 2005. 86

1831. M. Thomas, D. H. Simon, and V. Kadiyali. Do consumers perceive precise prices to be lower than round prices? Evidence from laboratory and market data. Research Paper Series #09-07, Johnson School, Cornell University, Sept. 2007. 86

1832. B. Thompson. The Bill Gates line. blog: Stratechery, May 2018. https://stratechery.com/2018/the-bill-gates-line. 110

1833. P. Thompson. How much did the Liberty shipbuilders forget? *Management Science*, 53(6):908–918, June 2007. 76, 77

1834. S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta. An empirical study on the maintenance of source code clones. *Empirical Software Engineering*, 15(1):1–34, Feb. 2010. 9, 358

1835. J. D. Tinder. ENTRY GRANTING REASSERTED MOTION TO DISMISS (Docket No. 34): DANIEL WALLACE, v. FREE SOFTWARE FOUNDATION, INC. Case 1:05-cv-0618-JDT-TAB, UNITED STATES DISTRICT COURT SOUTHERN DISTRICT OF INDIANA INDIANAPOLIS DIVISION, Mar. 2006. 70

1836. M. A. Tinker. The relative legibility of the letters, the digits, and of certain mathematical signs. *Journal of Generative Psychology*, 1:472–494, 1928. 196

1837. B. Tognazzini. Principles, techniques, and ethics of stage magic and their application to human interface design. In *Conference on Human Factors in Computing Systems*, INTERCHI'93, pages 355–362, May 1993. 7

1838. J. E. Tomayko. Computers in spaceflight: The NASA experience. NASA Contractor Report 182505, Wichita State University, Kansas, Mar. 1988. 114

1839. J. T. Townsend. Theoretical analysis of an alphabetic confusion matrix. *Perception & Psychophysics*, 9(1A):40–50, 1971. 23

1840. T. S. Traaen. The Brooks Act: An 8-bit act in a 64-bit world? An investigation of the Brooks Act and its implications to the Department of Defense information technology acquisition process. Executive Research Project S18, The Industrial College of the Armed Forces, National Defense University, Washington, D.C., May 1995. 105

1841. Transactions on Mathematical Software. Collected algorithms. organization website, Oct. 2020. http://www.acm.org/calgo. 177

1842. Transport, Department for. The accidents sub-objective. Transport Analysis Guidance Unit 3.4.1, Department for Transport, United Kingdom, Apr. 2011. 152

1843. L. M. Trick and Z. W. Pylyshyn. What enumeration studies can show us about spatial attention: Evidence for limited capacity preattentive processing. *Journal of Experimental Psychology: Human Perception and Performance*, 19(2):331–351, 1993. 48

1844. J. E. Triplett. Performance measures for computers. In *Deconstructing the Computer*, pages 99–139, Feb. 2003. 1

1845. D. Trippas, D. Kellen, H. Singmann, G. Pennycook, D. J. Koehler, J. A. Fugelsang, and C. Dubé. Characterizing belief bias in syllogistic reasoning: A hierarchical Bayesian meta-analysis of ROC data. *Psychonomic Bulletin and Review*, 25(2):2141–2174, Apr. 2018. 45

1846. K. S. Trivedi. *Probability & Statistics with Reliability, Queuing and Computer Science Applications*. John Wiley & Sons, Inc, second edition, 2002. 248

1847. J. S. Trueblood and J. R. Busemeyer. A quantum probability account of order effects in inference. *Cognitive Science*, 35(8):1518–1552, Nov.-Dec. 2011. 39

1848. C.-C. Tsai, B. Jain, N. A. Abdul, and D. E. Porter. A study of modern Linux API usage and compatibility: What to support when you're supporting. In *Proceedings of the Eleventh European Conference on Computer Systems*, EuroSys'16, page 16, Apr. 2016. 116

1849. N. P. Tschacher. Typosquatting in programming language package managers. Thesis (b.sc.), Department of Informatics, University of Hamburg, Mar. 2016. 165

1850. T. K. Tsingos. Enforceability of free/open source software licensing terms: A critical review of the global case - law. In *Fourth International Conference on Information Law*, ICIL 2011, May 2011. 70

1851. TSMC. TSMC historical operating data. http://www.tsmc.com/english/investorRelations/historical_information.htm, May 2017. 95

1852. M. Tufano, F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, A. De Lucia, and D. Poshyvanyk. There and back again: Can you compile that snapshot? *Journal of Software: Evolution and Process*, 29(4):e1838, Apr. 2017. 140

1853. T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. In *Proceedings of Usability Professionals Association*, pages 1–12, June 2004. 376

1854. J. Turley. Embedded processors. http://www.extremetech.com, Jan. 2002. 94, 299

1855. H. Turner and D. Firth. Bradley-Terry models in R: The `BradleyTerry2` package. *Journal of Statistical Software*, 48(9):1–21, 2012. 354

