Hubert Baumeister
Horst Lichter
Matthias Riebisch (Eds.)

LNBIP 283

Agile Processes in Software Engineering and Extreme Programming

18th International Conference, XP 2017
Cologne, Germany, May 22–26, 2017
Proceedings

Springer Open

Lecture Notes in Business Information Processing 283

More information about this series at http://www.springer.com/series/7911

Hubert Baumeister · Horst Lichter
Matthias Riebisch (Eds.)

Agile Processes in Software Engineering and Extreme Programming

18th International Conference, XP 2017 Cologne, Germany, May 22–26, 2017 Proceedings

Springer Open

Editors
Hubert Baumeister
Technical University of Denmark
Kongens Lyngby
Denmark

Horst Lichter
RWTH Aachen University
Aachen
Germany

Matthias Riebisch
University of Hamburg
Hamburg
Germany

ISSN 1865-1348 ISSN 1865-1356 (electronic)
Lecture Notes in Business Information Processing
ISBN 978-3-319-57632-9 ISBN 978-3-319-57633-6 (eBook)
DOI 10.1007/978-3-319-57633-6

Library of Congress Control Number: 2017937714

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

The 18th XP conference was held 2017 in the wonderful city of Cologne, Germany. In the spirit of past XP conferences, XP 2017 was a place where researchers and practitioners met to exchange new ideas and present their work. These proceedings contain the full research papers, short research papers, and doctoral symposium papers presented at the conference.

In all, 46 research papers were submitted (39 full and seven short papers). All submitted papers went through a thorough review process, with each paper receiving at least three reviews. Finally, the Program Committee accepted 14 papers as full research papers (an acceptance rate of 35%). Moreover, six papers — submitted as short or full research papers — were accepted as short research papers. The selected papers cover a wide range of agile techniques and approaches. Many of them present results of empirical studies aiming to systematically evaluate successful agile practices, others are technology studies that are relevant to both researchers and practitioners.

In the tradition of former XP conferences, the XP 2017 conference program offered many different session topics. Besides the scientific program, i.e., the research track, doctoral symposium, and scientific workshops, the conference featured an industry and practice track, experience reports, and Open Space sessions. Materials from all of these sessions are available on the conference website at www.xp2017.org.

Moreover, three keynotes were given by highly renowned speakers. Andrea Goulet from Corgibytes presented a talk on "Makers and Menders: Putting the Right Developers on the Right Projects" focusing on a group of developers called "menders" – people who love taking an existing project and making it better over time. In his keynote "End-to-End Agility at GitHub" Alain Hélaïli talked about the organization and the efficient workflows at GitHub. Finally, Claes Wohlin from Blekinge Institute of Technology answered the question "Evidence-Driven Change in Software Development: Is It Feasible?"

Many people contributed to the success of the XP 2017 conference. We would like to thank everyone, especially the authors and presenters of all papers, the Program Committee members, the volunteers, and sponsors. Furthermore, we want to express our gratitude to the XP 2017 organizers; they did a great job.

March 2017

Hubert Baumeister
Horst Lichter
Matthias Riebisch

Organization

Organizing Committee

General Chair

Jutta Eckstein	IT communication, Germany

Conference Chairs

Marc Clemens	codecentric AG, Germany
Nils Wloka	codecentric AG, Germany

Academic Program Committee

Academic Program Chairs

Hubert Baumeister	Technical University of Denmark
Horst Lichter	RWTH Aachen University, Germany
Matthias Riebisch	University of Hamburg, Germany

Scientific Workshops

Roberto Tonelli	University of Cagliari, Italy

Poster Chair

Ademar Aguiar	University of Porto, Portugal

PhD Symposium Chair

Stefan Wagner	University of Stuttgart, Germany

Industrial Program Committee

Tutorials/Workshops

Nancy Van Schooenderwoert	Lean-Agile Partners, USA

Working Software

Aslak Hellesøy	Cucumber, UK

Individuals and Interaction

Diana Larsen	FutureWorks Consulting, USA

Customer Collaboration

Ken Power	Cisco Systems, Ireland

Responding to Change

Jan Coupette	codecentric AG, Germany

Experience Reports

Rebecca Wirfs-Brock	Wirfs-Brock Associates, USA
Avraham Poupko	Cisco Systems, Israel

Open Space

Alexander Kylburg	Paragraph Eins, Germany

Program Committee (Research Papers)

Ademar Aguiar	University of Porto, Portugal
Mikio Aoyama	Nanzan University, Japan
Leonor Barroca	The Open University, UK
Hubert Baumeister	Technical University of Denmark, Denmark
Jan Bosch	Chalmers University of Technology, Sweden
Steve Counsell	Brunel University, UK
Torgeir Dingsøyr	SINTEF, Norway
Christof Ebert	Vector Consulting Services, Germany
Hakan Erdogmus	Carnegie Mellon University, USA
Juan Garbajosa	Technical University of Madrid, Spain
Alfredo Goldman	University of São Paulo, Brazil
Des Greer	Queen's University Belfast, UK
Peggy Gregory	University of Central Lancashire, UK
Rashina Hoda	The University of Auckland, New Zealand
Helena Holmström Olsson	Malmö University, Sweden
Casper Lassenius	MIT, USA
Horst Lichter	RWTH Aachen University, Germany
Lech Madeyski	Wroclaw University of Science and Technology, Poland
Michele Marchesi	University of Cagliari, Italy
Sabrina Marczak	Pontifícia Universidade Católica do Rio Grande do Sul, Brazil
Tommi Mikkonen	University of Helsinki, Finland
Alok Mishra	Atilim University, Turkey
Nils Brede Moe	SINTEF, Norway
Juergen Muench	Reutlingen University and University of Helsinki, Germany/Finland
Sridhar Nerur	University of Texas at Arlington, USA
Maria Paasivaara	Helsinki University of Technology, Finland
Kai Petersen	Blekinge Institute of Technology/Ericsson AB, Sweden

Matthias Riebisch	University of Hamburg, Germany
Pilar Rodríquez	University of Oulu, Finland
Knut H. Rolland	Westerdals Oslo School of Arts, Communication and Technology, Norway
Bernhard Rumpe	RWTH Aachen University, Germany
Kurt Schneider	Leibniz Universität Hannover, Germany
Helen Sharp	The Open University, UK
Darja Smite	Blekinge Institute of Technology, Sweden
Roberto Tonelli	University of Cagliari, Italy
Rini Van Solingen	Delft University of Technology, The Netherlands
Stefan Wagner	University of Stuttgart, Germany
Xiaofeng Wang	Free University of Bozen-Bolzano, Italy
Hironori Washizaki	Waseda University, Japan
Agustin Yague	Universidad Politecnica de Madrid, Spain

Reviewers (Industry and Practice)

Giovanni Asproni	Asprotunity, UK
Emily Bache	Bache Consulting, Sweden
Filipe Correia	Uphold, Portugal
Aino Corry	Metadeveloper, Denmark
Lisa Crispin	Pivotal, USA
Jutta Eckstein	IT communication, Germany
Sallyann Freudenberg	Sallyann Freudenberg Consulting, UK
Steve Holyer	Steve Holyer and Associates, Switzerland
Lise Hvatum	Schlumberger, USA
Allan Rennebo Jepsen	Core Agile, Denmark
Jason Kerney	Hunter Industries, USA
David Kramer	Agile New England, USA
Casper Lassenius	Aalto University, Finland
Olaf Lewitz	trustartist.com, Germany
Ralph Miarka	sinnvollFÜHREN, Austria
Maria Paasivaara	Alto University, Finland
Dana Pylayeva	Hudson's Bay Company, USA
Seb Rose	Cucumber, UK
Johanna Rothman	Rothman Consulting Group, USA
Aki Salmi	Ambientia, Finland
Andreas Schliep	Das ScrumTeam, Switzerland
Irina Tsyganok	Yoox Net-A-Porter Group
Nils Wloka	codecentric AG, Germany
Joseph Yoder	The Refactory, USA
Joe Wright	Arnold Clark Automobiles, UK

Additional Reviewers

Adam, Kai
Bjørnson, Finn Olav
Britto, Ricardo
Butting, Arvid
Da Silva, Tiago Silva
Díaz, Jessica
Edison, Henry
Fernández-Sánchez, Carlos
Fögen, Konrad
Kautz, Oliver
Kusmenko, Evgeny
Raco, Deni
Santana, Célio
Santos, Viviane
Stray, Viktoria
Vestues, Kathrine
Wang, Yang

Sponsors

"Cologne Cathedral" Sponsor

REWE digital

"Albertus Magnus" Sponsor

Accenture Interactive

"River Rhine" Sponsors

DATEV
EPLAN Software & Service
OPITZ CONSULTING
XebiaLabs

"Kölsch" Sponsor

Hänneschen and Bärbelchen

Organizer

codecentric

Contents

Doctoral Symposium Papers

Improving Agile Processes

Reflection in Agile Retrospectives

Yanti Andriyani[1(✉)], Rashina Hoda[1], and Robert Amor[2]

[1] SEPTA Research, Department of Electrical and Computer Engineering, The University of Auckland, Building 903, 386 Khyber Pass, New Market, 1023 Auckland, New Zealand
yand610@aucklanduni.ac.nz, r.hoda@auckland.ac.nz
[2] Department of Computer Science, The University of Auckland, Auckland, New Zealand
trebor@cs.auckland.ac.nz

Abstract. A retrospective is a standard agile meeting practice designed for agile software teams to reflect and tune their process. Despite its integral importance, we know little about what aspects are focused upon during retrospectives and how reflection occurs in this practice. We conducted Case Study research involving data collected from interviews of sixteen software practitioners from four agile teams and observations of their retrospective meetings. We found that the important aspects focused on during the retrospective meeting include identifying and discussing obstacles, discussing feelings, analyzing previous action points, identifying background reasons, identifying future action points and generating a plan. Reflection occurs when the agile teams embody these aspects within three levels of reflection: reporting and responding, relating and reasoning, and reconstructing. Critically, we show that agile teams may not achieve all levels of reflection simply by performing retrospective meetings. One of the key contributions of our work is to present a reflection framework for agile retrospective meetings that explains and embeds three levels of reflection within the five steps of a standard agile retrospective. Agile teams can use this framework to achieve better focus and higher levels of reflection in their retrospective meetings.

Keywords: Agile retrospective meeting · Reflection · Levels of reflection · Teams · Agile software development · Reflective practice

1 Introduction

Retrospective meetings embody the 'inspect and adapt' principle of Agile Software Development (ASD) [1, 2]. They are designed to enable agile teams to frequently evaluate and find ways to adjust their process [3]. There are several purposes for retrospective meetings, such as to evaluate the previous work cycle or sprint; to determine the aspects that need to be focused on as areas of improvement; and to develop a team action plan [4]. The purpose and the techniques of the retrospective meeting have been stated and described clearly as a guide for agile teams [2, 5, 6].

Much of the existing research focuses on the techniques of performing retrospective meetings and provides lesser detail about the reflection process involved [5–9]. The Reflective Agile Learning Model (REALM) [7] classified reflection in ASD practices

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 3–19, 2017.
DOI: 10.1007/978-3-319-57633-6_1

into *reflection-in-action* or reflection that occurs during a practice, and *reflection-on-action* or reflection that occurs post a practice based on definitions of the same by Argyris and Schön [10]. A retrospective meeting was seen to embody reflection-on-action where the agile teams reflect post finishing their sprint [7]. However, what is focused on during retrospectives and how reflection occurs in this practice is not well understood.

To address this gap, we conducted Case Study research by observing four agile teams and interviewing 16 of their members guided by the following research questions:

RQ 1: What aspects are focused on during the retrospective meeting?
RQ 2: How does reflection occur in the retrospective meeting?

2 Related Work

2.1 Agile Retrospective Meeting

There is a standard format commonly used to conduct an agile retrospective meeting which involves *setting the stage, gathering data, generating insight, deciding what to do and closing the retrospective meeting* [2]. *Setting the stage* involves welcoming and explaining the aim of the retrospective meeting. *Gathering data* involves agile teams sharing their review and feedback, reporting on what happened during the previous sprint and briefly discussing with other team members. In *generating insight*, agile teams participate in a further discussion and making agreements about what issues to focus on, and then on how to solve those issues and what areas that need to improve in the *deciding what to do stage*. *Closing the retrospective* involves summarizing the retrospective meeting and appreciating all team members' efforts.

There are several recommendations for embedding reflective practice within standard agile practices as it is related to team performance improvement [7–9]. Cockburn [8] introduced a reflection workshop which involves collecting issues and generating tasks and decisions. This workshop is performed regularly during or after the post-iteration workshop. Babb et al. [7] investigated reflection in agile practices based on Argyris and Schön's [10] classification and introduced the Reflective Agile Learning Model (REALM). REALM describes how some agile practices embody reflection-in-action and reflection-on-action. Retrospective meetings were seen to embody reflection-on-action where the agile teams reflect post finishing their sprint [7].

Most of the existing research focuses on the techniques of performing a retrospective or identifying a broad classification of the type of reflection that occurs, e.g. reflection-on-action [7]. What actual topics or aspects are discussed during a retrospective and how reflection occurs, however, is not well understood. We build upon these works by investigating the retrospective meeting in depth.

2.2 Reflective Practice

Reflective practice according to Osterman and Kottkamp [11], is defined as *"a means by which practitioners can develop a greater level of self-awareness about the nature*

and impact of their performance, an awareness that creates opportunities for professional growth and development".

Bain et al. [12] classified reflection into five levels: reporting, responding, relating, reasoning and reconstructing. Level 1 and 2 are *reporting and responding* and enable learners to share brief descriptions of their experience, their feelings about events, facts or problems that they encountered. Level 3 is *relating* and involves connecting experience with personal meaning. Understanding at this level occurs when learners try to highlight good points (e.g. their ability, successful work) and negative points (e.g. mistakes, failure) to learn and identify areas of improvement. Level 4 is *reasoning* where learners explore the information shared as well as background knowledge related to the occurrences. Level 5 is *reconstructing* which signifies a high level of learning where learners generate the general framework of thinking, which is specified in a plan or action for responding to similar obstacles in the future.

Our study refers to levels of reflection proposed by Bain et al. [12] and adjusts the levels into three main levels, i.e. *reporting and responding, relating and reasoning and reconstructing*, based on our observations of the agile retrospectives in practice. *Reporting and responding* are grouped together as the first level as these levels closely related to reviews sharing and discussions at the beginning of the retrospective meeting. *Relating and responding* are grouped as the second level as agile teams participate in a further discussion after they reported and responded to the reviews. The third level, the *reconstructing* level is embodied when agile teams discuss to formulate a plan as an improvement for the next sprint.

3 Research Method

The aim of this study is to investigate how reflection occurs in retrospective meetings. Understanding this is particularly important as agile teams are reported to focus more on their technical progress and tend to pay less attention to how reflection is performed thereby compromising their potential for improvement [7, 13].

This research is conducted by implementing the Case Study research method [14]. First, existing studies related to reflection in retrospective meetings were reviewed, as summarized in Sect. 2. The research gaps identified provided guidance on formulating the interview questions. To gain rich data from interviews, we developed semi-structured questions consisting of main questions and follow-up questions. The data collection method is described in Sect. 3.1 and participant demographics summarized in Table 1. All interviews and observation data were collected by the first author in person. The raw data and emerging findings from the analysis were discussed in detail with the supervisory team (co-authors) who provided feedback and guidance.

3.1 Data Collection

Participants. We wanted to include software practitioners with a minimum of 2 years' industrial agile experience to participate in our research. During one of the Auckland Agile meetups, we received interest in participation from an agile team lead working at

Table 1. Team and team members demographics (RMD: Retrospective meeting duration in minutes; P#: Individual participant number; FAP: first agile project)

Team Name	Interviewed/total members	Agile method	RMD and the frequencies	P#	Role	Agile experience (Year)	Agile projects (Total)
Jupiter	5 out of 10	Scrum	65 min (every two weeks)	P1	UI Designer	1	6–8
				P2	Developer	0.5	1
				P3	Developer	7+	6–7
				P4	Business analyst	7+	20+
				P9	Tester	3+	10+
Saturn	6 out of 10	Scrum	55 min (every two weeks)	P4	Business analyst	7+	20+
				P5	Developer	3	10+
				P6	Designer	1 month	FP
				P7	Designer	0.5	FAP
				P8	Tester	3+	6
				P9	Tester	3+	10+
Uranus	2 out of 3	Kanban	45 min (every two weeks)	P10	Tester, Developer	1	2
				P11	Scrum Master, Business Analyst, Product Owner	6	6
Neptune	4 out of 6	Scrum	15 min (when needed)	P12	Tester	2	1
				P13	Developer	1.5	FAP
				P14	Developer	1	FAP
				P15	Tester	<1 year	FAP
Working across all four teams				P16	Test chapter lead	7	10+

the largest online auction company in New Zealand, Trade Me. Trade Me had been practicing agile software development for over three years and provided access to four teams. Its headquarters are located in Wellington and the regional offices are in Auckland and Christchurch.

For confidentiality purposes, the teams are named Jupiter, Saturn, Uranus, and Neptune. The team names and team members' details can be seen in Table 1. Each team consisted of between 3 and 10 members. All members were invited and those willing were interviewed. All teams held retrospective meetings, which lasted for between 15 and 65 min. Sixteen individual practitioners from the four teams participated in the interviews and the observations. All team members had a dedicated role in their team

and there were three participants that committed across different teams: P4 was not only fully committed as a Business Analyst in Team Jupiter but also supported Team Saturn. Similarly, P9 was a tester in Team Saturn and a half tester in Team Jupiter. P16 worked as a test lead across all four teams.

Interviews and Observations. Face-to-face individual and one group interview (of six team members) were conducted to gain comprehensive explanations, which would help derive the real concerns from both individual and team perspectives. We conducted one-on-one interviews with all participants (P1–P16), where the duration of individual interviews varied from 35 to 50 min. We asked some semi-structured questions about their experiences and perspectives related to reflection in a retrospective meeting. Some sample questions include: "*Based on the three main points discussed in a retrospective (i.e. what went well, what went wrong, what can you improve), which one(s) do you think are most helpful for your team's reflection?", "How does your team use those points to find solutions and ways to improve? Could you give some real examples?"* and *"What is the outcome of your retrospective meeting? Does your team/scrum master preserve points from the meeting?"*. A sample question of the group interview includes: *"I noticed that your team exhibited some different ways of sharing knowledge, (e.g. post-it notes, verbal communication, drawing). Did it help your team to perform reflection? How?"*

The group interview was conducted immediately after the retrospective meeting of Team Jupiter with six of its team members. Given the variable meeting times, work commitments and deadlines for different teams, it was not possible to gain further team availability for group interviews with the remaining three teams over and above the individual interviews and team observations.

Observations were conducted during the retrospective meetings of all the teams and of their general workplace. The observations aimed to capture the details of the retrospective meeting (i.e. time spent, attendees, and discussion involved) and to help validate the findings from the interviews. Photographs were taken during the observations in order to document the actual situations in the meetings and the report presented by the agile team members. Notes were taken to highlight the important aspects being shared. The information collected (e.g. photographs and notes) from the observations were used to support individual interviews by including the photographs and describing the activities in the retrospective meetings as observed first hand. The duration of each observation depended on how long the team conducted the retrospective meeting. Three out of four teams conducted the meeting for around 40–60 min each and one team, Neptune, had a shorter 15 min' retrospective. Observational data (e.g. photographs and notes) were found to support the findings from the interview data analysis thereby strengthening them.

3.2 Data Analysis

This research involved sixteen individual interviews, one group interview (of six team members), and notes taken from retrospective meeting observations which were analyzed by conducting a thematic analysis. Thematic analysis is a method that aims to

recognize, analyze, evaluate and report patterns in data [15], which enables researchers to search across a data set of interviews. Braun and Clarke [15] classify the analysis into six phases: transcribing data, generating initial codes, searching for themes, reviewing themes, defining and naming themes and making a report.

Sixteen interviews were transcribed and imported into NVivo software to facilitate coding and thematic analysis. Generating initial codes involved code identification by analyzing interesting features of a sentence, which were highlighted and added as a node in NVivo representing a new code, such as *identifying and discussing obstacles* and *discussing feelings*. Searching for themes involved comparing data with different codes to see whether they have similar meanings or aspects. Parent themes were classified based on five (grouped into three) levels of reflection, where each code was classified based on the definition of each level.

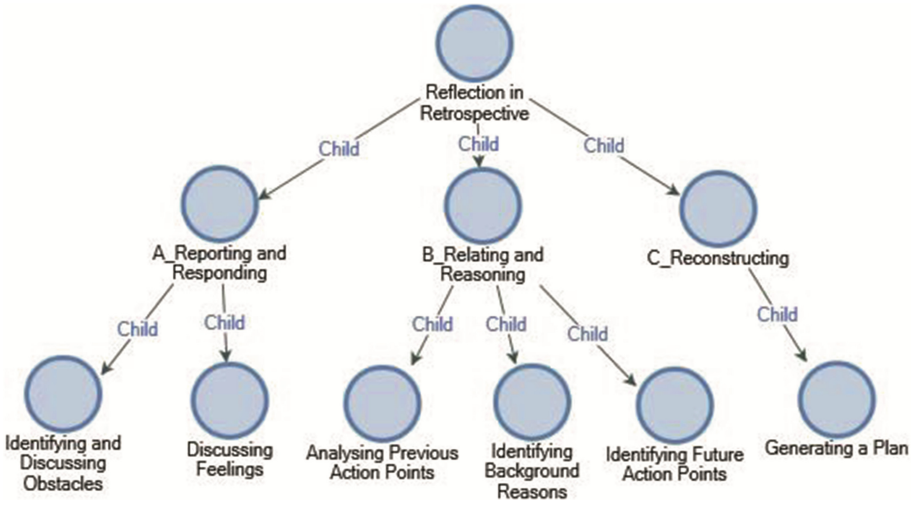

Fig. 1. Levels of reflection in retrospective meetings: a thematic map (using levels of reflection from Bain et al. [12])

Reviewing themes involved generating a thematic model to define the links and the relationships between the themes (see Fig. 1). Defining and naming themes involved the generation of several themes that emerged from the analysis, representing the aspects discussed during retrospective meetings, which was formulated and explained in this paper.

4 Findings

Following the thematic data analysis process, we identified seven themes that represent important topics or aspects discussed in the retrospective meeting, which were then mapped to the five (grouped into three) levels of reflection [12] (see Sect. 2.2).

Table 2 summarizes these themes along with their mapping to the reflection levels. These themes and levels are described below along with pertinent quotes and photographs from observations. The figures below (see Figs. 2 and 3) were captured during the observation and show a glimpse of Team Jupiter and Team Saturn's retrospective meeting.

Table 2. Themes representing topics discussed during retrospectives, their description, examples, and mapping to levels of reflection based on [12].

Levels of reflection	Themes/topics discussed	Description of themes	Examples
Reporting and Responding	Identifying and discussing obstacles	Problems, issues and concern causing blockages	Unfinished tasks and dependencies (e.g. expertise, activity, resource or entity and technical.)
	Discussing feelings	The Subjective response that reflects the situation, fact or events from the previous sprint	Negative and positive feelings
Relating and Reasoning	Analyzing previous action points	Evaluate the process improvement based on previous action points	Some improvement achieved or persisting obstacles
	Identifying background reasons	Analyzing some causes and aspects related to issues on team improvement	Testing environment issue related to external person in different location, who is difficult to contact
	Identifying future action points	Evaluating what areas need to be focused on more to be defined as future action points	Evaluate successful stories and failures
Reconstructing	Generating a plan	Define some action points for the next plan	Action points

Fig. 2. Team Jupiter's retrospective

Fig. 3. Team Saturn's retrospective

4.1 Reporting and Responding

Reporting and responding can be realized when an agile team shares some aspects (e.g. identifying and discussing obstacles and discussing feelings) while providing reviews and feedback of the previous sprint. Each team had different techniques of performing their reviews.

All teams were seen to engage in the reporting level of reflection by actively identified and discussed obstacles and feelings. Similarly, all four teams were seen to be actively involved in responding to their retrospective meeting discussions by providing brief comments on the obstacles and feelings being shared. Teams were seen to report on obstacles such as dependencies and unfinished tasks and respond with negative and positive feelings based on the previous sprint, described below.

Identifying and Discussing Obstacles. Obstacles reported in the retrospective meetings related to the aspects that hindered the team from making progress. During the retrospective meeting, agile teams gathered all the problems that occurred in the previous sprint, which would be useful for the teams to highlight areas of improvement. There were two specific obstacles reported: *dependencies and unfinished tasks*.

