



Dipl.Ing. Annemarie Harzl, BSc

On the Combination of FOSS and Kanban - Insights Gained from a Hybrid Student Free & Open Source Software Project

DOCTORAL THESIS

to achieve the university degree of
Doktorin der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Wolfgang Slany
Institute of Software Technology
Graz University of Technology

Graz, September 2018

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Dissertation identisch.

Graz, am _____
Datum

Unterschrift

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Graz, _____
Date

Signature

Abstract

Free and Open Source Software (FOSS) and Agile Software Development (ASD) are both very important software development methods. They have many success stories and share some similarities. However, there is a lack of research regarding the comprehensive integration of ASD and FOSS. This thesis attempts to consolidate these methods and to answer if FOSS and ASD can be combined successfully and for the benefit of the contributors, by studying a large hybrid student FOSS project, the Catrobat project.

First, problems the project experienced due to fast growth of the project and the internationalization of contributors are described and possible solutions are suggested. The implemented solutions enabled the subsequent Action Research (AR). Second, AR was conducted with one sub-team of the Catrobat project, and five AR cycles introducing the Kanban method into the FOSS project were performed. The Kanban practices *visualize the workflow, make policies explicit, limit Work In Progress (WIP), manage flow, and implement feedback loops* were examined during the AR cycles. The journey of the team and findings of this research are reported.

This thesis describes a real world situation, where Kanban is applied to a hybrid student FOSS project, and it determines that the combination is possible and experienced as beneficial by contributors. Study participants report a positive effect on communication with stakeholders as well as with other teams due to the use of the Kanban method. They regard their time acquiring knowledge about Kanban practices as well spent.

This work increases the knowledge about FOSS and ASD. Furthermore, it can help other FOSS projects to determine, if they want to introduce agile practices into their workflow and what they could gain from it. If they decide to do so, it can support them on their journey of adopting new practices.

Kurzfassung

Free and Open Source Software (FOSS) und Agile Softwareentwicklung (ASD) sind beide sehr wichtige Softwareentwicklungsmethoden. Beide weisen zahlreiche Erfolgsgeschichten auf und teilen einige Gemeinsamkeiten. Jedoch gibt es kaum Forschung zur umfassenden Kombination von beiden Methoden. Diese Doktorarbeit versucht beide Methoden zu vereinen und die Frage zu beantworten ob FOSS und ASD erfolgreich und zum Nutzen der Mitwirkenden kombiniert werden können. Zu diesem Zweck wird ein großes, hybrides studentisches FOSS Projekt, das Catrobat Projekt, untersucht.

Zuerst werden Probleme beschrieben, die durch das schnelle Wachstum des Projektes und die Internationalisierung der Mitwirkenden entstanden sind und mögliche Lösungen vorgeschlagen. Die Umsetzung dieser Lösungen ermöglichte die nachfolgende Action Research (AR). Mit einem Team des Catrobat Projektes wurden fünf AR Zyklen anhand der Kanbanmethode durchlaufen. Die Kanbanpraktiken *Mach Arbeit sichtbar*, *mach Prozessregeln explizit*, *limitiere den Work in Progress (WIP)*, *manage flow*, und *implementiere Feedback-Mechanismen* wurden während der AR Zyklen untersucht. Die Reise des Teams und die Ergebnisse der AR werden beschrieben.

Diese Doktorarbeit beschreibt eine Situation in der echten Welt, in der Kanban auf ein hybrides studentisches FOSS Projekt angewendet wird. Ergebnisse zeigen, dass die Kombination aus Kanban und FOSS möglich ist und von den Projektmitgliedern als vorteilhaft wahrgenommen wird. Studienteilnehmer berichten positive Effekte auf die Kommunikation mit Interessensvertretern und anderen Teams, die auf die Anwendung der Kanbanmethode zurückzuführen sind. Des Weiteren erachten sie die Zeit, die sie investiert haben um sich Wissen über Kanban anzueignen als gut investierte Zeit.

Diese Arbeit vergrößert das Wissen über FOSS und ASD. Darüberhinaus, kann sie anderen FOSS Projekten dabei helfen zu entscheiden ob sie agile Praktiken einführen möchten und was sie dadurch gewinnen können. Darüberhinaus kann diese Arbeit andere Projekte auf ihrem Weg agile Praktiken einzuführen unterstützen.

Thanks/Acknowledgments

First, I would like to thank my adviser, Professor Dr. Wolfgang Slany for the great opportunity to work on this topic I care so much about, his encouraging comments, endless patience, and help in all aspects.

I would also like to thank Professor Brian Fitzgerald PhD, Peggy Gregory BA MSc PhD, and Professor Helen Sharp for their insightful feedback, advice and encouraging comments at the XP 2015 PhD Symposium.

During my research and teaching at Graz University of Technology, I had the opportunity to meet so many wonderful and impressive students and I am forever grateful to have had these amazing years together with you. It was a pleasure working with you and you taught me many things! I would like to thank all my teaching assistants who helped me managing classes and therefore enabled me to conduct my research alongside teaching. Thanks also to all students of the Catrobat project, you are the project, you make all these great things possible. Special thanks to the Catrobat team, who allowed me to conduct research with them, I could not have asked for a better group of people to take this journey together. Last, but not least, my sincerest gratitude goes to Stephan Fellhofer, Markus Hobisch, Adrian Schnedlitz, Raphael Sommer, and Florian Winkelbauer, your impressive passion and great contributions made all this possible.

In addition, I thank all colleagues at the Institute for Software Technology, especially Birgit Hofer, Roxane Koitz-Hristov, Elisabeth Orthofer, Anja Petri, and Petra Pichler.

Finally, I am forever indebted to my parents, my whole family and friends. I owe you my deepest gratitude, this thesis would have not been possible without you.

Thanks to the Catrobat team¹: Anja Petri, Mario Kolli, Martina Edelhofer, Maria Schrack, Melanie Kleindienst, Lisa Kehrer, Davor Kirbis, Katharina Stadlmayr, Alexander Kalchauer, Othmar Ruprecht, Peter Treitler, Thomas Holzmann, Tom Spiss, Stefan Hohenwarter, Nikolaus Koller, Tobias Gritschacher, Oliver Prentner, Sercan Akpolat, Alexander Gütler, Peter Schmidl, Maximilian Fellner, Ferdinand Knapitsch, Daniel Markart, David Reisenberger, Daniel Burtscher, Jörg Hofer, Christian Hofer, Roman Mauhart, Stefan Mayer, Ainul Husna Abdul Muin, Jia Lin Chong, Denise Hoo, Manuel Zoderer, Martin Oswald, Dietmar Maurer, Christian Lesjak, Christoph Worgoetter, Safdar Zaman, Johannes Iber, Hans Peter Gugl, Florian Hubner, Christoph Kahr, David Kikelj, Jakob Strauss, Pulkit Chouhan, Smiai Ousama, Boris Kanjer, Thomas Rosmarin, Stephan Fellhofer, Alex Nicoara, Bernhard Trapp, Hans-Jürgen Schröttner, Raphael Sommer, Gerald Wagner, Fatin Ghazi, Deena Mugien, Walter Tang, Thomas Loidolt, Christian Hartinger, Angelika More, Peter Tielsch, Gerald Musser, Roland Dutzler, Philipp Taferner, Otto Touzil, Max Löffler, Florian Sumann, Andreas Maueder, Manfred Knapp, Tobias Stumpfl, Markus Hobl, Andrea Höfler, Valentin Rock, Herbert Christian Lierzer, Thomas Kaufmann, Michael Pletz, Jeton Arifi, Matthias Traub, Patrick Koch, Robert Painsi, Daniel Fritzsich, Dominik Schleicher, Ilija Simic, Majda Osmic, Marco Steger, Maximilian Sachs, Michael Münzer, Michael Rieder, Peter Kapfer, Hannes Hasenauer, Gerhild Grinschgl, Andreas Frühwirt, Christof Stromberger, Markus Schmid, Mattias Rauter, Stefan Simon, Stefan Oberacher, Vesna Krnjic, Simge Sezgin, Artur Termenji, Samitha Priyanath Jayathilaka, Florian Schitter, Markus Höfer, Petra Pupovac, Ingrid Reip, Patman Ghazi, Suemeyra Mutlu, Bernhard Ruttinger, Julia Schönhart, Thomas Huber, Michael Rabko, Daniela Zierler, Lukas Krisper, Eberhard Ferner, Gerhard Neuhold, Rudolf Andreas Stumptner, Severin Holzer-Graf, Philipp Koncar, Sandra Fuchs, Thomas Kugi, Arno Stauder, Tomislav Ausperger, Heimo Ernst Bischoffer, Tobias Oblak, Gerald Gsellmann, Norbert Spot, Sachi Alana Williamson, Saif Aldeen Alsaifi, Angelika Droisner, Thomas Gruber, Hakan Özkan, Fabian Tschitschek, Lukas Resch, Andreas Abraham, Michael Peitler, Bianca Teufl, Daniel Neuhold, Domenik Melcher, Elisabeth Heschl, Matthias Sebastian Schlesinger, Rene Obendrauf, Rudolf Josef Wagner, Armin Hutzler, Valentin Kassarnig, Johannes Singer, Stefan Pointner, Andreas Voraberger,

¹<http://developer.catrobat.org/credits>

Alexander Oberegger, Florian Winkelbauer, Karl Koch, Lukas Prokop, Philipp Weissensteiner, David Kolb, Martin Burtscher, Dominik Mößlang, Philipp Kremers, Dominik Widnig, Alexander Oberbacher, Christoph Pilz, David Strohmaier, Ewald Moitzi, Gottfried Selenko, Markus Hobisch, Matthias Eichhaber, Michael Koweindl, Samir Ramadani, Stefan Padureanu, Hedwig Höller, Eva-Maria Trummer, Stefan Galler, Simone Lemmerer, Christopher Jelinek, Abdelbasset Amara, Thomas Prem, Paul Melbinger, Maria Aumann, Rayna Nikolova, Hans Dieter Knaus, Franz Schreiner, Peter Pranter, Jakob Unterkofler, Stefan Jaindl, Bernhard Spitzer, Dominik Ziegler, Daniel Pail, Christine Pichler, Manuel Wallner, Thomas Fuchs, Marco Meiser, Andreas Müller, Viktoria Schlaipfer, Armend Zeqiraj, Stefan Neureiter, Thomas Kriechbaumer, Andreas Hofbauer, Philipp Neidhöfer, Richard Schumi, Max Reinthaler, Max Schafzahl, Daniel Neuhold, Florian Winkelbauer, Michael Stradner, Artur Knaus, Manuel Jelinek, Martin Erb, Armin Wieser, Marko Burazer, Matthias Schlesinger, Markus Hartmair, Christopher Immervoll, Christian M. Hofer, Elena Mashkina, Roxane Koitz, Christian Burghard, Thomas Laubreyter, Namik Delilovic, Jasmin Salihovic, Gregor Sitter, Mirhet Saracevic, Tatiyana Domanova, Roland Urbano, Patrick Trummer, Ajdin Vihric, Rudolf Weißenbacher, Johannes Lüftenegger, Andreas Gladik, Marc Slavec, Ralph Samer, Bernhard Winter, Daniel Ladenhauf, Karl Koch, Theresa Egger, Sebastian Krell, Thomas Anderhuber, Manuel Polzhofer, Adrian Schnedlitz, Andreas Voraberger, Stefan Pointner, Andrej Müller, Tobias Ibounig, Johannes Kühnel, Bernd Baumann, Gerald Lohnauer, Thomas Kohl, Thomas Mauerhofer, Daniel Ellmeier, Lukas Radacher, David Prott, Stephan Frühwirt, Christian Benkovic, Alexander Wieland, Fabian Hartl, Patrik Maier, Claudio Kirchmair, Michael Herold, Jürgen Wurzinger, Lukas Mayr, Nikolaus Tiesenhausen, Thomas Fötschl, Christian Reisinger, Thomas Schranz, Phillip Goriup, Markus Nager, Philipp Pfleger, Andrew Deutschmann, Matthias Müller, Wolfgang Wintersteller, Alexander Oberbacher, Patrick Radkohl, Werner Arnus, Gerulf Binder, Martin Prinz, Marc Schober, Darjan Salaj, Thomas Lienhart, Laura Bebek, Illya Boyko, Aiman Awwad, Michael Grebien, Anna Lickl, Mario Lins, David Marogy, Manuel Haid, Christof Rabensteiner, Bernadette Spieler, Patrick Radl, André Tropper, Benjamin Bristow, Christian Schindler, Pascal Steiner, Bianca Jakobitsch, Samuel Sprung, Josef Filzmaier, Christian Jung, Johannes Zenz, Michael Pittner, Philipp Eisner, Sebastian Schrimpf, Zulfiqar Ali, Richard Aigner, Georg Schober, Stefan Bürscher, Kirshan Kumar Luhana, Robert Riedl, Amel

Hamidovic, Adam Ujvari, David Wittenbrink, Thomas Hirsch, Lukas Fritz, Marco Wallner, Florian Weißensteiner, Lukas Loibnegger, Amra Dzombic, Wolfgang Karl, Paul Schreiner, Andrej Knaus, Michael Lang, Oliver Zott, Mario Pagger, Markus Reiter-Haas, Thomas Gruber, Patrick Klampfl, Denis Munter, Fabian Kofler, Sebastian Gabl, Tobias Striemitzer, Jochen Flachhuber, Harald Schaffernak, Joachim Lesser, David Niederkofler, Martin Kerschbaumer, Alexander Kargl, Eric Gergely, Dominik Lindenbauer, Patrik Hutter, Gregor Liebisch, David Kienreich, Christoph Pozvek, Thomas Leh, Manuel Schweiger, Reinhold Seiss, Stefan Reichnauer, Julia Viehberger, Ines Vorraber, Lukas Pichler, Christian Leopold, Benjamin Wullschleger, Clemens Stary, Dino Keskcic, Florian Schneider, Jakov Matic, Christian Huber, Peter Waldert, Jiaqi Ning, Anja Reibenbacher, Matthias Fuchs, Thomas Koinig, Josef Kraxner, Johanna Stefanzl, Patrizia Kamp, Kevin Haslinger, Nishant Thapliyal, Pushkar Sharma, Abduqodiri Qurbonzoda, Nikolaus Steininger, Christopher Lamprecht, and Michael Musenbrock.

Contents

1. Introduction	1
1.1. Research Questions	2
1.2. Contributions of this Thesis	3
1.3. Outline	4
2. Agile Software Development Methods	5
2.1. Extreme Programming (XP)	10
2.1.1. XP Values	11
2.1.2. XP Principles	12
2.1.3. XP Practices	13
2.2. Kanban	20
2.2.1. Kanban Principles	21
2.2.2. Little's Law	22
2.2.3. Kanban Practices	23
3. Free Libre Open Source Software	29
3.1. Free and Open Source Software (FOSS) Definitions	30
3.1.1. Free Software	31
3.1.2. Open Source Software	32
3.1.3. Free Software versus Open Software	34
3.1.4. Other Terms for Free and Open Source Software	35
3.2. The FOSS Development Model / Methodology	36
3.3. FOSS Motivation	39
3.4. FOSS Characteristics	40
3.5. FOSS Criticism	44
4. Catrobat	47
4.1. Catrobat History	50
4.2. Catrobat Characteristics	51

Contents

4.3.	Motivation in FOSS and Catrobat	56
4.4.	Software Development Approach	62
4.5.	Selection of the Project	65
4.6.	Selection of the Team	66
4.7.	Selection of Kanban	71
5.	Related Work	73
5.1.	Educational Settings	73
5.1.1.	Kanban and Academia	73
5.1.2.	FOSS and Academia	75
5.2.	FOSS and Agile Software Development	78
5.3.	Summary	84
6.	Action Research	85
6.1.	Basics of Action Research	85
6.2.	Study Approach	88
6.3.	Selection of Action Research	89
6.4.	Researcher Role	89
7.	Getting Ready for Combining FOSS and Kanban	91
7.1.	Abstract	91
7.2.	Introduction	92
7.3.	Related Work	94
7.4.	Optimizing Services for Distributed Participation	94
7.4.1.	User Management	95
7.4.2.	Communication	96
7.4.3.	Agile Development Management	98
7.4.4.	Knowledge Management	100
7.5.	Lessons Learned	102
7.5.1.	Human Related	102
7.5.2.	Technology Related	102
7.6.	Future Work	103
7.7.	Conclusion	103
8.	Combining FOSS and Kanban: An Action Research	107
8.1.	Background	107
8.2.	Study Participants	107

Contents

8.3. Action Research	108
8.3.1. Data Sources	109
8.3.2. Data Analysis	109
8.4. Action Research Cycles	110
8.4.1. Diagnosing	110
8.4.2. Cycle Zero	111
8.4.3. First Cycle	114
8.4.4. Second Cycle	115
8.4.5. Third Cycle	116
8.4.6. Fourth Cycle	131
8.4.7. Fifth Cycle	134
8.5. Analysis of Usage of Agile Practices	144
9. Results and Discussion	153
9.1. Results	153
9.2. Threats to Validity	155
10. Conclusion and Future Work	159
10.1. Conclusion	159
10.2. Future Work	159
A. Papers	163
B. Questionnaires	167
C. Supplementary Material	207
Bibliography	217

List of Figures

2.1. Modern resolution for all projects	6
2.2. Modern resolution agile versus waterfall from FY2011-2015	7
2.3. Waterfall Model	8
2.4. Kanban board	24
2.5. Kanban types of work	25
3.1. Structure and roles in FOSS communities	42
4.1. Pocket Code	48
4.2. Pocket Paint	49
4.3. Catrobat Organigram	51
4.4. Contributions to other FOSS communities	54
4.5. Structure and roles in FOSS communities	55
4.6. Structure and roles in the Catrobat community	55
4.7. Motivators in General 1	58
4.8. Motivators for Catrobat members in General 2	59
4.9. Motivators in Catrobat 1	60
4.10. Motivators in Catrobat 2	61
4.11. Agile knowledge in Catrobat	63
4.12. Agile experience in Catrobat	64
4.13. Team size in Catrobat	67
4.14. Age distribution in Catrobat	68
4.15. Age distribution in AR team	68
4.16. Education in Catrobat	69
4.17. Education in AR team	69
4.18. Fields of study in Catrobat	70
4.19. Fields of study in AR team	70
5.1. Learning of Team Work Competencies with Kanban	75

List of Figures

6.1. The Cyclical Process of Action Research	86
7.1. Default Jira workflow	98
7.2. Customized Jira workflow	99
7.3. Areas for improvement selected by the survey participants . .	104
8.1. Exemplary stakeholder analysis	113
8.2. Desired versus actual visualization	117
8.3. Estimated tickets with story points	124
8.4. Estimated tickets with story points desired value	125
8.5. Estimated and desired tickets with T-shirt sizes	126
8.6. Usefulness of story points estimation	127
8.7. Usefulness of T-Shirt size estimation	128
8.8. New workflow	132
8.9. New Jira workflow	134
8.10. The team's CFD	135
8.11. Is Kanban knowledge beneficial to project work	136
8.12. Is Kanban knowledge beneficial to other work	137
8.13. Is Kanban beneficial to communication to stakeholders	138
8.14. Is Kanban beneficial to communication in team	139
8.15. Time spent learning Kanban well spent?	140
8.16. Usage of customer acceptance tests	147
8.17. Usage of test-first design	147
8.18. Usage of pair programming	148
8.19. Usage of short releases	148
8.20. Usage of meeting	149
8.21. Usage of growth	149
8.22. Usage of artifact reduction	150
8.23. Morale	150
8.24. Usage of measuring flow	151
C.1. Usage of unit tests	208
C.2. Usage of continuous integration	208
C.3. Usage of refactoring	209
C.4. Usage of release planning	209
C.5. Usage of on-site customer	210
C.6. Usage of coding standard	210

List of Figures

C.7. Usage of collective code ownership	211
C.8. Usage of sustainable pace	211
C.9. Usage of simple design	212
C.10. Usage of metaphor	212
C.11. Usage of lessons learned	213
C.12. Usage of visualize workflow	213
C.13. Usage of WIP	214
C.14. Usage of making policies explicit	215

1. Introduction

Free and Open Source Software Development (FOSSD) and Agile Software Development (ASD) have proven to be successful ways to develop software and both have been researched a lot since the publication of the Agile Manifesto in 2001 (Dingsøy et al., 2012; Crowston, Wei, et al., 2012). Research about agile topics includes, for example, the adoption of agile methods (Boehm, 2002), implementation of agile methods in distributed settings (Boland and Fitzgerald, 2004; Ramesh et al., 2006), and research about Test Driven Development (TDD) (Fucci and Turhan, 2013). Research about FOSSD includes, for example, motivational factors and participation in Free and Open Source Software (FOSS) projects (Bonaccorsi and Rossi, 2006). Not only research, also ASD and FOSSD themselves flourished. Within the last two decades FOSSD and ASD increased in popularity and by now are integral processes in software development (Dingsøy et al., 2012; Crowston, Wei, et al., 2012). Although some studies have been conducted about combining some aspects of FOSS and ASD (Deshpande and Riehle, 2008; Düring, 2006a; Ahmad, Liukkunen, and Markkula, 2014; MacKellar, Sabin, and Tucker, 2015; Koch, 2004), review of literature showed no studies comprehensively combining them. This is also supported by the work of Gandomani et al. (2013).

Already in 2003 and 2004 Warsta and Abrahamsson (2003) and Koch (2004) showed that FOSSD and the definition of ASD methods are rather close. However, in 2009 research about agile development in the context of open source software was still identified as a future research area by Ågerfalk, Fitzgerald, and Slaughter (2009). The systematic literature review by Gandomani et al. (2013) in 2013 on relationships between ASD and FOSSD showed that there is still a gap in literature about the comprehensive combination of these methodologies. Nevertheless, the results of Gandomani et al. (2013) indicate that ASD can support FOSSD, mainly because both

1. Introduction

approaches share several concepts and principles. This gap in literature and the ever increasing use of ASD and FOSSD in industry (Lerner and Tirole, 2002; DeKoenigsberg, 2008; VersionOne, 2015) and education (Casson and Hawthorn, 2011; Ahmad, Liukkunen, and Markkula, 2014; MacKellar, Sabin, and Tucker, 2015; Pinto et al., 2017) show that there is a need for a more detailed view of both matters.

1.1. Research Questions

The goal of this thesis is to answer the questions if ASD methods, specifically the Kanban Method (Anderson, 2010) can be successfully applied to a specific real world FOSS project and if project members can benefit from it. Another goal is to determine how the agile process and its introduction into a FOSS project can be customized to better fit the needs of the project and its contributors. The following Research Questions will be answered:

1. Research Question (RQ)₁ Can FOSS and ASD be comprehensively combined?
2. RQ₂ Can FOSS projects benefit from using agile methods like the Kanban Method? To answer this question the focus will be split on different aspects and three sub-questions will be answered.
 - RQ_{2.1} Do FOSS contributors, who are coached in the Kanban Method, experience this knowledge as beneficial to their work?
 - RQ_{2.2} Do interaction or communication during meetings change with the use of the Kanban Method?
 - RQ_{2.3} Do FOSS contributors regard their time acquiring Kanban knowledge well spent?

Because FOSS contributors time is very limited, it is not only interesting to know, if Kanban is experienced as beneficial (see RQ_{2.1}), but also if it is so beneficial to their work, that contributors are willing to sacrifice programming time to learn about Kanban.

As a real world case a hybrid student FOSS project, which is situated at Graz University of Technology, was selected. Out of the whole project (consisting of more than 100 contributors and more than eight teams) one team

1.2. Contributions of this Thesis

was chosen to participate in the study, because conducting the study with the whole project at once, would not have been feasible. However, one part of the thesis was done for the whole project. This part is described in Chapter 7. The fact that the FOSS project was situated at the same university as the researcher, provided the opportunity to observe the research participants in their natural context and to conduct participatory Action Research (AR) with them. AR provides the possibility to pursue scientific outcomes and practical outcomes for the research participants at the same time. As ASD method the Kanban Method (Anderson, 2010; Leopold and Kaltefleiter, 2013) was chosen, because it is the most adaptive method (Kniberg and Skarin, 2010). The reasoning behind choosing this specific project, AR, and Kanban for this thesis will be explained in more detail in later chapters. The question, if the Kanban Method is an agile or lean approach or if the terms agile and lean are interchangeable or not (Fitzgerald, Musial, and Stol, 2014), is not relevant for this thesis, and will therefore not be discussed.

1.2. Contributions of this Thesis

This thesis increases the knowledge about ASD and FOSSD, and the possible integration of both, by applying Kanban to a hybrid student FOSS project. The main contributions of this thesis are:

- It describes a real world situation, where Kanban is applied to a hybrid student FOSS project.
- It determines that it is possible to comprehensively combine ASD and FOSS, in a specific project.
- It suggests two new Kanban practices specifically targeted at FOSS projects and their characteristics, which can support FOSS projects in introducing Kanban.
- It describes five AR cycles, discovering more details about combining Kanban and FOSS.
- It determines that the combination is experienced as beneficial by contributors. In fact, so beneficial that some study participants use personal Kanban to manage all their tasks.

1. Introduction

- It reports a positive change in communication and interaction to other teams and stakeholders due to the use of the Kanban method.
- It shows, that contributors regard their time learning about Kanban as worth their while.

1.3. Outline

The rest of the thesis is organized as follows: Chapter 2 explains different software development methods with a special focus on ASD methods. Methods relevant for this thesis are explained in greater detail. Chapter 3 discusses different types of Free and Open Source Software and the characteristics of Free and Open Source Software Development. The Free and Open Source Software project, which is examined in this thesis, is introduced in Chapter 4. Related work is presented in Chapter 5. Chapter 6 introduces the research methodology used in this thesis. Chapter 7 deals with the lessons learned during the introduction of processes and tools in the whole organization. The introduction of these processes and tools enabled the Action Research cycles conducted with one team, which are described in Chapter 8. The results and threats to validity of the Action Research are presented in Chapter 9. In Chapter 10 conclusions and future work are discussed. Appendix A includes the publications about the research and the author's contributions to them, as well as the questionnaires B used for this thesis. Appendix C contains supplementary material.