1856. H. Turner and D. Firth. *Generalized nonlinear models in R: An overview of the gnm package*. University of Warwick, UK, 1.0-8 edition, Apr. 2015. 318

1857. M. L. Turner and R. W. Engle. Is working memory capacity task dependent? *Journal of Memory and Language*, 28(2):127–154, Apr. 1989. 191

1858. R. Turner. *Weathering Heights: The Emergence of Aeronautical Meteorology as an Infrastructural Science*. PhD thesis, University of Pennsylvania, May 2010. 2, 92

1859. L. D. Tyson. *Who's Bashing Whom? Trade Conflict in High-Technology Industries*. Institute for International Economics, Nov. 1992. 61

1860. J. Tzelgov, V. Yehene, L. Kotler, and A. Alon. Automatic comparisons of artificial digits never compared: Learning linear ordering relations. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 26(1):103–120, 2000. 50

1861. UBM. Then, now: What's next? Embedded Market Study 2014, UBM Electronics, 2014. 114

1862. The ultimate Debian database. organization website, 2014. http://wiki.debian.org/UltimateDebianDatabase. 150, 221, 297

1863. G. Ülkümen, M. Thomas, and V. G. Morwitz. Will i spend more in 12 months or a year? The effect of ease of estimation and confidence on budget estimates. *Journal of Consumer Research*, 35(2):245–256, Mar. 2008. 127

1864. Unicode Consortium, The. *The Unicode Standard Version 11.0 – Core Specification*. The Unicode Consortium, June 2018. 106

1865. P. United States. Fiscal year 2018 activities. Aeronautics and space report of the president, National Aeronautics and Space Council, Sept. 2018. 2

1866. S. S. N. Upadhyay, K. J. Houghtonb, and C. M. Klin. Is "few" always less than expected?: The influence of story context on readers' interpretation of natural language quantifiers. *Discourse Processes*, 56(8):708–727, 2018. 153

1867. I. Utting, D. Bouvier, M. Caspersen, A. E. Tew, R. Frye, Y. B.-D. Kolikant, M. McCracken, J. Paterson, J. Sorva, L. Thomas, and T. Wilusz. A fresh look at novice programmers' performance and their teachers' expectations. In *Proceedings of the ITiCSE working group reports conference on Innovation and Technology in Computer Science Education-Working Group Reports*, ITiCSE-WGR'13, pages 15–32, June 2013. 361

1868. A. Vahabzadeh, A. M. Fard, and A. Mesbah. An empirical study of bugs in test code. In *International Conference on Software Maintenance and Evolution*, ICSME 2015, pages 101–110, Oct. 2015. 172

1869. M. Välimäki. Dual licensing in open source software industry. *Systemes d'Information et Management*, 8(1):63–75, 2003. 68

1870. J. van Angeren, C. Alves, and S. Jansen. Can we ask you to collaborate? Analyzing app developer relationships in commercial platform ecosystems. *Journal of Systems and Software*, 113:430–445, Mar. 2016. 91, 92

1871. O. Van den Bergh, S. Vrana, and P. Eelen. Letters from the heart: Affective categorization of letter combinations in typists and nontypists. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 16(6):1153–1161, 1990. 196

1872. K. G. van den Boogaart and R. Tolosana-Deldago. *Analyzing Compositional Data with R*. Springer, 2013. 348

1873. J.-B. van der Henst, L. Carles, and D. Sperber. Truthfulness and relevance in telling the time. *Mind & Language*, 17(5):457–466, Nov. 2002. 50

1874. E. van der Kouwe, D. Andriesse, H. Bos, C. Giuffrida, and G. Heiser. Benchmarking crimes: An emerging threat in systems security. In *eprint arXiv:cs.CR/1801.02381*, Jan. 2018. 366

1875. C. van der Merwe. An engineering approach to an integrated value proposition design framework. Thesis (m.s.), Faculty of Industrial Engineering at Stellenbosch University, Mar. 2015. 88

1876. M. J. P. van der Meulen. *The Effectiveness of Software Diversity*. PhD thesis, Centre for Software Reliability, City University, Nov. 2007. 130, 179, 197, 243

1877. M. J. P. van der Meulen, P. G. Bishop, and M. Revilla. An exploration of software faults and failure behaviour in a large population of programs. In *15th International Symposium on Software Reliability Engineering*, ISSRE'04, pages 101–120, Nov. 2004. 162

1878. M. P. van Oeffelen and P. G. Vos. A probabilistic model for the discrimination of visual number. *Perception & Psychophysics*, 32(2):163–170, 1982. 48

1879. K. E. van Oorschot, J. W. M. Bertrand, and C. G. Rutte. Field studies into the dynamics of product development tasks. *International Journal of Operations & Production Management*, 25(8):720–739, 2005. 134, 135

1880. P. Van Roy. Programming paradigms for dummies: What every programmer should know. In G. Assayag and A. Gerzso, editors, *New computational paradigms for computer music*, chapter 2. Delatour France, Jan. 2009. 197