Dependencies. Most of the participant (11 out of 16) mentioned dependencies as the specific type of obstacle most commonly reported in the retrospective meeting.

> *"If it's delayed at the first point, if something is wrong at the first point the next person feels it. So, if one brings it up [in the retrospective] and if it's a true concern you will have support because it does affect people." P16 – Test Chapter Lead (Across All Teams)*

By sharing problems about dependencies team members became aware of the other team members' tasks and how they related to their own tasks. By being aware of this issue the team could think about ways to solve the dependency problems.

Unfinished tasks. Unfinished tasks were mentioned by three participants as an obstacle reported in retrospective meetings. An unfinished task was a problem where team members could not accomplish the tasks they had planned or considered the team to be making slow progress.

> *"We were not achieving that daily goal and it is a kind of demotivating… let's say you plan 10 stories for the sprint and you achieve just two or three. The rest we couldn't complete for whatever reason. So, we say that is one thing which didn't go well." P12 – Tester, Team Neptune*

Surfacing this obstacle was helpful for teams to understand how much more effort was required to finish the tasks, what tasks were challenging and why the tasks were difficult to finish. For example, when Team Neptune faced a problem with a requirement that delayed finishing tasks, they asked for clarifications from the product manager. It was evident that dependencies led to unfinished tasks in some cases.

Discussing Feelings. Besides obstacles, agile teams also shared their feelings which were visualized in several forms, e.g. as drawings or journey lines. The feelings shared by team members represented the sense of facts and occurrences from the previous sprint, such as when they were feeling down or happy.

There was an example of positive feelings shared, which had a positive impact on the team's productivity, where their work can be distributed well. Team Neptune recruited an additional tester after they had a problem with tester resource. They felt happy because their team was complete and balanced between developers and testers.

> *"We do put down smiley. When we got a new tester on board, a new person we had a happy smiley saying that our squad is complete." P12 – Tester, Team Neptune*

These obstacles and feelings identified and discussed during the retrospective meeting were supported by our observations of the retrospective meetings of Teams Jupiter, Saturn, and Uranus. It was observed that Team Jupiter reported their review by defining some words on sticky notes (see Fig. 4(a)).

(a) Team Jupiter (b) Team Saturn (c) Team Uranus

Fig. 4. (a) Words to describe obstacles and feelings in the Retrospective meeting; (b) and (c) Journey lines visualizing emotions during a sprint in Retrospective meetings

For example, 'muddy' was used to describe a difficult situation where team members had difficulty in understanding the detailed description of specific user stories in the project. Upon asking a team member about what was the meaning of 'muddy', a participant explained:

> *"So, I think, he and I came up with the term of 'muddy'; from observation - they were really struggling to get the right data and really had to analyse the data for this project. I observed that and for me, I would pick out a description which would explain what I've observed; as a general team.", P1 – User Interface Designer, Team Jupiter.*

4.2 Relating and Reasoning

Relating and reasoning can be seen when agile teams compile the obstacles and the feelings shared (from the previous *reporting and responding level*) and investigate the relationship between those aspects. These levels consisted of activities such *as analyzing previous action points, identifying background reasons, identifying future action points*. The explanations below present the results from the individual interviews, which supported by group interview and observations.

Analyzing Previous Action Points. An 'action point' refers to a specific item selected by the team to focus on for improvement. In analyzing previous action points, agile

teams referred to the action points agreed upon by the team in the previous retrospective and evaluated the actual effort made by the team on that specific point.

> *"..that's how you define if you made any changes, we measure yourself based on your action points and that you've actually made changes for. You could make 200 action points of your 20 weeks, but not a single one of those was followed up on, you really haven't done anything." P4 – Business Analyst, Team Jupiter and Saturn*

From the example above, it was seen that agile teams reflected on the previous action points by measuring the outcomes achieved by the teams (i.e. good or slow progress). This statement was further supported by the observations where during the retrospective meetings, agile teams shared the process improvement or the failures of the previous sprint.

Identifying Background Reasons. The background reasons of the existing issues were identified when teams were not actively progressing, they would explore the reasons why and what blockers were related to this problem. By identifying the background reasons, teams would understand what aspects needed to be improved.

This point is supported by Team Jupiter's group interview, which a team member tried to identify the reason of the major problem during the retrospective meeting.

> *"I think we addressed like the major issues are causing the squad stuck at the moment and things like test environment and [..] dealing with an external dependency like platform team in [city name]" P4 – Business Analyst, Team Jupiter and Saturn*

During the retrospective meeting observation of Team Saturn, it was seen that there was a cause analysis discussion. For example, when team members shared their sad feelings experienced during the first week sprint, team members shared the reasons, such as unclear user stories or the user story was considered as a big task. The scrum master guided the team to identify the causes by asking why they used the sad feeling notation for the first week. Several reasons were shared, such as too many tasks, the previous estimation and the actual effort were different, the unclear scope of work restricted their progress. Discussing those reasons led to the point where the team realized the main background reason was about inaccurate estimation, i.e. the team had created high achievement expectations for the big tasks without considering the actual effort required.

Identifying Future Action Points. Identifying future action points happened when the teams analyzed previous action points and identified the background issues, which followed by identifying areas of improvement and asking ideas and agreement from the teams. From the discussion, the teams gained the understanding of the existing issues which lead to the thoughts of what areas need to improve and how to improve. Identifying future action points, the teams discussed areas of improvement, which were focused on the process improvement. For example, in the retrospective meeting, most agile teams stressed testing environment issues that delayed the team progress.

> *"we list down what didn't go well or problems or whatever, we usually derive action points on those things, which is a good way to improve maybe something immediately like getting a test environment set up so we can test something….like a more immediate thing… but there are also action points that are related to the squad as well; determine a team chart or something like that." P2 – Developer, Team Jupiter*

From the example above, it was seen that by knowing the existing issues the team to will understand several areas of the process that need to be focused on. To determine future action points the teams also discussed by asking each other's opinions.

> *"when we discussed it [a plan], we asked other people what they think about it, do they agree or don't they? If everyone says they think they agree with what you are saying, then we say so what the action for that?" P12 – Tester, Team Neptune*

During the retrospective observations of Team Saturn, an example of how the team identified their future action points was noted. Team Saturn had identified that the main reason for their slow progress was inaccurate estimation. Some ideas for addressing this included elaborating the stories into small tasks, providing the clear 'definition of done' for specific tasks, and asking for clarifications from the product manager about the scope of work. The team members were asked their opinions and perspectives about these ideas. Most team members agreed on asking for clarifications from the product manager and elaborating the stories into small tasks. Consequently, the Scrum master of Team Saturn made these ideas as official action points for the next sprint.

4.3 Reconstructing

The *Reconstructing* level of reflection seems to happen when a team constructs an agreement on a specific plan based on the team members' perspectives. There were three out of the four teams (Jupiter, Saturn, Uranus) that seemed to engage in the reconstructing level as they performed further discussions and finalized by generating action points.

Generating a Plan. In reconstructing, teams generated plans decided from their discussion in the retrospective meeting. Action points are an explicit outcome of the retrospective meeting. It is useful to remind all team members about the goal for the next sprint, who will responsible, and what are the associated deadlines.

> *"So, when they go up on their board and they are doing their sprint work, they can see, "Right, let's not forget what came out of this retro" and it is getting ticked off." P11 – Scrum Master, Business Analyst and Product Owner, Team Uranus*

This point was brought up in a group interview (of Team Jupiter) where most of the team members agreed that action points were used as a reference for evaluating improvement in the next retrospective meeting.

> *"umm we pulled out action points on the board. So, over the next two weeks, we will make sure that everything talked about we follow through on." P4 – Business Analyst, Team Jupiter and Saturn*

It was observed that Team Jupiter preserved their concrete action points on their Scrum board (see Fig. 5). Another evidence from the observations was Teams Saturn and Uranus did not have action points but their Scrum master made some notes during the meetings and shared verbally the points that needed to be focused on at the end of the retrospective meeting.

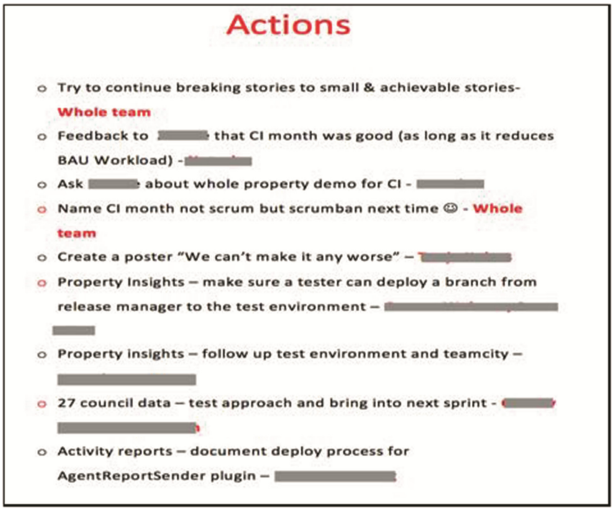

Fig. 5. Action points generated by team Jupiter posted on their Scrum Board (Photo taken during on-site observations)

5 Discussion

We now discuss the findings related to a reflection framework for agile retrospectives including the levels of reflection achieved by the teams studied, implications for practice and limitations of the study.

In response to the RQ1: *What aspects are focused on during the retrospective meeting?* We found that there are six important aspects discussed in the retrospective meetings: *identifying and discussing obstacles, discussing feelings, analyzing previous action points, identifying background reasons, identifying future action points and generating a plan*. In response to the RQ 2: *How does reflection occur in the retrospective meeting?* We found that the reflection that occurs in retrospective meetings can be classified into three levels of reflection [12], *reporting and responding, relating and reasoning, and reconstructing*.

5.1 A Framework of Reflection in Agile Retrospective Meeting

Based on these findings, we present a reflection framework for agile retrospectives (Fig. 6) that combines the five steps of the standard agile retrospective – *set the stage, gather data, generate insight and decide what to do, close the retrospectiv*e – and the levels of reflection – *reporting and responding, relating and reasoning, and reconstructing* [12] within those steps.

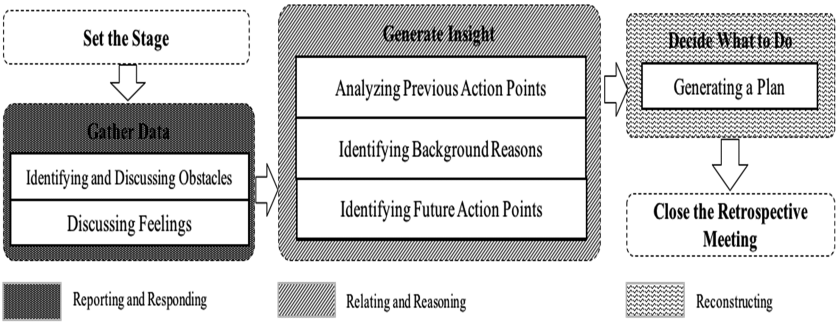

Fig. 6. Reflection in agile retrospective meeting (levels of reflection depicted in shaded areas based on [12])

Setting the stage involves welcoming and explaining the aim of the retrospective meeting. *Gathering data* step embodies the *reporting and responding* level of reflection as agile teams share their reviews (e.g. *identifying and discussing obstacles and discussing feelings*). Identifying and discussing obstacles and feelings in retrospective meetings was seen to correspond to 'descriptive reflection' [16] – a reflection which attempts to answer questions such as: *What is happening? What is this working, and for whom? For whom is it not working? How do I know? How am I feeling? What am I pleased and/or concerned about? What do I not understand?* The obstacles and feelings shared by all team members answer these questions. From the obstacles and feelings reported, the teams would be able to record and collect important points of the previous sprint. By having reviews (e.g. obstacles and feelings) of the previous sprint, team members can be prepared to deal with other similar experiences.

Generating insight step embodies the *relating and reasoning* level, where agile teams are involved in *analyzing previous action points*, *identifying* the *background reasons* behind identified issues and *identifying future action points*. Discussing these aspects was seen to be related to 'descriptive reflection', which attempts to answer questions: *does this relate to any of my stated goals and to what extent are they being met?* [16] and *why the issues happen in the previous sprint?* The answers to these questions support the reflection in the form of comparative analysis and looking back to the background issues, which help agile teams to *determine what areas needed to be focused on*. Agile teams move to deep analysis on ideas or perspective shared to identify future action points for the next sprint. It can be perceived that there is a transformation in the discussion from answering *what is happening?* in the previous sprint; to *what are the alternative views of what is happening?* and *what are the implications of the matter when viewed from these alternative perspectives?* [16]. These questions are answered when all team members provide their accounts about solutions of the obstacles or ways to improve.

In the *deciding what to do* step, agile teams have an explicit formulation which is generated in the form of action points (generating plans). The action points will be used as a reference for agile teams to act upon and improve the process. *Close the retrospective* step involves summarizing the outcomes of the retrospective meeting.

5.2 Levels of Reflection Build on Each Other

We now discuss the findings related to the levels of reflection achieved by the teams studied. A key finding of our study was that not all teams were performing on every level of reflection. So, while all teams performed retrospective meetings, not all achieved the higher levels of reflection, in particular *reconstructing*. Table 3 summarizes the levels of reflection achieved by each of the teams and the associated aspects or topics discussed in each level.

Table 3. Levels of reflection achieved by the teams (J: Jupiter; S: Saturn; U: Uranus; N: Neptune)

Levels of reflection	Aspects discussed in retrospective meeting	J	S	U	N
Reporting and responding	Identifying and discussing obstacles	✓	✓	✓	✓
	Discussing feelings	✓	✓	✓	X
Relating and reasoning	Analyzing previous action points	✓	✓	✓	✓
	Identifying background reasons	✓	✓	✓	✓
	Identifying future action points	✓	✓	✓	X
Reconstructing	Generating a plan	✓	✓	✓	X

Three teams were found to be fully engaged in all levels of reflection and one of the teams, Team Neptune, performed partially on the first two levels and did not achieve the final level of reflection, i.e. reconstructing. Based on the observation of their retrospective meeting, it was seen that they did not discuss their *feelings* explicitly and only discussed briefly the *obstacles* related to changing of task priorities needing confirmation with the product manager. They did not discuss it further as once they agreed on that obstacle then the product manager directly proceeded to the Scrum Board, discussed the issue and wrapped up the meeting. They did not record any outcomes, such as a plan or action points, from the meeting. There was little evidence of analyzing previous action points, identifying background reasons and identifying future action points. Besides, the duration of the meeting was also short, around 15 min, and they reported performing retrospective meetings only when it was necessary. Another interesting observation was that they had adapted the retrospective practice, which seemed too repetitive for them and people often seemed to have forgotten about what happened during the last two weeks' sprint. As result, they were used to placing all the individual reviews written up on sticky notes in a "retro box" – a box especially allocated to collect individual reflection. If there were no sticky notes during a two weeks' sprint, they would not perform a retrospective meeting.

The case of Team Neptune is likely related to the fact that three out of six members of Team Neptune were new to agile projects. They had in effect introduced a new reflective practice, that of using a retro box, as a way to identify the need for conducting a standard retrospective. However, a lack of reaching the reconstruction level suggests that they were not able to generate a plan for improvement as several aspects of the retrospective meeting were missing. Our findings confirm that the levels of reflection are related and build on each other [12]. Furthermore, we show that the highest level of

reflection, *reconstructing*, may not be reached at all or not reached effectively until the prior levels are accomplished effectively.

5.3 Implications for Research and Practice

For the researchers in the area of reflective practice and agile teams, our findings present a new perspective for exploring reflective practice in agile teams. Using the framework presented in the previous section, researchers can study agile teams' reflective practice in terms of levels of reflection both in retrospective meetings and other practices that involve reflection (e.g. daily standup, pair programming [7]). Future studies can explore new aspects or topics covered in each level and further explore how the levels build upon each other in different team contexts.

For agile practitioners, our findings show that not all agile teams reach all levels of reflection by simply performing retrospectives. By being aware of the different levels of reflection meant to be achieved in each retrospective step, teams can consciously strive to achieve the most out of their retrospective meetings. In particular, they can see that only *reporting and responding* and *relating and reasoning* levels are not enough rather *reconstructing* to generate action points and following up on those points in future meetings is critical to harnessing retrospective meetings to achieve continuous improvement. Thus, in order to maximize the benefits of their retrospective meetings, we recommend agile teams use our reflection framework (Fig. 6) to self-assess their level as a whole based on their personal understanding of their team context and track it in practice to achieve higher levels of reflection.

5.4 Limitations

A key limitation of this study lies in the fact that observations of a single retrospective meeting per team is not strong enough to establish and confirm a particular team's overall level of reflection. For example, it may be that in other retrospective meetings Team Neptune reached higher levels of reflection. However, the findings were arrived at by combining the data from interviews as well as the observations, which provides multiple perspectives that support the findings. Another related limitation is that the findings are limited to the contexts studied in this research, which in turn are dictated by the availability of participants. Further studies can confirm, adapt, or extend our framework to include different team contexts and reflective practices.

6 Conclusion

Previous studies have focused on specifying the techniques of conducting a retrospective meeting, with little focus on how the reflection in the retrospective meeting actually occurs. One of the key contributions of our work is to present a reflection framework for agile retrospective meetings that explains and embeds five (grouped into three) levels of reflection within the five steps of a standard agile retrospective meeting. Critically, we show that agile teams may not achieve all levels of reflection simply by performing

retrospective meetings. As the levels of reflection build upon each other, teams need to effectively identify and discuss their obstacles and feelings in the reporting and responding level, followed by analyzing previous action points, identifying background reasons, and identifying future action points in the relating and reasoning level and generating a plan in the reconstructing level. Embedding these levels of reflection into the retrospective meeting will help agile teams achieve better focus and higher levels of reflection from performing retrospective meetings. Another implication is an increase in their awareness of the main aspects that need to be discussed in the retrospective meeting and how to formulate these aspects to generate a plan for improvement.

Acknowledgement. This research is supported by The University of Auckland and the Indonesia Endowment Fund for Education (LPDP) S-669/LPDP/2013 as scholarship provider from the Ministry of Finance, Indonesia.

References

1. Deemer, P., Benefield, G., Larman, C., Vodde, B.: A Lightweight Guide to the Theory and Practice of Scrum Version 2.0, vol. 2015 (2012)
2. Derby, E., Larsen, D., Schwaber, K.: Agile Retrospectives: Making Good Teams Great. Pragmatic Bookshelf, Raleigh (2006). 0977616649
3. Fowler, M., Highsmith, J.: The agile manifesto. Softw. Dev. **9**, 29 (2001)
4. Sutherland, J., Schwaber, K.: The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game (2011)
5. Salo, O.: Systematical validation of learning in agile software development environment. In: Althoff, K.-D., Dengel, A., Bergmann, R., Nick, M., Roth-Berghofer, T. (eds.) WM 2005. LNCS (LNAI), vol. 3782, pp. 106–110. Springer, Heidelberg (2005). doi: 10.1007/11590019_13
6. Salo, O., Kolehmainen, K., Kyllönen, P., Löthman, J., Salmijärvi, S., Abrahamsson, P.: Self-adaptability of agile software processes: a case study on post-iteration workshops. In: Eckstein, J., Baumeister, H. (eds.) XP 2004. LNCS, vol. 3092, pp. 184–193. Springer, Heidelberg (2004). doi:10.1007/978-3-540-24853-8_21
7. Babb, J., Hoda, R., Nørbjerg, J.: Embedding reflection and learning into agile software development. IEEE Softw. **31**, 51–57 (2014). doi:10.1109/MS.2014.54
8. Cockburn, A., Highsmith, J.: Agile software development: the people factor. Computer **34**, 131–133 (2001)
9. Dingsøyr, T., Hanssen, G.K.: Extending agile methods: postmortem reviews as extended feedback. In: Henninger, S., Maurer, F. (eds.) LSO 2002. LNCS, vol. 2640, pp. 4–12. Springer, Heidelberg (2003). doi:10.1007/978-3-540-40052-3_2
10. Argyris, C., Schon, D.A.: Organisational Learning II: Theory, Method and Practice. Organisation Development Series. Adisson Wesley, Reading (1996)
11. Osterman, K., Kottkamp, R.: ReflectivePractice for Educators: Improving Schooling through Professional Development. Corwin Press, Newbury Park (1993)
12. Bain, J.D., Ballantyne, R., Packer, J., Mills, C.: Using journal writing to enhance student teachers' reflectivity during field experience placements. Teachers Teach. Theor. Pract. **5**, 51–73 (1999). doi:10.1080/1354060990050104
13. Hoda, R., Babb, J., Nørbjerg, J.: Toward learning teams. IEEE Softw. **30**, 95–98 (2013). doi: 10.1109/MS.2013.90

14. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Inc. (2003)
15. Braun, V., Clarke, V.: Using thematic analysis in psychology. Qual. Res. Psychol. **3**, 77–101 (2006)
16. Jay, J.K., Johnson, K.L.: Capturing complexity: a typology of reflective practice for teacher education. Teach. Teacher Educ. **18**, 73–85 (2002)

What Influences the Speed of Prototyping? An Empirical Investigation of Twenty Software Startups

Anh Nguyen-Duc[1(✉)], Xiaofeng Wang[2], and Pekka Abrahamsson[1]

[1] Department of Computer and Information Science (IDI), NTNU, 7491 Trondheim, Norway
{anhn,pekkaa}@ntnu.no
[2] Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
xiaofeng.wang@unibz.it

Abstract. It is essential for startups to quickly experiment business ideas by building tangible prototypes and collecting user feedback on them. As prototyping is an inevitable part of learning for early stage software startups, how fast startups can learn depends on how fast they can prototype. Despite of the importance, there is a lack of research about prototyping in software startups. In this study, we aimed at understanding what are factors influencing different types of prototyping activities. We conducted a multiple case study on twenty European software startups. The results are two folds; firstly we propose a prototype-centric learning model in early stage software startups. Secondly, we identify factors occur as barriers but also facilitators for prototyping in early stage software startups. The factors are grouped into (1) artifacts, (2) team competence, (3) collaboration, (4) customer and (5) process dimensions. To speed up a startup's progress at the early stage, it is important to incorporate the learning objective into a well-defined collaborative approach of prototyping.

Keywords: Prototype · MVP · Prototyping-learning loop · Validated learning · Speed · Software startups

1 Introduction

With the startup movement, software industry is witnessing a paradigm shift from serving customer requirements to creating customer value. The challenge for software companies is no longer primarily on implementing customer requirements, but rather on finding customer demands and providing a solution that delivers customer value [2]. Addressing uncertainty in both solution and problem domains has often been ad-hoc and based on guesswork, which becomes one of the main reasons for failing startup companies [3]. A demand on systematic approaches to manage the uncertainty has led to an increased research interest on Lean Startup [4], New Product Development (NPD) [5], software startups [6] and continuous experimentation [7].

In a competitive environment such as software industry, time-to-market is becoming more and more critical as a success factor for startup companies. Business ideas under development once revealed can be easily threatened by high speed copycats [9]. Moreover, competitors can also follow an on-going journey of validating product-market fit

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 20–36, 2017.
DOI: 10.1007/978-3-319-57633-6_2

and arrive faster in the destination. Regardless of company sizes and application domains, the knowledge of influencing factors for a quick learning loop is important for software startups to form best-fit strategy in developing business experimentation [10].

A 'Build-Measure-Learn' loop, as a central concept of the Lean Startup methodology, aims at speeding up the new product development cycle [4]. The central part of the loop is to build a representation of the business, a so-called Minimum viable product (MVP), to collect feedback from customers and to learn from that. Steve Blank emphasizes the goal of MVPs is "*to maximize learning through incremental and iterative engineering*" [2]. In the startup context, developers quickly and iteratively develop a software application to validate business ideas [12]. As such, the study of validated learning can be beneficial from Software Engineering (SE) concepts and practices, such as rapid prototypes and evolutionary prototypes [13–15]. Consequently, the time-to-release of prototypes is essential to determine the total time in the validated learning loop.

Software startup research is increasingly recognized by researcher's community, with many practical aspects, such as User Experience, Software practices, competences and startup ecosystem [6]. Despite of the importance, there is a lack of research about prototyping in software startups. In a multi-influenced context with funding, human resource and market concerns, it is crucial to understand how the speed of learning can be supported by prototyping activities and what are the influencing factors. In a previous study, we investigated how a prototype is built in software startups [12]. We found that prototyping activities as a core value of startup experimentation needed to be seen as a multifaceted phenomenon [12]. In this work, we are particularly interested in the factors that slow down the learning process and those that speed it up. The research question (RQ) is:

What factors influence the speed of prototyping in software startups?