2. Agile Software Development Methods

There exist various approaches to software development, more traditional ones like the waterfall or V-model and newer ones like ASD. All of them have something in common, the four control variables. In software development projects four variables are important to control the outcome. These four are:

- Cost
- Time
- Quality and
- Scope

Managers and customers should determine the values of three of these four, the project team should choose the fourth value. If managers or customers try to prescribe all four values, usually quality of the software suffers. Another effect of trying to control all variables is, that the product is delivered late, which happens quite often in software development. The CHAOS Report¹ shows that still many software projects fail or are challenged and only about a third is successful. Figure 2.1 shows the modern resolution for all software projects. Modern resolution means that the project was finished not only on time and within the budget, but also with a satisfactory result. Figure 2.1 shows an up-and-down in the percentage of successful, challenged and failed projects, but what remains the same is the general distribution of projects. In general the majority, around 50%, of all projects is challenged, around a third is successful and all other projects fail. For agile projects the picture is different. Figure 2.2 compares agile and waterfall projects and agile projects are far more often successful than

¹<http://www.standishgroup.com/> – retrieved on 10.01.2017

2. Agile Software Development Methods

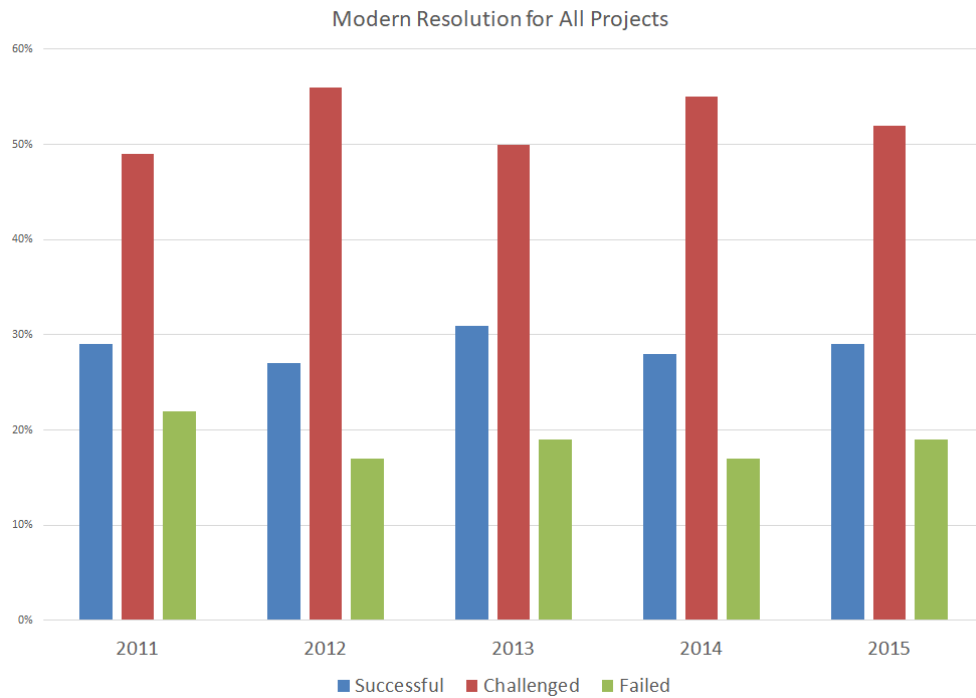


Figure 2.1.: The modern resolution (on time, on budget, with a satisfactory result) of all software projects from FY2011-2015 within the new CHAOS database [Source: adapted from InfoQ².]

waterfall projects (around 40% versus around 10%) and are also far less likely to fail (around 10% versus around 30%). The waterfall model is one of the traditional software development methods. It was introduced in the 1970s (Royce, 1970). The development stages are passed through sequentially, see Figure 2.3.

Traditional software development methods usually assume all four variables, cost, time, quality and scope, are fixed and all requirements are known in advance (Boehm, 2002). This often results in failed or challenged projects (see Figure 2.2). Budget and/or time are overdrawn, the software lacks in quality or not all functionality is implemented. The assumption that all requirements are and can be known in advance is questionable and even if that were true, often development models are not adhered to (Truex,

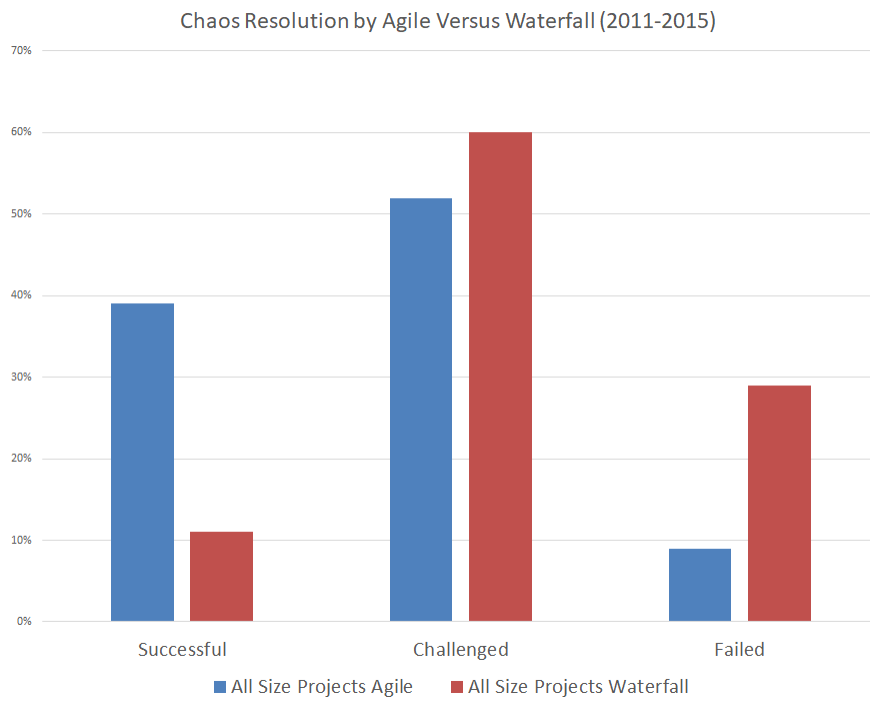


Figure 2.2.: The modern resolution (on time, on budget, with a satisfactory result) of software projects from FY2011-2015 segmented by the agile process and waterfall method within the new CHAOS database. Total number of software projects is over 10,000. [Source: adapted from InfoQ³.]

2. Agile Software Development Methods

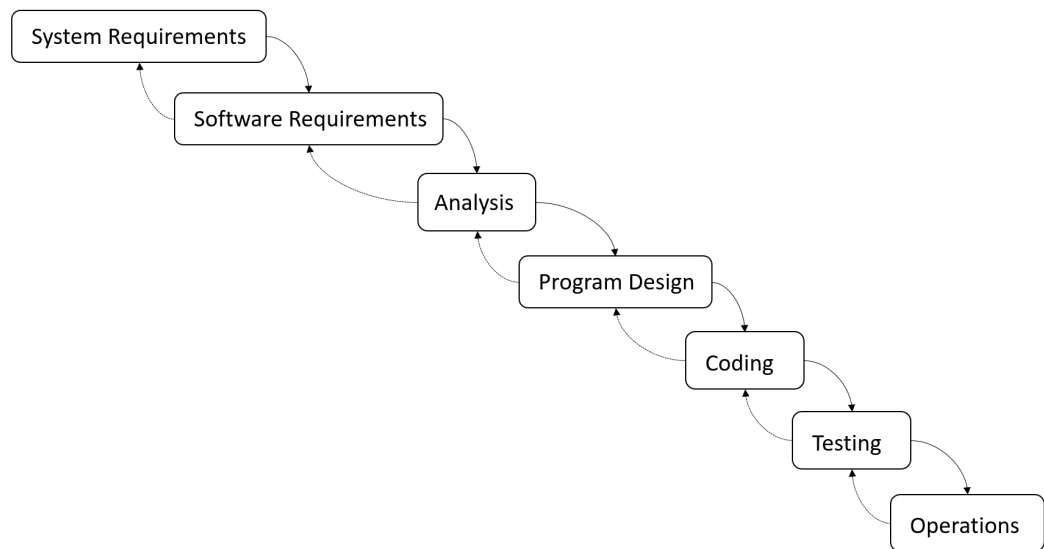


Figure 2.3.: Waterfall Model of Software Development [Source: adapted from Royce (1970).]

Baskerville, and Travis, 2000), at least not strictly. This additionally diminishes the chances to successfully finish a project. This is why other development methods like ASD were and are still developed.

Agile software development frameworks include but are not limited to:

- Adaptive software development (Highsmith, 2000)
- Agile unified process (AgileUnifiedProcess, 2006)
- Agile modeling (AgileModeling, 2017)
- Agile testing (Crispin and Gregory, 2009)
- Crystal clear methods (Cockburn, 2004)
- Disciplined Agile Delivery (Ambler and Lines, 2012)
- Dynamic systems development method (DSDMConsortium, 2008)
- eXtreme Programming (XP) (Andres and Beck, 1999; Jeffries, Anderson, and Hendrickson, 2000; Auer and Miller, 2002; Beck and Andres, 2004)
- Feature driven development (Palmer and Felsing, 2001)

- Kanban (Anderson, 2010; Benson and DeMaria, 2011; Kniberg, 2011; Leopold and Kaltenecker, 2013)
- Lean software development (Ohno, 1988; Poppendieck, 2007)
- Rational unified process (Kruchten, 2004)
- Scrum (Schwaber and Beedle, 2001; Schwaber and Sutherland, 2016; Kniberg and Skarin, 2010)
- Scrumban (Ladas, 2008; Reddy, 2015)

All ASD frameworks aim to counterbalance the decrease of quality and failing projects. They also embrace changes in requirements during the whole product life-cycle (J. Highsmith and A. Cockburn, 2001), which traditional software development models do not. The roots of the agile evolution (Abrahamsson et al., 2003) can be found in the late in 1990s when XP (Andres and Beck, 1999) was introduced. In 2001 the publication of the agile manifesto (Beck, Beedle, et al., 2001) followed. A group of 17 representatives from XP, Scrum, Crystal, Feature-Driven Development and other agile methods formulated and signed the manifesto around February 13 2001 in Utah.

The Agile Manifesto (Beck, Beedle, et al., 2001) states:

“We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
 Working software over comprehensive documentation
 Customer collaboration over contract negotiation
 Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.”

ASD keeps scope or time negotiable so quality is not sacrificed to meet deadlines or budget needs. It demands a constant information flow between development teams, customers and other stakeholders so that problems are detected earlier and countermeasures can be taken. Feedback is an integral part, as well as keeping everything as simple as possible and constantly improving the design, code and tests.

2. Agile Software Development Methods

Changes in requirements are expected and welcomed. This is especially important in today's fast paced software development environment, where today's idea can be tomorrow's mobile application. Users are used to bug fixes and new features being released on a regular basis and not only once a year. Web pages have to be able to serve different kinds and sizes of devices with a consistent look and feel on all platforms. Responsive web design (Bryant and Jones, 2012) is a prerequisite.

Agile projects can fail earlier and restart faster, which makes them more suitable for today's fast-paced software release cycles and also more successful, see Figure 2.2.

So no wonder ASD is increasingly popular in industry. The Annual State of Agile Report from VersionOne (now called CollabNet VersionOne), a company providing enterprise value stream management solutions, shows this every year (VersionOne, 2011; VersionOne, 2012; VersionOne, 2013; VersionOne, 2014; VersionOne, 2015). 70% of VersionOne (2015) respondents use Scrum or Scrum/XP hybrids at team level. And the number of respondents using Kanban grew by 8% from 2014 to 2015 (2015: 39%).

The following sections will describe the software development methods relevant for this thesis, XP and Kanban, in greater detail.

2.1. Extreme Programming (XP)

All XP values, principles, and practices described in this section are taken from Andres and Beck (1999) and the Shodan 2.0 Input Metric Survey in Williams, Krebs, and Layman (2004). This survey was also used to ask study participants about their use of agile practices. Differences to Beck and Andres (2004) will only be mentioned briefly, because the FOSS project Catrobat uses only Andres and Beck (1999).

To better understand the differences and the relationships between values, principles and practices, we will use Beck and Andres (2004) descriptions.

2.1. Extreme Programming (XP)

“Values are the roots of the things we like and don’t like in a situation. [...] Values are the large-scale criteria we use to judge what we see, think, and do. [...] Values bring purpose to practices.” (Beck and Andres, 2004)

Without a purpose and without explicitly stated values, practices might become meaningless routine tasks only performed, because one has to or out of habit. If employees do not understand and support the values behind a practice or if they do not know what a practice is supposed to achieve, they often perform it half-heartedly or stop using the practice if they can get away with it (Beck and Andres, 2004).

“Just as values bring purpose to practices, practices bring accountability to values.” (Beck and Andres, 2004)

Practices are concrete, clear but depend on the given situation, whereas values are universal. Principles bridge the gap between practices and values.

“Principles are domain-specific guidelines for life.” (Beck and Andres, 2004)

2.1.1. XP Values

The four values of XP are (Andres and Beck, 1999):

- Communication
- Simplicity
- Feedback
- Courage
- Beck and Andres (2004) contains a fifth value: Respect

Many problems in projects originate from communication problems. Important information is not passed on or information is misinterpreted. XP employs practices, which require communication, e.g., pair programming, task estimation (Andres and Beck, 1999).

Simplicity is the art of building the simplest thing that could possibly work and not more. Only the requirements of today are met and no guesswork

2. Agile Software Development Methods

about the future is done, because one cannot know if one will really need this functionality later on. You Ain't Gonna Need It (YAGNI) expresses this value (Andres and Beck, 1999).

Feedback means feedback of the system which is developed. Unit tests provide feedback about the system's state to the developers. Task estimation provides feedback to customers and functional tests also provide feedback about the system. Courage must be combined with the other three to be of value, otherwise it will result in hacking (Andres and Beck, 1999).

2.1.2. XP Principles

The five fundamental principles of XP are (Andres and Beck, 1999):

- Rapid feedback
- Assume simplicity
- Incremental change
- Embracing change
- Quality work

The time between action and feedback is important, the shorter the time period, the better. So system feedback delivered within seconds or minutes is better than within days, weeks, or months. So the test suite should be executable as fast as possible. Customer feedback should be fed back within days and not weeks or months. Every problem should be solved as simple as possible. Big changes usually do not work, small incremental changes do. Embracing change means keeping many options open (Andres and Beck, 1999).

Quality work is important because nobody wants to do bad work. People who are forced to deliver substandard work become frustrated and demotivated (Andres and Beck, 1999).

“Projects succeed when good people are allowed to do their best work.” (Crispin and Gregory, 2009)

Other (less fundamental) principles are (Andres and Beck, 1999):

- Teach learning

2.1. Extreme Programming (XP)

- Small initial investment
- Play to win
- Concrete experiments
- Open, honest communication
- Work with people's instincts, not against them
- Accepted responsibility
- Local adaption
- Travel light
- Honest measurement

The new principles in Beck and Andres (2004) are:

- Humanity
- Economics
- Mutual benefit
- Self-similarity
- Improvement
- Diversity
- Reflection
- Flow
- Opportunity
- Redundancy
- Failure
- Quality
- Baby steps

2.1.3. XP Practices

This section lists and explains the XP practices in the version (Andres and Beck, 1999), which the researched FOSS project uses. The updated practices from the second edition (Beck and Andres, 2004), will only be mentioned in brief detail.

2. Agile Software Development Methods

Release Planning / Planning Game

During release planning/planning games all stakeholders, customers and developers, determine the next release through discussion. Development issues are moved in or out of the next release based on their business value and costs. The technical effort necessary to implement the issue is estimated by the developers. The plan is continually refined (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Small / Short Releases

The focus is on delivering small meaningful frequent releases in a very short cycle. It is easier to plan for a few months than for half a year or more and frequent releases allow for more frequent feedback from the customers to the developers. It is also easier to adapt the following releases if deviations from the plan are detected by the customer (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Metaphor / System of Names

A single overarching metaphor, e.g., assembly line, spreadsheet, is used to describe the software. If there exists no suitable metaphor for the software a system of names is used, which describes all system components in a consistent manner. A metaphor or system of names helps to understand the basic elements of the project and their relationships to each other. It is also easier to explain ideas and concepts to the customer (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Simple Design

At first implement “the simplest thing that could possibly work” (Williams, Krebs, and Layman, 2004) and do not implement something only based on the assumption that it might be needed later. The YAGNI principle and the following quote from Knuth (1979) are often associated with this practice,

2.1. Extreme Programming (XP)

although often only the bold highlighted part of the quote is cited, leaving out the reference to efficiency.

“There is no doubt that the grail of efficiency leads to abuse. Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%.” (Knuth, 1979)

The design only describes what is needed and nothing more, future iterations are not considered for the current iteration. Simple design passes all the tests, does not have duplicate code, states every intention important to programmers and has the fewest possible classes and methods (Andres and Beck, 1999).

Testing

Every program feature has to have an automated test, unit tests written by developers and functional tests written by customers (Andres and Beck, 1999). Williams, Krebs, and Layman (2004) divided the practice of testing into unit tests, customer acceptance tests and TDD for their Shodan 2.0 Input Metric Survey.

- **Unit Tests**

Automated unit tests must be written and must run flawlessly otherwise development cannot continue (Andres and Beck, 1999). They verify if the code is working correctly and help determine if the integration of new code would break existing code (Williams, Krebs, and Layman, 2004).

- **Acceptance Tests**

All customer acceptance tests must pass before a product can be delivered to the customer (Williams, Krebs, and Layman, 2004).

2. Agile Software Development Methods

- **Test First Design / Test Driven Development (TDD)**

When TDD is employed tests are written before the actual code is implemented. The three steps of TDD according to Beck (2003) are red/green/refactor.

- “Red — write a little test that doesn’t work, perhaps doesn’t even compile at first”
- “Green — make the test work quickly, committing whatever sins necessary in the process”
- “Refactor — eliminate all the duplication created in just getting the test to work”

TDD increases the confidence in the code and code is changed more easily because errors or side-effects are detected through tests which fail.

Refactoring

Refactoring is the practice of rewriting or restructuring the code without changing the behavior to improve maintainability, performance, simplicity, flexibility, understandability, or code quality, e.g. code duplications are removed (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Pair Programming

When pair programming two people share one computer, one keyboard and one mouse. The person with keyboard and mouse, the driver, is thinking tactically, how this part of the code is implemented best. The second person, the navigator, thinks more strategically, if the approach will work, if some test cases might not work yet, if the current problem could be dissolved by simplifying the whole system. Programming roles, driver and navigator, and programming pairs should be switched on a regular basis (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004). Pair programming leads to higher code quality (Williams, 2000).

2.1. Extreme Programming (XP)

Collective Code Ownership

Collective code ownership means “Anyone can change any code anywhere in the system at any time.” (Andres and Beck, 1999). People do not have to wait for the code owner or expert to change a piece of code. Everybody knows every part of the system and is allowed to change the code if needed. Knowledge is spread throughout the team and it is no problem if an expert is busy, sick, on vacation or leaving the team.

Continuous Integration

Code is integrated and tested at least daily, ideally every few hours. Code integration and synchronization issues are minimized. A build machine can be used to build and test new code automatically and can notify developers if a build fails (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Sustainable Pace / 40h Week

People cannot work overtime for many weeks without losing energy, creativeness, productivity, confidence and morale. So it is important to keep the amount of working hours on a normal level (around 40 hours per week) and to also take vacations to recharge one’s batteries. If the team has to work overtime on a regular basis, there is probably a serious problem on the project (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

On-Site Customer

To ensure the product meets the needs of the customer, direct quick access to him or her is very important. The customer creates the requirements of the system, answers questions, gives feedback, and prioritizes what features should be built first. He or she is involved in release planning and is accessible to the team, ideally in person, or via synchronous communication tools (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

2. Agile Software Development Methods

Coding Standards

If collective code ownership, pair programming, and refactoring are practiced, a common set of coding practices to support those practices is needed. The team decides on and uses a coding standard collectively (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004).

Introspection

For their Shodan 2.0 Input Metric Survey Williams, Krebs, and Layman (2004) created questions concerning additional topics important for teams.

- **Stand-Up Meeting**

This question checks if stand up meetings are held everyday and are shorter than 15 minutes. It checks if they are to the point and concerns and successes are discussed openly (Williams, Krebs, and Layman, 2004).

- **Lessons Learned**

This question checks if the team reviews after every release how it can improve (Williams, Krebs, and Layman, 2004).

- **Growth**

This question checks if the team ensures that their skills are up to date.

“If you’re not learning, your (sic) falling behind!” (Williams, Krebs, and Layman, 2004).

- **Morale**

This question checks if XP is enjoyable (Williams, Krebs, and Layman, 2004).

- **Artifact Reduction**

This question checks if XP helps the team to have fewer or thinner versions of artifacts from classic techniques (Williams, Krebs, and Layman, 2004).

XP Practices from the Second Edition

In Beck and Andres (2004) the practices are divided into primary practices, which can be used independently, e.g. one can start using them in any order, and corollary practices, which should only be used after mastering the primary practices.

Practices identical to Andres and Beck (1999) are italic. For evolutionary practices (Beck and Andres, 2004), which are similar or related to standard practices (Andres and Beck, 1999) the according standard practice is mentioned in parentheses. According to Beck and Andres (2004) the primary practices are:

- Sit together
- Whole team
- Informative workspace
- Energized work (Sustainable pace)
- *Pair programming*
- Stories
- Weekly cycle (Planning game)
- Quarterly cycle (Planning game)
- Slack (Sustainable pace)
- Ten-minute build (Continuous Integration)
- *Continuous Integration*
- *Test-first programming*
- Incremental design (Simple design)

Most evolutionary practices are similar to the traditional practices, but names have changed and some traditional practices have been split into several practices. The practices metaphor and coding standard vanished.

The corollary practices are (Beck and Andres, 2004):

- Real customer involvement (On-site customer)
- Incremental deployment (Small/short releases)
- Team continuity
- Shrinking teams
- Root-cause analysis
- Shared code (Collective code ownership)

2. Agile Software Development Methods

- Code and tests
- Single code base
- Daily deployment
- Negotiated scope contract
- Pay-per-use

2.2. Kanban

Kanban is a method for knowledge work, which includes software development. There is an ongoing discussion if Kanban is ASD or rather a form of lean software development, but as this is not the focus of this thesis, this issue will not be discussed.

Kanban was originally developed for software development but has also been applied to other areas, e.g. human resources, marketing, organizations strategy. It can even be used to organize one's whole life with a personal Kanban board (Benson and DeMaria, 2011).

Kanban is inspired by the Toyota Production System (Ohno, 1988) and was first described by Anderson (2010). Anderson (2010) explains how his approach evolved from a theory of constraints approach (Goldratt and Cox, 1992) into the Kanban method.

Kanban is about changing organizations and not about optimizing people, which would not lead to huge improvements anyway, because according to Deming (2000)

“I should estimate that in my experience most troubles and most possibilities for improvement add up to the proportions something like this: 94% belongs to the system (responsibility of management) 6% special”

the longer lever for improvements lies within the organization.

Kanban is based on four principles and six practices, which will be explained in the following sections. These principles and practices are designed to initiate and constantly support evolutionary change in organiza-

2.2. Kanban

tions and help establish a culture of kaizen (continuous improvement) (Anderson, 2010; Leopold and Kaltenecker, 2013).

Kanban is not a silver bullet, it is not the solution to all problems, it supports problem solving by identifying existing problems. Kanban can be used on different flight levels of an organization. Flight level 1 is the operational level, where teams work. Level 2 is coordination and level 3 is strategic portfolio management (Leopold, 2017). Kanban never stops, there is always something to improve. People do not suffer changes, they are driving the change. Stakeholders, management and team are together in the same boat.

The “One day in Kanban land” comic ⁴ illustrates nicely how Kanban is supposed to work.

2.2.1. Kanban Principles

The four principles of Kanban are (Anderson, 2010; Leopold and Kaltenecker, 2013):

- Start where you are
- Pursue incremental, evolutionary change
- Respect the current processes, roles, responsibilities and titles
- Promote leadership at all levels

This means Kanban can be started without week long trainings for the whole organization, although a certain training is advantageous. Teams or organizations do not have to overthrow the whole development process. Teams and organizations can start right away with small incremental steps. They should not try to change everything at the same time, rather tackle one problem at a time, starting with the most important one. If the most important one is solved, they should move to the next one, and so on and so forth. Roles, titles, responsibilities and processes are not changed “overnight”. Neither do people lose their “power or reputation” because they get demoted from some kind of manager to team member, nor is “the

⁴<http://blog.crisp.se/2009/06/26/henrikkniberg/1246053060000> – retrieved on 17.05.2017

2. Agile Software Development Methods

boss” the all knowing all deciding entity. Everyone is invited to put their head into the process and contribute their knowledge and expertise. The people affected by the processes decide how to change them for the benefit of the organization and the people involved (Leopold, 2012; Leopold and Kaltenecker, 2013).

2.2.2. Little’s Law

Little’s Law will be used to explain the importance of limiting Work In Progress (WIP), therefore it will be explained shortly.

“Little’s Law says that, under steady state conditions, the average number of items in a queuing system equals the average rate at which items arrive multiplied by the average time that an item spends in the system.” (Little and Graves, 2008)

Little’s Law:

$$L = \lambda W \quad (2.1)$$

With:

L = “average number of items in the queuing system”

W = “average waiting time in the system for an item”

λ = “average number of items arriving per unit time”

In operations management and other areas the law is also used in the following form, e.g., Hopp and Spearman (2000):

$$Cycletime = \frac{WIP}{Throughput} \quad (2.2)$$

With:

Cycle time = “the average time from release of a job at the beginning of the routing until it reaches an inventory point at the end of the routing (that is, the time the part spends as WIP” (Hopp and Spearman, 2000)

WIP = “the inventory between the start and end points of a

product routing” (Hopp and Spearman, 2000)

Throughput = “the average output of a production process (machine, workstation, line, plant) per unit time” (Hopp and Spearman, 2000)

This form is equivalent to Little’s Law with throughput = λ , WIP = L and Cycle Time = W (Little and Graves, 2008).

$$WIP = Throughput * CycleTime \quad (2.3)$$

2.2.3. Kanban Practices

In this section all Kanban practices are explained in detail.