1881. H. VanLehn. *Mind Bugs: The Origins of Procedural Misconceptions*. The MIT Press, 1990. 23, 49

1882. Y. Vardi and E. Weitz. *Misbehavior in Organizations: Theory, Research, and Management*. Lawrence Erlbaum Associates, Sept. 2004. 78

1883. R. Vasa. *Growth and Change Dynamics in Open Source Software Systems*. PhD thesis, Faculty of Information and Communication Technology, Swinburne University of Technology, Melbourne, Oct. 2010. 111, 258, 291

1884. B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens. On the variation and specialisation of workload-A case study of the Gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008, Aug. 2012. 299, 300

1885. C. Vassallo, G. Grano, F. Palomba, H. C. Gall, and A. Bacchelli. A large-scale empirical exploration on refactoring activities in open source software projects. *Science of Computer Programming*, 180:1–15, July 2019. 139

1886. P. Vassiliadis, M.-R. Kolozoff, M. Zerva, and A. V. Zarras. Schema evolution and foreign keys: a study on usage, heartbeat of change and relationship of foreign keys to table activity. *Computing*, 101(10):1431–1456, Oct. 2019. 145

1887. VCDB. VERIS community database. https://github.com/vz-risk/VCDB, Mar. 2018. 151

1888. W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer, fourth edition, 2002. 383

1889. C. Vendome, M. Linares-Vásquez, G. Bavota, M. Di Penta, D. German, and D. Poshyvanyk. License usage and changes: A large-scale study on GitHub. *Empirical Software Engineering*, 22(3):1537–1577, June 2017. 69

1890. P. Verghese and D. G. Pelli. The information capacity of visual attention. *Vision Research*, 32(5):983–995, 1992. 57

1891. C. Verhoef. Quantitative IT portfolio management. *Science of Computer Programming*, 45(1):1–96, Oct. 2002. 128

1892. C. Vesel. Language bias in accident investigation. Thesis (m.s.), Lund University, Sweden, May 2012. 161

1893. I. Vessey. Cognitive fit: A theory-based analysis of the graphs versus tables literature. *Decision Sciences*, 22(2):219–240, Mar. 1991. 224

1894. A. Vetrò, R. Dürre, M. Conoscenti, D. M. Fernández, and M. Jørgensen. Combining data analytics with team feedback to improve the estimation process in agile software development. *Foundations of Computing and Decision Sciences*, 43(4):305–334, Dec. 2018. 138

1895. B. Veytsman and L. Akhmadeeva. Towards evidence-based typography: First results. *TUGboat*, 33(2):156–156, Apr. 2012. 240, 241

1896. Vgchartz global yearly chart: 2005-2016. VGChartz news site, Feb. 2017. http://www.vgchartz.com/yearly/2016/Global. 87

1897. V. B. Viard. Information goods upgrades: Theory and evidence. *The B. E. Journal of Theoretical Economics*, 7(1):1–34, 2007. 85, 86

1898. C. Vickrey and A. Neuringer. Pigeon reaction time, Hick's law, and intelligence. *Psychonomic Bulletin & Review*, 7(2):284–291, June 2000. 58

1899. N. M. Victor and J. H. Ausubel. DRAMs as model organisms for study of technological evolution. *Technological Forecasting & Social Change*, 69(3):243–262, Apr. 2002. 92

1900. S. Vidal, A. Bergel, J. A. Díaz-Pace, and C. Marcosa. Over-exposed classes in Java: An empirical study. *Computer Languages, Systems & Structures*, 46:1–19, Nov. 2016. 208

1901. F. Viénot, H. Brettel, and J. D. Mollon. Digital video colourmaps for checking the legibility of displays by dichromats. *COLOR research and application*, 24(4):243–252, Aug. 1999. 228

1902. V. Villard. Android version distribution history. http://www.bidouille.org/misc/androidcharts, 2015. 99, 229

1903. T. H. Vines, A. Y. K. Albert, R. L. Andrew, F. Débarre, D. G. Bock, M. T. Franklin, K. J. Gilbert, J.-S. Moore, S. Renaut, and D. J. Rennison. The availability of research data declines rapidly with article age. In *eprint arXiv:abs/1312.5670*, Dec. 2013. 11

1904. W. K. Viscusi and J. E. Aldy. The value of a statistical life: A critical review of market estimates throughout the world. Working Paper No. 9487, National Bureau of Economic Research, USA, Feb. 2003. 152

1905. R. Viseur and G. Robles. First results about motivation and impact of license changes in open source projects. In *11th IFIP WG 2.13 International Conference*, OSS 2015, pages 137–145, May 2015. 69

1906. H. M. Vollmer, J. J. McAuliffe, R. I. Hirshberg, and K. D. Moll. Organizational design – An exploratory study. R&D Studies Series AFOSR-67-2450, Stanford Research Institute, Dec. 1967. 71