The paper is organized as follows. Firstly, we present the background about business-driven experimentation in software projects, software prototype and a proposal of an analytical model of startup prototyping (Sect. 2). Then, we describe our research approach and the cases studied (Sect. 3). After that, the qualitative findings are presented (Sect. 4). Finally, we reflect on the findings, the threats to validity (Sect. 5), and draw the conclusion and future work (Sect. 6).

2 Background

2.1 Business Driven Experimentation

From SE perspective, validated learning means the focus on integrating business value in defining software development processes and practices. Even though experiment systems are recognized as beneficial to software projects, there are barriers in adopting them, such as integration of customer feedback, synchronizing vendors in short cycles and lack of reasoning about customer requirements [16, 17]. Bosch et al. [18] advocate for adjusting the Lean startup methodology to accommodate the development of

multiple ideas and to integrate them when time for their testing and validation is too long. Bosch suggested using 2-to-4-week experimentation iterations followed by exposing the product to customers in order to collect feedbacks. Fagerholm et al. present a model for continuous experimentation for start up companies [7], in which a key element is the ability to release a prototype with suitable instrumentation, to manage experiment plans, link experiment results with a product roadmap, and to manage a flexible business strategy. Olsson et al. present a Hypothesis Experiment Data-Driven Development model that integrates feature experiments with customer feedback in Agile projects [19]. While these work characterize a process-like approach in developing startups' software products, Paternoster et al. grounded a model from 13 software startups which describes a pattern that software startups often build evolutionary prototypes [20]. This study focuses on how startups are prototyping in reality and the influencing factors of the speed of learning by prototyping.

2.2 Prototype and Prototyping Activities

Brook mentioned "*In software engineering, at least, the concept of rapid prototyping has a name and a recognized value, whereas it does not always have the same status in computer design and in building architecture*" [21]. Prototyping implies a quick and economic approach that serves to achieve understanding of what final products should be [15]. From a technical perspective, prototypes can be differentiated according to its relation to later product development. Throwaway prototypes are used mainly for specification purposes; and they are not used as actual building blocks [15]. They are mostly used in exploratory and experimental prototyping. Evolutionary prototypes provide a basis for a real system, which is evolved out of the prototypes; they are used in evolutionary prototyping but can also be found in experimental prototyping (if it shows that they provide a good basis for a system) [15].

From a business perspective, startups can create a representation of product ideas, a so-called MVP, without actual product implementation. Eric Ries describes a classification of different types of MVPs [4], which are commonly used in the startup communities. For instance, a MVP can be a short animation that explains what a product does and why users should buy it. It can also be a user interface that looks like a real working product, but the actual business process is manually carried out (Wizard of Oz MVP). A concierge MVP is a manual service that consists of exactly the same steps users would go through with the product.

A few research paid attention on improving prototyping activities, such as the speed and effectiveness [28, 29]. Janssen et al. suggested code reuse to speed up writing code to prototype [28]. Grevet et al. described a 6-stage prototyping approach to speed up throw-away prototyping for new social computing systems using existing online systems [29]. In our work, we address the speed of prototyping from a socio-technical perspective, considering prototyping activities under human, market, finance and team factors.

2.3 A Prototype-Centric Learning Model in Software Startups

The Build-Measure-Learn loop is a key concept in Lean Startup [4]. The loop is used to manage and to operate software startups in finding a sustainable business model. A key idea is to minimize waste and to focus only on the elements, which will be tested. Lynn et al. describe another cycle, Probe and Learn, that is applicable to manage uncertainties about market, technology and time-to-market [25]. The authors suggest that startups should go to customers with an early version of a product to learn about the market, different applications and segments. Nguyen-Duc et al. propose a hunter-gather double loop to capture the evolution of startup activities from idea to achieving a product market fit [26]. The model visualizes the portion of product development vs. customer development activities across the startup stages. While these studies provide an emphasis on organization and evolution, they are well landed in an abstract space, not straightforward to apply from the SE perspective.

In the SE literature, Gordon et al. propose a rapid prototyping system approach to understand the prototype development of a system [27]. In the model, both low-fidelity and high-fidelity prototypes are essential parts of developing a system [27]. Preliminary product design activities create a throwaway prototype from the problem domain. A series of throwaway low-fidelity prototypes can be created to capture the ideas of what to built. Similarly, high-fidelity prototypes can also be evolved several times before reaching the product launch.

A literature survey of software development shows that startups often build a prototype in an evolutionary fashion and quickly learn from users' feedback [20]. We argue that both throwaway prototypes and evolutionary prototypes are important parts of startups' journey to a launched product. From the Lean startup perspective [4], learning is an input and also an outcome for a prototype. We tailored the double loop model in the previous work [26] by adapting Gordon's system prototyping elements [27] to capture the prototyping processes in the startup context, as shown in Fig. 1. The model focus on prototyping as the core concept and compose four loops:

- Idea-prototype loop: iterations of refining business idea through throwaway prototyping
- Throwaway prototype loop: iterations of constructing and learning from throwaway prototypes
- Evolutionary prototype loop: iteration of constructing and learning from evolutionary prototypes
- Pivot loop: starting a new cycle from the current product to a pivoted idea

Considering the model as a state-based system, it is possible to travel from a state to any other one. However, the typical flow would happen within two loops. It can also happen that a startup starts the loop from any state, for example, by doing a throwaway prototype before getting to a stated problem. In the scope of this work, we did not go in-depth about how these loops happen in our cases. The work will explore factors that occur during the startup progress and influence throw-away and evolutionary prototyping.

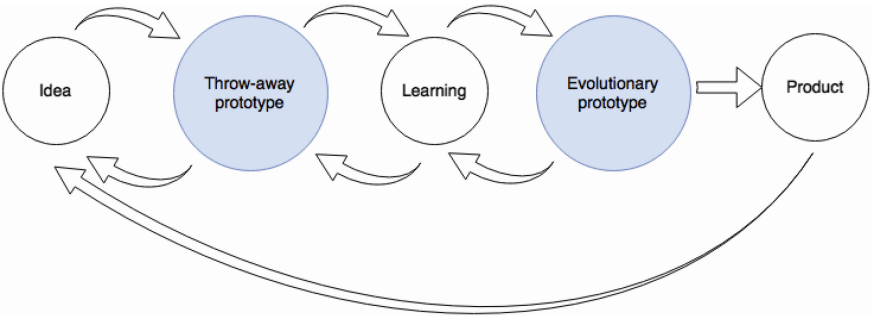

Fig. 1. A prototype-centric learning model in software startups

3 Research Approach

3.1 Multiple Case Study Design

This study is one part of a larger research activity that investigates the role of engineering activities in software startups. The objective is to explore commonalities, challenges and engineering patterns in software startups, from the business idea to a launched product. This study reports the findings from empirical data regarding prototyping activities. We conducted multiple case studies for a robust result in typical software startup population [11]. The unit of analysis is a startup company. We aimed at collecting as many startups as possible for a variety of the sample. As the aim is to reflect the state-of-practice rather than finding a secret recipe of success, we included startups in different stages and with different revenue statuses.

There is often a difficulty in identifying a real startup case among other similar phenomenon, such as freelancers, SMEs or part-time startups. We defined five criteria for our case selection: (1) a startup that operates for at least six months, so their experience can be relevant, (2) a startup that has at least a first running prototype, (3) a startup that has at least an initial customer set, i.e. first customer payments or a group of users, (4) a startup that has an intention to scale their business model, (5) a startup that has software as a main part of business core value.

The process of identifying and collecting data was done in 11 months, from March 2015 to February 2016. Cases were searched from four channels, (1) startups within the professional networks of the authors, (2) startups in the same town with the authors, (3) startups listed in Startup Norway and (4) Crunchbase database. The contact list includes 219 startups from Norway, Finland, Italy, Germany, Netherlands, Singapore, India, China, Pakistan and Vietnam. After sending out invitation emails, we received 41 feedbacks, approximately 18.7% response rate. Excluding startups that are not interested in the research, or startups that do not pass our selection criteria, the final set of cases are 20 startups, aliased as S1 to S20.

3.2 Data Collection and Analysis

Semi-structured individual interviews were used to collect data, since they enable the focus on pre-defined research topics and flexible structures to discover unforeseen information [28]. Methodological triangulation in data collection is also implemented by using evidence from documents and observations (in S01-S05, S09). Business documents, such as business model canvases and business plans were exposed to the research team as a preliminary step prepared for interviews. Observations were useful to understand how prototypes were implemented and used in the working environment.

The interviewees were asked questions about (1) business background (2) idea visualization and prototyping (3) product development (4) challenges and lessons learnt. The stories about startup ideas, prototypes and product development is organized into the schema as described in Fig. 1. Most of the interviews were conducted by the first author, with the attendance of a second researcher (the third author or sometimes external researchers in our network). This researcher has a long experience conducting interviews in software companies. Each interview lasted from 55 min to 70 min and the interviewees were informed about the audio recording and its importance to the study.

We used a thematic analysis – a technique for identifying, analyzing, and reporting standards (or themes) found in qualitative data [22]. We started by reading all interview transcripts and relevant documents, and coded them according to open coding [22]. A set of pre-determined categories were used to guide the coding process, as we have some interests in topics of (1) business original, (2) prototyping practices (3) pivoting (4) testing (5) challenges and (6) key performance indicators (KPIs). We attempted to label all meaningful text segments with appropriate codes. To feed data to this study, we filtered the codes that are related to prototyping, technical implementation, and testing activities prior to product launching. According to Sect. 2.2, throwaway prototypes were low-fidelity artifacts, such as mockup, wireframe, or simple code. Evolutionary prototypes were perceived as product building blocks, such as heavy code activities, i.e. feasibility testing of functionality, building new feature, etc. The relationship of the factors to the speed of prototyping or production was identified via text about challenges, or text specifying consequence on time-to-market or time to collect user feedback. We noted and reported evidence on prototyping as follows (1) factors that relate to prototyping activities in generals, (2) factors that slow down the prototyping activities and (3) factors that speed up the prototyping activities.

3.3 Case Description

The characteristics of our cases are given in Table 1. It is noticeable that a large number of the studied cases deliver peer-to-peer services as marketplaces or platforms (S01, S02, S03, S07, S08, S11, S13, S16, S20). There are also cases that deliver value in Business-to-Business model (B2B) (S04, S06, S10, S12, S15, S17). The cases are dominantly characterized by web-based and mobile-based software product with client-server architecture. We also identified the product focuses in early and later phases of the software startups [23]. Among them, there are some startups with annual revenue of one million euro or more (S06, S09).

Regarding the development strategy, interestingly, there are seven cases (35%) that have (parts of) product developed outside company boundary.

Table 1. Startup cases characteristics

Code	Product type	Early focus	Later focus	Dev. strategy	No. of prot.	Dev. method.
S01	Photo marketplace	Feature		Insource	2	Agile
S02	News generator	UX	New feature	Outsource	4	Agile
S03	Homemade food market	UX		Insource	2	Adhoc
S04	Construction management	Simple feature	New feature	Outsource	5	Distributed agile
S05	Underwater camera	Feasible technology		Outsourcing, subcontracting	7	Informal agile
S06	Sale visualization tool	UX	Flexible, scalable	Insource	3	Informal scrum
S07	Location recommendation	Feature, UX		Insource	3	Informal agile
S08	Ticket platform	Intuitiveness, friendliness	Scalable and new features	Outsource	2	Agile
S09	Educational quiz system	User friendliness	Scalable, Stable	Insource	5	From adhoc to distributed agile
S10	IoT OS platform	Ecosystem	Functionality	Insource	4	NO INFO.
S11	Ticket platform	User friendly, simple	More features, complexity	Insource	2	Adhoc
S12	Elearning platform	Feature		Insource	3	Agile
S13	Shipping services	NO INFO.	NO INFO.	Outsource	3	NO INFO.
S14	News services	Feature provider	Platform as a service	Insource	2+	Continuous development
S15	Smart grid application	NO INFO.	NO INFO.	Insource	NO INFO.	NO INFO.
S16	Secondhand marketplace	innovative feature	Product line	Insource	3	NO INFO.
S17	Simulation based training	UX, feature	Flexibility, Scalability	Insource	2 +	NO INFO.
S18	Open source messenger	Community	Feature	Open source	4	Adhoc
S19	Location based alert system	UX	Feature and enhanced UX	Insource	5	Agile
S20	Elearning system	User friendliness	Standardization	Insource	2	Agile

**Notation: NO INFO. means missing information*

The major reported development methodology is Agile, with iterative deliveries and frequent customer feedback: *"… Scrum based development, sprints of two weeks, standup, wrap-up meeting, we like to work in this way.*" (S06). In some cases, the company reports a type of informal Agile process: *"… fully informal but truly agile process with working release maintained, … iterative development of functionality and refactoring*" (S05)

One specific question asked to interviewees is how many prototypes have been made before product launching. The answers vary from two to seven prototypes, either throw-away or evolutionary ones, before a launch. In many cases, we considered prototypes as a tangible artifact that is experimented with (potential) users, customers and internal/external stakeholders.

4 Result

Figure 2 describes the influencing elements on throw-away prototyping (detail on Sect. 4.1) and evolutionary prototyping (detail on Sect. 4.2). It should be noticed that the direction of impact is not given. Some elements specifically show the positive/negative influences while other elements remain as general observations.

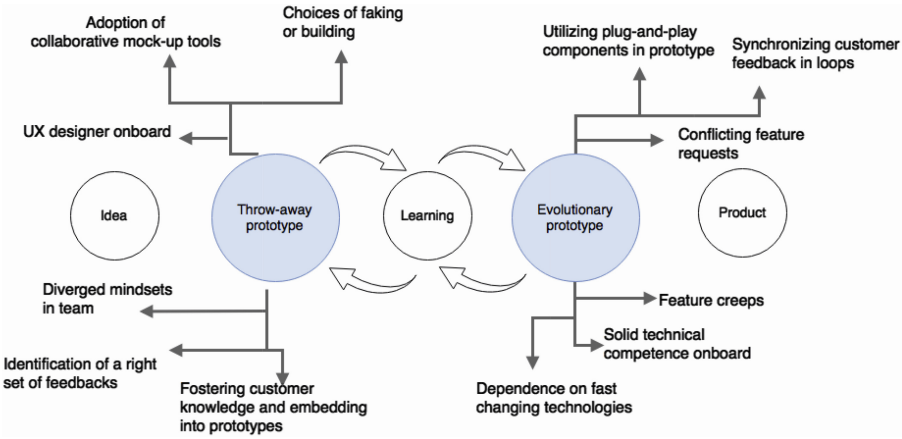

Fig. 2. Factors influencing the prototype-centric learning loops

4.1 Elements Influencing Throwaway Prototyping

4.1.1 Adoption of Collaborative Mock-up Tools

By adopting various tools, i.e. paper sketch, GUI mockups and wireframe tools, startups achieve a fast and an economic prototype without any technical expertise, as described in (S02, S09, S11, S13). In these cases, startups conducted very short iterations, from a few days (S02, S11, S13) to a few weeks (S09), from a product or a service idea to having the first user feedback. In S04, printing GUI layout in papers is reported as a good practice for teamwork, especially improving the customer involvement: "*normally we draw in the piece of paper first and then we make mock-ups… and then the customer joins us on that journey, then we click on the paper, we go to another one …*" (S04). It is also common that startups build mockups by using cloud-based software services. For such an online tool, the teamwork mode is reported as an important feature that facilitates collaborative design efforts among distributed team members (S02).

4.1.2 UX Designer Onboard

Business side of a startup (often CEOs) is always in a need of expressing and visualizing their ideas into more tangible artifacts. By doing that, sitting next to a designer is highly desirable for CEOs in early stages. In S02, the CEO expresses the need for a close collaboration with a designer in team: "*In this case, I would really like a designer that sits here together with us …"* (S2). The role of a design in mobile application is highlighted in another discussion with S2: "*You might think of user interface as a make-up for a person. But I think UI is the capacity that an app needs to interact with people."* It happens similarly in S12, when the CEO mentions about the process of designing the graphical part of their prototypes: "*The alternative is to create a specification … and just developing that document and all the process around it is typically very resource intensive. We talk about a future, … we make a prototype at a first phase implementation and then we adjust from there based on dialogues in between us*." (S12). For frontend-rich applications, a designer is a champion of the user experience, considering the viewpoint of users and keeping consistency among graphical elements across different platforms.

4.1.3 Choices of Faking or Building

There are often many uncertainties about customers and their expectations in the early stages of startups. Starting with a single-feature prototypes, or other approaches with implementation come always with a risk of wasting effort. It is considered time-saving to start with a clear mind about the throw-away strategy, by focusing on demonstrating business value rather than reusing the technical components (S02). Uncertainty about what to build and how to build often come with quick and dirty experiments without proper architectural designs, appropriate coding practices and documents. In this manner, frequent change of requirements or feature requests could lead to the increase of technical debts in later phase. Experimenting by the development of a runnable prototype was a costly and time-consuming experience in S09. In this way, the value of a prototype should exceed its cost. In S03, the development team has a clear plan for experimenting without "*making the product*" until they get the right product design. S11 applies the concept of "*fake it until you make it*", to simulate a final product without primary quality, both with functionalities and user experience. However, the focus on the speed has also led to the minimum part of viability. In S11, customer demonstration was done in a wizard of oz manner [4], customer interacting with an actual user interface, but business logics and backend functionality were done by manual work. Even though it is inefficient, the approach is easy and fast to build.

4.1.4 Collaboration Across Diverged Mindsets

We observed that in most of the cases, the ideas came from the CEOs, who are often business people or serial entrepreneurs. While the decisions about what the products should do come from a business mindset, they are implemented by developers with a technical mindset. In some cases (S01, S04, S05), there are challenges in communicating the product ideas and convincing the developers about the product value. In S04, it took as much time to discuss on the value proposition as to sketch a mockup. Vice versa, the communication of technical difficulties is also a time-consuming task, as mentioned by

a developer in S05: *"She [the CEO] is very sharp about business and finance stuffs, but it takes a long discussion to explain her about the importance of having flexible product design …" (S05).* The communication challenge might also happen between startups and customers, when no concrete prototypes are provided: *"We work with a customer organization, learn how they have worked with the current solutions and describe our proposal via the prototype. It is hard for them to realize the benefit without concrete examples…" (S04).* It also appears that a prototype is late released due to the wrong estimation of the CEO, who has no technical background. For example, in S1, the CEO insisted on a customer feedback having a new field in a frontend form, which caused the change of both business logic layer and data table structure.

4.1.5 Identification of a Right Set of Feedbacks

Steve Blank emphasizes the importance of early involvement of end users in product development [2]. Particularly, in startups developing products for mass market (or B2C business model), the feedback from the representative users of a market segment is essential. Nevertheless, not all users' input is equally valuable to product development. It was difficult to find the customer feedback that is useful for validating hypotheses in S02: "*I have attended a various types of events like that. To be honest, there are not so many interesting things there …*" (S02). The CEO wandered in town and talked to different people about the product idea. However, the approach is quickly found inefficient, as the users' feedbacks are often shallow. After that, the CEO targeted a group of innovative users from startups and research community and documented many interesting ideas for the product features. The integration of such lead users, "*whose strong needs will become general in a market-place months or years in the future*" [24], appears to be an important factor to accelerate the speed of startup learning. Lead users are also able to contribute via suggestions, testing and feedback, or even participate in the development and co-creation of new products or services, as observed in S14: "*We always do that in a close relation to our actual client stakeholders. Once we decide to narrow it on a new product area, the first thing we do is to get a partnership with a customer so that we can work together on a daily basis as stakeholders and product developers…*" (S14).

4.1.6 Fostering Customer Knowledge and Embedding into Prototypes

Prototypes can be seen from three different perspectives, function, look-and-feel and role, in which role is the representation of usability of the prototype [2]. In order to maximize lessons learned from a prototype, the vision on how end-users adopt a final product need to be visualized and captured in the prototype. As the actual end users are often not well known in the early phases, the integration of the user's role into the prototype design is a fuzzy task. The time pressure on prototyping makes startups skip a detailed analysis of users' behaviors. It seems that the adoption of customer/market analysis tools are not so common in our startup sample. In S02, the CEO emphasized the role of mapping tools, such as a customer journey map to describe the customer's experience: *"I have been told by my friends about the tool [a customer*

journey map]. We used it to describe how customer interact with the system and where could be the gap" (S02).

4.2 Elements Influencing Evolutionary Prototyping

4.2.1 Utilizing Plug-and-Play Components in Prototype

Utilizing ready-made components, such as Open source software (OSS) libraries and frameworks unlocks the capacity of experimenting functional as well as non-functional features. The adoption of OSS components was mentioned in all of the cases, from using tools (S19), integration of OSS code (S02, S03, S05, S20), to participation in OSS community (S18). The main benefits include reduced development cost and faster time-to-release, which were mentioned by the CTOs of (S19) and (S20): "*…we might not even come to the idea of making it happen if we do not have OSS as an experiment. Without OSS it would take a lot of time and very costly*" (S19). It is an even more obvious choice in open source type of platforms: "*It is very hard nowadays not to use OSS artifacts, especially when with Android development …*" (S20). It also appears that many advanced technologies were adopted via using OSS: "*A core part of our product includes a machine learning algorithm. We are lucky enough to find ml library in C++, entirely OSS, super cool*" (S02). By taking ready-made components, startups also reduce prototyping time by simplifying architectural aspects to some existing patterns.

4.2.2 Synchronizing Customer Feedback in Loops

Communication among team members or between a startup company and its external stakeholders is found as a significant factor delaying an iteration release. Insufficient communication due to misunderstanding, cultural difference, language barrier, lack of supporting tools happens often in outsourcing and remote partnership scenarios (S01, S09): "*Basically, we found some limitations that made it difficult to be efficient in the way to communicate. And since we're teams in different places it's really important that information flow works and also to make sure that all people—don't have to be involved in everything, and be able to group efficiently and create like projects, and store documents, and all these things, and have video-share links, and articles, and all these things.*" (S09). The misunderstanding and reworking also happens when customers are distant to developers and the customer feedbacks are not fully perceived. In S13, the CEO and sales people interacted with customers and collected insightful feedback from them. However, the feedback is not communicated efficiently to the development team in other locations. This leads to unnecessary re-work with communication and implementation effort and hence slows down the time to release.

4.2.3 Conflicting Feature Requests

It is a typical situation that evolutionary prototypes are built based on feature requests from the first customers. Gradually, when having more customers, new feature requests might vary from the business direction or even conflict with the previous functionalities. S14 describes how they handled such situation: "*either we solve them by providing them different products or we do ignore parts of the market… We make a very clear statement*

to what we think the future of journalism is, then we pursue that and the cost of that is neglecting parts of our market" (S14). Similarly, S15 expresses how their product evolved through different iterations: "*There will always be requirements arriving… Sometimes the new requirements disrupt the old requirements. At the moment, we are working to disrupt the old products*" (S15). Considering what to develop and which features to include adds complexity to future releases. Additionally, requests coming in the middle of the development sprint from large customers might influence the feature priority and delay the release further: "*We're in that situation all the time, it's very difficult to say no because giant customers telling you we need that functionality. If you're going to have us as customers you're going to have to make it, we need it in the contract that you have to make it. We also build it, we built it bigger and bigger*" (S11).

4.2.4 Feature Creeps

Many startups add new features to fit the prototype to a changing group of early customers. This leads to two possible challenges of satisfying customer demands, so-called (1) feature creep and (2) product portfolio. Feature creep refers to the addition of features to a product in a continuous manner: "*We are adding features all the time. This is not a product that will ever stop evolving. We will always have a strong engineering team to develop the product forward. We are not talking about maintenance here. We are talking about this being the core of the company's competence*" (S13). Startups rarely have a requirement management process to manage product complexity. Consequently, feature creeps are considered harmful to the production and enhancement of core features.

Moreover, this can be an unwanted expansion that requires changes also in the product architecture and even in the strategic direction. In S04, after the first two releases addressing a construction manager's requirements, the third release was developed for a construction operator's demands. Consequently, S04's product scope has grown from a single feature MVP to a supply-chain management system: "*So then we had a small one just for easy communication between users of the building and the maintenance guys… So the second feature was to manage document flow. And the third was to have a 3D model of the building. And all these things here we spent a lot of time and we were building in parallel with different prospects*" (S04).