Visualize the Workflow

Knowledge work is usually invisible, inside people’s heads. Kanban supports people and organizations to make knowledge work visible. It strives to establish a continuous flow of work through the systems and to make problems in the flow visible. WIP limits support the identification of bottlenecks. Work flowing through the system is not achieved through pushing work items to the next person, instead persons from a succeeding step pull work from a preceding step in the workflow, as soon as they are ready.

Limit Work in Progress

Limiting WIP is one of the basic foundations of Kanban. Figure 2.4 shows a Kanban board with WIP limits. In column *A* the WIP limit is broken because there are already two cards in the column. The WIP limit does not allow a third card in *A* so the third card, with a red cross, cannot be moved there and has to be moved back to *To do*. If one does not limit WIP it will be difficult to detect bottlenecks. They are where unfinished work piles up while other work stations have idle time. In knowledge work these work

2. Agile Software Development Methods

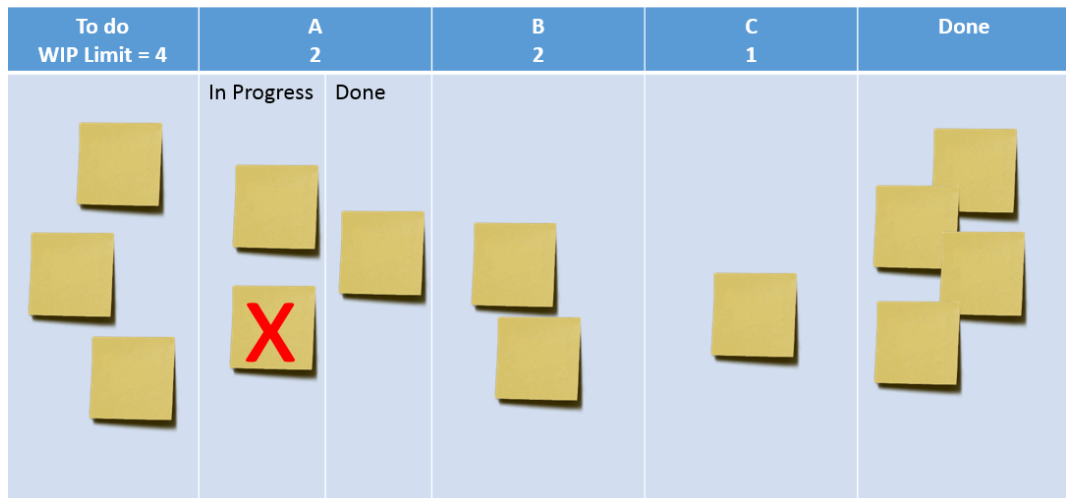


Figure 2.4.: Example of a Kanban board with WIP Limits. The card with the red cross cannot be pulled into column A, because this would break the WIP limit of two for this column.

stations would be colleagues waiting for their successor to finish their work, so that they can pull a new work item from their predecessor. The workflow is blocked and bottlenecks become visible to everyone. Because WIP limits prevent people from continuing their work, motivation is high to resolve bottlenecks immediately. WIP limits are a prerequisite for creating a pull system (Leopold, 2012; Leopold and Kaltenecker, 2013).

Another reason why this practice is important is, that unfinished work does not create revenue. The longer the cycle time the longer until a product can be sold and shipped. So economically speaking one finished piece of work is more valuable than ten unfinished pieces of work. As Little's Law (see Section 2.2.2) states one can influence cycle time only by reducing the WIP limit or by increasing a system's throughput. While increasing throughput is usually very hard to do, it is rather easy to reduce the WIP limit. So this should be the lever used to reduce cycle time (Leopold, 2012; Leopold and Kaltenecker, 2013).

Limiting WIP has another important advantage. It can improve relations to customers. If cycle time is reduced and feature requests which exceed












To do	A		B	C	Done
	In Progress	Done			
Type of work 1 (e.g. Features)					
Type of work 2 (e.g. Bugs)					
Type of work 3 (e.g. Change requests)					
Type of work 4 (e.g. Support)					

Figure 2.5.: Example of a Kanban board with swim lanes for different types of work.

the WIP limit are not accepted, promises made to customers can be kept more easily. As a consequence, customers will put more trust in the organization, which keeps promises, than in those which constantly fail to keep them, which is rather usual in software development. One goal of Kanban is to only make promises one can keep (Leopold, 2012; Leopold and Kaltenecker, 2013).

Manage Flow

Work should flow through the Kanban system as fast as possible. Blockades and bottlenecks need to be dealt with as quickly as possible. The performance of the Kanban system, not the employees, has to be measured, so one can detect if changes to the system had an effect on the workflow. If the system's performance is known, agreements with stakeholders are more likely to be met, which is important to build trusting relationships. If the Kanban system satisfies the current needs and demands of the business,

2. Agile Software Development Methods

no changes need to be made until new bottlenecks appear. However, in the more likely case, that the system does not meet all needs and demands, one has to look for further possibilities to improve it (Leopold, 2012; Leopold and Kaltenecker, 2013).

To manage the flow one has to know the performance of the system and one has to determine which types of work have to be done and how urgent they are, respectively which service level they have. Based on types of works and service levels Service Level Agreements (SLA) can be made. Within these agreements cycle times for specific work types or service levels are guaranteed. For an example see Figure 2.5. Every lane on the Kanban board could have a different guaranteed cycle time, e.g., one week to resolve a bug (Leopold, 2012; Leopold and Kaltenecker, 2013).

Communication is key, so all actions taken regarding measuring and controlling the flow go hand in hand. Team members have to communicate to ensure a constant workflow (Leopold, 2012; Leopold and Kaltenecker, 2013).

Make Process Policies Explicit

All teams use rules to organize their work. Often they are not written down or communicated clearly. (New) Team members only learn through experience that, e.g., Jane always writes the tricky tests, John always has the last word in discussions and so on. Kanban wants to make these informal rules transparent and visible to everybody. The team decides their rules jointly and communicates them clearly to everyone involved. Everybody, including stakeholders, has to stick to these rules, unless they become obsolete or have errors in them. Then the rules have to be changed. If standards and rules are not changed, when necessary, continuous improvement stops. Only if all stakeholders and the team adhere to the rules, mistakes in the rules can be recognized and repaired (Leopold, 2012; Leopold and Kaltenecker, 2013).

Another advantage of transparent roles and policies is, that teams can discuss problems more objectively and less emotional, because discussions concern the rules and not the mistakes of individual persons. Blaming

should cease to exist, although it might take a while and some professional moderating help to forsake this habit (Leopold, 2012; Leopold and Kaltenecker, 2013).

Implement Feedback Loops

Implement feedback loops on team level, e.g., daily stand-up meetings, retrospective, but also on a higher level along the whole value chain. Feedback gives everyone the opportunity to learn and learning is a prerequisite for continuous improvement (Leopold, 2012; Leopold and Kaltenecker, 2013).

In ASD feedback is a central part, e.g., tests provide feedback about the code, iteration releases provide feedback about the current condition of the project, and retrospectives provide feedback about the way the team works (Crispin and Gregory, 2009).

“If you don’t have meaningful feedback, then you’re not agile. You’re just in a new form of chaos.” (Crispin and Gregory, 2009)

Improve Collaboratively, Evolve Experimentally (using models and the scientific method)

Existing methodologies and models can and should be used to investigate and solve problems. So not everyone has to start from scratch. Many problems occur in every system and there are models to investigate such problems. Every organization should use the models and methodologies relevant for it and the current problem. Kanban does not prescribe specific models or how they should be applied. It only suggests that something should be done to improve things (Leopold, 2012; Leopold and Kaltenecker, 2013).

3. Free Libre Open Source Software

FOSSD offers some advantages, like cost savings, high software quality, a global testing pool, independent peer review and rapid development time, but it is no silver bullet, which will resolve all problems of the software industry (Scacchi et al., 2006; Fitzgerald, 2011). FOSSD has its own problems, e.g., usability is often of low concern (Levesque, 2004), stability and reliability are unpredictable, there is often a lack of documentation (Levesque, 2004; Fitzgerald, 2011) and code quality is sometimes comparable to proprietary software (Stamelos et al., 2002; Rusovan, Lawford, and Parnas, 2005).

Nevertheless FOSS proved that community-driven successful software development is possible in a distributed environment and can compete with traditionally developed software. Some of the large FOSS projects have numerous users and millions of lines of source code, e.g. the GNU Compiler Collection¹, OpenOffice², Eclipse³, KDE⁴, GNOME user interface packages⁵, Linux distributions⁶, the Apache web server⁷, and the Emacs text editor⁸. Some FOSS projects attract even people outside the Information and Communications Technology (ICT) community, e.g. GIMP drawing and image

¹<https://gcc.gnu.org/> – retrieved on 30.05.2017

²<https://www.openoffice.org> – retrieved on 30.05.2017

³<https://www.eclipse.org/> – retrieved on 30.05.2017

⁴<https://www.kde.org/> – retrieved on 30.05.2017

⁵<https://www.gnome.org/> – retrieved on 30.05.2017

⁶<https://www.linux.org/> – retrieved on 30.05.2017

⁷<https://httpd.apache.org/> – retrieved on 30.05.2017

⁸<https://www.gnu.org/s/emacs/> – retrieved on 30.05.2017

3. Free Libre Open Source Software

editor⁹ and Mozilla Firefox¹⁰.

Some FOSS projects even grow big enough to start a foundation or similar organizations, and receive enough donations or have other incomes, so they can pay some of their contributors. To support projects on this journey the Software Freedom Conservancy Inc.¹¹ offers their member projects services like donation handling, payment of key developers from these donations, fiscal oversight, taking care of copyrights, trademarks, and domain names¹².

And although various research exists about free or open source software, is its not known how many FOSS projects are out there. There are many hosting sites, some of the most popular ones include Github.com, GitLab.com, Assembla.com, SourceForge.net, Google Code, CodePlex.com, and Savannah.gnu.org, and it is impossible to determine the exact number of projects and contributors (Weber, 2004).

There also exist various terms for free or open source software, which describe the same set of software, but differ in some details and there is an ongoing controversy if the terms are interchangeable or not (Kelty, 2008)¹³¹⁴¹⁵. In this chapter the different terms, their definitions, the developing methodology, characteristics of FOSS projects and the motivation of contributors will be explained.

3.1. Free and Open Source Software (FOSS) Definitions

In this section Free Software (FS), Open Source Software (OSS), Libre Software (LS), Free, Libre and Open Source Software (FLOSS) and their relation

⁹<https://www.gimp.org/> – retrieved on 30.05.2017

¹⁰<https://www.mozilla.org/en-US/firefox/> – retrieved on 30.05.2017

¹¹<https://sfconservancy.org> – retrieved on 06.06.2017

¹²<https://sfconservancy.org/about/> – retrieved on 06.06.2017

¹³<https://fsfe.org/about/basics/freesoftware.en.html> – retrieved on 19.05.2017

¹⁴<https://opensource.org/faq#free-software> – retrieved on 19.05.2017

¹⁵<https://fsfe.org/freesoftware/basics/comparison.en.html> – retrieved on 19.05.2017

3.1. Free and Open Source Software (FOSS) Definitions

to each other will be described.

3.1.1. Free Software

The Free Software Definition (FSD) was first described in 1986 by Richard Stallman and contained the following two points

“The word ‘free’ in our name does not refer to price; it refers to freedom. First, the freedom to copy a program and redistribute it to your neighbors, so that they can use it as well as you. Second, the freedom to change a program, so that you can control it instead of it controlling you; for this, the source code must be made available to you.” (Stallman, 1986).

In 2010 he wrote the second edition (Stallman, 2010) with the following four freedoms, which can also be found online at the Free Software Foundation (FSF) website¹⁶:

- “The freedom to run the program, for any purpose.”
- “The freedom to study how the program works, and adapt it to your needs.”
- “The freedom to redistribute copies so you can help your neighbor.”
- “The freedom to improve the program, and release your improvements to the public, so that the whole community benefits.”

Only if all four freedoms are granted and people do not have to ask for permission or pay for any freedom, the software is regarded as FS (Stallman, 2010). FS is often compared to freedom of speech, “Because of these four freedoms, Free Software offers freedom to learn, freedom to teach, freedom of competition, freedom of speech and freedom of choice.” (FSFE, 2017a)

Everybody can use all freedoms or none, whatever the person wants. The freedoms include commercial use as well, so if a software is forbidding commercial use, it is not FS (Stallman, 2010).

¹⁶<https://fsfe.org/about/basics/freesoftware.en.html> – retrieved on 19.05.2017

3. Free Libre Open Source Software

Although Free Software is ambiguous in English the term is still used, because, according to the Free Software Foundation Europe (FSFE) free is easier to understand than open source and accessible source code only fulfills two of the four freedoms (FSFE, 2017a). According to the FSFE Free Software is harder to abuse than Open Source Software (FSFE, 2017a). The initiative to trademark *Open Source* for Free Software failed, so companies can call their software Open Source Software, even if only some parts of their source code are accessible. This is misleading because it suggests that that the OSS principle is applied, which is not the case (FSFE, 2017b). Another reason for using the term Free Software is, that the Free Software Definition of the FSF is currently the clearest definition and offers freedom (FSFE, 2017a). The FSFE claims that Free Software provides a philosophy from which companies can learn and profit, whereas Open Source Software “only” provides a technical model (FSFE, 2017a).

Free Software is also seen as a social movement, while Open Source Software Development (OSSD) is described as a software development methodology by Stallman and the FSF (Stallman and Gay, 2009).

3.1.2. Open Source Software

The Open Source Initiative (OSI) was set up in 1998 as a marketing campaign for FS. According to OSI both FS and OSS mean software with licenses, which guarantee certain freedoms and only differ in the way how they promote it. For OSI software freedom was more a practical issue whereas for FSF it was rather an ideological one, see the quote from the 1998 OSI FAQ page:

“The Open Source Initiative is a marketing program for free software. It’s a pitch for ‘free software’ on solid pragmatic grounds rather than ideological tub-thumping. The winning substance has not changed, the losing attitude and symbolism have.”¹⁷

Open Source (OS) was also seen as solution to the ambiguity of “free” in the English language. The author of the famous work *The Cathedral*

¹⁷<https://web.archive.org/web/20021217003716/http://www.opensource.org/advocacy/faq.html> – retrieved on 23.05.2017

3.1. Free and Open Source Software (FOSS) Definitions

and the Bazaar, Eric Raymond, also prefers the term OSS over FS because free is ambiguous and makes companies nervous (Raymond, 1998a; Fink, 2003) partly because of the viral nature of some “free” licenses (see Section 3.1.3).

Open source does not only mean the source code is available, it also means that the open source distribution terms in the license have to adhere to the Open Source Definition (OSD).

Open Source Definition The OSD was derived from the Debian Free Software Guidelines (Public Interest, 2004) and guarantees the four freedoms (see Section 3.1.1) as well. The OSD contains the following criteria (Perens, 1999; OpenSourceInitiative, 2007):

1. *Free redistribution* means that the license must not restrict or ask for fees for publishing or selling the software as part of a software collection with programs from several sources (Perens, 1999; OpenSourceInitiative, 2007).
2. *Source code* is included in the program and it is allowed to distribute the program in source code and compiled form. This enables users to study, experiment with, and modify the code (Perens, 1999; OpenSourceInitiative, 2007).
3. *Derived works* have to be allowed and it has to be allowed to distribute them under the same license as the original program (Perens, 1999; OpenSourceInitiative, 2007).
4. *Integrity of author's source code* enables the author of a program to restrict distribution of modified source code, if and only if the license allows other developers to distribute the source code together with “patch files”, which modify the program at build time. Derived works must have a different name or version number, if the license requires it, but it has to be allowed to distribute modified source code in a compiled form (Perens, 1999; OpenSourceInitiative, 2007).
5. *No discrimination against persons or groups* is allowed (Perens, 1999; OpenSourceInitiative, 2007). Everybody can use the software under the same terms.
6. *No discrimination against fields of endeavor* is allowed, e.g. commercial use must not be forbidden (Perens, 1999; OpenSourceInitiative, 2007).

3. Free Libre Open Source Software

7. *Distribution of license* means that no additional licenses, e.g. a Non Disclosure Agreement (NDA), can be added to close up the software (Perens, 1999; OpenSourceInitiative, 2007).
8. *License must not be specific to a product* closes another loophole regarding licenses. Redistributing a modified software distribution of a program grants everyone the same rights as the original software distribution (Perens, 1999; OpenSourceInitiative, 2007).
9. *License must not restrict other software* in its distribution, e.g., it must not require that all software distributed on the same medium is open source (Perens, 1999; OpenSourceInitiative, 2007).
10. *License must be technology-neutral* and not be based on any specific technology or type of interface (Perens, 1999; OpenSourceInitiative, 2007).

3.1.3. Free Software versus Open Software

Free Software and Open Source Software Licenses

One of the main differences between FS and OSS stems from licenses: FS usually uses the GNU general public licenses (GPL), whereas OSS may use GPL or other licenses, which allow the integration of non FS (Scacchi et al., 2006).

Otherwise the movements are similar in many aspects, which is also visible in literature. The GNU Project and the Free Software Foundation (GNU and FSF, 2017) and Scacchi et al. (2006) describe the relation between FS and OSS like this:

“[...]nearly all free software is open source, and nearly all open source software is free.”(GNU and FSF, 2017) and “[...]free software is always available as OSS, but OSS is not always free software”(Scacchi et al., 2006).

3.1. Free and Open Source Software (FOSS) Definitions

Copyleft

A common misunderstanding is that copyleft may be the difference between FS and OSS, but it is not. Copyleft is protective about the four freedoms. Even changes in the software do not make it possible to restrict the original software, e.g. make proprietary software out of FS or OSS. Copyleft licenses, e.g. the GPL, are sometimes described as having a viral nature because if FS software is incorporated or integrated into another software, this software is then also treated as FS (Scacchi, 2007). Non-protective licenses allow to share the software without the rights to study, share, improve or use the software. Both licenses are allowed under FS and OSS, so this does not qualify as a difference (Schiesle, 2017).

The Developing Model

The developing model is also not a distinguishing property of FS and OSS. Both terms describe the software model and not the development model. It is not a criterion if FS or OSS are developed in an open community or behind closed doors. It matters if the four freedoms or the OSD are fulfilled. Proprietary software can be developed in an open, collaborative development process and FS or OSS can be developed without a community or user interaction (FSFE, 2017a). Details about the FOSS Development Model will be given in Section 3.2.

3.1.4. Other Terms for Free and Open Source Software

Libre Software LS is another term for FS and was coined in 1992 by the European Commission to avoid the ambiguous term “free software” (see Paragraph 3.1.4) and the confusion around FS and OSS (FSFE, 2017b).

Freeware and Shareware The terms freeware or shareware are not clearly defined and are not related to OSS or FS. Users do not have the four freedoms and the conditions of the OSD are not met. The software is only free to use for e.g. private use, a 30 day trial period, a certain geographic area

3. Free Libre Open Source Software

or user group, or users have to pay a license fee etc. The source code is not available.

Free, Libre Open Source Software The terms FOSS and FLOSS are often used to avoid the controversy around FS and OSS, because they include all free, libre and open source software projects.

For this thesis the term FOSS is used to refer to free software, open source software, libre software or FLOSS alike, because the differences and controversy around it are not important for this work.

3.2. The FOSS Development Model / Methodology

One challenge of writing about FOSS, FLOSS, FOSSD, OSS, OSSD, FS, or LS is, that not only several names and definitions are used for FOSS, but more importantly

“There is no globally accepted open source software development process to define how open source software is developed in practice.” (Acuña et al., 2012) and “The FOSS development model does not have a formal, disciplined definition, since it works based predominantly on voluntary collaboration.” (Magdaleno, Werner, and Araujo, 2012)

When talking about the FOSS development model or methodology usually the text of Raymond (2001) “The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary” is mentioned. It compares the usual open collaborative FOSS development model, the bazaar, to the typical closed hierarchical development model of proprietary software, the cathedral. FOSS is an open collaborative software development approach and usually without a formal project management regime, schedule or budget (Scacchi et al., 2006). Source code, other development artifacts and development activities are visible and available on the Internet (Scacchi et al., 2006). Often most of the conversation is open as well,

3.2. The FOSS Development Model / Methodology

as mailing lists and open Internet Relay Chat (IRC) channels are used for communication.

West and O'Mahony (2005) identified the following models, how a FOSS community can be started:

- *Community-initiated projects* are founded by one or more developers independently of a company. Some examples are Linux, GNOME desktop environment project, and the Apache web server.
- *Sponsored or spinout projects* are initiated by releasing previously proprietary code to the public under a FOSS license. Some examples are Netscape founding the Mozilla project, IBM founding the Jikes and Eclipse project, Sun the OpenOffice project, and MySQL AB the MySQL project. Companies either try to build a community around the product or to collaborate with a community to receive feedback from customers (Mäenpää, Kilamo, and Männistö, 2016).

When a project is community-initiated development is often started by a single developer who wants to “scratch a personal itch” (Raymond, 2001; West and O'Mahony, 2005) and most FOSS developers, who join the project, are also end-users of the software. End-users without enough spare time or without software development skills often contribute through bug reporting, writing and maintaining Frequently Asked Questions (FAQ), documentation, giving feedback or suggesting improvements or new features (Scacchi et al., 2006). But how to grow from a one person endeavor to a bazaar? Raymond (2001) identifies some necessary preconditions for successful bazaar-style development:

- Programs cannot be started in bazaar style. The community needs code to test and to play with.
- The program must run and others must see potential in the program.
- Strong attractive basic design is critical, but coordinators must not have exceptional design skills if they are able to recognize good design ideas from others.
- Equally important or maybe even more important than good design skills are good people and communication skills of the project coordinator because they are important for building a strong development community and attracting many people.

3. Free Libre Open Source Software

Raymond (2001) also collected 19 lessons, which give more advice on how to build a successful project. In general respect and reputation are an important part of a FOSS community. Another very important lesson is the importance of tester and user involvement. This is emphasized in lessons 6, 7, 8, 10 and 11. For example, see lesson 10 and 11 from Raymond (2001):

“10. If you treat your beta-testers as if they’re your most valuable resource, they will respond by becoming your most valuable resource.”

“11. The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.”

Gacek and Arief (2004) identified six characteristics of successful FOSS projects. These are:

- An active *Community* with common interests which is developing and/or using the software.
- *Motivation* to contribute for free. Individuals and companies contribute for different reasons. Individuals mostly for personal satisfaction and companies mostly for economic reasons. Both profit from peer recognition when contributing to a FOSS project.
- *Developers are always users.*
- *The process of accepting submissions.*
- *Development improvement cycles.*
- *Code Modularity* is a prerequisite for distributed software development.

The success of FOSS software is highly dependent on the success of its community, they are codependent (Scacchi et al., 2006). This can also be a pitfall for sponsored or spinout projects. Releasing the source code to the public is only a small step. Afterwards it is essential to know how to build a community, how to engage contributors and keep them motivated. It is also important to find a balance between the FOSS and proprietary world (Mäenpää, Kilamo, and Männistö, 2016).

3.3. FOSS Motivation

A number of surveys concern the motivation of FOSS developers. Why do they contribute? The following reasons were determined:

1. *Learning* and sharing one's knowledge about software development is mentioned as providing the greatest benefit (Lakhani, Wolf, et al., 2002; Ye and Kishida, 2003; Shah, 2006).
2. FOSS experience and skills can sometimes also lead to a higher average salary and *better employment opportunities* compared to developers without any FOSS background (Hann, Roberts, Slaughter, and Fielding, 2002; Lerner and Tirole, 2002; Scacchi et al., 2006), therefore FOSS is used for career development (Hars and Ou, 2001; Hann, Roberts, Slaughter, and Fielding, 2002; Orman, 2008).
3. Developers have *fun* (Ghosh, 1998) and really enjoy their FOSS work (Hertel, Niedner, and Herrmann, 2003) and Stewart and Gosain (2001) found that they also enjoy the peer reputation.
4. *Peer reputation* is a driving factor for contributing to FOSS projects (Hann, Roberts, and Slaughter, 2004) to deliver high-quality code, because respected peers will review the code (Fitzgerald, 2011; Melian, 2007).

Other reasons to contribute to a FOSS project include (Lakhani, Wolf, et al., 2002): Code should be open, work functionality, non-work functionality, obligation from use, work with team, professional status, beat proprietary software, paid for contribution, and user needs (Lakhani and V. Hippel, 2003; Lerner and Tirole, 2005).

For paid contributors (Riehle et al., 2014) of course the payment and other workplace related topics will be a motivating or demotivating factor.

Besides socio-political reasons (e.g. peer reputation, community oriented idealism) Feller and Fitzgerald (2000) also identified technological (e.g. need for robust code, higher standards of quality, faster development cycles) and economical (corporate need for shared cost and risk) drivers for FOSS development.

3. Free Libre Open Source Software

3.4. FOSS Characteristics

There is no official comprehensive list of characteristics or processes FOSS projects should or must have (Fitzgerald, 2011). There is a variety of different FOSS projects, small, large, purely volunteer contributors, partly paid contributors etc. This section describes characteristics, which are often associated with FOSS like: “open sharing of source code, large-scale independent peer review, the community development model, and the expanded role of users” Fitzgerald (2011). Showing these characteristics does not automatically make a project FOSS nor does it mean that a project is not FOSS, if it does not show all of the characteristics.