1907. J. Volstorf. *Against all noise: On noise-robust strategies in the emergence of cooperation*. PhD thesis, Mathematisch-Naturwissenschaftlichen Fakultät II, Humboldt-Universität, Feb. 2013. 78

1908. K. G. Volz and G. Gigerenzer. Cognitive processes in decisions under risk are not the same as in decisions under uncertainty. *frontiers in Neuroscience*, 6:105, July 2012. 53

1909. K. von Fintel and L. Matthewson. Universals in semantics. *The Linguistic Review*, 25(1-2):139–201, 2008. 43

1910. A. von Rhein, J. Liebig, A. Janker, C. Kästner, and S. Apel. Variability-aware static analysis at scale: An empirical study. *ACM Transactions on Software Engineering and Methodology*, 27(4):18, Nov. 2018. 172

1911. S. L. R. Vrhovec, T. Hovelja, D. Vavpotič, and M. Krisper. Diagnosing organizational risks in software projects: Stakeholder resistance. *International Journal of Project Management*, 33(6):1262–1273, Aug. 2015. 135

1912. M. Wachs. When planners lie with numbers. *Journal of the American Planning Association*, 55(4):476–479, Apr. 1989. 127

1913. J. Wagemans, J. H. Elder, M. Kubovy, M. A. Peterson, S. E. Palmer, M. Singh, and R. von der Heydt. A century of Gestalt psychology in visual perception: I. Perceptual grouping and figure-ground organization. *Psychological Bulletin*, 138(6):1172–1217, 2012. 27

1914. J. Wai, M. Cacchio, M. Putallaz, and M. C. Makel. Sex differences in the right tail of cognitive abilities: A 30 year examination. *Intelligence*, 38(4):412–423, July-Aug. 2010. 21

1915. J. Wainer, C. G. N. Barsottini, D. Lacerda, and L. R. M. de Marco. Empirical evaluation in computer science research published by ACM. *Information and Software Technology*, 51(6):1081–1085, June 2009. 8

1916. L. Wakeham. Government policy on the management of risk, volume I: Report. HL Paper 183-I, Select Committee on Economic Affairs, UK House of Lords, June 2006. 153

1917. S. Waligora, J. Bailey, and M. Stark. Impact of Ada and object-oriented design in the flight dynamics division at Goddard space flight center. Technical Report SEL-95-001, Goddard Space Flight Center, Mar. 1995. 198, 216

1918. D. R. Wallace and D. R. Kuhn. Failure modes in medical device software: An analysis of 15 years of recall data. *International Journal of Reliability, Quality and Safety Engineering*, 8(4):351–372, Dec. 2001. 151

1919. H. Wang, H. Li, L. Li, Y. Guo, and G. Xu. Why are Android apps removed from Google play? A large-scale empirical study. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR'18, pages 231–242, May 2018. 110

1920. P. Wang. Chasing the hottest IT: Effects of information technology fashion on organizations. *MIS Quarterly*, 34(1):63–85, Mar. 2010. 6, 10

1921. P. Wang and K. T. Stolee. How well are regular expressions tested in the wild? In *Proceedings of the 26th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'18, pages 668–678, Nov. 2018. 172

1922. W. Wang. Toward improved understanding and management of software clones. Thesis (m.s.), University of Waterloo, Ontario, Canada, May 2012. 82

1923. Y. Wang. Language matters. In *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM'15, pages 1–10, Oct. 2015. 107

1924. Y. Wang and J. Zhang. The effort distribution of software development phases. *Computer Science and Application*, 7(5):428–437, May 2017. 127, 130

1925. L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava. A case for opportunistic embedded sensing in presence of hardware power variability. In *Proceedings of the 2010 International Conference on Power aware computing and systems*, HotPower'10, pages 1–8, Oct. 2010. 369

1926. G. Ward, L. Tan, and R. Grenfell-Essam. Examining the relationship between free recall and immediate serial recall: the effects of list length and output order. *Journal of Experimental Psychology: Learning, Memory, & Cognition*, 36(5):1207–1241, Sept. 2010. 34

1927. P. J. Ward. Euclid's Elements, from Hilbert's axioms. Thesis (m.s.), The Ohio State University, 2012. 148

1928. C. Ware. *Information Visualization Perception for Design*. Morgan Kaufmann Publishers, 2000. 26

1929. W. H. Ware, S. N. Alexander, P. Armer, M. M. Astrahan, L. Bers, H. H. Goode, H. D. Huskey, and M. Rubinoff. Soviet computer technology–1959. Research Memorandum RM-2541, The RAND Corporation, Mar. 1960. 8

1930. P. C. Wason. On the failure to eliminate hypotheses in a conceptual task. *The Quarterly Journal of Experimental Psychology*, XII(3):129–140, 1960. 25

1931. P. C. Wason. Reasoning about a rule. *The Quarterly Journal of Experimental Psychology*, 20(3):273–281, 1968. 44

1932. J. Waters. Variable marginal propensities to pirate and the diffusion of computer software. MPRA Paper No. 46036, Nottingham University Business School, Apr. 2013. 88