In a larger scale, the expansion could lead to deriving a product portfolio. Startups face with challenges of keeping both the focus to increase the quality of core delivered values and satisfaction of important customers. While not all good ideas can be turned into features, some ideas are selected to develop further and might become the core value providers for startups.

4.2.5 Solid Technical Competence Onboard

In several cases (S09, S01, S03, S06) the technical competence determines the speed of feature releasing. Startups' technical members are required to possess good technical skills and they also need to be productive in an ambiguous development environment: "*We don't hire people basically for them being cheap because we don't have time. Our challenge is time and to be more productive other kind of competing companies … it's*

much better to have people that can—within a short time, could produce good code" (S09). It is also important to write code in a clean and structured manner, to be quality-aware in the early phases: "*The back end was pretty good because he had hired my boss at my current company … there was some friction there in how to develop systems between the professional programmer, my boss, and the copy paste programmers. I think that also contributed to it not working.*" (S11). The combination of technical competence and customer understanding is emphasized in another case: "*… It is very hard to find people both good at technology and have a good sense of commercial edge…*" (S08).

4.2.6 Dependence on Fast Changing Technologies

Startups often struggle with thriving in a technical uncertainty, whether under market pull or technology push impacts [20]. Due to different reasons, e.g., specific devices, platforms or protocols becoming popular in market, or new technology gaining momentum, there are needs for changing the current product's features to accommodate new technology (S01, S09, S11). In a small scale, for instance, the adoption of new animation effects, a different type of map, etc. leads to an extension of the current or coming iterations. In S02, the development of an IOS application is delayed after the codebase and all dependent libraries were forced to be upgraded to a newer version of Swift. The team took time to resolve all the changes so the next release can be done in Swift 3.0. The technology uncertainty is expected with mobile applications, as stated by the CEO of S11: "*…at the moment we are changing the technology platform. This perhaps has been the biggest challenge we have decided where to stand and make a new platform on development technology… So next generation which will be out in the market place around summer next year will be quite heavily rearranged."* (S11). In a large scale, the technical change can lead to a change of business directions.

5 Discussion

5.1 Reflections on the Results

We captured what happened during the early phases of the studied twenty software startups. We identified the factors that are found to influence the speed of prototyping across different types of prototypes. They can be grouped into (1) Artifacts, (2) Team competence, (3) Collaboration, (4) Customer and (5) Process dimensions. **Artifacts** include collaborative tools and reusable components. The practices of adopting artifacts are important for saving time of prototyping user interfaces and functionalities. The issue here is to select the suitable tools and components to match the prototyping's purposes. The requirement of **team competence** might vary due to the type of prototyping and the type of products. For instance, UI-rich application would require a designer onboard at the early stage while a good developer in the later stage. **Collaboration**, including efficient communication of visions and tasks among startup teams and interaction with external stakeholders, is important for shorten the learning loops. Besides, how **customers** are involved in the prototyping loops has an impact on the duration of the

prototyping. While inappropriate customer feedback delays the learning and creates more prototyping loops, too many requests from customers delay the time-to-release and introduce complexity to product management. Last but not least, prototyping is performed under many uncertainty and dependencies. Defining practices and **processes** to support decision-making under uncertainties would help in prototyping.

5.2 Threats to Validity

There are several threats to validity worth discussing [1]. One internal threat to validity is the bias in the data collection, as the data might not represent the comprehensive case. This is worth discussing as most of the cases are represented by one interview. In order to mitigate this threat, we selected CTO and CEO as interviewees, who have the best understanding about their startups. We also use other types of data sources, such as documents and observations to increase our understanding about the cases (S01 – S05, S09). The participative observations in S01 and S02 enabled deeper insights that go beyond cross-sectional interviews. A construct validity threat is the possible inadequate descriptions of constructs. We tried at our best to collect contextual information about the startups, from social media and personal contacts. When analyzing data, the coding process of interview transcripts was assisted by the authors' prior knowledge about prototyping and validated learning. This helped to focus on the investigated phenomenon without losing relevant details.

The external validity is normally not addressed by case study research. Our result is grounded on twenty cases, with diversity in company size, application domain, financial model, and growth stage and organization structure, which adds the robustness to our findings. Many themes, such as Sect. 4.1.1, Sect. 4.2.1, Sect. 4.2.5, Sect. 4.2.6 are observed in more than half of the cases. Our sample is characterized by Norwegian software startups, with a small team and bootstrap financing model. We do not consider other types of startups, for example, internal cooperate startups, venture capital invested startups, and American startups. Hence, the results cannot be directly applied to other contexts, though analytical generalization may be possible in similar contexts.

6 Conclusions

To the best of our knowledge, this is the largest multiple case study research about software startups. Grounded on twenty European startups, we adopted an analytical framework to reveal different factors that influence the prototyping activities in early stages of software startups. We found that both throw-away and evolutionary prototypes were influenced by artifacts adoption approach, available team competence, collaboration and customer involvement. Even though there is certain limitation in our case sample, there are still valuable lessons learnt for practitioners. For startups that follow the Lean Startup approach, it is important to align the learning objective with a collaborative and well-defined approach of prototyping. Moreover, startups need to find a systematic approach to integrate relevant external feedback in all phases of prototyping.

This work does not address the evolution of startups according to the learning loops, i.e. what are lessons from idea to throw-away prototype, what are lessons from switching from throw-away prototypes to evolutionary ones. Besides, future work can investigate different types of learning brought by different types of prototypes. This work addressed validated learning through an important angle, which is the speed of prototyping loops. In the future work, we will explore another equally important aspect, which is the quality of learning. Further studies might also identify the effective prototyping and development patterns among software startups.

References

1. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. **14**(2), 131–164 (2009)
2. Blank, S.: The Four Steps to the Epiphany: Successful Strategies for Products that Win, 2nd edn. K & S Ranch Press (2013)
3. Giardino, C., Wang, X., Abrahamsson, P.: Why early-stage software startups fail: a behavioral framework. In: Lassenius, C., Smolander, K. (eds.) ICSOB 2014. LNBIP, vol. 182, pp. 27–41. Springer, Cham (2014). doi:10.1007/978-3-319-08738-2_3
4. Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business, New York (2011)
5. Cooper, R.G.: Stage-gate systems: a new tool for managing new products. Bus. Horiz. **33**(3), 44–54 (1990)
6. Unterkalmsteiner, M., Abrahamsson, P., Wang, X., Nguyen-Duc, A., Shah, S., Bajwa, S.S., Yagüe, A.: Software startups: a research agenda. e-informatica. Softw. Eng. J. **10**(1), 89–123 (2016)
7. Fagerholm, F., Guinea, A.S., Mäenpää, H., Münch, J.: The RIGHT model for continuous experimentation. J. Syst. Softw. (2016)
8. Houde, S., Hill, C.: What do prototypes prototype. In: Helander, M., Landauer, T., Prabhu, P. (eds.) Handbook of Human-Computer Interaction, 2nd edn. Elsevier Science (1997)
9. Accessed 1 Dec 2016. http://qz.com/771727/chinas-factories-in-shenzhen-can-copy-products-at-breakneck-speed-and-its-time-for-the-rest-of-the-world-to-get-over-it/
10. Cohen, M.A., Eliasberg, J., Ho, T.H.: New product development: the performance and time-to-market tradeoff. Manage. Sci. **42**, 173–186 (1996)
11. Yin, R.K.: Case Study Research: Design and Methods, 4th edn. Sage Publications Inc, Thousand Oaks (2008)
12. Duc, A.N., Abrahamsson, P.: Minimum viable product or multiple facet product? The role of MVP in software startups. In: Sharp, H., Hall, T. (eds.) XP 2016. LNBIP, vol. 251, pp. 118–130. Springer, Cham (2016). doi:10.1007/978-3-319-33515-5_10
13. Lichter, H., Schneider-Hufschmidt, M., Züllighoven, H.: Prototyping in industrial software projects-bridging the gap between theory and practice. IEEE Trans. Softw. Eng. **20**(11), 825–832 (1994)
14. Floyd, C.: A systematic look at prototyping. In: Budde, R., Kuhlenkamp, K., Mathiassen, L., Zullighoven, H. (eds.) Approaches to Prototyping, pp. 1–18 (1984)
15. Beaudouin-Lafon, M., Mackay, W.E.: Prototyping development and tools. In: Jacko, J.A., Sears, A. (eds.) Handbook of Human-Computer Interaction, Revisited edn, pp. 1006–1031. Lawrence Erlbaum Associates, New York (2007)

16. Karvonen, T., Lwakatare, L.E., Sauvola, T., Bosch, J., Olsson, H.H., Kuvaja, P., Oivo, M.: Hitting the target: practices for moving toward innovation experiment systems. In: Fernandes, J.M., Machado, R.J., Wnuk, K. (eds.) ICSOB 2015. LNBIP, vol. 210, pp. 117–131. Springer, Cham (2015). doi:10.1007/978-3-319-19593-3_10
17. Sauvola, T., Lwakatare, L.E., Karvonen, T., Kuvaja, P., Olsson, H.H., Bosch, J., Oivo, M.: Towards customer-centric software development: a multiple-case study. In: 41st Euromicro Conference on Software Engineering and Advanced Applications (2015)
18. Bosch, J., Holmström Olsson, H., Björk, J., Ljungblad, J.: The early stage software startup development model: a framework for operationalizing lean principles in software startups. In: Fitzgerald, B., Conboy, K., Power, K., Valerdi, R., Morgan, L., Stol, K.-J. (eds.) LESS 2013. LNBIP, vol. 167, pp. 1–15. Springer, Heidelberg (2013). doi:10.1007/978-3-642-44930-7_1
19. Olsson, H.H., Alahyari, H., Bosch, J.: Climbing the "stairway to heaven": a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In: 38th Euromicro Conference on Software Engineering and Advanced Applications (2012)
20. Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software development in startup companies: a systematic mapping study. Inf. Softw. Technol. **56**(10), 1200–1218 (2014)
21. Brooks, F.P.: The Design of Design: Essays From a Computer Scientist. Addison-Wesley Professional, Boston (2010)
22. Boyatzis, R.E.: Transforming Qualitative Information: Thematic Analysis and Code Development. Sage Publications, Thousand Oaks (1998)
23. Nguyen-Duc, A., Shah, S., Abrahamsson, P.: Towards an early stage software startups evolution model. In: 42nd Euromicro Conference on Software Engineering and Advanced Applications (2016)
24. Von Hippel, E.: Lead users: a source of novel product concepts. Manage. Sci. **32**(7), 791–805 (1986)
25. Lynn, G.S., Morone, J.G.: Marketing and discontinuous: the probe and learn process. Calif. Manage. Rev. **38**(3) (1996)
26. Nguyen-Duc, A., Seppnen, P., Abrahamsson, P.: Hunter-gatherer cycle: a conceptual model of the evolution of startup innovation and engineering. In: 1st Workshop on Open Innovation on Software Engineering, ICSSP (2015)
27. Luqi, F.K.: An introduction to rapid system prototyping. IEEE Trans. Softw. Eng. **28**(9), 817–821 (2002)
28. Jansen, S., Brinkkemper, S., Hunink, I., Demir, C.: Pragmatic and opportunistic reuse in innovative start-up companies. IEEE Softw. **25**(6), 42–49 (2008)
29. Grevet, C., Gilbert, E.: Piggyback prototyping: using existing, large-scale social computing systems to prototype new ones. In: 33rd Annual ACM Conference on Human Factors in Computing Systems; Seoul, Republic of Korea, pp. 4047–4056 (2015)

Key Challenges in Agile Requirements Engineering

Eva-Maria Schön[1,2(✉)], Dominique Winter[3], María José Escalona[1], and Jörg Thomaschewski[3]

[1] University of Seville, Seville, Spain
eva.schoen@iwt2.org, mjescalona@us.es
[2] CGI Deutschland Ltd. & Co. KG, Hamburg, Germany
[3] University of Applied Sciences Emden/Leer, Emden, Germany
dominique.winter@designik.de,
joerg.thomaschewski@hs-emden-leer.de

Abstract. Agile Software Development (ASD) is becoming more popular in all fields of industry. For an agile transformation, organizations need to continuously improve their established approaches to Requirements Engineering (RE) as well as their approaches to software development. This is accompanied by some challenges in terms of agile RE. The main objective of this paper is to identify the most important challenges in agile RE industry has to face today. Therefore, we conducted an iterative expert judgement process with 26 experts in the field of ASD, comprising three complementary rounds.

In sum, we identified 20 challenges in three rounds. Six of these challenges are defined as key challenges. Based on the results, we provide options for dealing with those key challenges by means of agile techniques and tools. The results show that the identified challenges are often not limited to ASD, but they rather refer to software development in general. Therefore, we can conclude that organizations still struggle with agile transition and understanding agile values, in particular, in terms of stakeholder and user involvement.

Keywords: Agile Software Development · Requirements Engineering · Challenges · Agile RE · Stakeholder and user involvement · Human-Centered Design

1 Introduction

Agile Software development (ASD) gains in popularity in today's business world due to enabling immediately changes in the direction of product development. These short-term changes in direction require a flexible approach to Requirements Engineering (RE) as well. In addition, agile methodologies (such as Scrum [1], Kanban [2] or Extreme Programming [3]) are often combined with Human-Centered Design (HCD) [4] activities in order to emphasize a value-driven approach to product development [5, 6]. To this end, the field of agile RE has emerged during the last decade.

Focusing on user needs and value delivery becomes an important aspect in product development due to the increasing competition in all areas. With regard to ASD, plan-driven organizations moved away to value-driven organizations. On the one hand,

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 37–51, 2017.
DOI: 10.1007/978-3-319-57633-6_3

people in plan-driven organizations often negotiate about project plans, pricing models and the amount of features they can develop with the available resources. They are emphasizing the generated outputs such as number of created features during a time period. On the other hand, people in value-driven organizations discuss visions, experiences and human values as well as the way to address them through the product. They focus on the outcomes that the delivered outputs entail.

Compared to sequential approaches to RE, which comprise a requirement analysis phase before the development can even begin, agile RE is carried out along with the development itself. Therefore, continuous management of requirements is a crucial attribute. Requirements are regularly described from a user perspective in the form of epics and user stories [7] instead of creating a requirements document [8]. Recent research is showing that there are several ways of running RE in an agile environment while involving users and stakeholders [5, 9–12].

Performing agile RE can lead to challenges organizations have to deal with. In literature, there can be found some studies investigating challenges in agile RE (see [11–15]). However, the related work still lacks in giving a general overview of the challenges in current industry.

This study pursues the main objective of identifying the most important challenges in agile RE industry has to address today. We aim to build a shared understanding concerning these challenges among voices that matter by means of experts in the field of agile RE. Thus, the research questions we pose are listed below:

- RQ1: What are the key challenges in Agile Requirements Engineering?
- RQ2: How can we deal with the identified key challenges?

The paper is structured as follows: Sect. 2 briefly summarizes the related work and points out the research gap. Section 3 presents the applied research method and describes the iterative expert judgement process. Then, Sect. 4 identifies the findings and discusses both on their meaning and on the limitations of this study. Finally, Sect. 5 provides the conclusions as well as an outlook on future research.

2 Related Work

There are related studies in the literature that investigate challenges in agile RE by means of different research methods. Table 1 shows an overview of the reported challenges and used research methods.

Analyzing the related work, we can state that the authors use two different kinds of research approaches in general. On the one hand, Ramesh et al. [13] and Bjarnason et al. [14] utilize case studies to investigate the challenges in the field. On the other hand, Inayat et al. [11], Heikkila et al. [15] and Soares et al. [12] report challenges in agile RE by analyzing primary studies with the aim to identify available evidence in existing research.

Table 1. Challenges in agile RE reported by related work

Authors	Research method	Reported challenges
Ramesh, Cao, Baskerville [13]	Multi-case study (16 companies)	Problems with cost and schedule estimation; inadequate or inappropriate architecture; neglect of non-functional requirements; customer access and participation; prioritization on a single dimension; inadequate requirements verification; minimal documentation
Bjarnason, Wnuk, Regnell [14]	Case study	Planning for agility; weak requirements prioritization; weak effort estimates; quality issues; system completed late; capturing innovation; lack of documented requirements; customer-proxy role; ensuring competence (RE, VV); motivating teams for requirements work; weak requirements at start
Inayat, Salim, Marczak, Daneva, Shamshirband [11]	Systematic literature review	Minimal documentation; customer availability; inappropriate architecture; budget and time estimation; neglecting non-functional requirements (NFRs); customer inability and agreement; contractual limitations; requirements change and its evaluation
Heikkila, Damian, Lassenius, Paasivaara [15]	Mapping study	Problems with client or customer representatives; insufficiency of user story format; difficulties in prioritization of requirements; growing technical debt; reliance on tacit requirements knowledge; imprecise effort estimates
Soares, Alves, Mendes, Mendonca, Spinola [12]	Systematic literature review	Requirement prioritization; non-functional requirements identification; lack of information; volatility of requirements; requirements definition; dependence among requirements; prediction of impacts of changes; user dependence; communication and collaboration with users; requirements validation

Ramesh et al. [13] results were published in 2010. However, as ASD is a rapidly changing research area and the body of knowledge has evolved over the last years, we need to clarify whether the reported challenges are still relevant today. For instance, NFRs may not be longer a challenge for industry since the concept of the Definition of Done and the usage of acceptance criteria are widely spread. Bjarnason et al. [14] carry out a case study in only one company, therefore the results may not be applicable to other companies and may not be representative in general. In comparison, Inayat et al. [11], Heikkila et al. [15] and Soares et al. [12] review primary studies by analyzing existing literature, which is a good approach to get an impression of relevant aspects from a theoretical viewpoint. Nevertheless, one could argue that this is not an appropriate approach to investigate the existing challenges in practice.

To this end, the aim of this study is to identify the most important challenges in agile RE industry has to face up today by getting insights from 26 experts in the field. To the best of our knowledge there is no existing study investigating these challenges by means of a qualitative study with practicing experts in ASD working for many different companies.

3 Research Method

We used an iterative expert judgement process rooted in a Delphi study [16–18] in order to respond to our RQs. We applied a modified Delphi study where measuring consensus and stability at group level among several iterations was not the most crucial part. On the contrary, we shifted the focus to applying the valuable features of Delphi for conducting our iterative expert judgement process [19]:

- Anonymity among experts to avoid influence of dominant individuals
- Iterative approach
- Controlled feedback with statistical group response

The main benefit of our modified approach was utilizing the learnings from a previous iteration for carrying out the following ones.

3.1 General Study Design

The study was performed in three complementary rounds. Figure 1 gives a general overview of the process. At the beginning of each round, we started designing the questionnaire, optimized by a pretest. Once finished, the invitation was sent to the experts via email. In the second and third round, we attached the results of the previous rounds to the invitation in order to share the outcomes among the panel. The experts had two weeks to fill in the questionnaire. During the following two weeks we evaluated the results, created the report, specified the criteria for dropping items for the following round and designed the questionnaire for the next round.

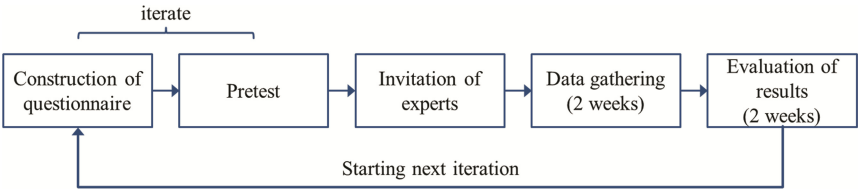

Fig. 1. General process of study

We conducted the study in German since most of the experts are native speaker. Since we are aware that the term agile RE is not very accepted in the agile community and some experts understand this as a contradiction in itself, we decided not to ask for challenges in agile RE directly. On the contrary, we phrased our questions differently and described the context of our study within the introduction part of each questionnaire.

We used google forms for the first and second round, whereas limesurvey was used for the third round due to the complexity of the questionnaire. In general, we decided to use 7-point Likert items since this has been proven to be the best choice in terms of avoiding interpolations within related research fields [20]. Besides, we adapted the quality criteria proposed by Diamond et al. [17] so as to ensure the quality of our study.

3.2 Panel of Experts

We selected our panelists specifically for their knowledge or position regarding the issue under study. As shown in previous work, the research field of agile RE is very close to existing work practices in industry [5]. To this end, we defined the reproducible criteria for selecting participants as follows:

- Many years of experience as professional in the field of ASD
- Working experience in one or more of the following roles: Product Owner, Scrum Master, Agile Coach, Consultant for Agile Transition, Kanban Expert or Lean Startup Expert

The panel consisted of 26 experts who are working in 19 different companies located in Germany and Switzerland. In general, they had 2–10 years of experience working in ASD (average = 6.14 years). In comparison, experts have about 0–16 years of experience with RE (average = 6.65 years). Even though one expert stated that he had no experience with RE at all, we decided to include his answers into the study, since he has long experience in ASD and in general there do not exist a specific role of a requirements engineer.

Figure 2 shows the kind of process models experts have been working with. It is worth mentioning that most of the experts have experience both with sequential approaches and with agile approaches.

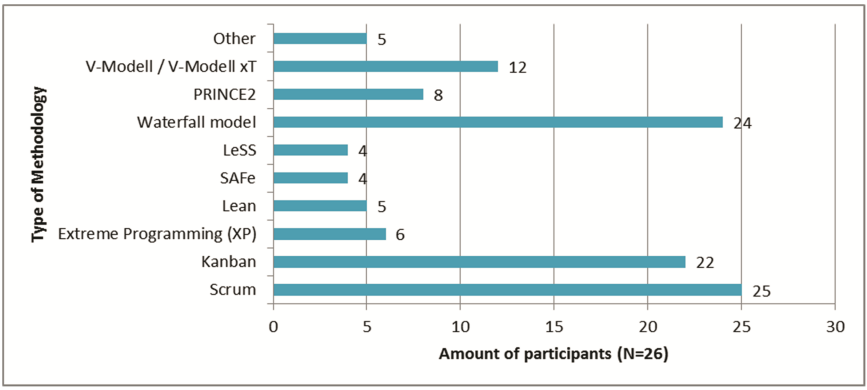

Fig. 2. Process models used by experts

In addition, Table 2 displays the know-how level in terms of ASD rated by experts themselves.

Table 2. Know-how of panel in terms of ASD

know-how very poor	1	2	3	4	5	know-how very high
	0.0%	0.0%	15.4%	69.2%	15.4%	

3.3 Round 1

The questionnaire of the first round comprised two open questions, repeated 15 times. On the one hand, the experts were asked what the most important challenges with requirements in terms of ASD were. On the other hand, they should give a statement for each challenge to clarify why they considered this challenge as important. The minimum number of required answers was 3, whereas the maximum was 15. In sum, we received 107 answers (items) from 26 experts. Table 3 shows an example of an item consisting of a challenge and a statement concerning importance. The full results can be found in [21].

Table 3. Exemplary item in round 1

Question round 1	Answer given by expert
What challenge do you perceive with requirements in terms of Agile Software Development?	Stakeholders affected by requirements or changing the system are not involved
Why do you consider this challenge as important?	In one of my projects, representatives of end users did not really knew the pain of end users. Even the early UI prototypes were tested by incorrect stakeholders, which led to risks of conflicts and failure

With respect to data analysis, each challenge was categorized by the authors during a workshop. Those items, which could not be categorized easily, were discussed within the group of authors. We used the following categories: stakeholder and user involvement, collaboration within the team, vision and big picture, iteration planning and estimation, granularity of requirements, dependencies of requirements, understanding agile and agile values, continuous delivery of value, roles and responsibilities, need for security, requirement validation, RE methods, format of requirements, clarity of requirements, prioritization, refinement, discovery and transparency.

Additionally, the reported challenges were categorized according to their agile RE activity (see Table 4).

Table 4. Agile RE activities

Agile RE activity	Description
Discovery	Eliciting new ideas/requirements
Refinement	Clarifying and analyzing new ideas/requirements
Prioritization	Measuring the value that the development will add to the product
Review	Checking if requirement is implemented in the manner to deliver value
Documentation	Capturing discussion and decisions around the requirement

3.4 Round 2

We checked each item of round one critically, whether or not it was appropriate for answering our RQs and being queried in the next round. Thus, items of round 1 were consolidated or excluded. In the end, we identified 34 items as relevant for assessing them in round 2. Based on those items, we created the questionnaire for the second round.

The resulting questionnaire assessed 34 items related to the following topics: stakeholder and user involvement (6 items), understanding agile and agile values (6 items), RE methods (10 items), iteration planning and estimation (6 items) and format of requirements (6 items).