(No) Formalities Some processes of traditional software development are often not or hardly present in FOSS projects, e.g., formal design process, risk assessment, measurable goals, monetary incentives (Fitzgerald, 2011), project planning (Mockus, Fielding, and Herbsleb, 2002; Scacchi et al., 2006; Howison, 2009) and deadlines (Shah, 2006).

(No) Management FOSS projects usually do not have management staff, which organizes, directs or improves the software development processes (Scacchi et al., 2006). Nevertheless, there is some form of leadership.

Leadership Some projects are lead by a benevolent dictator, often the project initiator, e.g. Linux (Moon and Sproull, 2000). Others move to a more democratic model with a voting committee, e.g. Apache (Fielding, 1999) and GNOME (Germán, 2003), or rotating dictatorship, e.g. Perl (Raymond, 1998b). Leadership may also change with time and growth of a project from a rather centralized decision-making process to a more decentralized one (Fitzgerald, 2006). For all types it is important to be transparent and considerate in their decision-making process because otherwise people who were not consulted might be alienated which is bad for the community (Jensen and Scacchi, 2005; Crowston, Wei, et al., 2012).

3.4. FOSS Characteristics

Strong Community A strong community is important for FOSS projects, because the success of FOSS software is highly dependent on the success of its community (Scacchi et al., 2006). However, many FOSS projects have only one or two contributors, the initiator(s) of the project (Krishnamurthy, 2002; Madey, Freeh, and Tynan, 2005; Krishna and Srinivasa, 2011; Theunissen, Kourie, and Boake, 2007; Scacchi, 2007) and most projects are inactive or have not yet released the software to end-users (Scacchi et al., 2006). Often the project initiator is not only the project lead but also the project maintainer (David, Waterman, and Arora, 2003), and most implementation is done in isolation (Ghosh, 2005; Koch and Schneider, 2002).

The Core FOSS projects often have a small group of core developers, which controls the architecture and direction of development and writes most of the code (Koch and Schneider, 2002; Mockus, Fielding, and Herbsleb, 2002; Crowston and Scozzi, 2004; Dinh-Trong and Bieman, 2005; Scacchi et al., 2006). This group typically consists of 10% to 20% of a team, and creates around 80% of the source code (Koch, 2004). Numbers vary from study to study, e.g. Mockus, Fielding, and Herbsleb (2002) found that 4% (15 out of 388) of contributors contributed 83% of modification requests and 66% of problem reports, Dinh-Trong and Bieman (2005) reported that 4% (15 of 354) developed 57% of source code and the top 47 developers are necessary to develop 80% of code and the top 15 contributed 40% of bug fixing changes. Although differing in numbers, all of these studies report a small group which is more active than the rest of the contributors.

Modularity The code base is usually modular so developers can work as independently from each other as possible. This modular approach is often also present in the project structure. Larger projects consist of smaller projects (Fitzgerald, 2011; Crowston and Howison, 2005).

Onion Model Many FOSS projects can be described by the onion model (Mockus, Fielding, and Herbsleb, 2002; Crowston and Howison, 2005; Mas-moudi et al., 2009; Teixeira, Robles, and González-Barahona, 2015), where

3. Free Libre Open Source Software

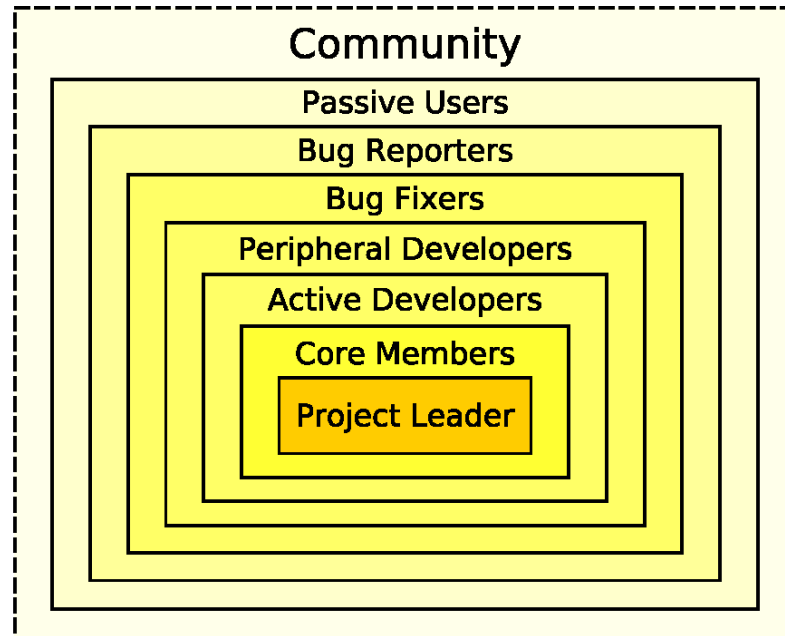


Figure 3.1.: Structure and roles in FOSS communities [Source: (Sommer, 2016) adapted from (Ye and Kishida, 2003; Crowston and Howison, 2005)]

every layer of the onion is larger by an order of magnitude. Figure 3.1 shows such an onion structure.

Voluntariness Although companies are sometimes paying employees to contribute (Riehle et al., 2014), see also Section 3.5, FOSS developers often volunteer their time, skills and hardware to contribute to a project (Scacchi et al., 2006).

Freedom of Choice In FOSS projects not only the four freedoms of FS are important, but also freedoms which are not explicitly formulated or protected by FS or OSS, namely the freedom of expression and the freedom of choice (Scacchi, 2007). They manifest themselves in the choices

3.4. FOSS Characteristics

- for what to develop. Tasks are self-selected not assigned by a superior (Mockus, Fielding, and Herbsleb, 2000; Mockus, Fielding, and Herbsleb, 2002; Crowston, Howison, et al., 2005; Crowston, Li, et al., 2007; Crowston and Scozzi, 2008). Nobody has the administrative authority to assign tasks to developers or to tell them what to do, how to do it or when to do it.
- how to develop it. The software development method is self-selected by the team versus prescribed by the employer.
- of which tools to use. Everyone uses tools they like most, not what is required by the employer.
- for when to release a product. Work quality is valued higher than keeping a deadline.
- of deciding yourself when and what to review
- and who to talk to with or without reservation (Scacchi, 2007).

Additionally, FOSS projects enable contributors to self-select their role (Ye and Kishida, 2003; Gacek and Arief, 2004; Fitzgerald, 2011) which might be more motivating than having a role assigned at the workplace. FOSS projects are partly characterized by believing in these freedoms of expression and choice and practicing them in their virtual organizations (Elliott and Scacchi, 2005; Scacchi, 2007).

User equals Developer Developers of the software are usually also end-users of the software (O'Reilly, 1999; Warsta and Abrahamsson, 2003), which is not the case in traditional software development. This has changed a little bit over the time, because FOSS software attracts many users, who are not developers, e.g. Firefox, GIMP, OpenOffice, but user involvement in the development process is generally higher in FOSS projects than in traditional software development (Gaughan, Fitzgerald, and Shaikh, 2009).

Multi-contribution FOSS developers often contribute to multiple FOSS projects, a few even contribute to ten or more (Hars and Ou, 2001).

Distributed Development Development is often globally distributed (Crowston, Wei, et al., 2012) and achieved by loosely coordinated software de-

3. Free Libre Open Source Software

velopers and contributors using complex software development processes (Hippel, 2001; Hippel and Krogh, 2003; Scacchi et al., 2006).

Tools Fogel (2005) identified tools, which are used in most FOSS projects, namely a project repository, a version control system, an issue tracking system, mailing lists and chat channels as team communication tools.

Electronic communication Developers interact through computer mediated communications tools like websites, email and online discussion (e.g. forums, mailing lists) (Raymond, 1998a; Monge et al., 1998; Wayner, 2000; Yamauchi et al., 2000). They have many values, beliefs and technical competencies in common (Crowston and Scozzi, 2002; Espinosa et al., 2002; Elliott and Scacchi, 2005). A specific pattern of centralized or decentralized communication is not a distinguishing characteristic of FOSS projects, at least not for bug-fixing tasks (Crowston and Howison, 2005).

3.5. FOSS Criticism

As already mentioned in the beginning of this chapter FOSS development is no silver bullet and has its own shortcomings. Usability is often disregarded (Levesque, 2004), often there is no documentation available (Levesque, 2004; Fitzgerald, 2011), code quality is not necessarily better than in proprietary software (Stamelos et al., 2002; Rusovan, Lawford, and Parnas, 2005), and knowledge and concepts (good and bad) from proprietary software development are rejected (Levesque, 2004). Development is usually feature-centric, because features are more fun (Levesque, 2004) and programmers often have other programmers as target audience and the software is designed accordingly (Levesque, 2004).

The high-quality feedback claim from FOSS proponents is called into question as well. Jørgensen (2001) found that simpler code gets feedback, but it is not that useful and design issues receive very little feedback. Moreover, they found that, while 99% of developers can identify 80% of the bugs, only

3.5. FOSS Criticism

1% of developers can identify the remaining, probably more difficult, 20% of bugs (Fitzgerald, 2011; Jørgensen, 2001).

Not only FOSS promises are challenged, also the popular Cathedral and Bazaar metaphor is under investigation. Elferink, Griffiths, and Zondergeld (2016) found that the metaphors generate confusion because they are interpreted differently by different people and both can have positive or negative emotional connotations. Elferink, Griffiths, and Zondergeld (2016) propose a revised pair of metaphors, where both vehicles are comparable items from the same domain. This allows for comparison between software development approaches. They use “the process which constitutes and maintains a Bazaar as an institution, and the process which constitutes and maintains a Shopping Mall.”

Voluntariness Although most FOSS developers are volunteers, there are also many people receiving payment for contributing. Some companies pay employees to contribute to a FOSS project (Scacchi et al., 2006). Hars and Ou (2001) found that 45% of contributors are directly or indirectly paid by companies to contribute. Regardless of the reasons for paid FOSS contributions, many people earn their living with FOSS (Lakhani and Wolf, 2003). In 2014 Riehle et al. (2014) reported similar results. They studied data from the Linux kernel between 2005 and 2011 and data from 5000 active projects from Ohloh (now called Black Duck Open Hub), a web service providing statistics about FOSS projects, between 2000 and 2007 and found that around 50% of all FOSS software development had been done during working hours, suggesting that the contributors were paid for their work. The ratio between paid and volunteer work did not change over the years although the combined Ohloh project data grew at a near exponential rate and the Linux kernel grew at a polynomial rate. Many small projects (one to two persons) consist only of paid developers, while larger projects have a paid work rate of 10% to 20%. The same range, 10% to 20%, of developers of the studied projects only contributed during working hours, when they are presumably paid for it. They made no contributions outside the working hours (Riehle et al., 2014).

4. Catrobat

Catrobat is a hybrid student FOSS project, which develops a visual programming language targeted at children and teenagers to acquire computational thinking skills. It also supports teachers, who want to use Pocket Code in their classes with educational material¹. For an example program see Figure 4.1. Catrobat is inspired by the Scratch programming system (Resnick et al., 2009) developed by the Lifelong Kindergarten Group at the MIT Media Lab. Although inspired by Scratch Catrobat is an independent FOSS project situated at Graz University of Technology (TUG) and focuses on mobile devices, which means children and teenagers do not need a computer to develop programs with Pocket Code (Slany, 2012; Harzl, Krnjic, et al., 2013a; Slany, 2014).

The Catrobat teams develop Integrated Development Environment (IDE) and interpreter apps natively for Android, iOS, Windows Phone (Harzl, Neidhoefer, et al., 2013) (canceled in January 2017), and HTML5 capable browsers. The Android version, Pocket Code, and the connected paint app, Pocket Paint are available on Google Play Store²³. All other versions are in development or only available to Beta testers. See Figures 4.1 and 4.2 for the user interfaces of Pocket Code and Pocket Paint.

¹<https://edu.catrob.at> – retrieved on 12.06.2017

²<https://play.google.com/store/apps/details?id=org.catrobat.catroid> – retrieved on 20.01.2017

³<https://play.google.com/store/apps/details?id=org.catrobat.paintroid> – retrieved on 20.01.2017

4. Catrobat

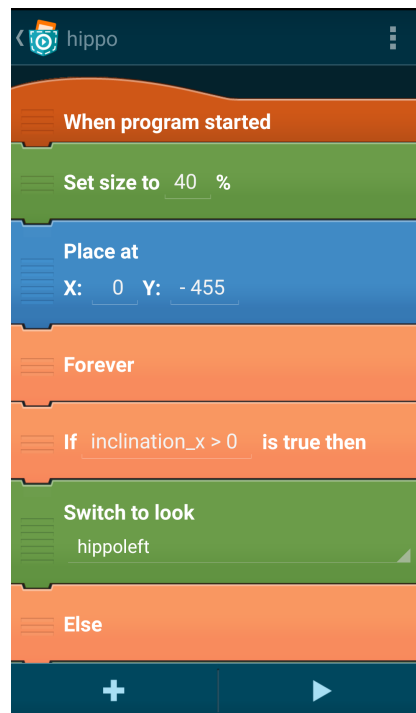


Figure 4.1.: Pocket Code Bricks.

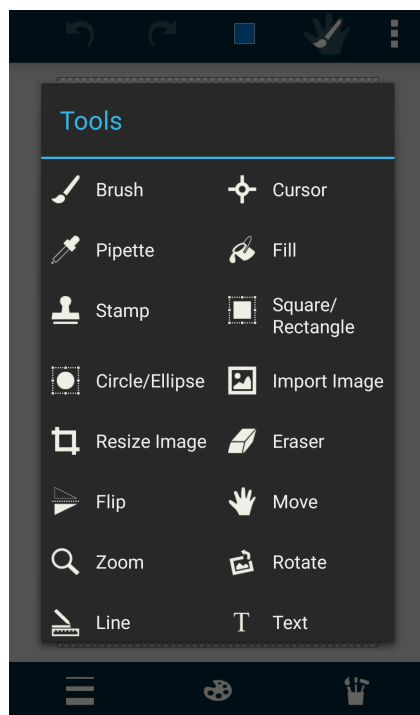


Figure 4.2.: Pocket Paint Tools View.

4. Catrobat

4.1. Catrobat History

The project was initiated as Catroid in 2010 by University professor Wolfgang Slany and five students of TUG. In 2013 the first public beta was released to Google Play Store. The project's name was changed from Catroid to Catrobat. By 2017 the project has around 120 developers (January 2017), 10 Usability and User Experience (UX) members, and 90 translators. Over the years more than 300 people have written more than 750.000 lines of code ⁴ in more than 10 different programming languages ⁵.

Between 100.000 and 500.000 people have downloaded Pocket Code from Google Play Store. Between 2013 and 2017 Catrobat has won the following awards:

- Reimagine Education Award Europe from the Wharton School of the University of Pennsylvania in Philadelphia, USA
- Internet for Refugees Award from the Internet Foundation Austria
- ICT 2015 "Young Minds" Grand Prix Best Connect Exhibitor Award from the European Commission
- Silver Winner of the Lovie Awards and a winner of the People's Lovie Awards from the International Academy of Digital Arts and Sciences, London, UK
- Austrian National Innovation Award for Multimedia and e-Business from the Austrian Ministry for Economics and Youth Development

Catrobat takes part in Google Summer of Code (GSoC)⁶, Google CS4HS⁷, Google Code-in⁸ and has many partners, e.g. Google, Samsung Austria, and the No One Left Behind project⁹.

⁴<https://www.openhub.net/p/catrobat> – retrieved on 20.01.2017

⁵https://www.openhub.net/p/catrobat/analyses/latest/languages_summary – retrieved on 20.01.2017

⁶<https://summerofcode.withgoogle.com/> – retrieved on 12.06.2017

⁷<https://www.cs4hs.com/> – retrieved on 12.06.2017

⁸<https://codein.withgoogle.com/> – retrieved on 10.12.2017

⁹<http://nolleftbehind.eu/> – retrieved on 12.06.2017

4.2. Catrobat Characteristics

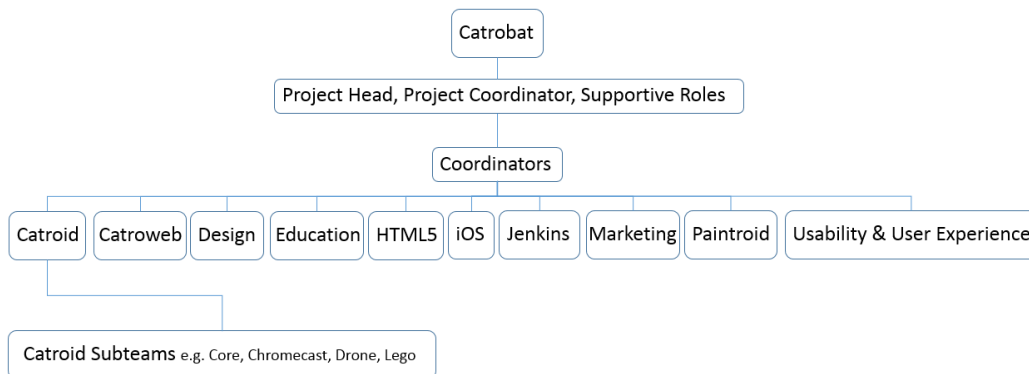


Figure 4.3.: Catrobat Organigram.

4.2. Catrobat Characteristics

Catrobat has a very flat organizational structure, as can be seen in Figure 4.3. There is only minimal central management. It is limited to the project head and one project manager, who takes care of all organizational activities, e.g. managing users, their accounts, project infrastructure, software licenses. There are a few supportive roles, e.g., taking care of interactions with schools or supporting the Continuous Integration (CI) (Jenkins) team. In the future the project manager responsibilities will be distributed between more people.

Work in Catrobat is accomplished in teams. Every platform, e.g. Android, iOS, is serviced by a team. There are also sub-teams, which develop a special functionality for the super-team, e.g., supporting Arduino elements, Lego NXT, or Parrot drones. There are also teams, which take care of CI, usability, design, marketing and education. Every team has a coordinator who takes care of organizational tasks (e.g. supervise Jira board), who has a good overview what everybody is doing and who is the primary contact person for the team. Other roles within a team are seniors, who are more experienced team members and are allowed to merge code into the main repository and contributors, who are team members without special rights and responsibilities. Contributors have to prove their technical abilities be-

4. Catrobat

fore they can become senior members. The team itself decides who is ready to assume senior responsibilities. The same is true for the coordinator role. Team members can volunteer for this role and the team collectively decides on their coordinator.

All development teams use an agile approach using elements of XP (Andres and Beck, 1999; Williams, Krebs, and Layman, 2004) and Kanban (Anderson, 2010). This approach will be explained in detail in Section 4.4.

Lack of knowledge and experience are in general a challenge for Catrobat. Because almost all Catrobat developers are students, the skill level is rather low. Skills about software development range from beginner to intermediate, depending on the work experience of contributors. Knowledge about agile methods is not wide-spread, some know very little and some only a fair amount about it. This is a problem, because “agile methods tend to need a richer mix of higher-skilled people.” (Boehm and Turner, 2003a). The best known methods are Scrum and Kanban (Sommer, 2016), see Section 4.4 for more details.

Another challenge is the high turnover rate. People are leaving and joining the project constantly which means a loss of experience and some initial effort to get new people started. Other FOSS projects also struggle with member retention. Shah (2006) reported that contributors to large projects leave within one year and David, Waterman, and Arora (2003) determined the median length of project participation as 1.2 years for large and small projects.

Catrobat and FOSS Characteristics

Catrobat adheres to the OSD, see Section 3.1.2, and therefore is a FOSS project.

Regarding the characteristics and processes, which FOSS projects often show the picture is not so clear. In this section we will discuss how Catrobat compares to the characteristics mentioned in Section 3.4. Table 4.1 shows a short summary of FOSS and Catrobat characteristics and some characteristics will be discussed in more detail.

4.2. Catrobat Characteristics

Table 4.1.: Comparing FOSS characteristics and Catrobat.

FOSS Characteristic	Catrobat
(No) Formalities	Sometimes soft deadlines.
(No) Management	Minimal management staff.
Leadership	Project initiator as benevolent dictator and project coordinator for organizational issues. After the study a voting committee (Product Owner group) was added.
Strong Community	Community is still in development.
The Core	Contributions are more evenly distributed because students have to contribute to receive a certificate.
Modularity	Project structure is modular, code base not always.
Onion model	Onion model, but layers are not larger by an order of magnitude.
Voluntariness	Most developers are students. Volunteers are mostly contributing in bug reports, feature requests and translations. Some contributors receive funding via the GSoC program.
Freedom of choice	Tasks are sometimes assigned to students (not to volunteers), if they are important and nobody volunteers. Software development method was chosen by project initiator. Free choice of tools, but team only supports certain tools, e.g. provides help with setting up the development environment. Release dates are determined by the project initiator. Free choice of review, talking to others and self-selection of role.
User equals Developer	Mostly through creation of Pocket Code programs, feature requests, bug reports and translations. UX team helps focusing on user needs.
Multi-contribution	Most Catrobat developers contribute only to Catrobat, see Figure 4.4.
Distributed development	Main development takes place in Austria, but team members are usually not co-located. International contributors come from all over the world.
Tools	All usual tools are present in Catrobat.
Electronic communication	Most communication is electronic, but direct personal communication is possible.

4. Catrobat

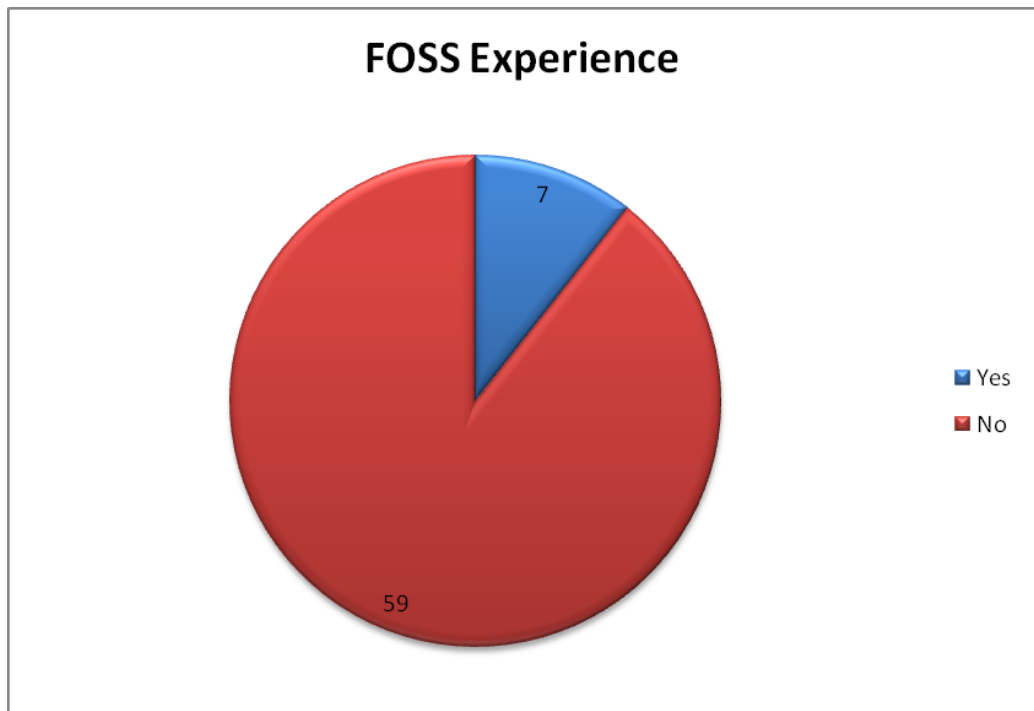


Figure 4.4.: Have you already contributed to another FOSS project? [Source: (Sommer, 2016)].

Catrobat does not exactly match the onion model as described in Mockus, Fielding, and Herbsleb (2002), Crowston and Howison (2005), Masmoudi et al. (2009), and Teixeira, Robles, and González-Barahona (2015). The layers are similar, see Figures 4.5 and 4.6, but some are named differently e.g. Project Head versus Project Leader and others do not exist in Catrobat, e.g. Core Members, Bug fixers. The project coordinator role is responsible for organizational things, like user accounts and giving a short introduction into the overall project. Catrobat's outer layers of the onion are not larger by an order of magnitude. The number of developers (120) is almost the same as translators (around 90) and bug reporters (around 40) combined. Translators and bug reporters were not checked for duplicates.

Most developers are students at TUG e.g. doing their bachelor thesis, master project or master thesis within the project. Therefore code contribu-

4.2. Catrobat Characteristics

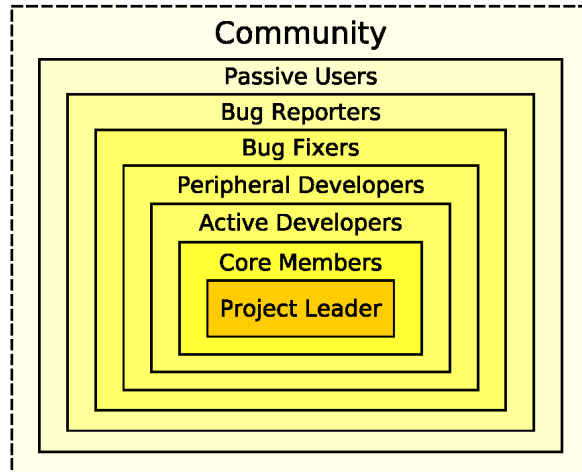


Figure 4.5.: Structure and roles in FOSS communities [Source: (Sommer, 2016) adapted from (Ye and Kishida, 2003; Crowston and Howison, 2005)].