1933. C. Watson and F. W. B. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 conference on Innovation technology in computer science education*, ITiCSE'14, pages 39–44, June 2014. 361

1934. V. M. Weaver and J. Dongarra. Can hardware performance counters produce expected, deterministic results? In *3rd Workshop on Functionality of Hardware Performance Monitoring*, pages 1–11, Dec. 2010. 370

1935. V. M. Weaver and S. A. McKee. Can hardware performance counters be trusted? In *IEEE International Symposium on Workload Characterization*, IISWC'08, pages 141–150, Sept. 2008. 370

1936. M. Webb, N. Bloom, N. Short, and J. Lerner. Some facts of high-tech patenting. Working Paper No. 18-023, Stanford Institute for Economic Policy Research, July 2018. 68

1937. E. U. Weber, A.-R. Blais, and N. E. Betz. A domain-specific risk-attitude scale: Measuring risk perceptions and risk behaviors. *Journal of Behavior and Decision Making*, 15(4):263–290, Apr. 2002. 53

1938. B. F. Webster. Patterns in IT litigation: Systems failure (1976-2000). A study, PriceaterhouseCoopers LLP, 2000. 124

1939. B. S. Weekes. Differential effects of number of letters on word and nonword naming latency. *The Quarterly Journal of Experimental Psychology*, 50A(2):439–456, 1997. 32

1940. D. M. Wegner. *The Illusion of Conscious Will*. The MIT Press, 2002. 20

1941. M. H. Weik. A survey of domestic electronic digital computing systems. Technical Report 971, Ballistic Research Laboratories, Maryland, Dec. 1955. 112, 365

1942. M. H. Weik. A third survey of domestic electronic digital computing systems. Technical Report 1115, Ballistic Research Laboratories, Maryland, Mar. 1961. 112, 365

1943. G. F. Weinwurm and H. J. Zagorski. Research into the management of computer programming: A transitional analysis of cost estimation techniques. Technical Documentary Report ESD-TR-65-575, United States Air Force, L. G. Hanscom Field, Bedford, Massachusetts, Nov. 1965. 127

1944. M. V. Welser. Opposing the monetization of Linux: McHardy v. Geniatech & addressing copyright "trolling" in Germany. *International Free and Open Source Software Law Review*, 10(1):9–20, 2018. 70

1945. J. West and J. Dedrick. Innovation and control in standards architectures: The rise and fall of Japan's PC-98. *Information Systems Research*, 11(2):197–216, June 2000. 96

1946. J. A. White. Grapher pics. `http://www.talljerome.com/mathnerd.html`, Oct. 2012. 230

1947. M. White. Scaled CMOS technology reliability users guide. JPL Publication 09-33 01/10, Jet Propulsion Laboratory, California Institute of Technology, 2010. 7, 166

1948. White House, The. Guidelines and discount rates for benefit-cost analysis of federal programs. OMB Circular A-94, U.S. Government, 1992. 64

1949. D. Whitfield. Cost overruns, delays and terminations: 105 outsourced public sector ICT projects. ESSU Research Report 3, European Services Strategy Unit, Dec. 2007. 122

1950. R. M. Whyte. Order Re Sun's Motions for Preliminary Injunction Against Microsoft. Re: Sun Microsystems v. Microsoft, Case No. 97-20884 RMW(PVT). Opinion, UNITED STATES DISTRICT COURT FOR THE NORTHERN DISTRICT OF CALIFORNIA, 1998. 110, 172

1951. W. F. Whyte. *Money and Motivation: An Analysis of Incentives in Industry*. Harper Torchbooks, Jan. 1970. 73, 74

1952. J. M. Wicherts, M. Bakker, and D. Molenaar. Willingness to share research data is related to the strength of the evidence and the quality of reporting of statistical results. *PLoS ONE*, 6(11):e26828, Nov. 2011. 11

1953. W. A. Wickelgren. Size of rehearsal group and short-term memory. *Journal of Experimental Psychology*, 68(4):413–419, 1964. 34

1954. G. Wiederhold. What is your software worth? *Communications of the ACM*, 49(9):65–75, Sept. 2006. 84

1955. A. Wierzbicka. *Semantics: Primes and Universals*. Oxford University Press, 1996. 43

1956. I. S. Wiese, J. T. da Silva, I. Steinmacher, C. Treude, and M. A. Gerosa. Who is who in the mailing list? Comparing six disambiguation heuristics to identify multiple addresses of a participant. In *International Conference on Software Maintenance and Evolution*, ICSME 2016, pages 345–355, Oct. 2016. 384

1957. Wikipedia. List of most expensive video games to develop. organization website, 2018. https://en.wikipedia.org/List_of_most_expensive_video_games_to_develop. 63

1958. R. Wilcox. *Introduction to Robust Estimation & Hypothesis Testing*. Elsevier, 3rd edition, 2012. 263

1959. J. Wiley. Expertise as mental set: The effects of domain knowledge in creative problem solving. *Memory & Cognition*, 26(4):716–730, 1998. 40