The experts rated each item using 7-point Likert items (see Fig. 3). Moreover, they could choose giving no statement. To sum up, we received responses from 23 experts. For each item we calculated mean, variance and standard deviation. Additionally, we created a diagram showing the distribution of experts' opinion (see Fig. 3) and discussed on the meaning of findings. The results of round two can be found in [22].

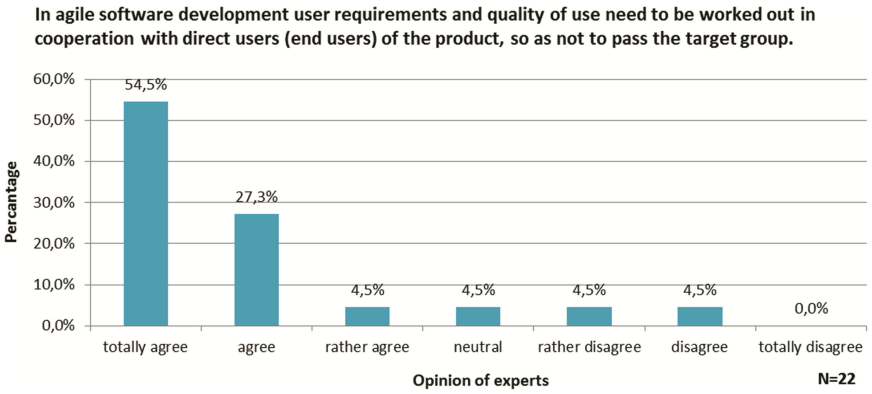

Fig. 3. Exemplary item of round 2

3.5 Round 3

We reduced the number of items when designing the questionnaire for the third round. Considering items from round 2, we assessed each item according to (a) its relevance in terms of our RQs, (b) the importance in terms of the attributes of agile RE, (c) the opinion of the experts and the comprehensibility of the items.

The final questionnaire comprised two parts. The first part queried in sum 20 potential key challenges of agile RE (see Appendix). The experts were asked to rate each item, whether or not it is a challenge in agile RE. Moreover, they had the option to choose giving no statement. Then, the second part evaluated those items that experts identified as challenge in terms of importance, following 7-point Likert items (totally important, important, rather important, neutral, rather unimportant, unimportant, totally unimportant, no statement). In addition, experts optionally had the chance to provide a solution for solving the challenge.

In sum, 22 experts filled in the questionnaire. We classified each of the 20 items as challenge in Agile RE since we derived all items from the results of the previous rounds. Besides, we calculated the number of experts who rated each item as a challenge. Then, we defined challenges as key in those cases where 2/3 of the experts' answers were: "Yes, it is a challenge". Finally, we calculated the importance for those items. The results of round 3 can be found in [23].

4 Results and Discussion

Summarizing the results of the three complementary rounds, we derived 20 challenges that companies have to cope with in terms of agile RE (see Appendix). We categorized such challenges into stakeholder and user (3 items), requirements management (7 items), methods and artifacts (5 items) and format of requirements (5 items).

4.1 (RQ1) What Are the Key Challenges in Agile Requirements Engineering?

We identified six key challenges industry has to face today in terms of agile RE (see Table 5). In general experts weighted the identified challenges as important [23] and none of them rated one of the six key challenges as unimportant.

Table 5. Key challenges in agile RE

ID	Key challenge	N	Yes	No
C1	In agile software development functional or technical dependencies with other teams are a challenge because a considerable coordination effort is required	17	14 (82.4%)	3 (17.6%)
C2	In agile software development it is a challenge that stakeholders understand that the development team can make independent (detailed) decisions	20	15 (75.0%)	5 (25.0%)
C3	In agile software development it is a challenge not to lose sight of the big picture during the implementation of complex requirements	20	15 (75.0%)	5 (25.0%)
C4	In agile software development continuous management of requirements is a challenge since not all of them are fixed at the beginning and they may change over the course of the project	22	16 (72.7%)	6 (27.3%)
C5	In agile software development it is a challenge to work out user requirements and quality of use in cooperation with direct users (end users) of the product	18	13 (72.2%)	5 (27.8%)
C6	In agile software development it is a challenge to involve stakeholders throughout the whole development process in regular iterations, so that product development will succeed	20	14 (70.0%)	6 (30.0%)

All challenges related to the category stakeholder and user are classified as key challenges (C2, C5, C6). Therefore, we can conclude that organizations still struggle to the agile transition. Evolving an agile mindset within a whole organization even in parts that are not close to development is still a challenge companies have to address.

Typically, agile transformation starts in development-oriented parts of an organization. Transforming an organization to become more agile implies a change within the whole organization. The results show that there is a gap between knowledge and understanding agile values [24] within organizations. Development-oriented techniques

evolve rapidly. In comparison, there are still challenges involving stakeholders and users into the agile processes (C2, C5, C6).

Two challenges (C1, C4), related to category requirements management, are key in agile RE. On the one hand, companies have an issue with the continuous management of requirements. On the other hand, they have a problem with technical or functional dependencies due to raising effort in coordination. Besides, one challenge of methods and artifacts (C3) is a key challenge.

ASD is commonly used in environments where people have to solve complex adaptive problems [25]. Concerning C1, C3, and C4 we can state that there are still challenges to be solved, due to the complexity of problems, which are not addressed by agile techniques properly. To this end, existing techniques and methods must be adapted or new techniques need to be found.

Figure 4 offers an overview of the categorized key challenges.

Stakeholder and user	• understanding of agile values of the stakeholders (C2) • refine requirements in collaboration with users (C5) • involve stakeholder iteravely (C6)
Requirements management	• technical or functional dependencies to other teams (C1) • continuous requirements management (C4)
Methods and artifacts	• staying focused on the big picture (C3)

Fig. 4. Categorized key challenges in agile RE

4.2 (RQ2) How Can We Deal with the Identified Key Challenges?

Experts recommend techniques, methods and tools in order to deal with the challenges in agile RE. Below, we will list the techniques and methods proposed by the panel for each key challenge.

C1: In agile software development functional or technical dependencies with other teams are a challenge because a considerable coordination effort is required.
More than three experts recommended using scaled frameworks such as LeSS, SAFe or Scrum of Scrum. Moreover, they proposed the use of the following techniques: creating a common understanding among all, enhancing continuous communication and collaboration, training the ability to solve dependencies, holding weekly coordination meetings, organizing teams in matrix management, building communities of practices for transcending topics, release planning (SAFe), team-transcending availability of product und sprint backlogs, involving temporary representatives in other teams, enforcing continuous integration, improving API-driven development and microservices.

C2: In agile software development it is a challenge that stakeholders understand that the development team can make independent (detailed) decisions.
The following techniques were suggested: continuous coordination and presenting possible solutions to stakeholder, providing transparency about rationales of the decisions, strengthening product owner with competency in decision making and helping stakeholders become aware of the consequences of interfering into detailed decisions.

More than three experts recommended providing alternative solutions for one requirement. In addition, it is useful to demonstrate that the recommended solution of a stakeholder is an alternative out of many. In previous rounds, more than one expert stated that product owner and stakeholder altogether decide what to be developed. In contrast, the development team decides how the requirement should be developed.

C3: In agile software development it is a challenge not to lose sight of the big picture during the implementation of complex requirements.
The following techniques were recommended: creating a shared understanding regarding the meaning of the big picture by means of a product vision, defining epics or subgoals in the beginning, managing the big picture as a responsibility of the product owner, providing transparency concerning changes among all, understanding connections among user stories by means of story mapping, visualizing customer journey in the beginning, involving users continuously in order to focus on the problem to be solved and identifying central contact person for related topics to enable rapid coordination. Moreover, the experts advised to use visualization by means of roadmaps, sketches of the system and processes, and value streams.

C4: In agile software development continuous management of requirements is a challenge since not all of them are fixed at the beginning and they may change over the course of the project.
The experts proposed the following techniques, methods and tools: collaborating closely with the requesting stakeholder, communicating regularly within the team, refining and prioritizing continuously the product backlog, grooming on demand (Kanban), describing in detail the requirements in the sprint backlog, reviewing the results regularly, discussing the maturity level of a requirement with the team, grouping user stories to epics, using Kano analysis, screening and scoring the theme, weighting relatively, utilizing spike stories to evaluate uncertainty in requirements and using ticketing tools (e.g. JIRA).

C5: In agile software development it is a challenge to work out user requirements and quality of use in cooperation with direct users (end users) of the product.
The experts recommended utilizing the following techniques: prototypes, interviews, observing users by the think aloud method, A/B testing, UX labs, analyzing usage behavior, friendly user tests, alpha/beta/silent launches, improving continuously a released version, utilizing a UX-board for play back user insights and testing hypotheses with real users. In addition, one expert suggested adapting user research to ASD by reducing the methods to the minimal, evaluate within the team without report creation, reduce financial restrictions for user involvement as well as problems of accessing real user by means of panels or a prior recruitment.

C6: In agile software development it is a challenge to involve stakeholders throughout the whole development process in regular iterations so that product development will succeed.
The following techniques were proposed: defining stakeholders and their involvement in regular iterations, proposing goals instead of prescribing solutions, involving all possible stakeholders in the beginning and reducing the amount of people over time.

More than eight experts suggested involving stakeholders by regular planning and review meetings to gather feedback and useful information. In light of this, they recommended clarifying the purpose of the meetings and the importance of the outcomes to be discussed beforehand.

4.3 Meaning of Findings

Comparing our findings to the identified challenges of the related work (see Table 1), we can conclude that 16 out of our challenges are not reported by the related studies.

Our key challenge C5 (user involvement) is reported by all related studies. In addition, three studies [11–13] report issues with non-functional requirements, which is comparable to our challenge C13. There is also a relation between the key challenge C4 (continuous requirements management) and the challenge "requirements change and its evaluation" reported by [11]. Moreover, the key challenge C1 (technical or functional dependencies to other teams) is reported by [12] in a slightly different manner since they phrase it like "dependence between requirements".

Moreover, the results show that the identified challenges are often not limited to ASD, but they rather refer to software development in general. Therefore, we can conclude that organizations still struggle with agile transition and understanding agile values, in particular, in terms of stakeholder and user involvement.

4.4 Limitations

We are aware that the design of a questionnaire is important for the process of data gathering. To this end, we made several pretests of each questionnaire we used with participants matching our criteria of expert selection. Nevertheless, we observed two experts struggling with the user experience of the questionnaire tool (Google Forms) used in round 1. Therefore, we decided to use another tool (LimeSurvey) for the questionnaire in round 3, which was more complex than the previous two.

To carry out the study, the group of authors created summaries of the results and made decisions concerning the kind of items they had to query in the following rounds. That may lead to bias in the opinion building process of the panel. We tried to prevent this point by being very accurate in terms of data analysis and by creating the reports. In addition, we selected items for the following rounds through the selection criteria defined earlier.

5 Conclusions and Future Work

This paper has addressed the identification of the most important challenges in agile RE industry has to face up today. Moreover, we examined how to deal with those challenges. For that purpose, we carried out an iterative expert judgement process comprising three complementary rounds. The learnings from previous iterations were used for carrying out the following ones. Our panel consisted of 26 experts in the field of ASD working for 19 different companies.

We have contributed to the body of knowledge of software development by identifying 20 challenges industry has to address at present in terms of agile RE. Six of these challenges have been defined as key challenges. In addition, we have analyzed options to deal with those key challenges by means of agile techniques recommended by the experts.

Future research may specifically identify challenges in agile RE by means of an international panel of experts, for instance with experts from Scandinavian countries. Our aim is to conduct a comparative analysis among the statements of German-speaking experts with the viewpoint of international experts. In addition, we are creating a tool that supports practitioners solving the identified challenges using agile techniques. Therefore, we are working on agile RE patterns. Some experts stated that the queried challenges are not limited to ASD. To this end, future studies may analyze whether the challenges appear in terms of RE in general.

Acknowledgements. First of all, we would like to thank all experts for their participation and sharing their valuable knowledge. Moreover, we would like to thank all participants in our pretests for their collaboration. This research has been supported by the MeGUS project (TIN2013-46928-C3-3-R), Pololas project (TIN2016-76956-C3-2-R) and by SoftPLM Network (TIN2015-71938-REDT) of the Spanish Ministry of Economy and Competitiveness.

Appendix

See Table 6.

Table 6. Challenges in agile Requirements Engineering

ID	Challenge in agile RE	N	Yes	No
C1	In agile software development functional or technical dependencies with other teams are a challenge because a considerable coordination effort is required	17	14 (82.4%)	3 (17.6%)
C2	In agile software development it is a challenge that stakeholders understand that the development team can make independent (detailed) decisions	20	15 (75.0%)	5 (25.0%)
C3	In agile software development it is a challenge not to lose sight of the big picture during the implementation of complex requirements	20	15 (75.0%)	5 (25.0%)
C4	In agile software development continuous management of requirements is a challenge since not all of them are fixed at the beginning and they may change over the course of the project	22	16 (72.7%)	6 (27.3%)
C5	In agile software development it is a challenge to work out user requirements and quality of use in cooperation with direct users (end users) of the product	18	13 (72.2%)	5 (27.8%)
C6	In agile software development it is a challenge to involve stakeholders throughout the whole development process in regular iterations so that product development will succeed	20	14 (70.0%)	6 (30.0%)
C7	In agile software development it is a challenge that the requirements to be implemented are clearly defined from the development start since the priorities often change in the short term	21	13 (61.9%)	8 (38.1%)
C8	In agile software development it is a challenge to analyze requirements with regard to the past development in order to avoid side effects	15	9 (60.0%)	6 (40.0%)
C9	In agile software development it is a challenge to formulate requirements as objectives that describe the problem area so that the creativity in solution finding is not restricted	22	13 (59.0%)	9 (41.0%)
C10	In agile software development it is a challenge to slice requirements in such a way that they offer added value for the product	20	11 (55.0%)	9 (45.0%)
C11	In agile software development it is a challenge to justify the benefits of the requirements in order to make the added value of the implementation clear as well as decisions for a specific requirement comprehensible	21	11 (52.4%)	10 (47.6%)
C12	In agile software development it is a challenge to document changes to the requirements comprehensibly	18	9 (50.0%)	9 (50.0%)
C13	In agile software development it is a challenge to establish non-functional requirements	19	9 (47.4%)	10 (52.6%)
C14	In agile software development it is a challenge to focus only on the refinement of the requirements for the short-term iterations	22	10 (45.5%)	12 (54.5%)
C15	In agile software development it is a challenge to develop an outlook on the next iterations without making it a binding one	21	9 (42.9%)	12 (57.1%)
C16	In agile software development it is a challenge to design requirement documents in such a way that they can be adapted to changing surrounding factors at reasonable effort	21	9 (42.9%)	12 (57.1%)
C17	In agile software development it is a challenge to use methods for elicitation and evaluation of requirements in which the findings are shared with the development team	20	8 (40.0%)	12 (60.0%)
C18	In agile software development it is a challenge to capture requirements in such a way that detailed test cases can be derived from them for quality assurance	21	8 (38.1%)	13 (61.9%)
C19	In agile software development it is a challenge to formulate clear and comprehensible requirements in order to avoid uncertainties in the development	22	7 (31.8%)	15 (68.2%)
C20	In agile software development it is a challenge that elicitation and evaluation of requirements are not fast enough in the project context	17	5 (29.4%)	12 (70.6%)

References

1. Schwaber, K.: Agile Project Management with Scrum. Microsoft, Redmond (2004)
2. Anderson, D.J.: Kanban - Successful Evolutionary Change for your Technology Business. Blue Hole Press, Sequim (2010)
3. Beck, K.: Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading (2000)
4. International Organization for Standardization: ISO 9241-210:2010 - Ergonomics of human-system interaction - Part 210: Human-centred design for interactive systems (2010)
5. Schön, E.-M., Thomaschewski, J., Escalona, M.J.: Agile requirements engineering: a systematic literature review. Comput. Stand. Interfaces **49**, 79–91 (2017)
6. Schön, E., Winter, D., Uhlenbrok, J., Escalona, M.J., Thomaschewski, J.: Enterprise experience into the integration of human-centered design and Kanban. In: Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016), Lisbon, Portugal, pp. 133–140 (2016)
7. Cohn, M.: User Stories Applied: For Agile Software Development (2004)
8. Sommerville, I., Sawyer, P.: Requirements Engineering: A Good Practice Guide. Wiley, New York (1997)
9. Silva da Silva, T., Martin, A., Maurer, F., Silveira, M.: User-centered design and agile methods: a systematic review. In: 2011 AGILE Conference, pp. 77–86. IEEE (2011)
10. Brhel, M., Meth, H., Maedche, A., Werder, K.: Exploring principles of user-centered agile software development: a literature review. Inf. Softw. Technol. **61**, 163–181 (2015)
11. Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S.: A systematic literature review on agile requirements engineering practices and challenges. Comput. Hum. Behav. **51**, 915–929 (2015)
12. Soares, H.F., Alves, N.S.R., Mendes, T.S., Mendonca, M., Spinola, R.O.: Investigating the link between user stories and documentation debt on software projects. In: 2015 Proceedings of the 12th International Conference on Information Technology - New Generations, pp. 385–390. IEEE (2015)
13. Ramesh, B., Cao, L., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Inf. Syst. J. **20**, 449–480 (2010)
14. Bjarnason, E., Wnuk, K., Regnell, B.: A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In: Proceedings of the 1st Workshop on Agile Requirements Engineering - AREW 2011, pp. 1–5. ACM Press, New York (2011)
15. Heikkila, V.T., Damian, D., Lassenius, C., Paasivaara, M.: A mapping study on requirements engineering in agile software development. In: 2015 Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 199–207 (2015)
16. Dalkey, N., Helmer, O.: An experimental application of the DELPHI method to the use of experts. Manage. Sci. **9**, 458–467 (1963)
17. Diamond, I.R., Grant, R.C., Feldman, B.M., Pencharz, P.B., Ling, S.C., Moore, A.M., Wales, P.W.: Defining consensus: a systematic review recommends methodologic criteria for reporting of Delphi studies. J. Clin. Epidemiol. **67**, 401–409 (2014)
18. Linstone, H.A., Turoff, M.: The Delphi Method - Techniques and Applications (2002)
19. Dalkey, N.: An experimental study of group opinion. Futures **1**, 408–426 (1969)
20. Finstad, K.: Response interpolation and scale sensitivity: evidence against 5-point scales. J. Usability Stud. **5**, 104–110 (2010)
21. Schön, E.-M., Winter, D., Thomaschewski, J., Escalona, M.J.: Results of "Challenges in Agile Requirements Engineering" (Round 1) (2017). doi:10.13140/RG.2.2.34571.28961

22. Schön, E.-M., Winter, D., Thomaschewski, J., Escalona, M.J.: Results of "Challenges in Agile Requirements Engineering" (Round 2) (2017). doi:10.13140/RG.2.2.32893.56802
23. Schön, E.-M., Winter, D., Thomaschewski, J., Escalona, M.J.: Results of "Challenges in Agile Requirements Engineering" (Round 3) (2017). doi:10.13140/RG.2.2.16116.35201
24. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development. http://www.agilemanifesto.org/
25. Schwaber, K., Sutherland, J.: Scrum Guide. http://www.scrumguides.org/scrum-guide.html

Eeny, Meeny, Miny, Mo...

A Multiple Case Study on Selecting a Technique for User-Interaction Data Collecting

Sampo Suonsyrjä(✉)

Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
sampo.suonsyrja@tut.fi

Abstract. Today, software teams can deploy new software versions to users at an increasing speed – even continuously. Although this has enabled faster responding to changing customer needs than ever before, the speed of automated customer feedback gathering has not yet blossomed out at the same level. For these purposes, the automated collecting of quantitative data about how users interact with systems can provide software teams with an interesting alternative. When starting such a process, however, teams are faced immediately with difficult decision making: What kind of technique should be used for collecting user-interaction data? In this paper, we describe the reasons for choosing specific collecting techniques in three cases and refine a previously designed selection framework based on their data. The study is a part of on-going design science research and was conducted using case study methods. A few distinct criteria which practitioners valued the most arose from the results.

Keywords: Agile software development · User-interaction data · Multiple case study · Software data collecting

1 Introduction

In the last few years, the world has witnessed a tremendous progress in the ways software is developed with. On one hand, this has already benefited both customers and vendors by improving productivity, product quality, and customer satisfaction [1]. On the other hand, the acceleration of release velocity has been such a strong focus point, that the evolution of the means of understanding user wants and needs could not have kept up the pace. For example, Mäkinen et al. [2] describe that customer data analytics are still used sparingly. Similarly, research related to the techniques of automatic collecting of post-deployment data and its use to support decisions still seems to be in its infancy [3]. This feels partly unfortunate, because agile software development has always had the intention of faster responding to changing customer requirements – and to achieve this, both rapid releasing and rapid understanding of customers are needed.

Addressing this, one of the promising solutions is to track users in the user-interface level, then analyze that data to understand how they use the software,

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 52–67, 2017.
DOI: 10.1007/978-3-319-57633-6_4

and finally make decisions based on the analysis [4]. To start such a process, the first thing to do is to select a collecting technique that is suitable for the case. There are many restrictions to this, however, and these make the selecting a rather problematic task. Therefore, guidelines for evaluating and selecting a suitable collecting technique are needed. In our previous work [5], we have designed such a selection framework, which should serve as a guideline and help practitioners in these tasks. The objective of this study is to evaluate and refine that selection framework.

In this paper, we describe the reasons for choosing specific collecting techniques in three different case contexts and evaluate and refine the previously presented selection framework based on their data. The study is a part of ongoing design science research in which we have already designed the selection framework. This part uses the case study method to evaluate and refine the previous design and explore its contexts. Specifically, we address the research question:

– **What reasons software teams have for selecting a specific technique for user-interaction data collecting?**

To answer this overarching research question, we have derived two sub-questions. Firstly, the process of choosing a collecting technology will be explained. Secondly, we try to find out if some of the criteria we presented in our previous work are more significant than others or if there are completely other and more relevant reasons for choosing the technologies. The sub-questions for the study are declared as follows:

1. **How were the collecting techniques selected in each case?**
2. **What kind of criteria for choosing a certain technique were the most significant in each case?**

The rest of the paper is structured as follows. In Sect. 2, we present the background of the study, namely the selection framework which consists of selection criteria and a process. In Sect. 3, we explain how and why we used case study methods and describe the cases involved. In Sect. 4, we describe the process and criteria for choosing a specific technique for user-interaction data collecting in each case. In Sect. 5, we discuss those results to evaluate and refine the selection framework and in Sect. 6 we present the final conclusions of the study.

2 Background

To the best of our knowledge, related work for selecting techniques for user-interaction data is very limited. For example, a recently published systematic mapping study by Rodriguez et al. [6] identified the analysis of why certain technologies for monitoring post-deployment user behavior are selected over other similar existing solutions as a concrete opportunity for future work. However as a background for this study, we revisit the basics of the previously designed selection framework for user-interaction data collecting techniques.

The selection framework forms the basis for this study, as our goal is to evaluate the framework and refine it where necessary. It consists of a set of selection criteria and a process for the selecting. In addition, we introduce different techniques for user-interaction data collecting. These techniques and their evaluations are presented in a more detailed manner in [5]. They are mentioned nonetheless here for an overlook to the different alternatives that software teams have when they start collecting user-interaction data and for demonstrating the criteria part of the selection framework.

2.1 Selection Framework for a Collecting Technique

Criteria. The selection framework guides software teams to evaluate user-interaction data collecting techniques in terms of the technique's *timeliness, targets, effort level, overhead, sources, configurability, security, and reuse.* In the following list, each criterion is described by demonstrative questions which could be asked as a team evaluates collecting techniques.

- *Timeliness.* When can the data be available? Does it have a support for real-time?
- *Targets.* Who should benefit from the data? What is the intended use? Does it support many targets? Does it produce different types of data?
- *Effort level.* What kind of a work effort is needed from the developers to implement the technique?
- *Overhead.* How does it affect performance, e.g. system response time to user-interactions?
- *Sources.* Does it support many source platforms?
- *Configurability.* Can the collecting be switched on and off easily? Can it change between different types of data to collect?
- *Security.* Can the organization who developed the collecting technology be trusted with the collected data? Is the data automatically stored by the same organization?
- *Reuse.* Is the collecting always a one-time solution or can it be reused easily?

Process. The first thing to do when selecting a technique for user-interaction data collecting, is to rapidly **explore the case** to get a grasp of the most critical technical limitations. These include things such as the size of the code base, availability of automated tools and AOP libraries for the target application's language and platform, and access to the UI libraries and execution environments.