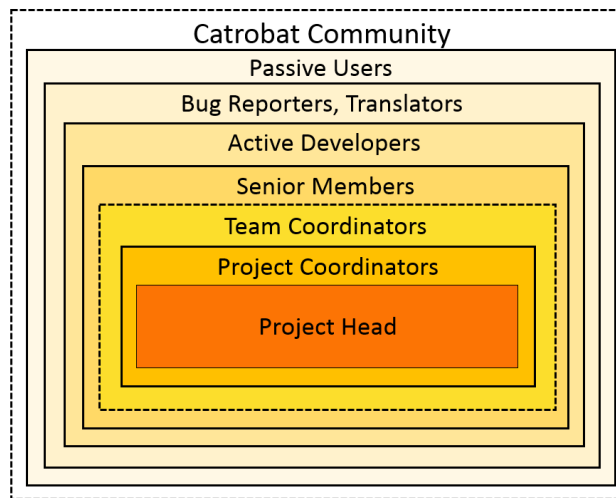


Figure 4.6.: Structure and roles in the Catrobat community [adapted from (Sommer, 2016)].

4. Catrobat

tions are more evenly distributed in Catrobat than in usual FOSS projects, where a small percentage of project members develops most of the software (Kagdi, Hammad, and Maletic, 2008).

Students sometimes work at the university, but most of the development work is accomplished at home or anywhere on earth. Catrobat has no core team or core developers (Torres et al., 2011), because developers change all the time, and students only stay with the project from six months to two years (with breaks). So there are (hardly) any developers who stay with the project for several years who have experience and tacit knowledge about the project. This is a big disadvantage for the project. A wiki system helps to keep important information, but still every time someone leaves the project, something is lost.

International contributors are working mostly on translations for the user interface, provide bug reports or feature requests, and create tutorials, Youtube videos or example projects. Young users mainly contribute indirectly through sharing their projects on the Pocket Code website ¹⁰ under an Open Source and Creative Commons license. Some engage in bug reports or feature requests.

4.3. Motivation in FOSS and Catrobat

Motivation in FOSS settings is described in detail in Section 3.3. In this section we will compare motivating factors in FOSS projects in general and motivating factors for student contributors in Catrobat. Motivation in Catrobat was studied in 2016 by Sommer (2016). Answers of 66 Catrobat members were collected and analyzed. Studies about FOSS motivation often mention learning, reputation, career development and fun as main motivational factors to contribute to FOSS. Although learning and career path are also motivators in Catrobat, only learning is under the top five. Reputation is only *rather present* in Catrobat and is only a minor motivating factor for the survey respondents in general. Fun was not in the survey.

¹⁰<http://pocketcode.org> – retrieved on 12.06.2017

4.3. Motivation in FOSS and Catrobat

The motivational factors most present in Catrobat are trust/respect, working conditions and autonomy. Figures 4.7 and 4.8 show the motivational factors, which are most important for Catrobat members in general sorted by their general motivational effect in descending order.

Figures 4.9 and 4.10 show to what extent they are present in Catrobat. Motivators are sorted by their presence in the Catrobat project in descending order.

4. Catrobat

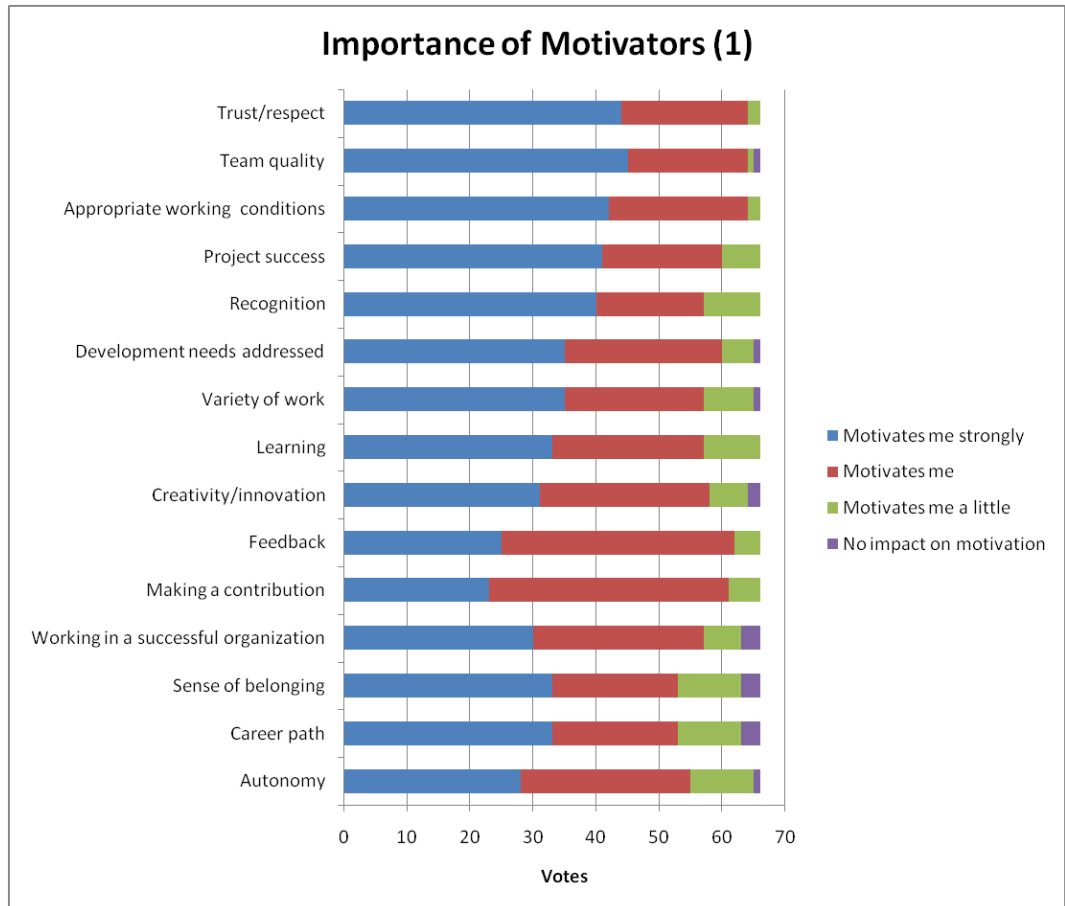


Figure 4.7.: Importance of Motivators for Catrobat members in General Part 1 [Source: (Sommer, 2016)].

4.3. Motivation in FOSS and Catrobat

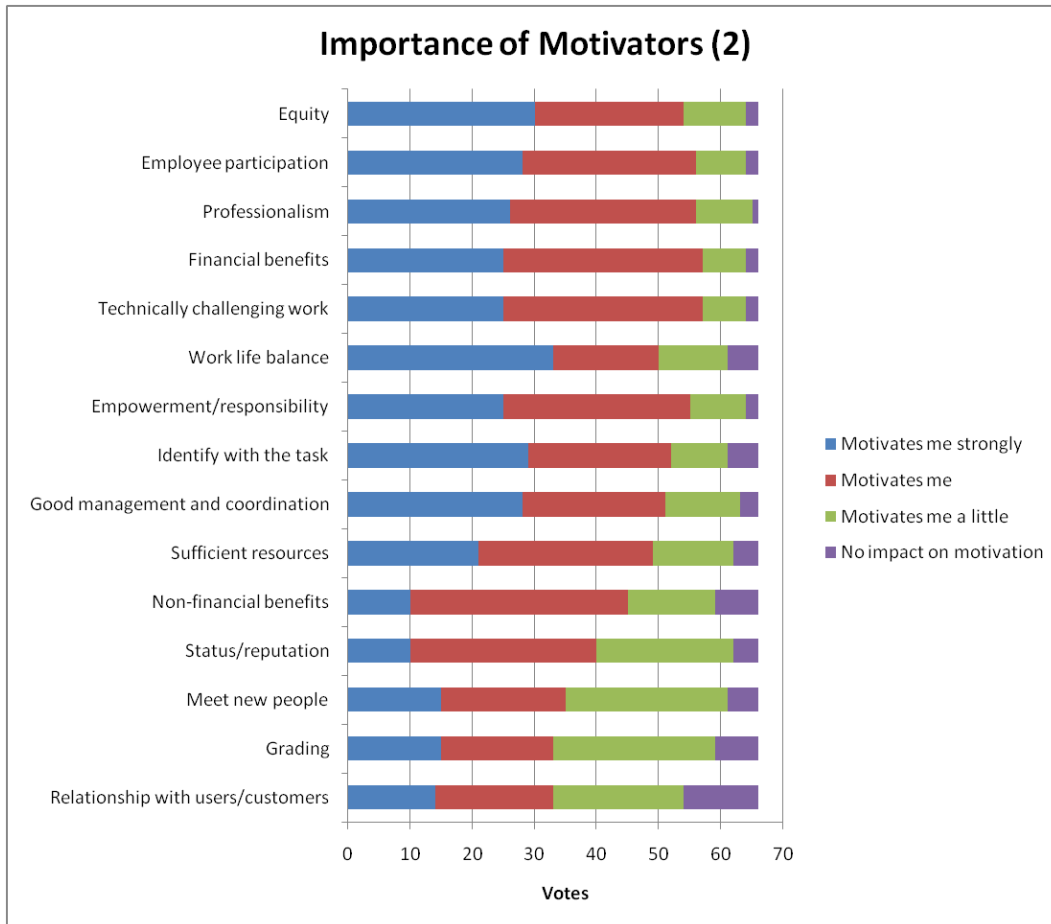


Figure 4.8.: Importance of Motivators in General Part 2 [Source: (Sommer, 2016)].

4. Catrobat

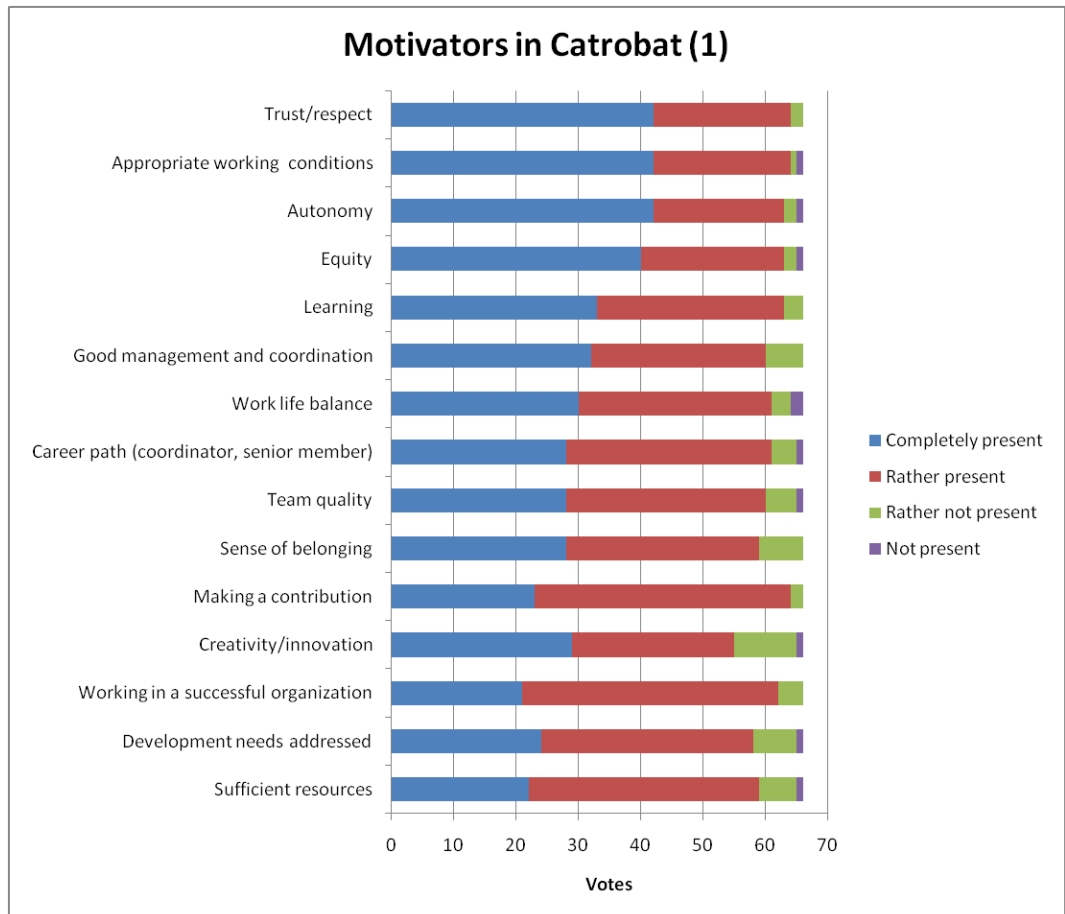


Figure 4.9.: Presence of Motivators in Catrobat Part 1 [Source: (Sommer, 2016)].

4.3. Motivation in FOSS and Catrobat

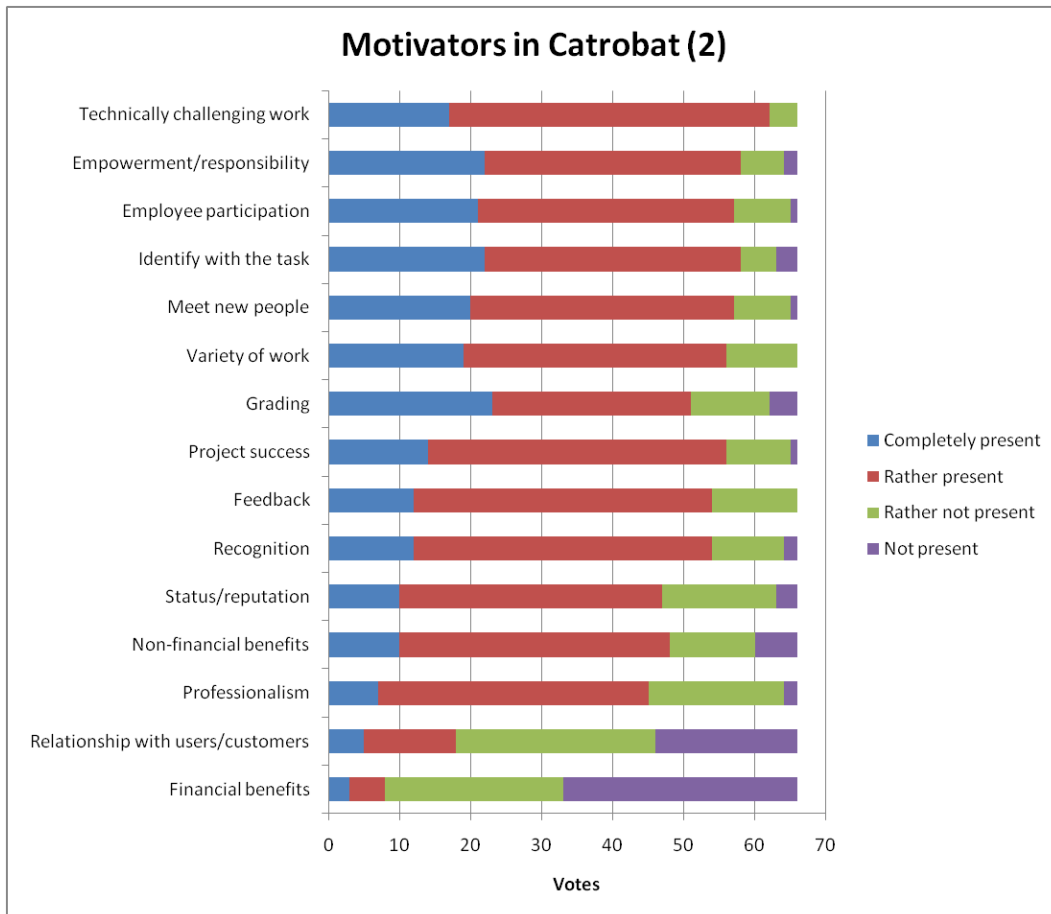


Figure 4.10.: Presence of Motivators in Catrobat Part 2 [Source: (Sommer, 2016)].

4. Catrobat

4.4. Software Development Approach

Catrobat uses a software development approach closely related to XP in Andres and Beck (1999), but not all practices are applied. XP, its values, principles and practices are explained in detail in Section 2.1. Catrobat teams use the following practices: *automated unit tests, pair programming, refactoring, release planning, short releases, CI, coding standards, collective code ownership, simple design, and regular meetings*, and a Kanban board. These practices are used to a greater or lesser extent, depending on the team. Other XP practices, like *acceptance tests, TDD, customer access, sustainable pace, system metaphor, lessons learned, growth, artefact reduction* (Williams, Krebs, and Layman, 2004) are hardly used or are not used at all. Meetings are usually held weekly, the coordinator meeting is held every two weeks, release planning occurs in irregular intervals, some teams do it every few months, others do it not even once a year. Testing is an important issue for the project head and all teams should employ TDD (Beck, 2003), but students often lack the experience to do so and write the tests only after finishing the code and sometimes code is not tested. Figures 4.11 and 4.12 show the detailed answers of the respondents of the 2016 survey for best known and most used ASD methods.

4.4. Software Development Approach

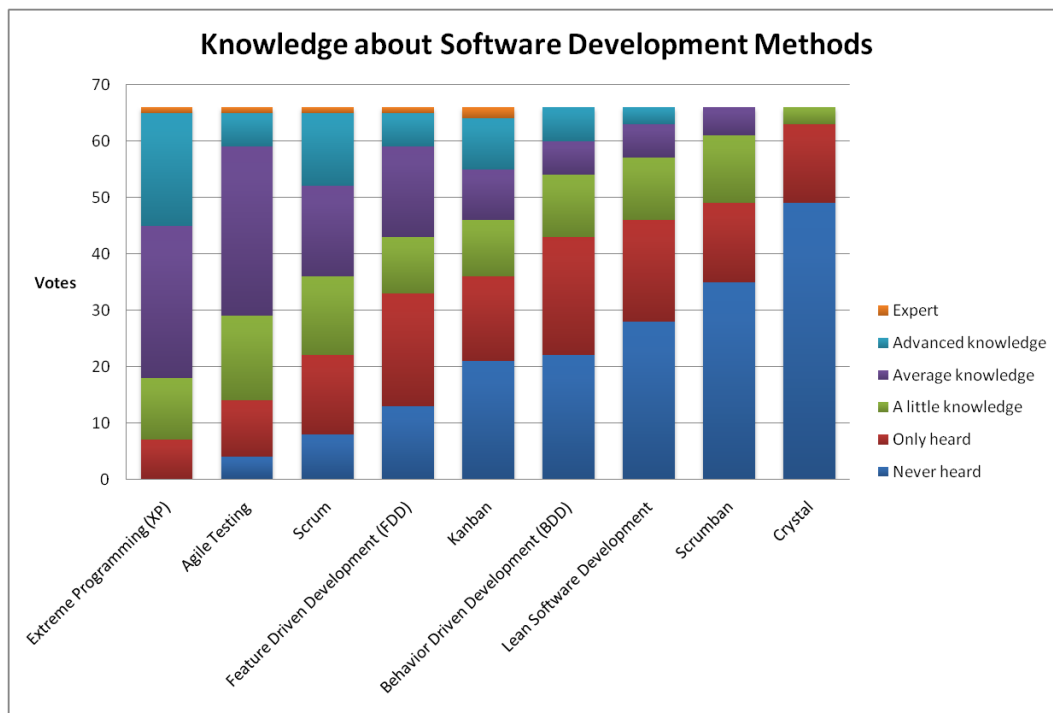


Figure 4.11.: Answers to the question: How well do you know the following software development methods? [Source: Sommer (2016)]

4. Catrobat

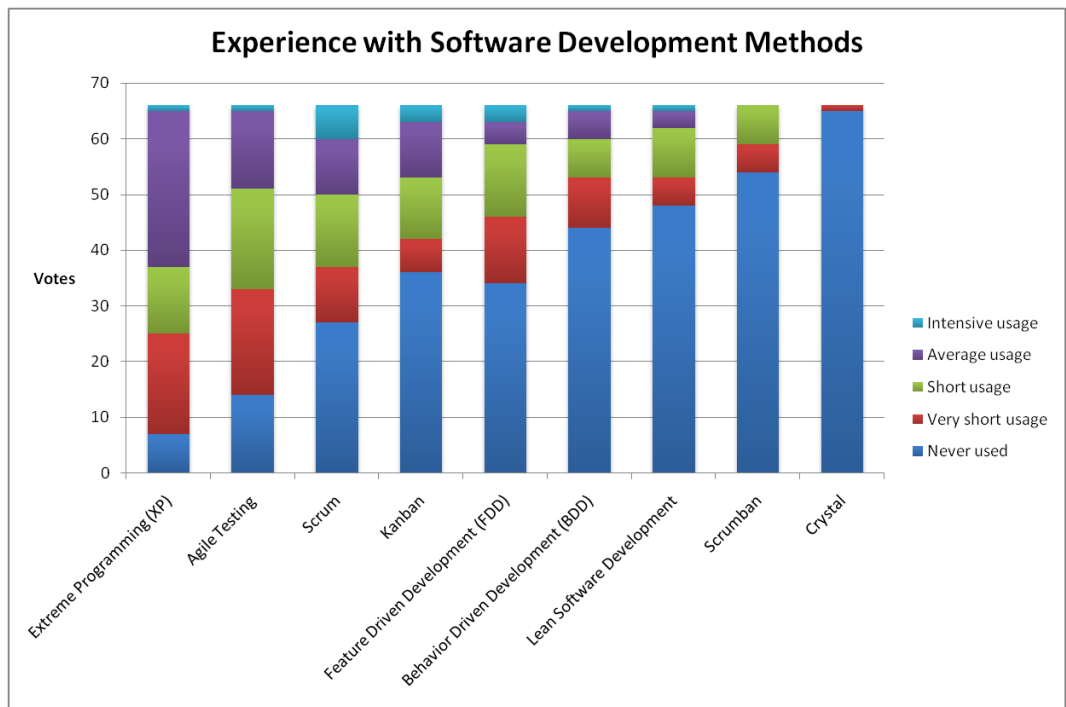


Figure 4.12.: Answers to the question: How much experience do you have with the following software development methods? [Source: Sommer (2016)]

4.5. Selection of the Project

This part is based on Harzl (2016) and Harzl (2017).

The Catrobat project was selected due to the following reasons:

- **Personal contact:** The setting with most developers at TUG allows direct personal contact. Written information, e.g., in mailing lists, only conveys a small fraction of human communication and interaction and is often misunderstood (Schafer, 2000), if not used correctly. If somebody wants to change a process, people affected by these changes need to trust this person and trust is easier established through personal contact. Furthermore, it is easier to receive feedback on multiple levels and to refine the research methodology and researcher skills through personal contact. Personal contact is not seen as prerequisite, but as facilitating the research process.
- **Experiments and evaluation:** Evaluations and experiments are an important part of AR. Students are often used to research and to experimenting with different approaches and willing to evaluate them. Non-student contributors may be more reluctant to do so. Although this setting with mainly student developers is rather unusual, it allows to conduct more questionnaires and evaluations, which is important for research purposes. Moreover, students work on many FOSS projects and are not atypical FOSS contributors.
- **Time and access:** Reduced bonding time and easy access to team members and artifacts are other reasons to select this project. It can take a very long time to build a good reputation within a FOSS project and to gain enough trust to be allowed to change work processes. A basic trusting relationship with members was already established hence the bonding period could be minimized and made it possible to conduct the AR within a reasonable time frame. If one has direct access to people and artifacts, e.g., whiteboards and flip charts, discussions can be done in a shorter time and it is easier to acquire all material used in the discussion for later analysis.
- **FOSS Characteristics:** Many characteristics are the same or similar to other FOSS projects, see Section 4.2. Even the rather unusual face-to-face gatherings are not unprecedented in FOSS projects (Düring,

4. Catrobat

2006a).

Therefore, this project is a good starting point to explore Kanban in the context of FOSS development. It should of course not remain the only case, due to its limitations, which will be discussed in Section 9.2.

4.6. Selection of the Team

The actual team was co-selected by the participants of the AR. The team coordinator asked for help regarding the team's motivation and workflow, probably because of the researcher's supporting role in the umbrella project, which is described in more detail in Section 6.4. The team did not know how to overcome their problems and agreed to participate in research to achieve practical outcomes, which would hopefully improve their situation. This resulted in a bias for action, which contributed to the decision to select AR as the research methodology.

Asking for help shows some commitment, which is usually needed to achieve action outcomes. This contributed to the decision to conduct the study with this team. Other factors for the decision were that the team (six to eight people, varying over time) has roughly the average size of teams in this FOSS project (see Figure 4.13), it uses the same agile workflow as the other teams and direct personal contact with the members of the team is possible. All team members are students, the only non-student developer left the project before the AR started. Apart from team size also other demographic details, like age, studies, and education, are quite similar to the whole Catrobat project, see Figures 4.14, 4.15, 4.16, 4.17, 4.18, and 4.19 for comparison between the demographics of Catrobat as a whole and the studied team.