1960. M. V. Wilkes. *Memoirs of a Computer Pioneer*. The MIT Press, 1984. 147

1961. M. V. Wilkes, D. J. Wheeler, and S. Gill. *The Preparation of Programs for an Electronic Digital Computer*. Addison–Wesley, second edition, 1957. 115

1962. J. H. Wilkinson. *Rounding Errors in Algebraic Processes*. Dover Publications, 1994. 150

1963. L. Wilkinson. *The Grammar of Graphics*. Springer, second edition, 2005. 225

1964. J. C. Williams. A data-based method for assessing and reducing human error to improve operational performance. In *Fourth Conference on Human Factors and Power Plants*, pages 436–450, June 1988. 23

1965. P. Williams and B. Curtis. A matched project evaluation of modern programming practices: Scientific report on the ASTROS plan. Technical Report RADC-TR-80-6, Vol II, General Electric Company, Feb. 1980. 141

1966. R. R. Willis, R. M. Rova, M. D. Scott, M. I. Johnson, J. F. Ryskowski, J. A. Moon, K. C. Shumate, and T. O. Winfield. Hughes Aircraft's widespread deployment of a continuously improving software process. Technical Report CMU/SEI-98-TR-006, Raytheon Systems Company, May 1998. 83

1967. H. E. Willman, Jr., T. A. James, A. A. Beaureguard, and P. Hilcoff. Software systems reliability: A Raytheon project history. Final Technical Report RADC-TR-77-188, Rome Air Development Center, Griffiss Air Force Base, June 1977. 151

1968. L. M. Wills. Automated program recognition by graph parsing. A.I. Technical Report No. 1358, MIT Artificial Intelligence Laboratory, July 1992. 185

1969. M. P. Wilmot and D. S. Ones. A century of research on conscientiousness at work. *PNAS*, 116(46):23004–23010, Nov. 2019. 56

1970. R. Wiltbank and W. Boeker. Returns to angel investors in groups. Working Paper 1028592, US universities, Nov. 2007. 93

1971. K. Winter, H. Femmer, and A. Vogelsang. How do quantifiers affect the quality of requirements? In *eprint arXiv:cs.SE/2002.02672*, Feb. 2014. 162

1972. J. C. Wise, D. L. Hannaman, P. Kozumplik, E. Franke, and B. L. Leaver. Methods to improve cultural communication skills in special operations forces. ARI Contract Report 98-06, United States Army Research Institute for the Behavioral and Social Sciences, July 1998. 106

1973. K. Wnuk, J. Kabbedijk, S. Brinkkemper, B. Regnell, and D. Callele. Factors affecting decision outcome and lead-time in large-scale requirements engineering. *Journal of Software: Evolution and Process*, 27(9):647–673, Sept. 2015. 133

1974. C. Wohlin, P. Runeson, and J. Brantestam. An experimental evaluation of capture-recapture in software inspections. *Journal of Software Testing, Verification and Reliability*, 5(4):213–232, 1995. 376

1975. R. W. Wolverton. The cost of developing large-scale software. *IEEE Transactions on Computers*, c-23(6):615–636, June 1974. 131

1976. W. E. Wong, S. S. Gokhale, and J. R. Horgan. Quantifying the closeness between program components and features. *The Journal of Systems and Software*, 54(2):87–98, Oct. 2000. 185

1977. A. Wood. Software reliability growth models. Technical Report 96.1, Tandem Computer, Sept. 1996. 157, 158

1978. R. Woodfield. Undergraduate retention and attainment across the disciplines. Report, The Higher Education Academy, York, UK, Dec. 2014. 361

1979. World Semiconductor Trade Statistics. Semiconductor monthly sales volume: 1975–2016. corporate website, Mar. 2016. https://www.wsts.org. 7

1980. D. Wren. Passmark website. http://www.passmark.com, July 2014. 375, 376

1981. J. D. Wren, A. Valencia, and J. Kelso. Reviewer-coerced citation: case report, update on journal policy and suggestions for future prevention. *Bioinformatics*, 35(18):3217–3218, Sept. 2019. 11

1982. R. Wright. *The Evolution of GOD*. Little, Brown Book Group, 2009. 96

1983. A. Wrzesniewski, C. McCauley, P. Rozin, and B. Schwartz. Jobs, careers, and callings: People's relations to their work. *Journal of Research in Personality*, 31(1):21–33, Mar. 1997. 74

1984. S. D. Wu, C. Rossin, K. G. Kempf, M. O. Atan, B. Aytac, S. A. Shirodkar, and A. Mishra. Extending Bass for improved new product forecasting. Wagner Prize, Apr. 2009. 87

1985. G. Xiao, Z. Zheng, B. Jiang, and Y. Sui. An empirical study of regression bug chains in Linux. *IEEE Transactions on Reliability*, 69(2):558–570, June 2020. 151