If any critical limitations are faced, the next step is to **reject the unsuitable techniques** accordingly. For example, if there are many security issues related to the data being collected or if data needs to be sent in real-time, collecting techniques using 3rd party tools might have critical limitations that cannot be avoided resulting in the rejection of the technique.

The following step is to **prioritize the evaluation criteria**. In addition to the explored case information, one should find out the goals different stakeholders

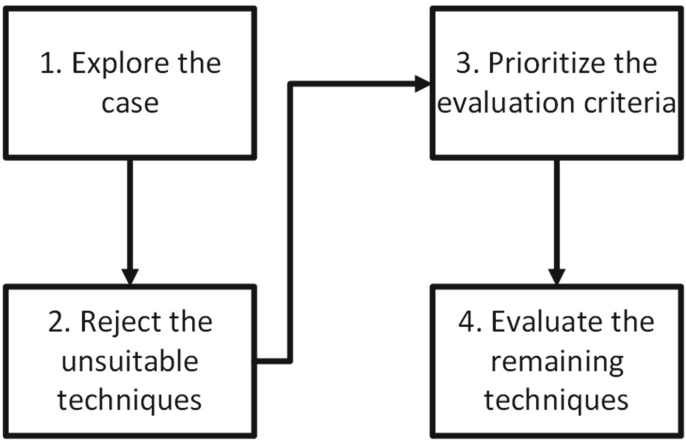

Fig. 1. Selection framework for user-interaction data collecting techniques.

have for the usage data collecting as these can have a major impact on the approach selection. If the goals are clearly stated, and the aim is e.g. to simply find out which of two buttons is used the most, manual instrumentation can work sufficiently. However, if the goal is stated anything like "to get an overall view of how the system is used" or if the goal is not stated at all, the more automated and more configurable approaches most likely become more appealing. Therefore, one of the most crucial things to find out in this step is to understand what different stakeholders want to accomplish with the collected data.

After this, the final step is to **evaluate the remaining approaches**. The plus and minus signs used in Table 1 work as guidelines in this, but their emphasis obviously varies on a case to case basis. To summarize, the selection framework is illustrated in Fig. 1.

2.2 Techniques for User-Interaction Data Collecting

Firstly, in **manual instrumentation** (Manual) developer adds extra statements to the relevant locations of the software. On one hand, this highlights the flexibility of the technique but on the other, adoption to new targets and sources would require significant rework making reuse practically impossible.

Secondly, there are multiple **tools for automated instrumentation** (Tools) of the code, e.g. GEMS [7], for various data logging, quality assurance and performance monitoring purposes. This technique frees the programmers from the manual work and reduces the probability for errors lowering the effort significantly.

Thirdly in between of the above techniques, **aspect-oriented programming approach** (AOP) is something of a mixture from the two. The research presented e.g. in [8,9] use aspect-oriented programming as a tool for code instrumentation. Aspect-based instrumentation allows the instrumentation to be

Table 1. Summary of the technique evaluations.

Criteria	Techniques				
	Manual	Tools	AOP	UI Lib.	E.E.
Timeliness	+	−	+	+	−
Targets	+	−	+	−	−
Effort	−	+	+	+	+
Overhead	+	−	+	−	−
Sources	+	−	−	−	−
Configurability	+	−	+	+	−
Security	+	−	+	+	−
Reuse	−	+	−	−	+

+ = Supports selecting
− = Technique has limitations

system and application specific, which focuses the collecting better on the relevant targets.

Fourthly, **an alternative implementation of a user-interface library** (UI Lib.) can be set to automatically collect user-interaction data. Because user-interaction is usually implemented with standard UI libraries, their components can be altered so that they include the collection of user-interaction data within them. Finally, the data collection can also be integrated into the environment without modifications to the original application. For languages like Java and JavaScript the virtual machine is an **execution environment** (E.E.) where method and function calls can be monitored by instrumenting critical places.

We have summarized the evaluations of different collecting techniques for the basis of the selection framework, i.e. Table 1, giving each technique either a plus if it has a positive impact or if it does not have restrictions in terms of the criterion. A technique is marked with a minus sign if it limits the selection or the use of a data collecting implementation according to a criterion.

3 Research Approach

The study was conducted using case study methodology. It allowed us to explore and describe the case specific situations and their circumstances related to the selection framework from deeper and more insightful viewpoints than if a research method with set variables had been used. Case study investigates contemporary phenomena in their real-life context [10], and this suited the purposes of the study well. The study is a part of on-going research effort, where we design, evaluate, and diffuse the selection framework by design science guidelines presented in [11] and with the process presented in [12]. The design science method of the underlying research effort affected this study as well especially in how actively the researchers had to take part in the cases. This participation was

obviously required because the automated collecting of user-interaction data and its use for software development was still an unknown area for each of the case organizations. Moreover, the researchers had a substantial expertise considering the designed selection framework.

3.1 Explanatory Case Study

The selection framework, as presented in [5], includes predetermined criteria for evaluating the collecting technologies. These criteria could have been used straightforwardly as variables of a study with more experimental setting. However, the criteria have been derived from a literature survey and from only one case study. Therefore, we acknowledge that there can be other criteria that affect the selection as well, and perhaps with a greater impact. To allow the inclusion of these other possible factors into the selection framework, we have chosen to use specifically *multiple* case study method and gather data from three different cases.

This study uses *explanatory* case study methodology because its aim is at finding the reasons why software teams choose a specific collecting technology. The results of this explanatory case study are used for evaluating and refining the designed selection framework where necessary. Runeson & Höst [13] have categorized case studies by their purposes into exploratory, descriptive, explanatory, and improving. Since explanatory case studies are "...seeking an explanation of a situation or a problem, mostly but not necessary in the form of a causal relationship", their aims are well-suited for the study.

Case Selection. Given the purpose of the studied selection framework, its potential users are mainly software teams that are only beginning to collect user-interaction data. This limited the potential cases for this study to software teams that had not yet selected a technique for user-interaction data collecting but were still willing to try such collecting out. Clearly, the selected cases had to be open enough that publishing the results reliably was possible and also accessible in the first place for the first author to do the research with them.

Similarly, the number of the cases selected for the study was affected by the fact that the first author had to spend considerable effort in each case. As suitable software teams for this study had not tried out user-interaction data collecting or explored its techniques, the first author had to have access to a potential software team to tell about the possibilities of such data collecting and initiate these tasks. All of these limited the number of selectable cases to few, and finally three software teams were selected for the study.

Data Collection. The data was gathered from February to December 2016. Main parts of the data consist of meeting notes written down by the first author of this paper. Workshop type of meetings were held in each of the cases. Since collecting user-interaction data was a novelty for each participating software team, simple interviewing would not have worked. Rather, the meetings were organized as workshops where the first author motivated the software team to

try out user-interaction data collecting and described the different possible techniques for doing so. In addition to the data gathered in meetings, the first author had designated work desks in the same rooms where the software teams were working in cases A and B. Therefore, data was also gathered by observation and by participating in informal meetings. However, these data were only used for verifying some of the previously collected meeting note data, such as how many standup meetings a team have in a week. Although these observational data were not collected in a formal fashion, for the first author it improves the reliability of the results in terms of data triangulation.

Validity and Reliability Considerations. Although this study tries to investigate what kind of things have an effect on the decisions of software teams, the aim is not to find definitive proofs or certain amounts of statistical significance in these relations – rather to broaden the scope of possible causes. Therefore, the internal validity needs especially careful considering. Firstly, selections in earlier cases can have had effects on later ones. This was obviously not intentional but still surely possible because the same researcher explained the different options for the teams in each case. However, the author of this paper separated himself from the decision making in each of the cases and the decisions were made only by the software teams.

Secondly, the criteria presented with the selection framework can have guided the author of this paper to identify only those as the reasons for selection. Consequently, there can have been reasons that have not been mentioned aloud in the meetings but which still have had an effect on the decision. For example, a technology might have been seen as an unsuitable option in such an indisputable manner that the software team has not even mentioned it. This risk was mitigated in cases A and B by not only gathering data from meetings, but also by observing the working of the teams in the their offices and participating in their informal meetings.

The results of this study will not be generalizable for any software team. However, they provide a detailed look on the reasons these three software teams had for choosing a user-interaction data collecting technique. The three case organizations are different from each other in many ways, and therefore the results can give interesting insights to a wide audience. Although only one researcher gathered the data in each case, the meeting notes were shown to and accepted by team members in each case.

3.2 Case Organizations

Case A. Organization A is a large international telecommunications company. The software team that was involved in this case consisted of around eight members. The border of one team in this organization is quite flexible as employees work for many products. The team members had titles of software architect, UX designer, software developer, and line manager. Their products consist primarily of software in the field of network management, and these range from Java software

to web based systems. The software development method used in their team has some properties from agile development methods such as Scrum. They, for example, have bi-daily standup meetings and they use Kanban boards to organize their work. New versions of their product are released usually a few times a year.

Case B. Organization B is a privately held software company in Finland. At the time of the study, they had around 300 employees and offices in three major cities in Finland and they primarily develop software in projects for their customers as ordered. The software team involved in this case, however, develops their own software-as-a-service solution. As in case A, the software team in case B also uses things such as daily standup meetings, Kanban boards and retrospective sessions familiar from some of the agile development practices. On the contrary however, they are releasing new versions of their product to the end-users far more often – usually biweekly. Their software team consists of seven members with titles such as product owner, UX specialist, software architect, and software developer.

Case C. Organization C is a research and education center of around 10000 students and 2000 employees. The case C software team is part of a research group who have specialized in embedded systems design. They have developed Kactus2, which is an "open source IP-XACT-based tool for ASIC, FPGA and embedded systems design"[1]. The software has created traction from users world wide. It has been downloaded around 5500 times during the last year requests coming mainly from the USA and from middle Europe. The development team consists of four employees with the titles of software developer, software architect and business architect. The developed tool itself is an installable software system and installer packages for Windows and Linux tar-packages of its new versions are released three to four times a year.

4 Results

The results of the study are twofold. Firstly, we describe the processes with which the techniques were selected in each case. Secondly, we dive into the reasons the software teams had for their selection.

4.1 The Processes of Choosing a Collecting Technology

Case A. In February 2016, members of the software team of Case A explained to the researcher that they had an overall interest in trying out the use of user-interaction data for the further development of their software products. The researcher had presented the different technological approaches for collecting such data in a previous informal meeting. These were the same approaches as described in [5]. Two of the software team's products had been then analyzed by

[1] http://funbase.cs.tut.fi/.

the Organization A in terms of the suitability of the products in experimenting with user-interaction data collecting. The first of the two was Tool X written in Java, and the second one a JavaScript based Web-system Y. The team decided to carry on the collecting efforts with the System Y.

After this decision, the team had a meeting with the researcher to give a short presentation about the code base of the System Y and its software architecture. The meeting was arranged as a workshop to find out what kind of user-interaction data the team wanted to have collected. In addition, the team described what is important for the collecting technology and its implementation.

From this point on, the job of the researcher in the eyes of the software team was to develop a demonstrative collecting tool for their product. The researcher then used the criteria from the selection framework and was left with only one suitable technology approach – developing a new tool for **monitoring the execution environment**. After developing a prototype of such a collecting tool, the researcher presented it in a demo show for the team in March and got a thumbs up from the team to go on with experimenting with the actual System Y. A testing day with eight users from within the Organization A was held in December 2016 to try out an improved version of the collecting tool implemented in a lab version of the System Y. The developed collecting tool is available in GitHub[2].

Case B. In case B, a similar workshop meeting as in Case A was held by the researcher with the software team in March 2016. The team explained the method they use for developing their software and what kind of a software the product is architecturally. It turned out, however, that this team had more experiences with collecting use related data even at that point. For example, they had tried out Google Analytics with some default settings for their product already. After explaining that the data was mainly collected for debugging, two of the team members and the researcher worked out also new targets in their software development process which could be improved with user-interaction data. These ideas ranged from prioritizing their product backlog to improvements in the user interface of the product.

The team was well motivated to try out user-interaction data collecting. However, as its return on investment was still unclear the first few tasks for data collecting were agreed upon to be completed with as little work effort and changes to the software architecture as possible. Therefore, three very specifically described places in the UI of the product were selected to be improved with the help of user-interaction data collecting. As the team had already tried out Google Analytics on the same product, it was a straightforward choice for the storing and analyzing the data of the tasks at hand as well.

At that point, the researcher described the same technological approaches to the team as in Case A. Also similar to Case A, the selecting of the collecting technology was an obvious pick since the three tasks were specified so explicitly. The team members and the researcher made an unanimous decision to use **manual implementation** for instrumenting the required places of the source code.

[2] https://github.com/ssuonsyrja/Usage-Data-Collector.

The researcher was then given rights to change the source code. After applying the collecting code to six places in it, the version was sent to end-users for a two week collecting period in April 2016.

Case C. In case C, an initial meeting was held with two members of the software team and the researcher in September 2016. Similar to the previous cases, the team members described the environment for which they develop software and the architecture of their product. The meeting then continued as a workshop, where each participant tried to figure out ways for how user-interaction data collecting could be used for their software development. Such targets were plenty, and no specific tasks were selected at that point. The researcher then explained the same technological approaches for user-interaction data collecting to the team members. The option of monitoring execution environment was rejected at this point, but the rest still remained possible for selecting.

The evaluation criteria from the selection framework were then used for the analysis of the product and its environment. Since the aspect-oriented approach raised the most interest among the software team, it was decided that the availability of AOP libraries and their suitability to the product were to be examined. An alternative implementation of a UI library was considered as a second choice, but the rest of the alternatives were rejected at this point. During the fall of 2016, the **aspect-oriented approach** was implemented technically successfully to the product. The first data collecting period is planned to be held during the spring of 2017 with a student group as experimental end-users.

4.2 Reasons for Choosing a Collecting Technique

Case A. The first decision made by the Organization A was that they selected to try out user-interaction data collecting with System Y. This decision was based on **the sources and the reuse possibilities** of the collecting effort, because the motivation was to specifically try out this kind of data collecting as a technical concept rather than immediately produce actionable insights from exact places of a product. Had the collecting effort been carried out with the Tool X, the reuse would have been practically impossible since its environment was not as common as with the System Y.

Although the overall motivation was to test user-interaction data collecting conceptually, the team wanted to focus the requirements of the data collecting after the selection of the specific **source**, i.e. product. Finding a technology that could be easily reused with as little implementation **effort** as possible became a goal. This made the option of manual instrumentation heavily unfavorable. The team also emphasized how **the security and configurability** were important for the collecting technology. For example, the environment of their product was such that the collecting should be easy to be left out of the whole product when necessary. Consequently, the unobtrusiveness of the technology was highly valued.

Although the need for low configuring effort increased the attractiveness of using an automated tool for instrumentation, the security concerns were so heavy

that the use of a tool developed outside the organization was not recommended. Therefore, the option of finding and using 3rd party tools was quickly rejected. In addition, the availability and effects of AOP libraries to things such as the **overhead** were unknown in the environment of System Y. Possibly the most significant of all, there was no motivation to make as big **a change to the software architecture** as needed by the aspect-oriented approach. The same reason applied for rejecting the option of an alternative UI library, because having different versions of the libraries was not acceptable for the delivery pipeline.

Case B. Similar to case A, the motivation for the team of case B in user-interaction data collecting was to try it out as a concept. On the contrary however, this resulted in this case in a faster and a narrower scoped experiment. In other words, the **targets** and the **source** of their data collecting were very clearly defined in the first place. At the same time, this resulted in the lack of significance of the implementation **effort** because it would be so low even with the manual approach. Similarly, **reuse** was not considered as a significant reason, since there were no guarantees that the data collecting mechanism would be ever reused. All this resulted in a very straightforward choice of the manual approach. It was by far the easiest approach to implement on a small scale and it allowed the team to try out if user-interaction data collecting in a fast and low-effort way.

Case C. Being a new thing for the case C software team, the user-interaction data collecting was again designed as a demonstrative experiment similar to the case A. Likewise, the interests of the team in this case were technical in the sense that they firstly wanted to find out a suitable technique for user-interaction data collecting. In the best case scenario, this technique could be then used with their actual product and actual end-users after the initial experiment. Because there was no simple access to experiment the collecting with real users, in the manner of case B, and the **security** requirements were weighted a lot heavier, the technical design of the collecting was the primary focus. Although the possible user-interaction data types and collection places, i.e. **sources**, were plenty, they were to be considered only secondly after validating the technical setup for the collecting.

This affected the evaluation of the collecting techniques in terms of prioritizing the criteria from the selection framework. Not limiting the **sources and targets** became important, because the collecting technique would not be selected and designed for just a one time try out. Although not mentioned out loud by the team, this could hint towards them valuing the **reuse** possibilities. All of these resulted in the attractiveness of the techniques enabling lower work **effort** spend on each distinct collecting place. Further on, the whole collecting was required to be able to be switched off as easily as possible. In other words, the **configurability** of the collecting technique was valued high.

5 Discussion

In each case, the process of choosing a collecting technology for user-interaction data was more or less the same. Members of the software team and the researcher had a meeting, where the researcher described the different technologies overall. After finding out what was the underlying goal for the team in the user-interaction data collecting, the most important criteria for the selecting became quite clear for both the researcher and the team members.

Comparing those criteria with the ones in the selecting framework, it is safe to say that most of the evaluation criteria from the selection framework were used without the researcher pushing the team towards those specific points. However, **timeliness** was never mentioned by the teams, which could signal either its insignificance or that its need is self-evident. On the contrary, **overhead** rose up in each case as a conversation topic but similar to the timeliness it did not seem to have any effect on the selecting in any case.

For both of these, it is worth mentioning that none of the techniques had a known disadvantage nor a limitation in terms of these criteria (timeliness and overhead) that would have been significant enough to get the whole technique rejected. However, in the original selection framework they were marked with minus signs for the monitoring execution environment technique. Therefore, the summary table with the evaluation criteria from the original selection framework, i.e. Table 1, requires some refining.

Firstly, the evaluations should consist of a wider scale than a plain plus or a minus sign. In these cases, some of the criteria affected the selection clearly a lot more than others. For example, the timeliness and overhead criteria did not seem to have an effect on the selection but on the other hand, the effort level of the manual technique had it rejected. Therefore, we propose an additional exclamation mark to the evaluations in case the criterion is a possible ground for a rejection. We have gone through the rest of the summary evaluations and added an exclamation mark where necessary based on the cases.

Secondly, some of the evaluations are not clearly pluses nor minuses. Therefore, we have added an option of $+/-$ marking for the evaluation, if the technique does not definitely support nor limit the selection in terms of the specific criterion. Adding this option has had effects especially on the evaluations of the techniques that are heavily intertwined with specific tools. For example, the minus signs in the execution environment column of timeliness and overhead rows can be then replaced with this option. We have reviewed the evaluations and changed the original signs into $+/-$ markings where necessary.

Thirdly, the *effort* criterion should be divided into two and renamed to *scalability*. The intention of the criterion is to depict the work effort that is required from the software developers to implement collecting snippets to the different places of the source code. Finally, however, there was a clear need for an evaluation criterion of how great an effort is needed from the software developers to change the software architecture and/or environment of the moment to support the collecting technique. This criterion could be named as the *change* that is

Table 2. Refined summary of the technique evaluations.

Criteria	Techniques				
	Manual	Tools	AOP	UI Lib.	E.E.
Timeliness	+	+/−	+	+	+/−
Targets	+	−	+!	−	+/−
Scalability	−!	+	+!	+	+!
Overhead	+	−	+/−	−	−
Sources	+	−	−	−	−
Configurability	+	−	+	+	+/−!
Security	+	+/−!	+/−	+	+/−
Reuse	−	+	−	−	+!
Change	+!	+	−!	−!	+

+ = Supports selecting
− = Technique has limitations
+/− = No clear support nor limitations
! = A possible ground the rejection

required. With these refinements to the criteria and evaluations, the summary table of the evaluations is as listed in Table 2.

In addition to the changes in the evaluations, the original selection framework requires some refinements based on the cases as well. First of all, in these cases the underlying goal of the whole collecting effort was the most important driver in the selection process. In cases A and C the delivery pipelines did not allow fast and flexible releases of new software versions with user-interaction data collecting capabilities, and so the software teams decided to develop their environment so that the collecting would be possible in the future. This became their real target, where as the team in case B did not have to develop their environment. On the contrary, they had the luxury of aiming straightforwardly at just testing out the collecting and the resulting user-interaction data with a minimum effort.

Therefore, the first step of the selection framework, *exploring the case*, should be clarified and replaced by a step of *defining a main goal* for the collecting effort. Based on these cases, it would be easy to then *remove the irrelevant evaluation criteria* after defining such a goal. For example, in case B the *scalability* of the collecting technique was seen unnecessary after the collecting was designed to be implemented as a one time solution.

Exploring the case still included important things that should be part of the selecting framework. Thus, the next thing of the process should be to *find out the critical limitations*. The rest of the original selection framework worked out as it was in these cases, and so no other changes were required to the final refined version of the selection framework. This framework is illustrated in Fig. 2.

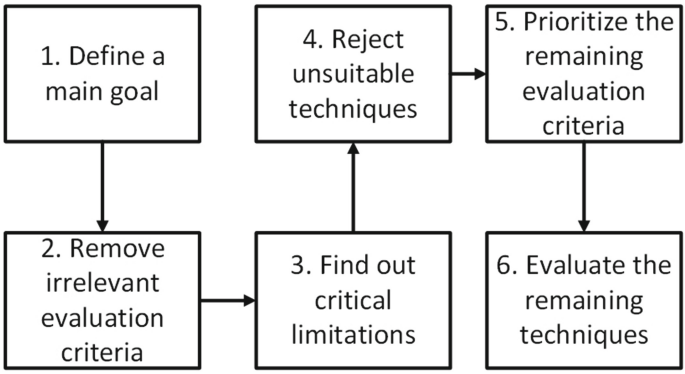

Fig. 2. Refined selection framework for user-interaction collecting techniques.

6 Conclusions

In this paper, we studied three cases where software teams selected techniques for user-interaction data collecting. More specifically, we examined the reasons the software teams had for the selection. To complement this, we evaluated our previously designed selection framework and refined it based on the data gathered from the cases.

In these cases, two of the most valued criteria for the selection were the scalability of the technique and the lack of changes required to the software architecture and deployment pipeline of the moment. Additionally, teams appreciated the reuse, security, and configurability of the techniques as well as the support for a wide range of monitoring targets. On the other hand, the rest of the criteria presented with the original selection framework, i.e. timeliness, overhead, and support for different source applications, did not seem to have a significant effect on the selections.

The original evaluations of the different user-interaction data collecting techniques were refined to include markings for the different levels of significance. In addition, the original selection framework was fixed to better support these more detailed evaluations. With these changes, we think the selection framework and its complementary technique evaluations can help practitioners greatly to the beginning of their journey of user-interaction data collecting.

Acknowledgments. The authors wish to thank DIMECC's Need4Speed program (http://www.n4s.fi/) funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes/) for its support for this research.