Initially the AR team was a stand-alone team but was transferred into a sub-team of another team. It implements a larger feature within the super-team's mobile application, so the team has to coordinate their efforts with the super-team. The part developed by the sub-team has not been released to the public, therefore end users are not part of the sub-team's workflow

4.6. Selection of the Team

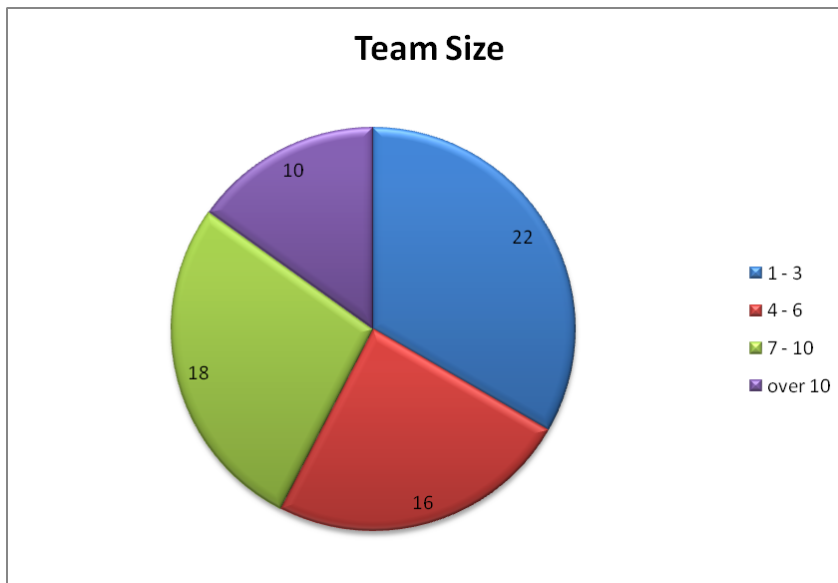


Figure 4.13.: Answers to the question: What is the size of your team? [Source: Sommer (2016)]

for now. From the beginning the team used elements of XP and a Kanban board, like all other sub-projects. The applied XP practices include *automated unit tests*, *pair programming*, *refactoring*, *release planning* (occurs in irregular intervals), *short releases*, *continuous integration*, *coding standards*, *collective code ownership*, *simple design and regular meetings* (weekly). *Visualize the workflow* was the only Kanban practice applied, in the form of an agile board, but members did not know, that this was a Kanban practice. The Shodan 2.0 Input Metric Survey in Williams, Krebs, and Layman (2004) was used to determine the use of agile practices within the team. Other XP practices, like *acceptance tests*, *TDD*, *customer access*, *sustainable pace*, *system metaphor*, *lessons learned*, *growth*, *artifact reduction* (Williams, Krebs, and Layman, 2004) were initially not known, hardly used or not used at all by team members.

4. Catrobat

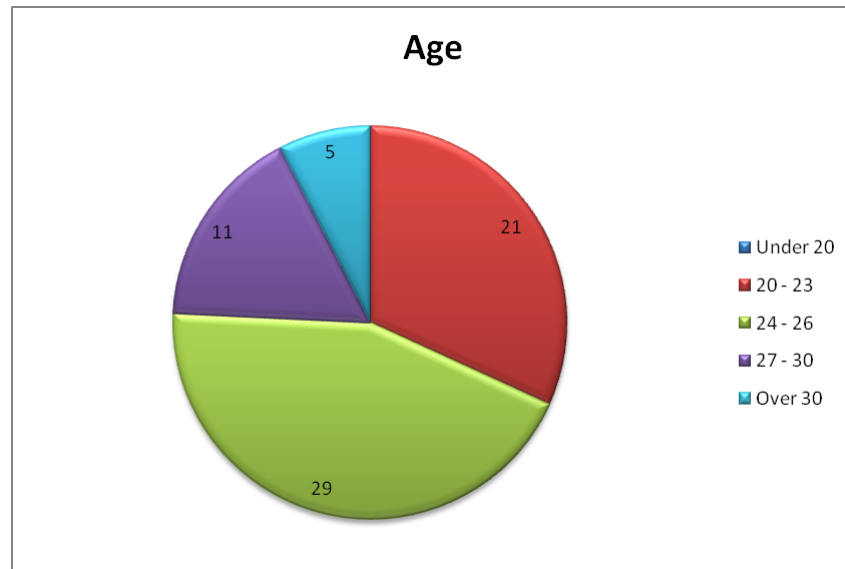


Figure 4.14.: Age distribution in Catrobat [Source: Sommer (2016)]

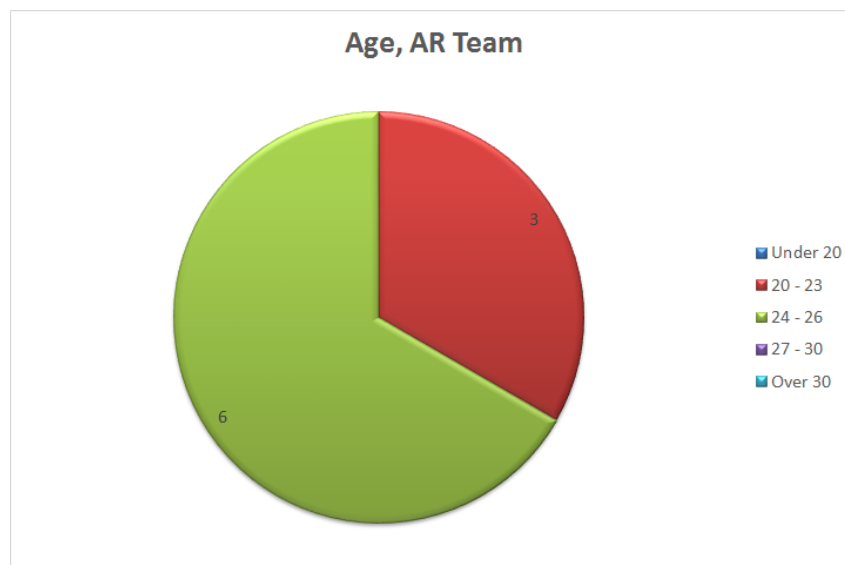


Figure 4.15.: Age distribution in AR team

4.6. Selection of the Team

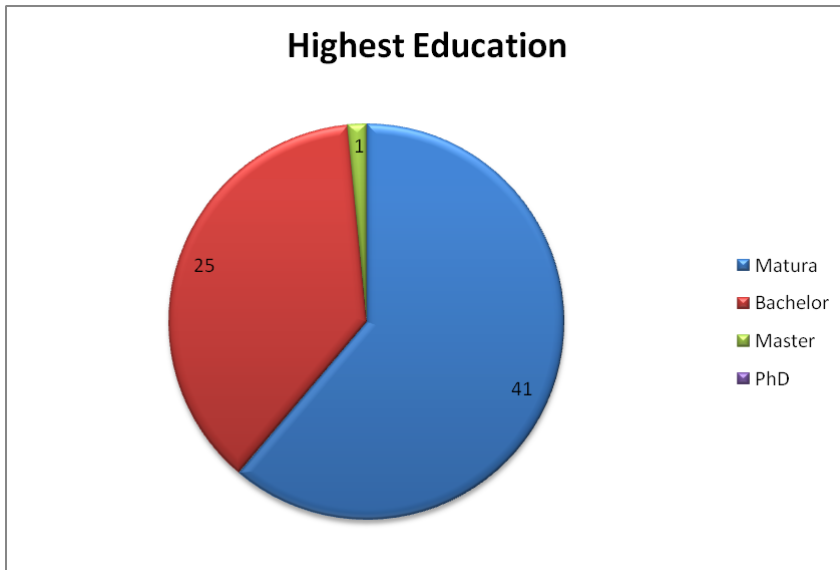


Figure 4.16.: Highest education in Catrobat [Source: Sommer (2016)]

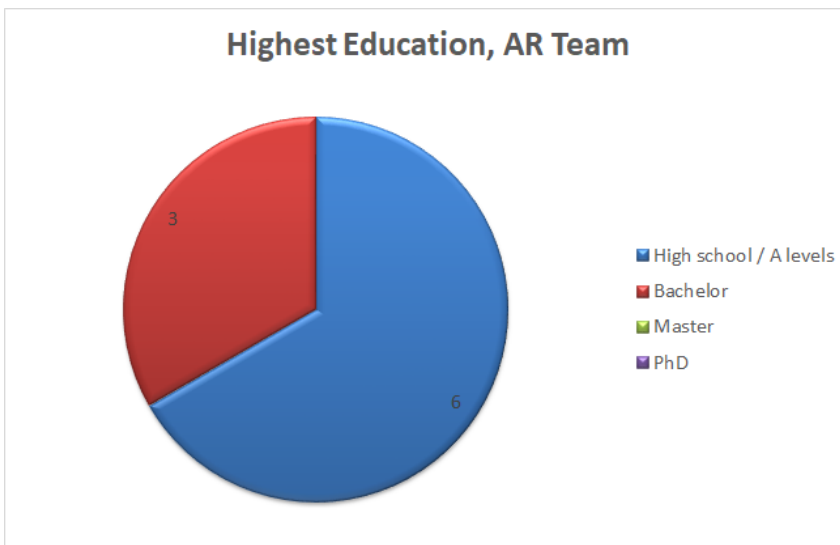


Figure 4.17.: Highest education in AR team

4. Catrobat

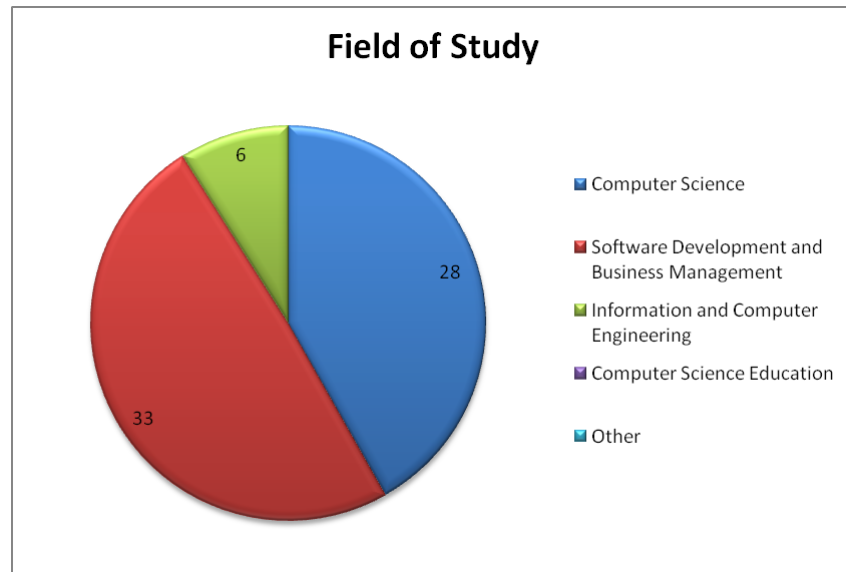


Figure 4.18.: Fields of study in Catrobat [Source: Sommer (2016)]

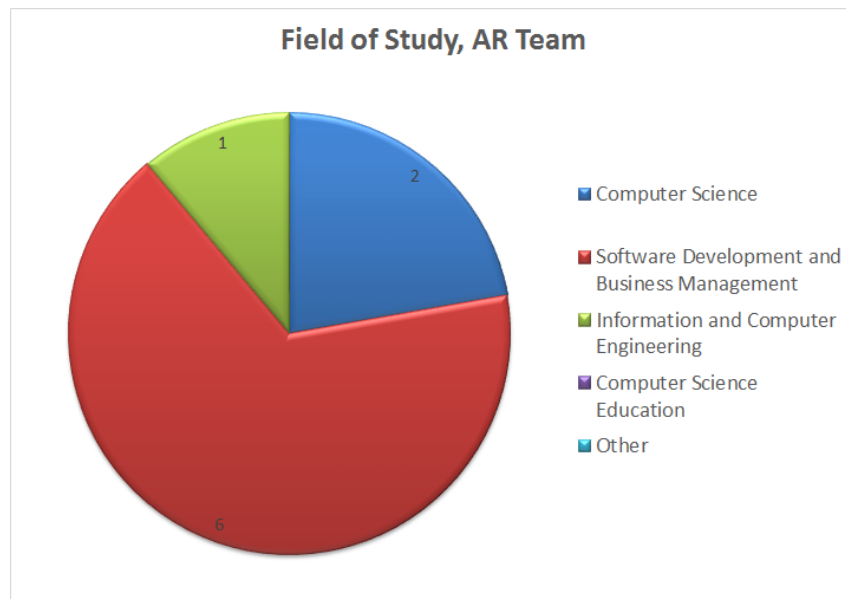


Figure 4.19.: Fields of study in AR team

4.7. Selection of Kanban

Catrobat teams already use Kanban-like boards. They are Jira Kanban boards, but without WIP limits, swim lanes or SLA. By choosing the Kanban method, the team could keep existing tools and the initial changes to its workflow would not be overwhelming.

Literature suggests that Kanban is useful for teaching inexperienced developers about software engineering and a good pedagogical tool (Ahmad, Liukkunen, and Markkula, 2014). Furthermore in this study Kanban exhibited a short learning curve and low adoption threshold and helped students to improve their team work skills, e.g. improved collaboration and communication. Because many inexperienced software developers contribute to Catrobat, it also serves teaching purposes. This makes Kanban an appropriate and lightweight approach for the student and the FOSS part of Catrobat. The entrance barrier for the FOSS part is kept low, so potential contributors will not be scared off, and additionally Kanban fosters collaboration, which is important for both parts and the project as a whole.

According to Kniberg and Skarin (2010) Kanban is the most adaptive method. It does not require an all or nothing approach, where all processes have to be introduced at the same time, instead small evolutionary changes are welcome. No expensive week-long up-front training is necessary. Initial job titles and responsibilities stay the same, so people do not feel demoted because their job titles vanish. Small incremental changes enable an adoption speed appropriate for the team versus steamrolling the team with all changes at once. This would require positional power, which is not available in a FOSS community. In Kanban people are encouraged to use their own mind. People affected by changes are actively involved and participate in the process and do not simply endure the changes. This active involvement of affected parties makes Kanban also a good fit for AR as a research methodology, which is discussed in more detail in Chapter 6.

5. Related Work

Extensive literature review did not reveal previous work comprehensively combining Kanban and FOSS, so related work regarding Kanban or FOSS in academia and work about the relation between FOSS and ASD will be presented.

5.1. Educational Settings

5.1.1. Kanban and Academia

Ahmad, Liukkunen, and Markkula (2014) investigated students' perceptions of the Oulu Software Factory Laboratory including the constructs of Kanban boards and collaborative learning. A Software Factory provides students with a realistic environment with close customer involvement, where they work intensively in teams and use modern software development tools and processes, thus, improving their learning experience (Fagerholm, Oza, and Münch, 2013). Students worked on real projects with real customers. Apart from the mandatory use of Kanban boards, students were free to use any agile or lean software development practices. Students overall perceived the Kanban boards as positive and one respondent (out of 19 responses) even suggested that they should be used more in school, which would help students to keep track of their on-going work.

Ahmad, Markkula, and Oivo (2014) reported positive results regarding the usability and effectiveness of Kanban as a pedagogical tool for teaching software engineering in Software Factory projects. Students had to answer various questions about the use of Kanban on a five-point Likert scale ranging from 1 (strongly disagree) to 5 (strongly agree). Students had in general

5. Related Work

a very positive perception of Kanban. The majority said it helped them understand project activities, identify bottlenecks and provided them with a better view of the project. 38 out of 51 (approx. 75%) respondents agreed or strongly agreed with *Kanban helps to identify bottlenecks in project* and only 4 out of 51 (approx. 8%) disagreed or strongly disagreed. Also 38 out of 51 (approx. 75%) agreed or strongly agreed with *Kanban helps in understanding project activities*, and only 5 out of 51 (approx. 10%) disagreed or disagreed strongly. When answering *Kanban provides a better project view* 40 out of 51 (approx. 78%) agreed or agreed strongly and only 2 out of 51 (approx. 4%) disagreed. 39 students (approx. 76%) intend to use Kanban in the future in real software engineering projects. Answers to open-ended questions included

- “One can see the progress of the project at one glance.”
- “The timing for internal deadlines among project members was also aided by the use of board.”
- “...the Kanban board helps me follow my project procedure when I am lost.”
- “The Kanban board is a very good tool for task management. The boards were good to use for discussion in my project. It was good to use, we discussed the problems of our project when they occurred. It serves us well and we use it to analyze the task at hand. It is also good when work is not going forward due to some problem.”

Ahmad, Markkula, and Oivo (2014) concluded that Kanban appears to be useful for inexperienced software developers like students, provides them with a good overview of the project progress, makes problems immediately visible, and helps them understand project activities rather easily, thus, supporting their learning experience. Team work competencies were improved in the Software Factory Kanban projects. Students reported positive effects of the projects on their team working skills. They were able to acquire and practice their skills within this setting. Figure 5.1 shows the answers to the questions if Kanban Software Factory projects helped students to acquire team work competencies.

Ahmad, Markkula, and Oivo (2014) concluded after a 1.5 year long study with Master’s degree students, that Kanban is a good pedagogic tool for

5.1. Educational Settings

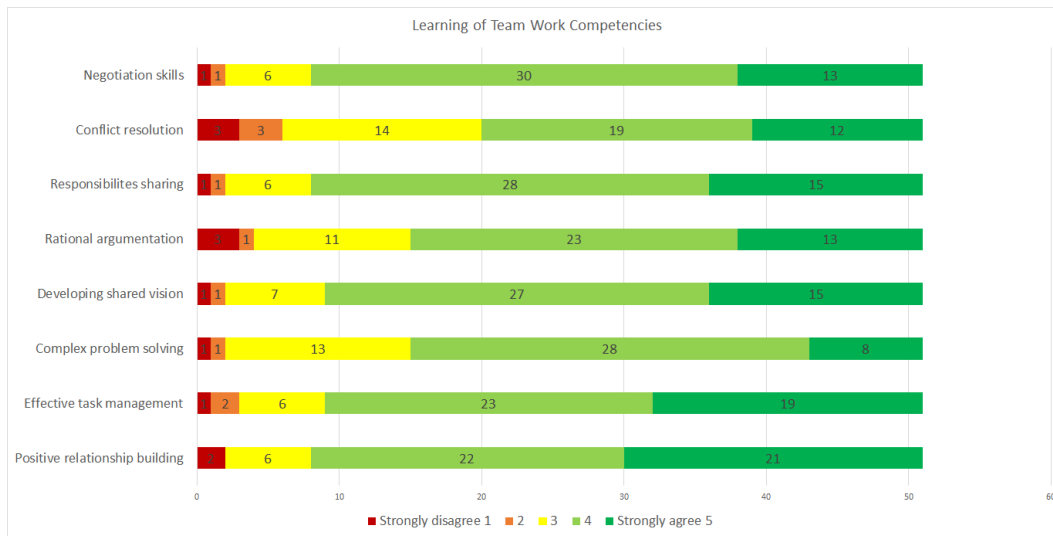


Figure 5.1.: Learning of Team Work Competencies in Kanban Software Factory Projects [Source: adapted from (Ahmad, Markkula, and Oivo, 2014)]

teaching software engineering in practical project courses and that students find Kanban relevant to their learning. Software Factories using Kanban are effective for teaching software engineering, because students are enabled to acquire important interpersonal skills. Moreover, Kanban helps students to achieve their goals in their project work, helps them to understand software projects and practices currently applied throughout the industry. “Kanban also facilitates project development, promotes team communication and supports collaboration.” (Ahmad, Liukkunen, and Markkula, 2014). Kanban should have a low adoption threshold and short learning curve, because it is easy to learn and and to apply, for professionals and inexperienced software engineers.

5.1.2. FOSS and Academia

Liu (2005) describes GROW (Gradually Ripen Open-source Software) an educational software process, which is cross-term, cross-team and enables

5. Related Work

teachers to use complex, real-world projects in one-term courses. This software process was tested in a senior-level software engineering course in 2004. Partner organizations visited the class and presented their initial requirements. Students worked with the partner organizations and developed software projects for them. To ensure that the software reaches a level, where actual users feel comfortable using it, project artifacts had to be released as OSS and were reused by subsequent courses, which improved and extended the project. So students did not contribute to a pre-existing FOSS project, only their course artifacts were provided as OSS to students of the following years.

Pedroni et al. (2007) describe a programming course where students had to contribute to an existing FOSS project. They evaluated the experience through a motivation measuring technique and analyzed students' efficiency and commitment over time. The study shows that students were more anxious about their success than the control group, working on a "toy" project, but they rated their achievements higher (compared to the control group) and considered to keep contributing to the assigned project after the course. There was no information given which software development approach was or which practices were used.

Gehring (2011) reports the result of a survey of managers of FOSS projects. Managers of projects, which successfully included contributions of students, were asked how they interacted with classes and what instructors could do to improve interaction. Most of these projects were either started by faculty or caught the attention of members of the faculty, e.g. the H-FOSS project (Humanitarian Free & Open-Source Software). The study indicates that course instructors should contact the specific projects weeks or months in advance and should also be personally involved in FOSS development. They should prepare their students especially in the areas of design and testing, so they can contribute more easily. Because students usually need more help getting started than more experienced developers, some projects created tutorials especially for students or support pages for new developers. Others had an IRC channel for students or provided day long code sprints. The paper gives some examples of FOSS projects, which worked with students.

Marmorstein (2011) describes a software engineering course during which

5.1. Educational Settings

small student teams contributed to a FOSS project. Students should familiarize themselves with the FOSS community, the design strategies and work flow of a large project. In contrast to academic “toy” projects, where code is often developed from scratch, students could implement new features, do software maintenance tasks, e.g., bug fixing, or write documentation. Almost all students rated the project as instructive and most of them answered in a survey that they had learned about the design phase, as well as the implementation and maintenance of a project. One problem was communication with the FOSS projects, many students rated the helpfulness of community members as rather low.

MacKellar, Sabin, and Tucker (2015) developed a teaching approach which uses open source development practice(s) and client-oriented hands-on software development. Instead of writing code for small “toy projects” simply for educational purposes, company-sponsored proprietary software, internship courses, or large FOSS projects, students developed relatively small projects for local non-profit organizations. The developed code is open source and can be re-used, extended or adapted by other students or organizations. MacKellar, Sabin, and Tucker (2015) call this approach Client-Oriented Free and Open Source Software Development (CO-FOSS). Local non-profit organizations received software tailored to their needs without draining their budgets, students gained valuable real world experience and the open source code enabled instructors, students and organizations to re-use, extend or adapt the software for different clients over several semesters or at other universities. MacKellar, Sabin, and Tucker (2015) adapted the framework from Allen Tucker Morelli, Lanerolle, and Tucker (2012) to their institutions. They present their framework, their adaptation experiences and guidelines on how to incorporate the CO-FOSS approach into software engineering courses. Advantages of CO-FOSS include flexible team size and task assignment, FOSS tools, which are open source as well and therefore affordable for universities, and FOSS practices, which are suitable for students because they support asynchronous, distributed work. Because the code can be re-used, adapted and extended, project parts and team sizes can be adapted to the actual course. For example, if a course focuses on databases, students of this course can implement only the database part, whereas another course can involve development for several parts of a project, e.g. user interface, database and help system.

5. Related Work

Team size is also flexible. Larger teams enable team members to split up to work on parts of the system, they are comfortable with, smaller teams enable team members to work on all parts of a project. Students familiarize themselves with FOSS tools, which are often used in companies as well, e.g., version control and issue tracking systems, and FOSS practices, e.g., code reviews, testing, team communication in geographically distributed teams, which are also important in industrial settings.

Ellis, Morelli, et al. (2007), Ellis, Purcell, and Hislop (2012), Ellis, Hislop, and Purcell (2013), Hislop and Ellis (2015), Ellis and Hislop (2016), Ellis and Hislop (2017), and Ellis, Hislop, and Burdge (2017) report on students contributing to Humanitarian Free and Open Source Software (HFOSS) projects. The goal was to prepare students for the real-world through engaging them in large real-world projects, where they could gain the technical (e.g. programming, development processes) and social (e.g. teamwork) skills needed to work in the fast-paced software development industry. They identified one major challenge for teachers, to identify appropriate HFOSS projects. Important factors for choosing the right project are the license, programming language, rate of activity, number of contributors, availability of an issue tracker, size of the project, openness towards / welcoming of new contributors, community norms and user base.

5.2. FOSS and Agile Software Development

Koch (2004) compares the ASD and FOSS movements based on several criteria taken from the principles of ASD and the FOSS description of Raymond (2001). Data, wherever appropriate, was sampled from Apache and Mozilla (Mockus, Fielding, and Herbsleb, 2002), GNOME (Koch and Schneider, 2002), and Sourceforge ¹. There is no practical application of combining ASD and FOSS included, nor does the paper mention projects which comprehensively combine ASD and FOSS. Nevertheless, he concluded that they show many similarities, e.g. craftsmanship and self-organization, and only a few differences, e.g. team co-location, self-selection

¹<https://sourceforge.net/> – retrieved on 02.09.2017

5.2. FOSS and Agile Software Development

of tasks. This is also supported by Warsta and Abrahamsson (2003) and Gandomani et al. (2013).

Theunissen, Kourie, and Boake (2005) studied the chances of combining agile and open source practices in the context of corporate software development and suggested an approach how to accomplish this. They identified several tensions between FOSS and corporate culture as well as between ASD and FOSS, e.g. monitoring developers vs. volunteers without supervision and adaption to remote communication vs. co-located developers' communication face-to-face. Theunissen, Boake, and Kourie (2005) propose a theoretical hybrid approach including the best of ASD and FOSSD, while minimizing the tension points, which is suitable for companies. This work was extended by Theunissen, Kourie, and Boake (2007) who report about a fictional ASD development team focusing on using FOSS products. Both papers do not report on any practical application of this hybrid approach to an actual company or FOSS project.

Turnu, Melis, Cau, Marchesi, et al. (2004) and Turnu, Melis, Cau, Setzu, et al. (2006) introduced TDD to their open source simulation model to research the effects of TDD on FOSS. They used a simulation model for their research, because empirical data of FOSSD with and without TDD are very difficult to obtain. Turnu, Melis, Cau, Marchesi, et al. (2004) made the following assumptions to introduce TDD into their simulation model: While the average time necessary to write a line of production code increases (automated tests have to be written as well), the number of defects injected during coding and the debugging time to fix a single bug decrease (Turnu, Melis, Cau, Marchesi, et al., 2004; Turnu, Melis, Cau, Setzu, et al., 2006). The model simulated the evolution of a FOSS project after the initial kernel of the system was developed by a small number of core contributors before it was released to the public to join development (Turnu, Melis, Cau, Setzu, et al., 2006). For more details regarding the calibration of the model read Turnu, Melis, Cau, Marchesi, et al. (2004) and Turnu, Melis, Cau, Setzu, et al. (2006). Their simulations predicted a higher code quality for the model with TDD than for the model without it, while productivity of the average contributor stayed the same. Introducing TDD to the simulation model had no effect on the number of contributors. Only one practice was introduced to the FOSS simulation.