1986. J. Yan and W. Zhang. Compiler-guided register reliability improvement against soft errors. In *Proceedings of the 5th ACM International Conference on Embedded software*, EMSOFT'05, pages 203–209, Sept. 2005. 167

1987. M. Yang, G.-R. Uh, and D. B. Whalley. Efficient and effective branch reordering using profile data. *ACM Transactions on Programming Languages and Systems*, 24(6):667–697, Nov. 2002. 203

1988. M. C. K. Yang and A. Chao. Reliability-estimation & stopping-rules for software testing, based on repeated appearances of bugs. *IEEE Transactions on Reliability*, 44(2):315–321, June 1995. 175

1989. X. Yang, Z. Wang, J. Xue, and Y. Zhou. The reliability wall for exascale supercomputing. *IEEE Transactions on Computers*, 61(6):767–779, June 2011. 167

1990. Y. Yang, Y. Zhou, H. Sun, Z. Su, Z. Zuo, L. Xu, and B. Xu. Hunting for bugs in code coverage tools via randomized differential testing. In *IEEE/ACM 41st International Conference on Software Engineering*, ICSE'19, pages 488–499, May 2019. 165, 358

1991. M. J. Yap, S. J. R. Liow, S. B. Jalil, and S. S. B. Faizal. The Malay lexicon project: A database of lexical statistics for 9,592 words. *Behavior Research Methods*, 42(4):992–1003, Nov. 2010. 196

1992. J. Yates. *Structuring the Information Age: Life Insurance and technology in the Twentieth Century*. The Johns Hopkins University Press, Nov. 2008. 105

1993. Y. C. B. Yeh. Triple-triple redundant 777 primary flight computer. In *Proceedings Aerospace Applications Conference (vol 1)*, pages 293–307, Feb. 1996. 167

1994. J. R. Yost. *Making IT Work: A History of the Computer Services Industry*. The MIT Press, 2017. 105

1995. A. G. Yu. *Managing Application Software Suppliers in Information System Development Projects*. PhD thesis, Department of Management and Organisation, University of Stirling, Nov. 2003. 131, 132

1996. L. Yu and S. Ramaswamy. A study of SourceForge users and user network. AAAI Technical Report FS-13-05, Association for the Advancement of Artificial Intelligence, Nov. 2013. 202

1997. D. Yuan, S. Park, and Y. Zhou. Characterizing logging practices in open-source software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE'12, pages 102–112, June 2012. 168

1998. T. Yuki and S. Rajopadhye. Folklore confirmed: Compiling for speed = compiling for energy. Technical Report CS13-107, Computer Science Department, Colorado State University, Aug. 2013. 368

1999. A. Zaidman, B. V. Rompaey, A. van Deursen, and S. Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. Technical Report TUD-SERG-2010-035, Software Engineering Research Group, Delft University of Technology, 2010. 172

2000. M. J. Zbaracki. The rhetoric and reality of total quality management. *Administrative Science Quarterly*, 43(3):602–636, Sept. 1998. 78

2001. S. F. Zeigler. Comparing development costs of C and Ada. Technical report, Rational Software Corporation, Mar. 1995. 58

2002. A. Zeileis, K. Hornik, and P. Murrell. Escaping RGBland: Selecting colors for statistical graphics. *Computational Statistics & Data Analysis*, 53(9):3259–3270, July 2009. 228

2003. M. V. Zelkowitz. The effectiveness of software prototyping: A case study. In *ACM Washington Chapter 26th Annual Technical Symposium*, pages 7–15, June 1987. 133

2004. A. Zeller, R. Gopinath, M. Böhme, G. Fraser, and C. Holler. *The Fuzzing Book: Tools and Techniques for Generating Software Tests*. Authors, Dec. 2019. 172

2005. A. Zeller, T. Zimmermann, and C. Bird. Failure is a four-letter word-A parody in empirical research-. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, PROMISE'11, pages 5:1–5:7, Sept. 2011. 268, 282

2006. A. Zerouali and T. Mens. Analyzing the evolution of testing library usage in open source Java projects. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering*, SANER 2017, pages 503–507, Feb. 2017. 145

2007. J. Zhang and H. Wang. The effect of external representations on numeric tasks. *The Quarterly Journal of Experimental Psychology*, 58(5):817–838, Oct. 2005. 49

2008. J. Zhang, M. Zhu, D. Hao, and L. Zhang. An empirical study on the scalability of selective mutation testing. In *Proceedings 25th International Symposium on Software Reliability Engineering*, ISSRE'14, pages 277–287, Nov. 2014. 175

2009. Q. Zhang, C. Sun, and Z. Su. Skeletal program enumeration for rigorous compiler testing. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI'17, pages 347–361, June 2017. 172

2010. T. Zhang, D. Yang, C. Lopes, and M. Kim. Analyzing and supporting adaptation of online code examples. In *eprint arXiv:cs.SE/1905.12111*, May 2019. 69