References

1. Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V.P., Itkonen, J., Mäntylä, M.V., Männistö, T.: The highways and country roads to continuous deployment. IEEE Softw. **32**(2), 64–72 (2015)

2. Mäkinen, S., Leppänen, M., Kilamo, T., Mattila, A.L., Laukkanen, E., Pagels, M., Männistö, T.: Improving the delivery cycle: a multiple-case study of the toolchains in finnish software intensive enterprises. Inf. Softw. Technol. **80**, 175–194 (2016)
3. Fabijan, A., Olsson, H.H., Bosch, J.: Customer feedback and data collection techniques in software R&D: a literature review. In: Fernandes, J., Machado, R., Wnuk, K. (eds.) ICSOB 2015. LNBIP, vol. 210, pp. 139–153. Springer, Cham (2015). doi:10.1007/978-3-319-19593-3_12
4. Suonsyrjä, S., Mikkonen, T.: Designing an unobtrusive analytics framework for monitoring Java applications. In: Kobyliński, A., Czarnacka-Chrobot, B., Świerczek, J. (eds.) IWSM/Mensura -2015. LNBIP, vol. 230, pp. 160–175. Springer, Cham (2015). doi:10.1007/978-3-319-24285-9_11
5. Suonsyrjä, S., Systä, K., Mikkonen, T., Terho, H.: Collecting usage data for software development: selection framework for technological approaches. In: Proceedings of The Twenty-Eighth International Conference on Software Engineering and Knowledge Engineering (SEKE 2016) (2016)
6. Rodriguez, P., Haghighatkhah, A., Lwakatare, L.E., Teppola, S., Suomalainen, T., Eskeli, J., Karvonen, T., Kuvaja, P., Verner, J.M., Oivo, M.: Continuous deployment of software intensive products and services: a systematic mapping study. J. Syst. Softw. **123**, 263–291 (2017)
7. Chittimalli, P.K., Shah, V.: GEMS: a generic model based source code instrumentation framework. In: Proceedings of the Fifth IEEE International Conference on Software Testing, Verification and Validation, pp. 909–914. IEEE Computer Society (2012)
8. Chen, W., Wassyng, A., Maibaum, T.: Combining static and dynamic impact analysis for large-scale enterprise systems. In: Jedlitschka, A., Kuvaja, P., Kuhrmann, M., Männistö, T., Münch, J., Raatikainen, M. (eds.) PROFES 2014. LNCS, vol. 8892, pp. 224–238. Springer, Cham (2014). doi:10.1007/978-3-319-13835-0_16
9. Chawla, A., Orso, A.: A generic instrumentation framework for collecting dynamic information. SIGSOFT Softw. Eng. Notes **29**(5), 1–4 (2004)
10. Yin, R.K.: Case Study Research: Design and Methods. Sage Publications, Thousand Oaks (2013)
11. Von Alan, R.H., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
12. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A design science research methodology for information systems research. J. Manage. Inf. Syst. **24**(3), 45–77 (2007)
13. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Softw. Eng. **14**(2), 131 (2008)

Comparing Requirements Decomposition Within the Scrum, Scrum with Kanban, XP, and Banana Development Processes

Davide Taibi[1(✉)], Valentina Lenarduzzi[1], Andrea Janes[1], Kari Liukkunen[2], and Muhammad Ovais Ahmad[2]

[1] Free University of Bolzano/Bozen, Bolzano, Italy
{davide.taibi,valentina.lenarduzzi,andrea.janes}@unibz.it
[2] University of Oulu, Oulu, Finland
{kari.liukkunen,muhammad.ahmad}@oulu.fi

Abstract. **Context:** Eliciting requirements from customers is a complex task. In Agile processes, the customer talks directly with the development team and often reports requirements in an unstructured way. The requirements elicitation process is up to the developers, who split it into user stories by means of different techniques. **Objective:** We aim to compare the requirements decomposition process of an unstructured process and three Agile processes, namely XP, Scrum, and Scrum with Kanban. **Method:** We conducted a multiple case study with a replication design, based on the project idea of an entrepreneur, a designer with no experience in software development. Four teams developed the project independently, using four different development processes. The requirements were elicited by the teams from the entrepreneur, who acted as product owner and was available to talk with the four groups during the project. **Results:** The teams decomposed the requirements using different techniques, based on the selected development process. **Conclusion:** Scrum with Kanban and XP resulted in the most effective processes from different points of view. Unexpectedly, decomposition techniques commonly adopted in traditional processes are still used in Agile processes, which may reduce project agility and performance. Therefore, we believe that decomposition techniques need to be addressed to a greater extent, both from the practitioners' and the research points of view.

1 Introduction

Eliciting requirements from customers is a complex task. In Agile processes, the introduction of the product owner usually facilitates the process, suggesting that the customer talk directly with the development team and thus reducing the number of intermediaries. However, the product owner, especially when he or she is not an expert in the project domain, reports requirements in natural language, in their own words, and often in an unstructured way.

The requirements elicitation process is up to the developers, who usually split it up into user stories in the case of Agile processes.

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 68–83, 2017.
DOI: 10.1007/978-3-319-57633-6_5

To the best of our knowledge, there are no studies that have attempted to understand how requirements are decomposed in Agile processes and, moreover, no studies that compare requirements decomposition among different Agile processes or other processes.

To bridge this gap, we designed and conducted the first such empirical study, with the aim of comparing the requirements decomposition process of an unstructured process and three Agile processes, namely XP, Scrum, and Scrum with Kanban [21]. We conducted the study as a multiple case study with a replication design [1] since it was not possible to execute a controlled experiment because of the unavailability of developers for the major effort required. We selected four groups of second-year master students as participants, which constitute a good sample of the next generation of developers entering the job market. They were perfectly suited for this task since the project did not require the use of new technologies unknown to the students, and they can thus be viewed as the next generation of professionals [10–13]. Students are perfectly suitable when the study does not require a steep learning curve for using new technology [13, 17].

We selected a project idea to be developed by means of an idea contest for entrepreneurs, selecting an idea from a designer with no experience in software development. This project idea was then developed by four teams using four different development processes. The requirements were elicited by the teams from the same entrepreneur who acted as product owner with all four groups.

The results show interesting differences regarding requirements decomposition. The team that developed in XP decomposed a lot more stories, followed by the one using Scrum with Kanban, then the one using Scrum, and finally the team using the unstructured process. Another interesting result is related to the development effort, which was perfectly inversely proportional to the number of user stories decomposed, resulting in the highest effort for the unstructured process and the lowest for the XP one.

This paper is structured as follows. Section 2 introduces the background and related work. Section 3 presents the multiple case study and Sect. 4 the results obtained. Section 5 describes the threats to validity and Sect. 6 draws conclusions and future work.

2 Background and Related Work

The term “user story decomposition” describes the act of breaking a user story down into smaller parts [8]. User stories are typically decomposed into parts that have a scope that is large enough to provide value to the customer but small enough so that the effort for implementing the story can be estimated with a low risk of being wrong. A story with a smaller scope is likely to be less complex than a story with a large scope. Moreover, if the scope is large, more things can go wrong, e.g., unknown details might emerge, the architecture may be inadequate, and so on [4]. Altogether, the expectation is that it should be easier to estimate the effort for developing a small story than that for a large one. As a consequence, sprint planning, i.e., defining which stories the team should be able to complete during a sprint, is more likely to be accurate with small user stories.

Additionally, developing stories with a smaller scope allows the team to complete a user story more often than if it were to develop only a few large user stories. This allows

it to regularly deliver business value to the customer, with the consequence that the customer can provide feedback earlier, allowing the team to learn faster which requirements the system being developed should fulfill.

A popular technique for decomposing user stories is "User Story Mapping" [9], which decomposes the stories from the user's point of view, i.e., it decomposes the flow of user activities "into a workflow that can be further decomposed into a set of detailed tasks" [8]. User Story Mapping uses the terms "activity", "task", and "subtask" to describe high-level activities (e.g., "buy a product"), tasks (e.g., "manage shopping cart"), and subtasks, i.e., the decomposed user stories, which are the ones assigned to developers (e.g., "add product to shopping cart"). Rubin uses the terms "epic", "theme", and "sprintable story" to apply it within Scrum [8].

Outside of an Agile context, the decomposition of requirements into different parts has been discussed to prepare their technical implementation: for example, [14] describes techniques used in service-based applications to decompose complex requirements in order to reuse relatively simple services; in [15], the authors develop a technique for matching parts of the requirements to COTS components; in [16], the authors discuss how to decompose architectural requirements to support software deployment in the cloud; in [17, 20], the authors study conflicting requirements; in [18], the authors propose an extension to UML to allow decomposing use case models into models at several levels of abstraction; and in [19], the authors decompose requirements to identify security-centric requirements.

All these examples rather describe decomposition as an activity to devise a specification that describes the system to be built. Within an Agile context, decomposition is used to reduce the risk of providing a constant flow of value; therefore, user stories are typically decomposed following the principle that each one should deliver value to the customer. To the best of our knowledge, no peer-reviewed works exist that describe decomposition techniques used within an Agile context. However, various other techniques worth mentioning have been developed by practitioners, who describe them on their blogs. In the following, we will describe the approaches they propose.

As a general approach, Lawrence [3] suggests two general rules of thumb: choosing a decomposition strategy that allows deprioritizing or throwing away parts of a story, thus isolating and removing unnecessary smaller parts of a larger user story, and then choosing a strategy that results in equally sized small stories. Verwijs [5] distinguishes between two ways to break down user stories: horizontal and vertical. Horizontal breakdown means dividing user stories by the type of work that is needed or the layers or components that are involved, e.g. separating a large user story into smaller user stories for the UI, the database, the server, etc. He suggests avoiding this type of break-down as user stories will no longer represent units of "working, demonstrable software", as it will be hard to ship them separately to the user, as it increases bottlenecks since developers will tend to specialize in types of user stories, e.g., the "database guy", and as it is hard to prioritize horizontally divided stories. Verwijs suggests breaking down user stories "vertically", i.e., "in such a way that smaller items still result in working, demonstrable, software [5]." Recent works also support Verwijs proposal, suggesting to decompose the user stories incrementally, starting from the minimum viable product [16] and decomposing each functionality vertically, so as to also improve the user stories

effort estimation accuracy [7, 15] and the testing easiness [20]. However, this process is more suitable for projects started from scratch with SCRUM instead of project where SCRUM has been introduced later [14].

As these are specific techniques for decomposing a large user story into smaller ones in an Agile context, we integrated their proposals into the following list:

1. Input options/platform [5]: decompose user stories based on the different UI possibilities, e.g., command line input or a graphical user interface;
2. Study conjunctions and connecting words (like "and") to separate stories [4];
3. Data types or parameters [3, 5]: user stories are split based on the datatypes they return or the parameters they are supposed to handle; for example, during a search process, one could define different user stories for the different search parameters the user is allowed to define;
4. Operations, e.g. CRUD [3, 5]: whenever user stories involve a set of operations, such as CRUD (create, read, update, delete), they are separated into smaller versions implementing each operation separately;
5. Simple/Complex [3, 5]: a complex user story is decomposed into a simple, default variation and additional variations that describe special cases or additional aspects;
6. Major effort [3, 5]: a complex user story is decomposed into smaller ones isolating the difficulty in one user story;
7. Workflow steps [3–5, 8]: the temporal development of the user story is studied by imagining the process that the typical user has to follow to accomplish the user story in order to develop (smaller) step-by-step user stories;
8. Test scenarios/test case [4, 5]: user stories are divided based on the way they will be tested. If they will be tested by first executing a sequence of steps and then executing another sequence of steps, these two groups of steps will be implemented as two separate user stories;
9. Roles [5]: one functionality is formulated as separate user stories describing the same story for different user roles (personas) in each user story;
10. Business rules [3, 5]: user stories are extended by "business rules", i.e., constraints or rules that are defined by the context in which the system has to be developed, e.g., a specific law that has to be fulfilled, and the single constraints and rules are used to formulate more fine-grained user stories;
11. Happy/unhappy flow [5]: separate user stories are created for successful variations and unsuccessful variations of the user story;
12. Browser compatibility [5]: if there is a large effort connected to particular technologies, e.g., a text-based browser, [5] recommends splitting user stories according to browser compatibility. Having separate user stories for different browsers allows the product owner to prioritize the work;
13. Identified acceptance criteria [4, 5]: acceptance criteria are defined for user stories that can be used to develop a refined set of (smaller) user stories;
14. External dependencies [5]: user stories can be separated based on the external systems to which they have access;
15. Usability requirements [5]: user stories are separated based on particular usability requirements, e.g., particular implementations for color-blind users;

16. SEO requirements [5]: user stories are separated based on search-engine-optimization requirements, e.g., separate landing pages for specific keywords;
17. Break out a Spike [3]: a user story that is not well understood is divided into one that develops a prototype, a so-called "spike", and one that implements the actual functionality; and
18. Refinement of generic words (like "manage") into more concrete user stories [4].

3 The Multiple Case Study

As stated in the introduction, the objective of this research is to compare the requirements gathering processes and user story decomposition in Agile and unstructured development processes. Therefore, we designed this study as a multiple case study with a replication design [1].

As depicted in Fig. 1, we first identified the research questions, then selected the case studies and their design.

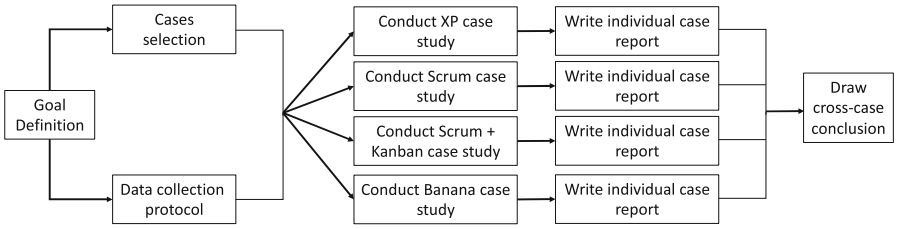

Fig. 1. Study design (adapted from [1])

In this section, we present the study process we adopted, the goal, the research questions, and the metrics for the case study. This is followed by a description of the designs used for the case study, the measurement instruments, and the results obtained.

3.1 Study Goal

According to our research objective, we formulated the goal of the case study following the GQM approach [2] as follows:

Analyze requirements decomposition in user stories (or tasks)
For the purpose of comparison
With respect to the granularity
From the point of view of software developers
In the context of Scrum, Scrum with Kanban, XP, and an ad-hoc development process

Note that in the case of Agile processes, we refer to user stories, whereas in the case of the ad-hoc development process, we refer to tasks. This leads to the following research question:

RQ1: How does the requirements decomposition of user stories differ between the four considered development processes?

For this research question, we defined six metrics:

M1: Number of requirements: the number of requirements provided by the product owner;
M2: Number of user stories: the number of decomposed user stories;
M3: Number of tasks/user stories per requirement: describes how each requirement was decomposed in each team for each requirement;
M4: Total effort (actual development time): time spent (hours) to develop the whole application;
M5: Total effort for each requirement: time spent (hours) to implement requirements among the different teams;
M6: Total effort per task/user story: time spent (hours) to implement each task/user story; and
M7: Strategy used to decompose requirements into tasks or user stories: strategy described in the literature used to decompose each requirement, assigned to two researchers of this paper studying the names of the decomposed requirements.

3.2 Study Design

We designed our study as a multiple-case replication design [1]. We asked four development teams to develop the same application. The four sub-case studies are sufficient as replications since the teams have similar backgrounds. All the teams received the same requirements provided by the same entrepreneur, who acted as product owner and within the same timeframe.

One team was required to develop the project in Scrum, one in Scrum with Kanban, another one using XP, and the last one was free to develop using an ad-hoc process, as shown in Fig. 2. We call the ad-hoc development process the "Banana" process, since this term is used among practitioners to describe processes that produce immature products, which have to "ripen" after being shipped to the customer, like bananas.

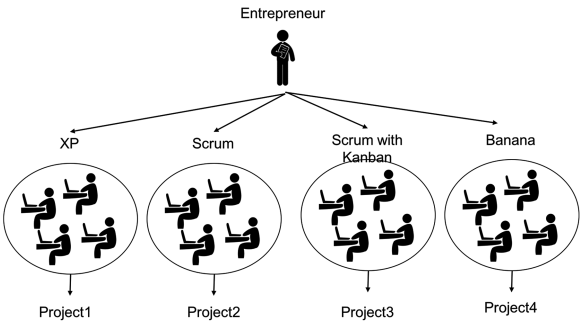

Fig. 2. Study design

As the population of our study, we selected four groups of master students in computer science from two universities involved in the Software Factory Network [6]. One group was from the Master in Computer Science curriculum of the University of Bolzano-Bozen (Italy) and the other four groups were from the Master in Information Processing Science curriculum of the University of Oulu (Finland). The groups had very similar backgrounds since they were all master students in computer science and both universities have similar programs due to their participation in the European Master in Software Engineering (EMSE, http://em-se.eu/) program and having taken classes on agile software development and software engineering. International students took part in this project, originating from Finland, India, Nepal, China, Russia, Bangladesh, Germany, Italy, and Ghana.

The students were randomly assigned to each group taking into account that each team needed to have at least one experienced developer.

The groups were asked to develop the same application. The application requirements were proposed by the product owner, a designer from Bolzano with no experience in software development who described the requirements to the groups with the same schedule and using the same terminology.

The developers elicited the requirements, translating them from the "designer language", a non-technical language, to a more technical one. The groups working with Scrum (with and without Kanban) and XP decomposed the requirements into user stories, while the group using "Banana" decomposed them into tasks.

The developed project. The teams were required to develop an Android application called Serendipity. The idea was selected in a contest for entrepreneurs, where entrepreneurs were asked to submit the minimum viable product [16] description of their project ideas that could be implemented in the software factory lab (http://ideas.inf.unibz.it/). Serendipity is an Android application and a web application intended to share a set of sounds in a specific location, so as to have the user recall special moments by listening to the sounds.

The entrepreneur, a designer from Bolzano, initially defined the project idea as: "Serendipity means "fortunate happenstance" or "pleasant surprise". This project is meant to be an experience that mixes places and sound to enable you to see places you usually go to with new eyes, in a more poetic, more ecstatic way. While taking walk, you will have access to six music tracks, developed from the actual ambient sound of those places themselves. I specifically chose very popular meeting points in my town (Bolzano), where many people go without even realizing anymore what the place looks like. On a map displayed on your smartphone, these locations are highlighted. When you arrive there, you can listen to the soundtrack created to allow you to enjoy the moment. It should be a discovery process. The perk is that this concept is applicable to any city/place – it would be nice to spread it and let the sound go local".

The entrepreneur acted as product owner and described the project to the groups, which elicited the requirements (Req) independently. The requirements were intentionally stated such as to allow vertical break-down [5] decomposition and were proposed to the groups within this timeframe:

Week #0:

Req 1: Minimal Viable Product, with all pages with fake content. The parts of the product comprised: Sign-in/Login; Maps; Listen to sound; Record sound; Rules; and About.
Req 2: Show the list of available sounds on a map.
Req 3: Allow only registered users to record tracks.
Req 4: The main sound can only be played when the user is exactly in the correct location.

Week #3:

Req 5: No more than three sounds allowed within a radius of 300 m.
Req 6: Sounds cannot be downloaded but only played.
Req 7: Any user (registered or not) can listen to sounds.
Req 8: Users are allowed to listen to an ambient sound within a radius of 300 m from the main sound.

Week #5:

Req 9: Play a notification when entering the notification area, so as to alert the user to a sound in the neighborhood.
Req 10: Due to the lack of accuracy of GPS signals in smartphones, the main sound must to be playable within a radius of 10 m instead of only at the exact point, as previously required in Req 6.

Week #7:

Req 11: Create a "liking" system for each sound, allowing users to "like" a maximum of one sound per spot. In this way, sounds with a lower number of likes can be replaced by new sounds after three weeks.
Req 12: Create a web application to allow users to login to their profile with the only purpose of uploading sounds, especially for professional users who would like to upload high-quality or edited sounds.
Req 13: Allow users to register with their Facebook account.

The teams in Oulu that started the development in February were asked to develop the same tool with the same requirements proposed with the same schedule. To ensure the correct succession of requirements and to prevent the development of the previous project in Bolzano to influence the entrepreneur's perception of her project, we recorded every requirement elicited in Bolzano so as to ask her to request the same things without revealing any details to the other teams.

3.3 Study Execution

The web application was developed at the Software Factory Lab of the two participating universities. The participants were initially informed about the study and about the usage of the collected data. The development took place at the University of Bolzano-Bozen (Italy) from October 2015 until the end of January 2016 and at the University of Oulu

from February 2016 to the end of April 2016. The groups were required to spend a minimum effort of 250 h on their work.

Three groups were composed of second-year master students in computer science at the University of Oulu (Finland), while one group was composed of second-year master students in computer science from the University of Bolzano-Bozen. The selected students represent typical developers entering the market. It is therefore interesting not only to understand how they break down requirements but also to observe their work processes. All of the teams had iterations lasting two weeks. The Banana team also met the entrepreneur every two weeks in order to be updated on the requirements.

The first group (Kanban, https://github.com/Belka1000867/Serendipity) was composed of five master students who developed in Scrum with Kanban. The second group (Scrum, https://github.com/samukarjalainen/serendipity-app and https://github.com/-samukarjalainen/serendipity-web) was composed of five master students who developed in Scrum with 2-week sprints, while the third group (XP, https://github.com/davidetaibi/unibz-serendipity) was composed of four master students who developed in Extreme Programming (XP). The fourth group (Banana, https://github.com/Silvergrail/Serendipity/releases) was composed of six master students who developed in an unstructured process, which we defined as "Banana" process.

3.4 Data Collection and Analysis

The measures were collected during meetings with the developers. They also used the collected data to draw burn-down charts and track results. We defined a set of measures to be collected as follows:

- number of sprints;
- opening and closing date for each user story;
- user story description;
- responsible developer for each user story; and
- the actual effort for each user story.

The requirements were elicited from the entrepreneur. However, to avoid interference with the development process, two researchers attended the requirements elicitation meetings and reported the requirements independently.

Three sets of decisions were used to measure pairwise interrater reliability in order to get a fair/good agreement on the first process iteration. In order to resolve any differences, where necessary, we discussed any incongruity to get 100% coverage among the authors.

We associated user stories/tasks with each requirement defined by the entrepreneur. Then we calculated sums, medians, and averages.

4 Study Results

The teams developed the project according to the assigned development process. The XP team developed with a test-first approach, while the two Scrum teams (Scrum and

Scrum with Kanban) developed test cases during the process. The Banana team developed a limited set of test cases at the end of the process. All four teams delivered a final product with the same set of features, with no requirement missing.

The three Agile teams delivered the first version of the product with a limited set of features that could be evaluated by the customer after two sprints, while the Banana team delivered the application, with nearly all the features implemented, only three weeks before the end of the development and then started to implement tests. This result was expected because of the structure of the process, since they decomposed the requirements by means of a horizontal break-down. For example, they developed the whole server-side application first, starting from the design of the database schema, and then the Android application connecting the frontend with the server-side functionalities.

The three Agile teams decomposed the requirements by means of a vertical breakdown [5], so as to deliver to the entrepreneur a working product with the required features as soon as possible. For example, Req 2 (Show the list of available sounds on a map) was decomposed by the XP team into: "Show a Google map centered on the user location in the Android app" and "Show existing sounds as map placeholders", while the Scrum team and the Scrum with Kanban team decomposed this into: "Show a Google map", "Centered on the user location in the Android app," and "Show existing sounds as map placeholders."

As expected, the groups decomposed the 13 requirements into different subsets of user stories/tasks. As reported in Table 1 and Fig. 3, the team working in XP is the one that decomposed the requirements with the lowest granularity (46 user stories), followed by the team using Scrum with Kanban (40 user stories) and the team using Scrum (27 stories). However, the team using the Banana approach decomposed the requirements into only 13 tasks. Moreover, they merged two requirements into one single task. Considering the number of decomposed user stories or tasks per requirement, the results are obviously similar to the total number of user stories and tasks reported.

Table 1. Summary of metrics results

Metrics	XP	Scrum	Scrum+Kanban	Banana
M1 (# of requirements)	13	13	13	13
M2 (# of user stories/tasks)	46	27	40	19
M3 (user stories per requirements)	3.54	2.08	3.08	1.46
M5 (effort per requirement)	23.04	36.77	24.46	48.85
M6 (effort per user story/task)	6.51	17.70	7.95	33.42
Total effort all user stories/tasks	299.5	478	318	635
Other effort	92	10	0	481
M4 (total effort entire project)	391.5	488	318	1116

Taking into account the required effort, the team developing with Scrum with Kanban was the most efficient one, spending a total of 318 h on development. The XP and Scrum teams followed with an effort of 391 h for XP and 478 h for Scrum. The Banana team, unexpectedly, spent dramatically higher effort (1116 h), nearly 3.5 times more than the teams developing with Scrum and Kanban. Considering the effort spent on other tasks not related to user stories, such as database design, server setup, and such, the team using Scrum with Kanban was also the most efficient one, spending no effort on these tasks.

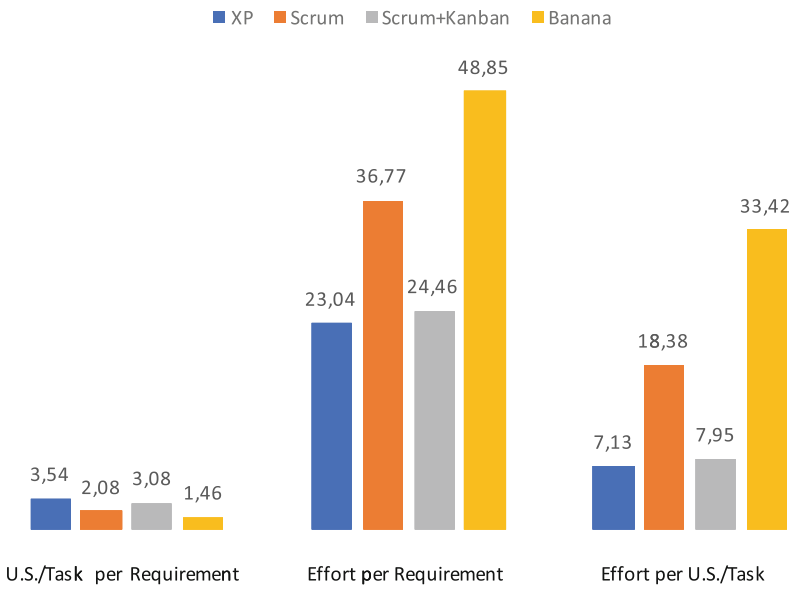

Fig. 3. Comparison of user stories and task decomposition and effort (hours)

The Scrum team only spent 10 h on other activities (2%), the XP team spent 92 h (23%), and the Banana team 481 h (43%).