5. Related Work

Düring (2006b) and Düring (2006c) and Sigfridsson et al. (2007) studied the PyPy FOSS project, an alternative implementation of the Python language. Düring (2006c) studied the effects sprint-driven development had on the project. PyPy is a hybrid project employing practices from Agile and Distributed Development in the context of an Open Source community. It uses tools and practices from both parts, e.g., sprint-driven development, TDD, open communication climate, focus on collaborative approaches, a public issue tracker, continuous integration, and code review. It was partly funded by the European Union from 2004 to 2006. In the case of PyPy sprint-driven development means, that people meet in person for up to one week and jointly work on the code base. These sprint meetings were supplemented by 30 minute time-boxed weekly sync-meetings, which provided a time for regular discussions and integration of ongoing work.

Between 2003 and 2004 six sprints were arranged in different European cities, which encouraged participation. After the EU-funding sprinting was carried out every 6th week. As a consequence of the sprinting the number of subscribers on the development list increased from around 150 to over 250. Düring (2006c) concluded that the PyPy sprint-driven approach makes Agile and Distributed Development more combinable and helps ensuring quality in projects with hybrid cultures and methodologies. Moreover sprints improved the cohesion as well as the community through personal contact (with core developers), served as training sessions for coding and in the development methods, and supported design decisions, high-level requirements discussions, while at the same time they minimized the risks of Distributed Development. Düring (2006c) also concluded that sprint-driven development as a methodology needs an agile group of people, so called CRACK performers (“Collaborative, Representative, Authorized, Committed, Knowledgeable”) (Boehm and Turner, 2003b). She even went so far as to say that in some sense the people are the methodology and cites Alistair Cockburn

“The fundamental characteristics of ‘people’ have a first-order effect on software development, not a lower-order effect.”

Only some XP practices were integrated into PyPy and

“I think it would be a mistake to say this is a combination of agile and open source. I think it has applied one of the ideas of

5.2. FOSS and Agile Software Development

agile development to the very difficult problem of distributed software development.”

said James Shore, a leading practitioner of agile methodology, in Goth (2007).

Sigfridsson et al. (2007) studied the effects of sprint-driven development on learning, the dissemination of knowledge and the expansion of the FOSS community in PyPy. They saw PyPy sprints as a perfect example of situated learning. At the same time they ensured the sustainability of the PyPy community by introducing newcomers to the project through mentoring. Only some XP practices were integrated into PyPy and only sprinting was evaluated regarding its effects on learning.

Porruvecchio et al. (2007) studied the relationship between ASD and FOSS by integrating a Health Information System into an Italian hospital following an XP approach. XP was not introduced to a FOSS project, it was used by a software development team developing code extending the functionality of the FOSS project.

Deshpande and Riehle (2008) analyzed the code contribution size of more than 5000 active FOSS projects over the projects' life spans. They concluded that FOSSD did not change code integration practices and claimed that CI did not significantly influence FOSS developers behavior up until then. Either FOSS projects did not adopt CI or that they always used it, so that the appearance of agile methods did not change anything.

Adams and Capiluppi (2009) investigated the effects of sprinting on productivity in two FOSS projects, Plone and KDE PIM. Sprinting in this context means, that a small group of people co-locates for a short period of time to work on a specific aspect of the overall project. They wanted to determine if sprinting increases productivity and if FOSS projects remain more productive after sprints. To this end code repository logs were analyzed and average commits per day were used as metric. Productivity was measured 7 days before, 7 days after the sprint and during the sprint. For Plone no overall increase in general productivity could be measured during the sprint for the selected six sprints. For KDE PIM sprints an increase in general productivity could be found for the majority of the selected sprints. After the sprints the productivity of the Plone project was

5. Related Work

not larger than compared to before the sprint for the majority of the sprints, which changed slightly for the last sprint events. For the KDE PIM project an increase of productivity (compared to the general base rate) was still accomplished during the week after the sprints.

Tsirakidis, Koebler, and Krmar (2009) studied success and failure factors of two agile teams of a FOSS organization regarding their team performance. For both teams the following agile practices were assumed: Scrum, TDD, collective code ownership, refactoring, coding standards, and co-located and distributed pair programming. Preliminary results showed four success factors: constant and synchronous communication, FLOSS development experience in accepting and handling the environmental limitations, consistency in the methodological development approach, geographical dispersion management through an extensive testing culture, and two failure factors: information hiding in separate mailing lists, inconsistency in the methodological development approach (Tsirakidis, Koebler, and Krmar, 2009). The practices were only assumed and did not include all XP or Scrum practices.

Wusteman (2009) studied the OJAX project, where open source design methods and usability testing were used iteratively. The results of the tests were used to develop agile uses cases. Apart from iterative design and usability testing, they did not report on a specific ASD methodology or agile practices. As the next step the adoption of agile development methods by the library community was mentioned.

Lavazza et al. (2010) report their experiences of using Scrum (Schwaber and Beedle, 2001) for the development of a FOSS Java tool. Only the tool was FOSS, the people involved were paid for their contributions. Lavazza et al. (2010) applied a modified Scrum process to the development of MacXim (Model And Code Xml-based Integrated Meter), a tool in the QualiPSo² project. MacXim extracts static measures from source code (Crisà, Bianco, and Lavazza, 2006). There was a distinct project manager monitoring project requirements and deadlines. This role later changed to product owner i.e., “the QualiPSo manager in charge of tool development”. This role worked three days per week on the project. After one year a dedicated development team was formed, including two (later three) junior developers working

²<http://qualipso.icmc.usp.br/OMM/> – retrieved on 02.09.2017

5.2. FOSS and Agile Software Development

full-time on MacXim. Developers were spread over several kilometers. For daily stand-ups a forum and a video conferencing system were used. The team met online every two weeks (four hour meetings in video conference) and additionally they met in person every month to mitigate risks due to lack of direct interaction. Lavazza et al. (2010) concluded that it seems to be possible to integrate Scrum into an ongoing distributed FOSS project process and it supported the control of the development process, but did not change the quality of the product or the productivity of the team significantly. They also reported improved communication between developers. Regarding the introduction of Scrum into a FOSS development community they concluded

“As a final consideration, applying the pure Scrum methodology to an OSS development community is not possible because of geographical, cultural, and communication problems.”

Gary, Enquobahrie, et al. (2011) describe a case study of the IGSTK (Image-Guided Surgical Toolkit) project, which is a safety-critical system (Bowen and Stavridou, 1993) and a FOSS project. A set of best practices (Gary, Ibáñez, et al., 2006) was used. To ensure code quality there was a code review at the end of every sprint, unit testing, CI and full code coverage were employed. To accommodate the safety-critical domain the best practices lightweight requirements management, safety-by-design, CI and testing, and architecture validation were used. IGSTK adapted two ASD approaches to their needs, Scrum management practices and XP coding practices. A FOSS community supports their work.

Magdaleno, Werner, and Araujo (2012) conducted a quasi-systematic literature review of papers regarding the reconciliation of plan-driven, ASD and FOSS software development models. Their review indicates, that

- there are very few studies reconciling all three software development models.
- the large number of studies is reconciling plan-driven and agile models.
- work about reconciling ASD and FOSS development models is still incipient. In general, only one particular practice is integrated, e.g., introducing TDD into FOSS.

5. Related Work

- organizations can reconcile them on organizational, group or process level.

Okoli and Carillo (2012) compared FOSSD, ASD and plan-driven software development by comparing their pragmatic perspective in the different stages of a software development project and not specific projects. For example, number of developers is usually larger (>20 people) in plan-driven methodologies than in ASD (<20 people), and FOSS communities can have one or hundreds of contributors. Regarding the organizational culture in plan-driven approaches it is usually centralized and well-organized, in ASD decentralized and highly flexible and FOSS can work with centralized and decentralized cultures.

5.3. Summary

Some FOSS projects apply certain agile practices, e.g., Eclipse employs TDD, refactoring and Feature Driven Development since the beginning, Netbeans employs Behavior/Feature Driven Development since the beginning and partially uses automatic testing (Murgia et al., 2009), and CI, PyPy employs TDD and CI, and PyPy, Plone and Canonical use sprint-driven development. But all of them apply only a few ASD practices (sometimes only one) (Magdaleno, Werner, and Araujo, 2012) to a university course or a FOSS project or use FOSS elements for university courses.

This makes Harzl (2016) and Harzl (2017), the first study about comprehensively combining ASD and FOSS. Gandomani et al. (2013) confirm the lack of literature regarding the combination of both methods.

6. Action Research

This chapter describes the research methodology used for this thesis.

6.1. Basics of Action Research

The term AR was initially coined by Kurt Lewin in 1946 (Lewin, 1946) to describe a new approach in social research. This approach combined the change of a social system through research acting in or on the social system and theory creation (Susman and Evered, 1978). Rapoport (1970) defines AR like this:

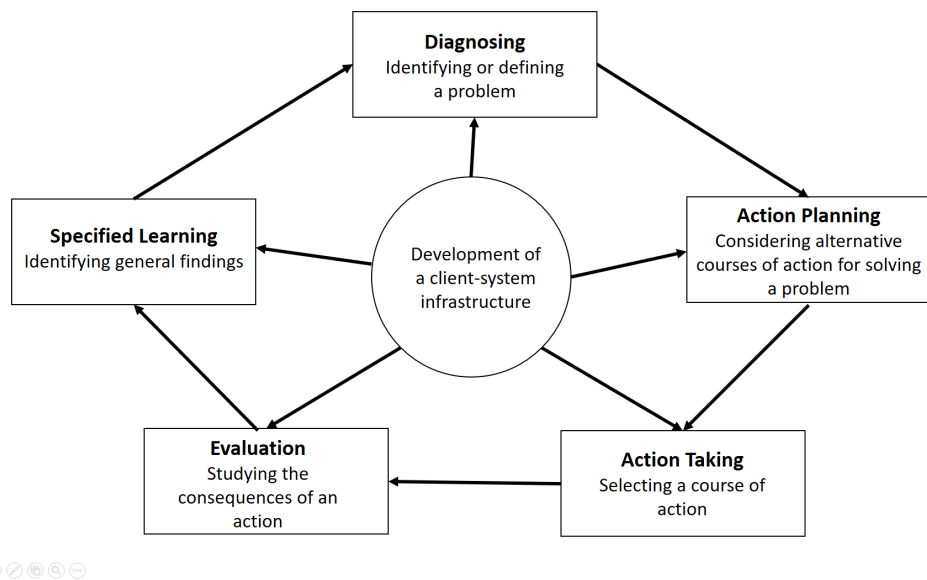
“Action research aims to contribute both to the practical concerns of people in an immediate problematic situation and to the goals of social science by joint collaboration within a mutually acceptable ethical framework.”

Susman and Evered (1978) extend this definition by a third aim, “to develop the self-help competencies of people facing problems”. In AR researchers and practitioners collaborate to solve real world problems through theoretically informed actions (Greenwood and Levin, 2007; Harzl, 2016). Another view of AR is that of a cyclical process containing five phases; diagnosing, action planning, action taking, evaluating, and specified learning (see Figure 6.1).

The characteristics of AR according to Susman and Evered (1978) are:

- “AR is future oriented”, because it strives to solve real-world problems of people and to create a more desirable future for them.
- “AR is collaborative”, because researchers and the client system work together and determine the direction of the research process jointly.

6. Action Research



⏪ ⏩ ⏴ ⏵ ⏶ ⏷

Figure 6.1.: The Cyclical Process of Action Research [adapted from (Susman and Evered, 1978)].

6.1. Basics of Action Research

- “AR implies system development”, because it generates an infrastructure of the system, which alleviates the current, sub-optimal or problematic, situation and generates new knowledge about the system’s processes.
- “AR generates theory grounded in action.” Theory informs the diagnosis of an organization and appropriate actions to take. In turn the evaluation of the consequences of these actions can inform theory.
- “AR is agnostic.” Theory and prescriptions for actions have to be re-examined and reformulated, based on the consequences of the actions taken, throughout the research process because they are the product of actions previously taken. The researcher cannot fully know the consequences of selected actions ahead of time.
- “AR is situational.” Every research situation is unique based on people, relationships between people and so on. The AR researcher takes actions based on the current understanding of the situation and the stakeholders involved. Actions are planned in consensus with stakeholders, so they are going to produce the intended outcomes.

Chein, Cook, and Harding (1948) differentiate between different types of AR, depending on how many or which phases are carried out together by the researcher and the client system.

- Diagnostic AR: Researcher is only involved in data collection for diagnosis, and feeding the data back to the client system.
- Empirical AR: Researcher evaluates the client system’s actions and feeds data back to it.
- Participant AR: Researcher and client system jointly conduct diagnosing and action planning.
- Experimental AR: Researcher and client system jointly conduct (almost) all phases.

AR is usually conducted in iterative cycles of plan, act, observe, and reflect (Lewin, 1948) or variations of this approach, e.g., the spiral of self-reflective cycles (Kemmis, McTaggart, and Nixon, 2014) with planning, acting and observing, reflecting, re-planning, acting and observing or the approach from Susman and Evered (1978). For more information on this approach see Section 6.2. Often the stages of the cycles overlap and are not

6. Action Research

clearly separated. The AR methodology is more responsive to the situation than other methodologies (Kemmis, McTaggart, and Nixon, 2014). As Kemmis, McTaggart, and Nixon (2014) puts it

“For critical participatory action research, the criterion of success is not whether participants have followed the steps faithfully, but whether they have a strong and authentic sense of development and evolution in their *practices*, their *understandings* of their practices, and the *situations* in which they practice.”

This was very important for this study, because the study participants had a strong interest in improving their situation and work practices.

6.2. Study Approach

For this thesis a participatory AR approach, namely a modified version of Susman and Evered’s approach (Susman and Evered, 1978), was used as a research method. Figure 6.1 shows this cyclical model with its five stages *diagnosing*, *action planning*, *action taking*, *evaluating*, and *specified learning*. All steps were discussed and conducted together with the study participants. In the classification of Chein, Cook, and Harding (1948) it would be an experimental AR.

The following phases were conducted:

- Diagnosing
- Cycle zero including a user analysis and a stakeholder analysis. It was added after the diagnosing phase at the beginning of the study.
- Five AR cycles consisting of action planning, action taking, evaluation and specified learning.
- Two coaching sessions, one at the beginning of cycle zero and one at the beginning of the second AR cycle, see Section 8.3.

Finally the results were analyzed, presented to the team and published in Harzl (2016) and Harzl (2017).

6.3. Selection of Action Research

Several reasons contributed to the selection of AR as research methodology. The main reasons are:

- Strong focus on practical outcomes and not only scientific outcomes by the research participants. According to Dick (2000) AR is well suited to achieve both at the same time.
- Increased commitment of the study participants through participation in the research (Dick, 2000).
- AR is a flexible approach, which allows to explore areas, where theory is not fully developed yet (Edmondson and McManus, 2007; Kampenes, Anda, and Dybå, 2008).
- AR allows for changes in later cycles, if the observations from earlier cycles suggest them.
- AR investigates the actual practices, not proclaimed practices.

“It involves learning about the real material, concrete, particular practices of particular people in particular places.” (Kemmis, McTaggart, and Nixon, 2014)

6.4. Researcher Role

The AR is designed as insider in collaboration with other insiders, but power relations could still play a part. In Catrobat the researcher was responsible for organizational and supporting processes, see Project Coordinator in Figure 4.6, for example creating user accounts and giving a short introduction into the overall project. Other roles like developer, senior member or team coordinator were not taken on. However, project coordinator and project founder, Mr. Slany, work closely together. Mr. Slany is not only an integral part of the project, but also the professor grading the students, who do their Bachelor thesis or Master project in the FOSS project. Thus, although researcher and project coordinator are not official hierarchical roles, contributors probably see the roles as having informal power within the organization.

7. Getting Ready for Combining FOSS and Kanban

This chapter describes phenomena and improvements for the whole Catrobat organization and not only for the studied sub-team. Because the issues discussed in this chapter did concern all teams and the solutions were rather time and resource consuming it did not make sense to apply them just to one team.

FOSSD is highly distributed software development, very knowledge intensive and community-based. Because of these characteristics FOSSD has to tackle knowledge management challenges (Ciborra and Andreu, 2001; Edwards, 2001; Becking et al., 2005; Crowston, Wei, et al., 2012). In this chapter changes to the organization and tools of the Catrobat project will be described. They were changed to address knowledge management (see Section 7.4.4) and other challenges. Basic project management tools and processes were introduced, which helped enable future research. Before these improvements were put into place the number of project members was unknown, development information was scattered over different platforms and other information e.g. meeting notes, to-do lists were stored over various services. Nobody knew which information was available and much information was lost in private cloud services when a member left the project. This chapter is based on Fellhofer, Harzl, and Slany (2015).

7.1. Abstract

This chapter describes problems that arose with the scaling and internationalization of the FOSS project Catrobat. Problems included lack of a

7. Getting Ready for Combining FOSS and Kanban

centralized user management, insufficient scaling of communication channels, and the necessity to adapt agile development techniques to remote collaboration. To solve the problems a mix of open source tools (Git, IRC, LDAP) and commercial solutions (Jira, Confluence, GitHub) was chosen, because this mix fitted the needs of the project best. Other projects can benefit from the lessons learned during the reorganization of Catrobat's knowledge base and communication tools, as infrastructure changes can be very labor-intensive and time-consuming.

7.2. Introduction

Scaling and internationalizing a FOSS project is not an easy task. The project grew from five contributors in 2010 to over 130 contributors in 2014, which lead to various organizational problems. In this paper experienced problems will be shared, as well as approaches to solve them, and lessons learned along the way. Other FOSS projects can profit from these experiences.

Details about Catrobat can be found in Chapter 4. Catrobat was initiated by Wolfgang Slany and a team of students from Graz University of Technology, seeking challenges besides their studies and ways to practice what they learned at the university. This eagerness to apply software development principles taught in courses heavily influenced the basic structure of the project, with all its advantages and disadvantages.

On the positive side the usage of agile development methods such as Kanban (Anderson, 2010; Hiranabe, 2008) enabled project members to stay flexible and to easily adjust the scope of the project on the fly. XP, especially pair programming (Andres and Beck, 1999), facilitated knowledge transfer between developers. TDD (Beck, 2003) ensured that the code remained working and testable while new developers joined and former developers left the project. A dedicated UX team applying the personas method (Husain et al., 2012) and other usability techniques focused on the users. This is particularly important because, in contrast to most FOSS projects, Catrobat's developers (mostly university students) are not a subgroup of Catro-

bat's targeted users (mostly children and teenagers). The UX team helps to create an understanding of user needs in all developing teams.

On the negative side most of Catrobat's contributors are more or less inexperienced as software developers and they have to familiarize themselves with the principles used, which steepened the learning curve. Communication was mainly face-to-face, which is good for localized agile teams, but leaves many decisions undocumented, which is disadvantageous for a larger, distributed FOSS project. Discussions tend to get started over and over again, when nobody remembers why and based on what information a decision was made in the first place. Communication problems will be discussed later in more detail, because they are at the core of Catrobat's difficulties with scaling and internationalizing.

The increasing number of contributors from five contributors in 2010 to over 130 contributors in 2014, and the participation of international contributors in the project made organizational problems visible. Documentation was not equally available for everyone, the entire current project status was not visible online and important communication channels were missing. For example most of the contributors did not use IRC, which is often an integral infrastructure of FOSS projects and compulsory for the participation in GSoC¹, in which Catrobat participates since 2011. To address these problems and to be able to integrate more and international contributors, the project infrastructure, the ways to communicate and some tools had to be changed.

In Section 7.4 the problems will be identified in greater detail and approaches to solve them will be described. Section 7.5 contains the lessons learned on the way from a small, localized project to a larger, more international project.

¹A global program from Google to support FOSS projects by sponsoring *students* and pairing them with *mentors* to develop given tasks over summer (<https://developers.google.com/open-source/soc/>)

7. Getting Ready for Combining FOSS and Kanban

7.3. Related Work

The main goal was to enable and facilitate contributions from contributors around the world. Various studies ((Korkala and Abrahamsson, 2007; Layman et al., 2006; Poole, 2004; Scacchi, 2010; Schümmer and Schümmer, 2000)) highlight how important communication is in a FOSS project or in projects which use agile development techniques.

The work done focused on optimization of communication for FOSS and agile software development projects. Layman et al. (2006) recommended among other things that *“when face-to-face, synchronous communication is infeasible, use an email listserv”* and *“use globally-available project management tools”*. Korkala and Abrahamsson (2007) recommended to *“enable and support direct communication between the developers”*. Some technologies and common practices used in FOSS development such as instant messaging, IRC, news postings, how-to guides, FAQ, or Wikis are listed in Scacchi (2002) and Scacchi (2010). In Yamauchi et al. (2000) technologies such as versioning systems and to-do lists are mentioned. Difficulties for newcomers like *“selection of a suitable task”*, *“lack of up-to-date development documents”*, or *“no response from core developers for their doubts”* are mentioned in Shibuya and Tamai (2009).

Section 7.4 contains the implementation of recommendations from this Section.

7.4. Optimizing Services for Distributed Participation

The Catrobat project grew faster than the supporting infrastructure, which led to organizational problems. For each part of the infrastructure the initial situation will be described first, then problems which occurred over time and finally how the problems were resolved. Usually solutions were determined through collecting requirements and looking for appropriate tools and researching other FOSS projects. One important criterion was, that software solutions should be free of charge for FOSS projects.

7.4.1. User Management

Initial Situation. In the beginning there was no user management. Every piece of infrastructure (for example: instant messaging, source code repository) had its own built-in user management, and accounts were created manually on demand. Some services were used through shared user accounts. This made accountability impossible.

Resulting Problems. The effort necessary for user management and maintenance increased tremendously with the increasing number of contributors, because every user account had to be created manually and every change had to be populated manually to all platforms. This resulted in missing and outdated account information. Rights management was not even a topic. Sometimes this lack of rights management caused inexperienced contributors to inadvertently change or delete infrastructure. Shared accounts made it impossible to trace who made changes to project services. On the side of the contributors the account management was elaborate too, as for every service, contributors had to use different credentials.

The goal was to simplify the management of user databases and to support contributors by providing them with only one account for (almost) all Catrobat infrastructure.

Method of Resolution. Most parts of the project's infrastructure have a built-in support for Lightweight Directory Access Protocol (LDAP), so LDAP was the most suitable solution to simplify user management. LDAP groups are used for various reasons:

- different experience levels need different rights
- different contributor groups need different resources and services
- no shared accounts, so there are clear responsibilities
- infrastructure administration should be left to experienced contributors

The goal was to design a user management which serves experts and beginners. Experts should have all the rights they need, and beginners should not be overwhelmed by too many services and rights too soon.

7. Getting Ready for Combining FOSS and Kanban

Unfortunately not all services support LDAP. For example externally hosted services such as GitHub do not support foreign user directories and still need extra maintenance. Other services like Crowdin² support OAuth³ as authentication method, but to take the user groups and corresponding rights needed for Catrobat into account, the service's Application Programming Interface (API) or an additional configuration interface must be used.

7.4.2. Communication

Catrobat is allowed to use a room at TUG, where local contributors can meet, discuss, and code. In the beginning all contributors fit in the room, and communication was mainly face-to-face. The reliance on face-to-face was very beneficial in the beginning but caused some communication and documentation problems later on as already mentioned in Section 7.2.

Instant Messaging

Initial Situation. Other means of communication were and still are e-mail, mailing lists⁴ as recommended in Layman et al. (2006) and Instant Messaging (IM). In the beginning a Skype group chat was used for project discussions with all contributors.

Resulting Problems. Many contributors joined the Skype group chat but did not participate actively, because they preferred face-to-face communication, e-mail, or were overwhelmed by the number of messages that did not concern them. This massive number of messages was a direct result of the growing number of contributors.

One problem with Skype is that group chats are invite-only, which makes it difficult for aspiring contributors to join the discussion. They first had to find a project member to invite them. Another problem with the Skype

²Tool to help non-developers to translate texts (<https://crowdin.com/>)

³<http://oauth.net/>

⁴<http://catrob.at/maillinglist> – retrieved on 20.02.2015

7.4. Optimizing Services for Distributed Participation

group was the language used. Almost all messages were in German, because all of the initial team members spoke German.

In an attempt to open up the project and allow for internationalization, an IRC channel was created, which was open to everyone and where it was obligatory to use English.

This attempt failed, because project issues were, as a matter of habit, still discussed in Skype and hardly anyone used IRC. So theoretically the project maintained an IRC channel, but interested contributors still got *“no response from core developers for their doubts and support request”* (Shibuya and Tamai, 2009).

As a consequence the project appeared nontransparent to remote contributors, and it became obvious that face-to-face communication and chats on Skype, as useful and familiar as they are in everyday life, are not enough for an international FOSS project.

Method of Resolution. A decision was made to switch the whole synchronous communication to IRC and to delete the Skype group chat. The reasons for this decision were:

- One IM platform for all purposes.
- Everybody can join channels, he or she is interested in.
- “Irrelevant” messages are reduced, because messages are posted only to the channels where they belong.
- Faster responses to questions from aspiring contributors due to increased online time of project members.
- Topic specific channels can be created and deleted easily, when needed.

To make the communication with IRC more attractive for the contributors they were provided with an IRC bouncer which records all messages when the user is offline and replays them when the user goes online.

One disadvantage of IRC as communication platform is that the technology seems old-fashioned to Catrobat’s contributors and most of them have never used IRC before. Another disadvantage is that it is more time consuming to configure IRC with the bouncer than it is to configure Skype.