2011. X. Zhang. *An Analysis of the Effect of Environmental and Systems Complexity on Information Systems Failures*. PhD thesis, University of North Texas, Aug. 2001. 102

2012. Y. Zhang, Y. Jiang, C. Xu, X. Ma, and P. Yu. ABC: Accelerated building of C/C++ projects. In *Asia-Pacific Software Engineering Conference*, APSEC 2015, pages 182–189, Dec. 2015. 199

2013. Y. Zhang, J. W. Lee, N. P. Johnson, and D. I. August. DAFT: Decoupled acyclic fault tolerance. In *Proceedings of the 19th International Conference on Parallel architectures and compilation techniques*, PACT'10, pages 87–98, Sept. 2010. 167

2014. M. Zhao, J. Grossklags, and P. Liu. An empirical study of web vulnerability discovery ecosystems. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS'15, pages 1105–1117, Oct. 2015. 109, 151

2015. M. Zhao and P. Liu. Empirical analysis and modeling of black-box mutational fuzzing. In *International Symposium on Engineering Secure Software and Systems*, ESSoS 2016, pages 173–189, Apr. 2016. 158, 159

2016. Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: A large-scale empirical study. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ASE'17, pages 60–71, Oct.-Nov. 2017. 140

2017. J. Zheng, L. Williams, N. Nagappan, W. Snipes, J. P. Hudepohl, and M. A. Vouk. On the value of static analysis for fault detection in software. *IEEE Transactions on Software Engineering*, 32(4):240–253, Apr. 2006. 170

2018. Q. Zheng, A. Mockus, and M. Zhou. A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies. In *Proceedings of the 10th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'15, pages 637–648, Aug.-Sept. 2015. 380

2019. H. Zhong and Z. Su. An empirical study on real bug fixes. In *Proceedings of the 37th International Conference on Software Engineering*, ICSE'15, pages 913–923, May 2015. 164

2020. H. Zhou and A. Fishbach. The pitfall of experimenting on the web: How unattended selective attrition leads to surprising (yet false) research conclusions. *Journal of Personality and Social Psychologs*, 111(4):493–504, Oct. 2016. 361

2021. J. Zhou, S. Wang, C.-P. Bezemer, Y. Zou, and A. E. Hassan. Bounties in open source development on GitHub: A case study of Bountysource bounties. In *eprint arXiv:cs.SE/1904.02724*, Apr. 2019. 154, 155

2022. K. Zhou, P. Huang, C. Li, and H. Wang. An empirical study on the interplay between filesystems and SSD. In *7th International Conference on Networking, Architecture and Storage*, NAS'12, pages 124–133, June 2012. 373, 374

2023. M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the 18th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, FSE'10, pages 137–146, Nov. 2010. 58

2024. S. Zhou, B. Vasilescu, and C. Kästner. What the fork: A study of inefficient and efficient forking practices in social coding. In *Proceedings of the 27th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE'19, pages 350–361, Aug. 2019. 145

2025. Y. Zhou, L. Wu, Z. Wang, and X. Jiang. Harvesting developer credentials in Android apps. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec'15, page 23, June 2015. 206

2026. X. Zhu, E. J. Whitehead, Jr., C. Sadowski, and Q. Song. An analysis of programming language statement frequency in C, C++, and Java source code. *Software–Practice and Experience*, 15(11):1479–1495, Nov. 2015. 241, 242

2027. A. Ziegler, V. Rothberg, and D. Lohmann. Analyzing the impact of feature changes in Linux. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS'16, pages 25–32, Jan. 2016. 199

2028. T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction. In *Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT symposium on the Foundations of Software Engineering*, ESEC/FSE 2009, pages 91–100, Aug. 2009. 170

2029. T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for Eclipse. In *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE'07, May 2007. 164

2030. J. O. Zinn. The proliferation of 'at risk' in The Times: A corpus approach to historical social change, 1785-2009. *Historical Social Research*, 43(2):313–364, 2018. 152

2031. P. M. Zislis. An experiment in algorithm implementation. Technical Report CSD-TR 96, Purdue University, June 1973. 37, 38, 182, 197

2032. F. Zlotnick. *The POSIX.1 Standard: A Programmer's Guide*. The Benjamin/Cummings Publishing Company, 1991. 115

2033. W. Zou, W. Zhang, X. Xia, R. Holmes, and Z. Chen. Branch use in practice: A large-scale empirical study of 2,923 projects on GitHub. In *19th International Conference on Software Quality, Reliability and Security*, QRS 2019, pages 306–317, July 2019. 138, 139

2034. K. Zuse. Über den allgemeinen Plankalkül als mittel zur formulierung schematisch-kombinativer aufgaben. *Archiv der Mathematik*, 1:441–449, 1949. 113

2035. O. Zwikael and S. Globerson. Benchmarking of project planning and success in selected industries. *Benchmarking: An International Journal*, 13(6):688–700, 2006. 120