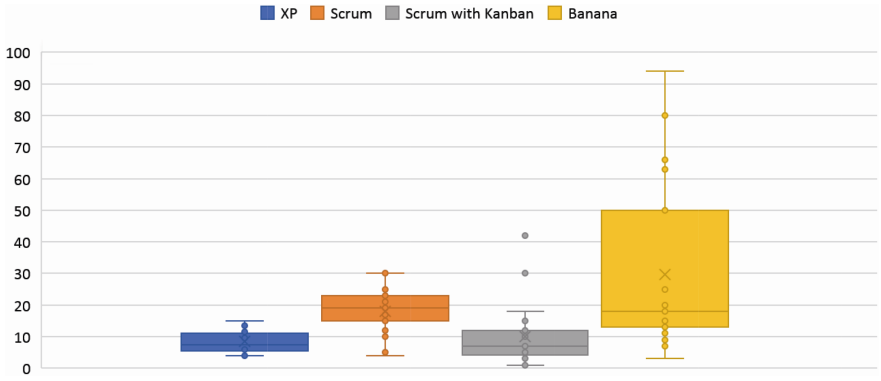

Fig. 4. Boxplot of the effort spent per user story/task

When analyzing the average effort spent to implement each requirement, the teams developing with XP and Scrum with Kanban obtained similar results, while the Scrum and the Banana teams spent similar amounts of effort per requirement, nearly 2.5 times more than the XP and Scrum with Kanban teams. Taking into account the distribution of effort depicted in Fig. 4, there is a similar distribution of effort spent on user stories

between the Agile teams, while, as expected, the Banana team had the highest variability of effort. Looking at the decomposition for each task (Table 2), other differences among the groups emerge. Req 6 was not implemented by all the teams since it was related to "not implementing" the download sound feature. The Banana team also considered zero effort for Req 7 since they merged the tasks with the activities related to Req 4.

Table 2. Effort and user stories/tasks per requirement

Requirement	XP			Scrum			Scrum with Kanban			Banana		
	Effort	# of user stories	Effort/ user story	Effort	# of user stories	Effort/ user story	Effort	# of user stories	Effort/ user story	Effort	# of tasks	Effort/ task
R1	40.5	6	6.8	81	3	27.0	77	5	15.4	140	5	28
R2	20.5	2	10.3	47	3	15.7	17	3	5.7	50	1	50
R3	94	11	8.5	57	3	19.0	112	9	12.4	150	3	50
R4	66.5	7	9.5	46	2	23.0	27	4	6.8	94	1	94
R5	9	2	4.5	16	1	16.0	5	1	5.0	19	1	19
R6												
R7	4	1	4.0	6	1	6.0	5	1	5.0			
R8	13	1	13.0	10	1	10.0	1	1	1.0	18	1	18
R9	17	2	8.5	12	2	6.0	15	1	15.0	25	1	25
R10	10.5	1	10.5	2	1	2.0	1	1	1.0	13	1	13
R11	10	1	10.0	21	1	21.0	2	4	0.5	21	2	10.5
R12	7	1	7.0	165	8	20.6	44	9	4.9	87	2	43.5
R13	7.5	1	7.5	15	2	7.5	12	1	12.0	18	1	18

Table 3 illustrates the various methods applied by the various teams to break down the requirements into user stories (or tasks for the Banana approach), i.e., the results of collecting metric M7. To obtain this table, two researchers studied the user stories and tasks provided by the teams and compared the approach adopted to break down the requirements with the approaches described in the literature. All disagreements in the classification were discussed and clarified based on the description of the broken-down user stories or tasks as well as the description of the approaches found in the literature.

To also be able to classify approaches not recommended in an Agile project, we added the three horizontal break-down strategies described by Verwijs [5]: divide user stories by (1) the type of work that is needed, (2) the layers that are involved, or (3) the components that are involved.

All teams used the approaches "Input options/platform" and "Conjunctions and connecting words". All Agile teams used the approaches "Data types or parameters", "Operations", and "Simple/Complex". Only the Banana team adopted the "Workflow steps" approach and only the XP team adopted the approaches "Test scenarios/test case" and "Roles". The approach "Major effort" was used by the teams XP, Scrum with Kanban, and Banana.

Unexpectedly, the Banana team was not the only one that adopted horizontal break-down approaches such as dividing user stories or tasks based on the layers of the solution, types of work, or components. Typically, Agile teams avoid such types of break-down since this contradicts with the principle that a user story should provide value to the user. We conjecture that the frequent application of horizontal break-down approaches by the

Scrum team was the reason for their bad performance in terms of total effort, compared to the other Agile teams. This also shows that the experiment was conducted with university students with little experience in the field. Nevertheless, their behavior is comparable to professionals at the beginning of their careers. We did not involve freshmen students in the study, as recommended by [10].

Table 3. Requirement decomposition strategies adopted by the studied teams

Strategy	XP	Scrum	Scrum with Kanban	Banana
Vertical decomposition strategies				
Input options/platform [5]	×	×	×	×
Conjunctions and connecting words [4]	×	×	×	×
Data types or parameters [3, 5]	×	×	×	
Operations e.g. CRUD [3, 5]	×	×	×	
Simple/Complex [3, 5]	×	×	×	
Major effort [3, 5]	×		×	×
Workflow steps [3–5, 8]				×
Test scenarios/test case [4, 5]	×			
Roles [5]	×			
Business rules [3, 5]				
Happy/unhappy flow [5]				
Browser compatibility [5]				
Identified acceptance criteria [4, 5]				
External dependencies [5]				
Usability requirements [5]				
SEO requirements [5]				
Break out a Spike [3]				
Refinement of generic words [4]				
Horizontal decomposition strategies				
Layers, e.g. database, GUI [5]	×	×		×
Type of work, e.g. testing, coding [5]		×		×
Components, e.g. server, client [5]		×		×

5 Threats to Validity

Concerning the internal validity of the study, even though we did our best to select developers with a similar background, the results could be partially dependent on the subjects. A replication study could confirm or reject our findings. Concerning the external validity of the study, the use of students to investigate aspects of practitioners is still being debated but considered very close to the results of real practitioners in the case of master students [9] and when one is interested in evaluating the use of a technique by novices or non-expert software engineers [10–13, 17].

6 Conclusion and Future Work

In this work, we conducted a preliminary multiple case study with a replication design with the aim of comparing the requirements decomposition process of an ad-hoc process and Extreme Programming, Scrum, and Scrum with Kanban.

With this study, we contribute to the body of knowledge by providing the first empirical study on requirements decomposition in the Agile domain.

To achieve this purpose, we first provided an overview of the different requirements decomposition techniques and then a description of the study we executed.

Although some results might depend on the participants' skills, we observed the usage of different decomposition techniques in our groups, which often adopted traditional decomposition techniques, which are more suitable for waterfall processes, in combination with other Agile techniques.

The teams developing with Scrum with Kanban and with XP decomposed the requirements into the highest number of user stories, while the team working with an unstructured process, as expected, decomposed the requirements into a very limited number of tasks. Two decomposition approaches were adopted by all processes, namely "Input options/platform" and "Conjunctions and connecting words". All Agile teams used the "Data types or parameters", "Operations", and "Simple/Complex" approaches, while, as expected, only the Banana team adopted the "Workflow steps" approach and only the XP team adopted the approaches "Test scenarios/test case" and "Roles".

Unexpectedly, the Banana team was not the only one that adopted horizontal break-down approaches such as dividing user stories or tasks based on the layers of the solution, types of work, or components. We suppose that the bad performance in terms of total effort of the Scrum team compared to the other Agile teams was probably due to the application of horizontal break-down approaches.

The main result of this work is that requirements decomposition is not only team-dependent but also process-dependent, and that therefore decomposition techniques need to be addressed to a greater extent in order to improve the efficiency of the development process.

Therefore, we recommend that developers investigate requirement break-down approaches more thoroughly and that researchers study the impact of different approaches, so as to identify the most effective ones in different contexts.

In the future, we plan to validate the results obtained with studies involving more students and practitioners and using larger projects.

References

1. Yin, R.K.: Case Study Research: Design and Methods, 4th edn. Sage, Thousand Oaks (2009)
2. Basili, V.R., Caldiera, G., Rombach, H.D.: The goal question metric approach. In: Encyclopedia of Software Engineering (1994)

3. Lawrence, R.: Patterns for splitting user stories. Agile For All Blog, 28 October 2009. http://Agileforall.com/patterns-for-splitting-user-stories/. Accessed 8 Dec 2016
4. Irwin, B.: Boulders to gravel: techniques for decomposing user stories. VersionOne Blog, 9 May 2014. https://blog.versionone.com/boulders-to-gravel-techniques-for-decomposing-user-stories/. Accessed 8 Dec 2016
5. Verwijs, C.: 10 useful strategies for breaking down large User Stories (and a cheatsheet). Agilistic Blog. n.d. http://blog.agilistic.nl/10-useful-strategies-for-breaking-down-large-user-stories-and-a-cheatsheet/. Accessed 8 Dec 2016
6. Taibi, D., Lenarduzzi, V., Ahmad, O.M., Liukkunen, K., Lunesu, I., Matta, M., Fagerholm, F., Münch, J., Pietinen, S., Tukiainen, M., Fernández-Sánchez, C., Garbajosa, J., Systä, K.: Free innovation environments: lessons learned from the software factory initiatives. In: The Tenth International Conference on Software Engineering Advances, ICSEA 2015 (2015)
7. Lenarduzzi, V., Lunesu, I., Matta, M., Taibi, D.: Functional size measures and effort estimation in agile development: a replicated study. In: 16th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP2015 (2015)
8. Rubin, K.S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process, 1st edn. Addison-Wesley Professional, Boston (2012)
9. Patton, J., Economy, P.: User Story Mapping: Discover the Whole Story, Build the Right Product, 1st edn. O'Reilly Media, Inc., Sebastopol (2014)
10. Runeson, P.: Using students as experiment subjects – an analysis on graduate and freshmen student data. In: Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering (2003)
11. Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., Rosenberg, J.: Preliminary guidelines for empirical research in software engineering. IEEE Trans. Softw. Eng. **28**(8), 721–734 (2002)
12. Tichy, W.F.: Hints for reviewing empirical work in software engineering. Empirical Softw. Eng. **5**(4), 309–312 (2000)
13. Salman, I., Misirli, A.T., Juristo, N.: Are students representatives of professionals in software engineering experiments? In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence (2015)
14. Lavazza, L., Morasca, S., Taibi, D., Tosi, D.: Applying SCRUM in an OSS development process: an empirical evaluation. In: 11th International Conference on Agile Processes in Software Engineering and Extreme Programming, XP 2010, pp. 147–159 (2010)
15. Diebold, P., Dieudonné, L., Taibi, D.: Process configuration framework tool. In: 39th Euromicro Conference on Software Engineering and Advanced Applications (2014)
16. Taibi, D., Lenarduzzi, V.: MVP explained: a systematic mapping on the definition of minimum viable product. In: 42th Euromicro Conference on Software Engineering and Advanced Applications 2016, Cyprus (2016)
17. Basili, V.R., Shull, F., Lanubile, F.: Building knowledge through families of experiments. IEEE Trans. Softw. Eng. **25**(4), 456–473 (1999)
18. Wang, H., Zhou, S., Yu, Q.: Discovering web services to improve requirements decomposition. In: 2015 IEEE International Conference on Web Services, New York, NY (2015)
19. Abbasipour, M., Sackmann, M., Khendek, F., Toeroe, M.: Ontology-based user requirements decomposition for component selection for highly available systems. In: Proceedings of the 2014 International Conference on Information Reuse and Integration (2014)

20. Morasca, S., Taibi, D., Tosi, D.: OSS-TMM guidelines for improving the testing process of open source software. Int. J. Open Source Softw. Process. **3**(2), 1–22 (2011)
21. Ahmad, M.O., Markkula, J., Oivo, M.: Kanban in software development: a systematic literature review. In: 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), pp. 9–16, September 2013

Effects of Technical Debt Awareness: A Classroom Study

Graziela Simone Tonin[1(✉)], Alfredo Goldman[1], Carolyn Seaman[2], and Diogo Pina[1]

[1] Institute of Mathematics, Statistics and Computer Science, University of Sao Paulo, São Paulo, Brazil
{grazzi,gold,diogojp}@ime.usp.br

[2] Department of Information Systems, University of Maryland Baltimore County, Baltimore, USA
cseaman@umbc.edu

Abstract. Technical Debt is a metaphor that has, in recent years, helped developers to think about and to monitor software quality. The metaphor refers to flaws in software (usually caused by shortcuts to save time) that may affect future maintenance and evolution. We conducted an empirical study in an academic environment, with nine teams of graduate and undergraduate students during two offerings of a laboratory course on Extreme Programming (XP Lab). The teams had a comprehensive lecture about several alternative ways to identify and manage Technical Debt. We monitored the teams, performed interviews, did close observations and collected feedback. The results show that the awareness of Technical Debt influences team behavior. Team members report thinking and discussing more about software quality after becoming aware of Technical Debt in their projects.

Keywords: Technical debt · Technical debt awareness · Technical debt impact · Extreme programming

1 Introduction

Several studies have shown that agile methods have provided significant gains in software projects [19]. However, it is also known that when prioritizing delivery speed, as may happen in agile projects, Technical Debt may be incurred. Much of this debt is not even identified, monitored or managed. Technical Debt that is not well managed runs the risk of high maintenance costs.

The term Technical Debt was introduced by Cunningham, who explained it in the following way [4], "…Although the immature code may work fine and be completely acceptable to the customer, excess quantities will make a program unmasterable, leading to extreme specialization of programmers and finally an inflexible product. Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite […]. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt…". Technical Debt is recognized as a critical problem for software companies [2] and has received a lot of attention in the recent years from both practitioners and researchers [16, 17].

H. Baumeister et al. (Eds.): XP 2017, LNBIP 283, pp. 84–100, 2017.
DOI: 10.1007/978-3-319-57633-6_6

Lim et al. [18] emphasize that: "…most project teams now recognize that Technical Debt is unavoidable and necessary within business realities. So managing Technical Debt involves finding the best compromise for the project team…", but a project team cannot do this if they are not aware of Technical Debt. Also, Lim et al. highlighted that when the development team is not aware of Technical Debt, it will probably result in challenges for maintenance and evolution tasks. Given this scenario, our motivation was to observe the effects of Technical Debt awareness in teams in an academic setting.

The Extreme Programming Laboratory (XP Lab) is a course that has Undergraduate and Graduate students at the University of São Paulo since 2001. The aim of this course is to provide the experience of a real software development scenario using the Extreme Programming values and practices [1].

Extreme Programming emphasizes teamwork; managers, customers and developers are all equal partners in a collaborative team. The main values of Extreme Programming are communication, simplicity, feedback, respect and courage [3].

The objective of our research is to characterize the impact on the team when Technical Debt items are visible, based on team members' perceptions. This study aims to answer the following research question (RQ):

- **What is the impact on the team when Technical Debt is explicitly considered?**

The study was applied in two editions of XP Lab. Four teams were followed in the 2013 edition and five teams in the 2014 edition. We conducted the study and collected data through questionnaires and interviews, and analyzed the source code of the projects with Sonar Qube and Code Climate tools to identify the impact on the teams that explicitly considered Technical Debt (TD).

In the next section, related work is described. In Sect. 3, we describe the context of the Extreme Programming Laboratory. After, in Sect. 4, we provide a description of the research steps, data collection and analysis. Section 5 describes the results. In Sect. 6, we discuss the findings and present the threats to validity. Finally, in Sect. 7, we present the final considerations and future work.

2 Related Work

Few studies deal directly with the technical debt awareness. The study of Kruchten [22] showed that agile teams believe that they are immune to TD, because they use an interactive development process. Therefore, he explains that in these teams, TD items could be contracted rapidly and massively, because code is often developed and delivered very rapidly, without time to devote to good design or to think about longer term issues. This could result in contracting TD items such as a lack of rigor or systematic tests. To deal with TD and to avoid accumulating too much TD, he suggests*: "The first step is awareness: identifying debt and its causes. The next step is to manage this debt explicitly, which involves listing debt-related tasks in a common backlog during release and iteration planning, along with other "things to do."*. Bavani [21] shows that if teams are unaware of the context of meaning of the term TD, they can consider trivial issues or technical tasks as a TD. These teams have to improve the awareness of it, so he proposes

a quadrant to help teams to better recognize and understand the TD concept. The study of Martini [23] listed some causes of architecture technical debt and one of the reasons he found was the lack of awareness about the dependencies between the specific architectural TD and the other parts of the software. Furthermore, there are many related studies on not managing TD and how this affects software quality, such as in the studies of Guo [20], Sterling [24], Li [16], Lim [18], and Curtis [25]. McConnell [26] emphasizes that when a team makes the decision to contract a debt or not, they are really deciding between two ways to complete the current development task, one faster and the other resulting in better quality. Bavani [21] talks about management of TD items in distributed agile teams, and he emphasizes that the management of TD items directly affects the economics of software maintenance and according to him, the key for success in the current global economy is building and maintaining software under optimal costs. Sterling [24] said that TD exists and is detrimental to the maintenance of software quality. Buschmann [27] suggests that teams doing a refactoring in the code should also pay the TD items and improve internal quality. A recent report showed that one of the consequences of incurring TD is the impact on quality [28].

3 Context: Extreme Programming Laboratory

The XP Lab is a regular course offered at the University of São Paulo, to graduate and undergraduate Computer Science students. The motivation is to provide them an opportunity to learn agile software development methods on real projects. In the 2013 offering, there were four teams, with five or six students each. In the 2014 offering, five teams with six students each attended the course. XP Lab students have the support of meta-coaches who are experts in agile methods. They provide agile mentoring for all the teams with the professor's help. Each team also has a coach, who is a student that has more experience in agile methods. The teams develop real projects with on-site customers. The teams have to follow some agile practices, for instance; pair programming, automated tests, continuous improvement, continuous integration, etc. In both studies, the teams worked in pairs and in threes, and the groupings changed many times during the course, sometimes according to the tasks they needed to develop. The course requires a minimum attendance of at least 8 h a week of dedication (four hours in the laboratory and four hours of extra classes), and there is a lunch once a week, to encourage the students' presence in the lab and to allow the students to share experiences. On some weeks, there are short presentations about some difficulties that the teams are facing, where a specialist explains and discusses specific topics. A complete description of the course settings can be found in [1].

3.1 Projects

In Table 1, we briefly describe each of the projects involved in our study:

Table 1. Extreme programming projects

Project	Description
Arquigrafia	Arquigrafia is a public digital collaborative environment, nonprofit, dedicated to the dissemination of architectural images, with particular attention to Brazilian architecture [6].
Games-VidaGeek	A platform for games that support the teaching of programming (with games for Scala, Java, Html, CSS, SQL and other languages being produced) [7].
TikTak	A project focused on collecting feedback data from users and display it in a web dashboard [5].
Mezuro	A framework for monitoring source code metrics [8].
Monitoring system	An online system where students can apply to be a teacher assistant of a regular courses [14].
System specialist in sport	An application to enable researchers working with physiological data to apply metabolic mathematical models [15].
Social networking startups	A social network for Startup, with the goal of creating a community of highly connected and committed entrepreneurs [5].
Family tree	A genealogy community where each individual can create a family tree and from time to time the system attempts to "link" the trees [5].
CoGroo	Portuguese grammar corrector used by LibreOffice [9].

3.2 The Informative Workspace

Each team had its private informative workspace[1] [10, 11], where they physically displayed TD items. In the XP Lab 2013 offering, all teams had a TD board (Figs. 1 and 2). In the XP Lab 2014 offering, each team decided by themselves how to manage the TD items in their informative workspaces. Some teams decided to have the TD board and other teams kept the TD items list on a Kanban board.

3.2.1 Boards

In the TD board (Fig. 1) a team placed the TD items that were incurred and/or identified. On the top of the board, there is a supply of blank cards called 'Fichas'. These cards were used to document the TD items.

Figure 2 shows another team's board where they kept the list of TD items that were incurred and identified. On the right side, they have a reserve of blank cards.

[1] The informative workspace is the place where the teams put all the physical boards and graphics, with the metrics they used to manage the project development also the list of the task they will develop in each sprint.

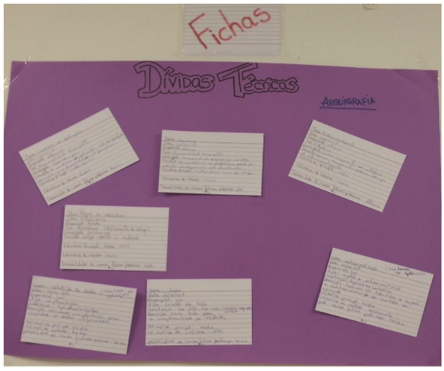

Fig. 1. Technical debt board

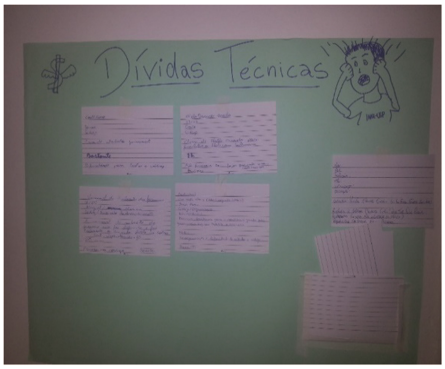

Fig. 2. Board of the technical debt list.

Figure 3 shows one TD item about *duplicated code*. Each card had nine categories to fill out. Below we transcribe the data contained in Fig. 3.

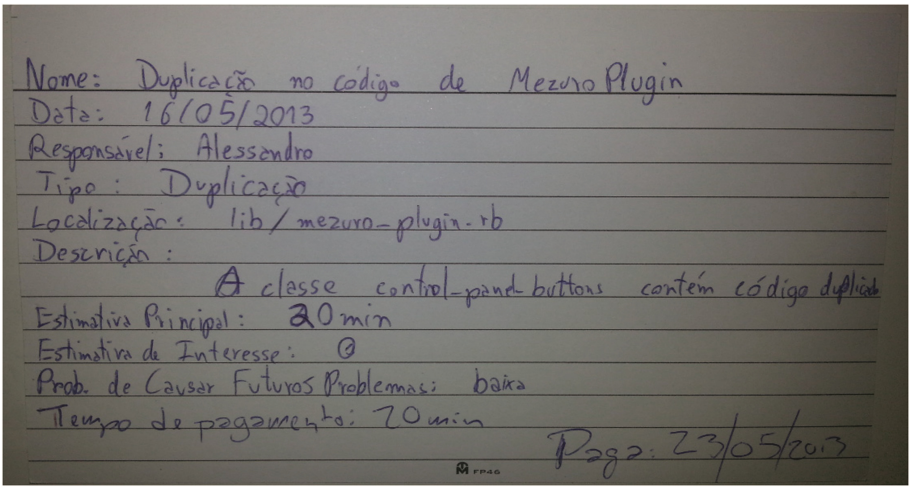

Fig. 3. Technical debt item

In this case the Name of TD item was *Duplicated code in the Mezuro plugin,* the Date (when the item was identified), was 05/16/2013, the Responsible (the person that incurred or found the TD item, in this case Alessandro), the Type was *duplication,* (could be test, documentation, design, etc.), the Location (which part of the code the items was related), was in *lib/mezuro-puglin-rb*. The Description (a brief description of the TD item), *the class control-panel-buttons* has duplicated code. The Estimated Principal, was *twenty minute*s (how much expected time they need to spend now if they implemented that task in the correct way, if they did not know how much time, then they could use a scale of *high* - if they probably will spend a large amount of time -, *medium* - if they probably will not spend much time - and *low* - if they probably will solve it quickly).