7. Getting Ready for Combining FOSS and Kanban

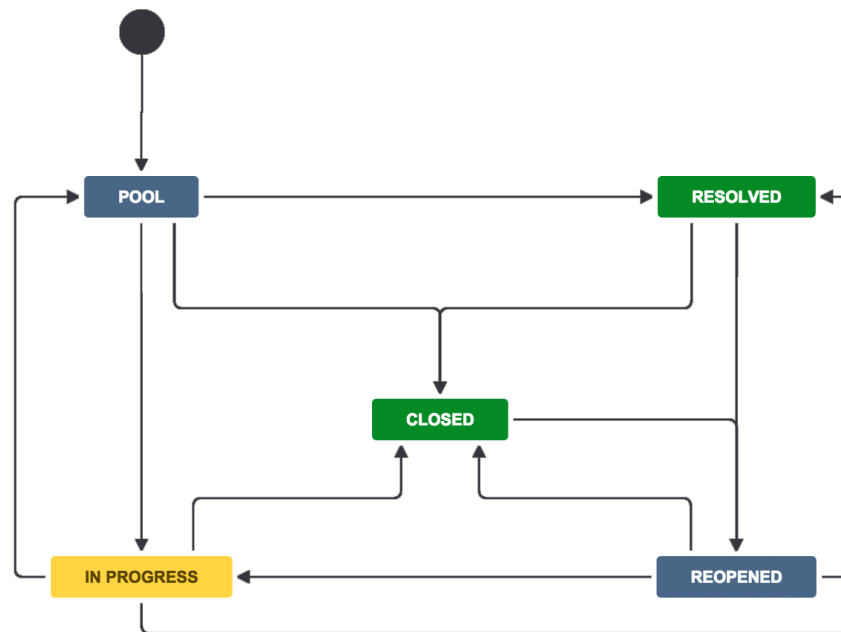


Figure 7.1.: The default Jira workflow for issues.

Contributors have to authenticate to freenode⁵ and to the project bouncer with different credentials.

Meanwhile IRC is widely accepted by the community and the communication improved compared to Skype because contributors only have to join and read channels they are interested in and people are by now used to IRC.

7.4.3. Agile Development Management

Initial Situation. As already mentioned Catrobat is allowed to use a room at the TUG for the project. There several things are available to the project contributors, like, whiteboards serving as Kanban boards, as well as story cards on paper. Initially every software development team had its own

⁵IRC network where the project runs all its channels (<https://freenode.net/>)

7.4. Optimizing Services for Distributed Participation

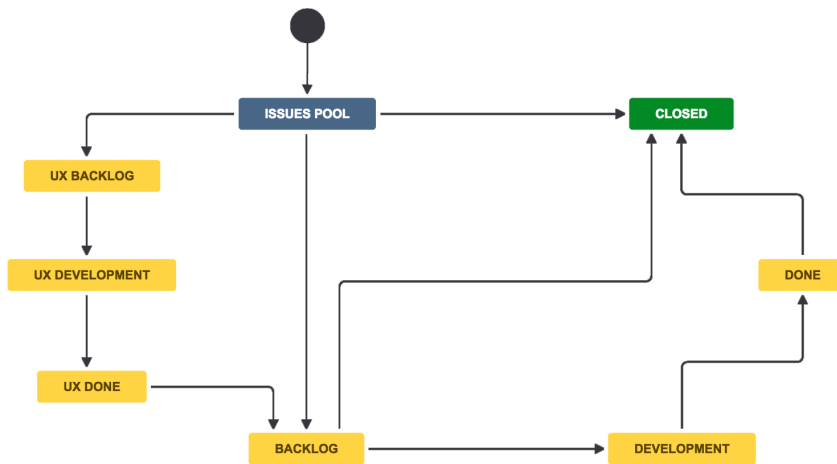


Figure 7.2.: This is the customized workflow for issues with additional steps for reviewing by experienced developers, the UX team, and code acceptance.

Kanban board in the room, though this became impossible as the number of projects grew. The project uses GitHub⁶ as source code repository. The integrated issue tracker was used not just to track bugs and issues reported by users but also as a digital version of the local Kanban boards. Labels of issues were used to indicate which kind of issue it was (bug, story, enhancement, ...), the current working status (to do, in development, done, ...), the priority (low, medium, high, or critical) and which part it affected (development environment or interpreter). To enable children and teenagers to report bugs and issue requests without bothering with Github, a Google Group⁷ was created.

Resulting Problems. User stories and bug reports had to be synchronized between three different platforms (local Kanban board, Github, Google Group). It is not hard to imagine that this had negative consequences: the local Kanban board was often outdated and the issue tracker did not cover stories, which were created locally on the Kanban board. The Github issue

⁶<https://github.com/Catrobat> – retrieved on 20.02.2015

⁷<http://catrob.at/PocketCodeUserForum> – retrieved on 20.02.2015

7. Getting Ready for Combining FOSS and Kanban

tracker did not support the Kanban board and lacked the functionality of hiding security relevant issues from the public.

Method of Resolution. In 2013 the decision was made to test distributed agile development with online agile boards. The choice fell on Jira⁸, because it is used by other FOSS projects like Apache⁹ or Cyanogenmod¹⁰, is free for FOSS projects and provides much more functionality like, for example Redmine¹¹, a FOSS version of online agile boards.

As a pilot test Jira was used during GSoC 2013. Every mentor/student pair received their own project with a simple Kanban board. Over this trial period it became apparent, that the original workflow of issues had to be customized to fit Catrobat's developing principles (specifically review of newly created issues and code acceptance by experienced developers) and to integrate usability reviews into the workflow. Figure 7.1 shows the original workflow and Figure 7.2 shows the customized workflow.

After the successful pilot test Jira was introduced as a globally-available project management tool (recommended by Layman et al. (2006)) and as replacement for the local whiteboards. This proved to be another important step towards becoming an international FOSS project.

7.4.4. Knowledge Management

Initial Situation. Initially Google Docs, Dropbox, and a Wiki system were used for distributing documents and information. The Wiki system grew more or less naturally and the structure was seldom revised. For a group of five people, which communicates mainly face-to-face, a less formal documentation management is useful, but once the number of contributors grew, severe problems surfaced.

⁸Planning and tracking tool for agile project management by Atlassian (<https://www.atlassian.com/software/jira>)

⁹<https://issues.apache.org/jira/secure/Dashboard.jspa> – retrieved on 20.02.2015

¹⁰<https://jira.cyanogenmod.org/secure/Dashboard.jspa> – retrieved on 20.02.2015

¹¹<https://www.redmine.org/> – retrieved on 20.02.2015

7.4. Optimizing Services for Distributed Participation

Resulting Problems. Document owners left the project and the owner rights were not transferred. Documents were not updated, became outdated, and there was no general overview of documents and their content. File sharing tools like Google Drive and Dropbox are neither integrated in the Wiki nor are they supporting the new user management with LDAP. These facts make it harder to share and find current information, documents, and their owners.

The Wiki is not well maintained on all of its pages, and the organically grown structure can be an additional hurdle if one does not take the time to learn how to use the included tools. As discussed in Shibuya and Tamai (2009), outdated development documents lead to difficulties for newcomers. Even some of the experienced project members tend to avoid using the Wiki because of its partly outdated content and complex structure.

Method of Resolution. Current systems would need major rework and would not integrate well with Jira, therefore the decision was made to use Confluence¹² instead. Confluence is used in FOSS projects like Apache¹³, and issues from Jira can be easily referenced. The switch of systems provides the opportunity to revise the structure and to eliminate or correct outdated information. This will make it easier to find up-to-date and useful information.

Scacchi (2010) introduced *FOSSD informalisms*, which are easy to use and publicly accessible resources like: threaded discussion forums, group blogs, news postings, how-to guides, to-do lists, and FAQ. Most of these technologies are supported either by Jira or Confluence and they will support contributors in their search for information.

¹²Wiki system by Atlassian which is free for FOSS projects - <https://www.atlassian.com/software/confluence>

¹³<https://cwiki.apache.org/confluence/dashboard.action> - retrieved on 20.02.2015

7. Getting Ready for Combining FOSS and Kanban

7.5. Lessons Learned

7.5.1. Human Related

Introducing and switching to IRC as the new IM service was time consuming and resulted in resistance. The confusions described in Section 7.4.2, the out-dated and uncomfortable user interface of IRC clients, and the under-estimation of the need for change management were mainly responsible for the long (and still ongoing) resistance. To reduce this resistance to change techniques described in Aladwani (2001) and Kotter and Schlesinger (1979) were used during the introduction of Jira. More developers were involved during the configuration period of Jira and the workflow adaptation. The benefits of the new system were communicated more clearly. This communication of benefits, training, involvement of GSoC mentors and optimizing the workflow probably led to a smooth change of the issue tracker and the introduction of Jira as a project management tool.

7.5.2. Technology Related

Centralized management of team member accounts and information about them should ideally be introduced from the very beginning as it tremendously simplifies the administration of contributors and saves a lot of time later. User rights management should be well structured and at the same time adjustable to future changes. Integrated services are preferable from a maintenance perspective, because they save time and effort related to organizational tasks. This time can then be spend on project goals. Not all external services, e.g., GitHub support LDAP. But if an API is available for that service, and user maintenance for this service consumes a lot of time, at least some parts should be automated by scripts.

7.6. Future Work

At the publication of this paper the transition of the services was not finished for every team, but will be finished later. Nevertheless, improvements never stop and further tool and process changes are happening and are going to happen. The new Wiki contains how-to guides, FAQ, blog-like news entries, and meeting notes to be more transparent and to provide new contributors with the most relevant and up-to-date information (Scacchi, 2010). To support the change (Aladwani, 2001; Kotter and Schlesinger, 1979) of the Wiki system a survey was conducted to detect main concerns and problems with the current Wiki and senior contributors were involved in the development of the new Wiki. The survey was answered by 25 members. Figure 7.3 shows the main areas for improvement selected by the survey participants. The most selected topics were clear arrangement, up-to-dateness of the information and completeness of the information.

Answers to the open ended question “Please describe in detail what should be improved, so we can derive concrete improvements” most often included the suggestion to group similar topics under one umbrella term so to provide a more hierarchical structure and a neatly arranged start page. These two points were then implemented in the new system, Confluence. Also implemented were the improvement ideas of a section for newcomers with information how to get started and distinct team pages. The issue of up-to-dateness of information, which was also mentioned very often in the open-ended question is and will remain a constant challenge. At least Confluence offers convenience features like automatic notifications, when pages are changed or blog entries, which automatically disappear from the start page after a while, making it less overloaded with outdated information. The whole survey can be found in Appendix B.

7.7. Conclusion

As explained in Section 7.3 communication and documentation are essential parts of ASD and even more important in distributed development (Shibuya and Tamai, 2009). Face-to-face communication is suitable

7. Getting Ready for Combining FOSS and Kanban

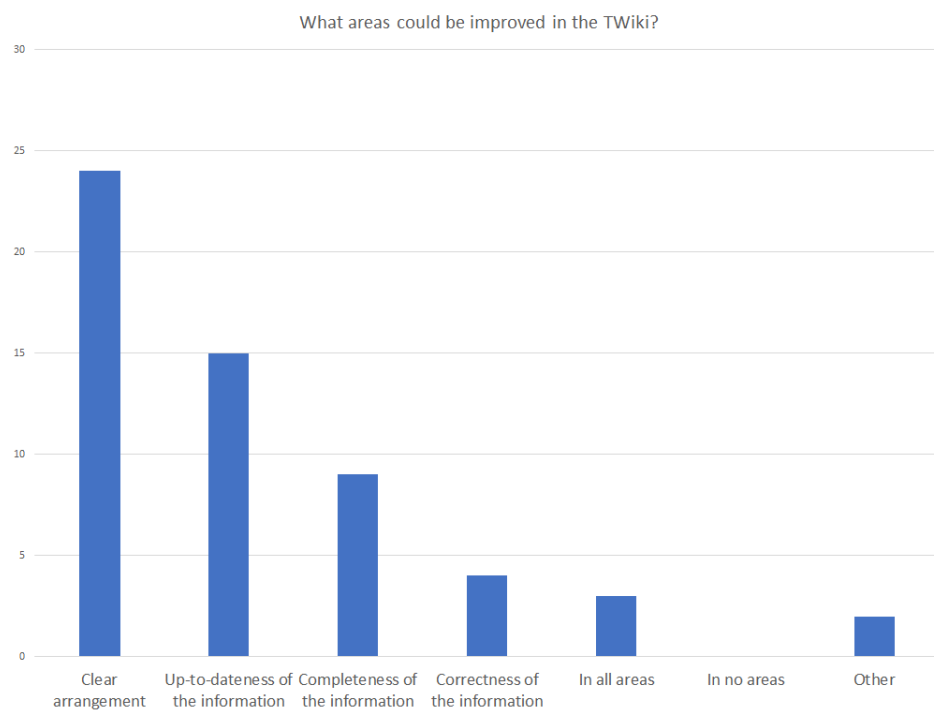


Figure 7.3.: Areas for improvement selected by the survey participants. Multiple answers could be selected.

7.7. Conclusion

at the beginning, but with the growth and internationalization of a project, good communication channels and project management tools have to be introduced. Every change of workflow or established tools is time consuming and needs proper change management to succeed. So before changing major aspects it should be properly considered, if the changes are worth the effort. Appropriate user and rights management simplifies the administration of infrastructure and contributors. It saves time and supports contributors by giving them access to the project's services ideally with one account.

8. Combining FOSS and Kanban: An Action Research

This chapter is based on Harzl (2016) and Harzl (2017).

8.1. Background

The FOSS project under study is described in Chapter 4, the researchers' role within the project is explained in Section 6.4, and reasons for integrating Kanban into a hybrid student FOSS project are given in Section 4.7. The research methodology is described in Section 6.2.

8.2. Study Participants

Nine people in total participated in the study, but due to people leaving and joining the project, only six to eight people were part of the team at any time. Initially six people started the AR, two people joined the team after cycle zero, so eight team members took part in the first and second cycle. Eight people took part in the third cycle, but individual members changed, one person left the team after the second cycle was completed, but filled in the questionnaires about agile estimating and motivation of the third cycle, due to extensive experience with story point estimation. Questions about T-shirt size estimation were not answered or answered neutrally (did not use it, undecided) by this person. One person joined in the beginning of the third cycle, but did not take part in the agile estimation and motivation questionnaires due to lack of experience in the team. The

8. Combining FOSS and Kanban: An Action Research

fourth and fifth cycle were finished by seven people, because one person left after the third cycle.

8.3. Action Research

As already explained in Section 6.2, a modified version of Susman and Evered's approach (Susman and Evered, 1978) was used as a research method. The cyclical model contains the five stages *diagnosing*, *action planning*, *action taking*, *evaluating*, and *specified learning*. One distinct diagnosing phase and five regular AR cycles consisting of action planning, action taking, evaluation and specified learning were realized. Apart from the distinct diagnosing phase, diagnosing was performed continuously. For this purpose data from the online ticket system Jira were examined and questionnaires and meeting notes were analyzed. Results of the continuous diagnosing will be discussed in the related cycles. An additional cycle zero was added after the diagnosing phase at the beginning of the study. A participatory AR approach was used, all steps were discussed with the study participants and decided jointly.

As already described in Section 4.4 the sub-project uses elements of XP and Kanban. However, an initial questionnaire showed, that team members assess their knowledge about both methods quite differently. While all members think that they have average to very good knowledge about XP, only 17% think that they have very good knowledge about Kanban. The other 83% think that they possess little to no knowledge about Kanban.

This supported the decision to conduct a Kanban coaching session at the beginning of cycle zero and the second cycle. In these sessions topics like Kanban in general, its principles and practices, and terms like flow and kaizen were explained. These sessions were based on two books (Anderson, 2010; Leopold and Kaltenecker, 2013) and one video¹. It was necessary to provide the participants with some theoretical background about Kanban so they could understand its principles and practices. Furthermore, it was also important to enable them to decide how to integrate Kanban

¹<https://youtu.be/6n0Ua6E0250>

practices into their workflow in a way, which would allow them to benefit from these changes. Due to observations made during the diagnosing phase, a cycle zero was added. It included a user analysis and a stakeholder analysis. The first AR cycle was designed to introduce two practices into the team, namely *visualize (the workflow)* and *make policies explicit*. The second AR cycle should then familiarize the team with the principles *limit WIP* and *manage flow*. Before the third cycle participants were provided with a video (Leopold, 2012) and the opportunity to discuss Kanban with the coach during meetings to refresh their Kanban knowledge. The next three cycles examined the following Kanban practices again, *visualize the workflow*; *make policies explicit* and the new practice *implement feedback loops*. Initially it was planned to also introduce *improve collaboratively, evolve experimentally*, this plan was revised during the study, because AR cycles showed, that some practices had to be revisited.

8.3.1. Data Sources

Various types of data sources were used as empirical basis: questionnaires, weekly notes from team meetings written by the participants, researcher notes taken during meetings and discussions, the artifacts produced as part of the user and stakeholder analysis, as well as the artifacts (pictures of the whiteboards) produced during the workflow meeting and the feedback meetings, the team's Kanban board and Cumulative Flow Diagram (CFD).

8.3.2. Data Analysis

Firstly, all questionnaires were statistically analyzed. Secondly, all researcher notes taken during meetings and discussions, were examined for issues of interest to the research and recurring topics or problems. For this purpose statements were "coded" (Corbin and Strauss, 1998) and grouped together, if they had a theme or problem in common, e.g., communication, interaction. These themes then inspired topics for new AR cycles. Thirdly, the results of the team's user analysis and information about the sub-project's

8. Combining FOSS and Kanban: An Action Research

target group, retrieved from the project head, were compared. For this purpose statements from the team and the project head were compared one by one and discrepancies identified. Finally, to accomplish a better visualization of the team's workflow, the Kanban board was analyzed for its adherence to Kanban principles and practices, e.g. pull instead of push and limiting work in progress.

8.4. Action Research Cycles

This section explains cycle zero, the five AR cycles and their phases as described in Section 8.3.

8.4.1. Diagnosing

The sub-project of the FOSS project was inspired by a programming exercise done during a university programming course in 2012. The results showed some promising ideas for a new application, which would fit nicely into the portfolio of the umbrella project Catrobat. However, the code was neither finished (many functions were only rudimentarily implemented) nor was it as structured and neat as it should be, because the course was part of a Bachelor study program and lasted only one semester. Thus, only the ideas remained and the code had to be rewritten.

Some of the students of the programming course decided to do their Bachelor thesis within the scope of the FOSS project. Together with some other students they started to develop the software anew. After a while it became apparent, that the team had been too ambitious and had ignored agile principles, such as working in small iterations and implementing the simplest thing that could possibly work. The team tried to implement too many features in parallel and was overwhelmed by the amount of work necessary to finish it. This situation became even worse when some contributors decided not to use the XP practice pair programming and accomplished most of their work alone, leaving the rest of the team clueless about their contribution to the code. As a result the team ended up with a heap of

8.4. Action Research Cycles

unfinished code and after around a year the sub-project came to a halt. The team decided to start over again.

The second attempt to restart was not successful either. The application did not meet its usability goals and was abandoned again.

By this time the team members' motivation was understandably very low. They struggled with the code, their workflow and their team spirit. Disillusioned by the failed attempts the team coordinator of this sub-project thought that the team could not solve their problems all by themselves and asked me for help.

The main problems of this team seemed to be: no clear goals, underestimation of the tasks at hand, insufficient adoption of agile practices and the overall workflow resulting in frustration and lack of motivation.

In the diagnosing phase regular attendance at weekly meetings made observations of interactions in the team possible. Additionally it helped in establishing more frequent contact and better relationships with team members. Observations made during diagnosing indicated that the team was targeting a different main user group than the project head envisioned. Previous conversations with him about the sub-project and its target group painted a clear picture. He wanted to target beginners in the domain and the team seemed to target people with at least intermediate knowledge in the domain. This observation strongly suggested to investigate this topic further. Therefore, a cycle zero was added to the study. Before starting the AR cycles it was important to determine if there was indeed a misunderstanding between the team and the project head.

8.4.2. Cycle Zero

In cycle zero a user analysis was conducted with the team. It yielded some unforeseen results and led to a re-positioning of the sub-project within the umbrella project with some major changes for the team and its interaction with other teams. As a consequence, a stakeholder analysis was performed as well.

8. Combining FOSS and Kanban: An Action Research

Action Planning and Action Taking

The user analysis was done as a simple brainstorming exercise, where all possible user groups team members could think of were collected on a whiteboard. Afterwards, the team chose the main user groups for which they were developing the application.

The stakeholder analysis was conducted according to Leopold and Kaltefleiter (2013). First, the team determined all stakeholders and listed them on individual pieces of paper. Each paper was sized differently, reflecting the importance of the stakeholder for the team's long-term success. Then, the pieces were put on a table and arranged around the team's mission, which is at the center of the analysis. Stakeholders, who are affected more by the team's day to day work and possible changes, are placed nearer to the center. Stakeholders, who are affected less, are placed further away. Afterwards, the frequency of relationships between all stakeholders was determined. The stronger the relationship, the more lines were drawn between two stakeholders. At last the quality of these relationships was determined as *friendly*, *adversarial*, *love-hate* or *unknown*. For an exemplary stakeholder analysis see Figure 8.1. The actual stakeholder analysis will not be published, because it would threaten the anonymity of the study participants and other project members.

Evaluating and Specified Learning

The user analysis showed some discrepancy between the main user group the team was targeting and the main user group the project head envisioned. This was pointed out to the team and they arrived at the conclusion that the team had to talk to the project head. Luckily he was available at the time of the user analysis and joined the discussion. And indeed there was a misunderstanding. He gave invaluable input, explained to the team which user group they should target and why and it became obvious, that due to a misunderstanding the team had been developing their application for a different target group. This was a very unexpected result for the team and the whole software development came to a halt. Together with the project head the team had to redefine their goals and to rethink their application.

8.4. Action Research Cycles

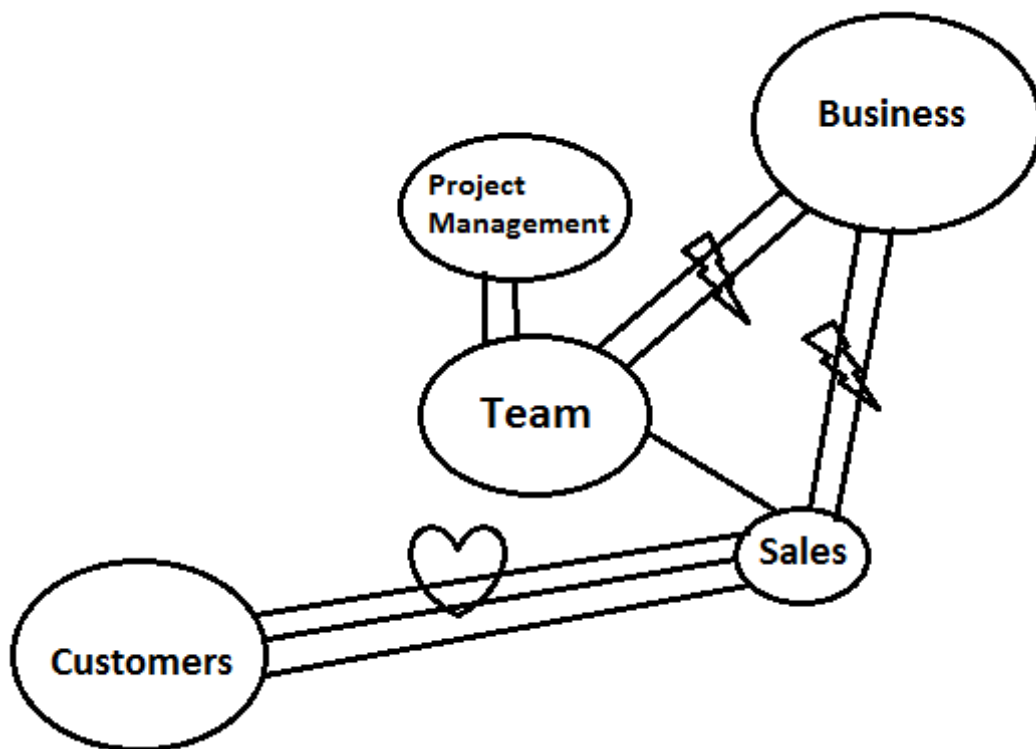


Figure 8.1.: Exemplary stakeholder analysis, adapted from Leopold and Kaltenecker (2013).

8. Combining FOSS and Kanban: An Action Research

Two meetings with the project head followed and the UX team was contacted as well. The project head and the team discussed the goals for their sub-project in detail. The UX team did a user inquiry with four members of the targeted user group. The results of the user inquiry were discussed with the developing team. Based on the input of the four possible users, the developing team and the project head, the UX team designed a digital mock-up of the application. This mock-up showed an absolutely different Graphical User Interface (GUI) than the current application. It was more intuitive for novice users, but more elaborate to develop. Thus, the team had to re-implement the whole GUI. More importantly the former independent stand-alone application was integrated into another application of the umbrella project. This decision was made jointly by the sub-team, the super-team and the project head. The main reasons for this decision were: the sub-project will extend the functionality of the super-application with some very important features. The sub-project will reach more users as an extension than as a stand-alone application because the super-application is already publicly available and has a steadily growing user base.

As a result of this repositioning from stand-alone application to extension and because it is recommended in Leopold and Kaltenecker (2013), the team performed a stakeholder analysis. The analysis showed that the intensity and quality of some relationships between stakeholders were unknown to the team and that the team had to work on intensifying some relationships, especially the one with the new super-team.

8.4.3. First Cycle

Action Planning and Action Taking

The team already used a Jira Kanban board to visualize the workflow. The board contained the following columns: *backlog*, *in development*, *done*, *done and accepted*.

Team policies were determined in an open discussion. Team members collected all policies on a white board and discussed their meaning and importance jointly. After agreeing on a set of policies, they were transferred

8.4. Action Research Cycles

to the project wiki, so everybody could look them up easily and they were not forgotten after the meeting.

Evaluating and Specified Learning

The main focus in this cycle was *making policies explicit*. *Visualize the workflow* was regarded as finished, because the team already used a board. Additionally, as a consequence of cycle zero the software development was put on hold and the team focused on redefining their goals, hence there was no activity on the board at the time.

Making policies explicit yielded some interesting effects: Team members discussed their unspoken policies for the first time and discrepancies showed. Some were of semantic nature others reflected a different view of processes. During the discussion some problematic habits were identified and immediately discussed. To prevent these habits from reoccurring, new team policies were stipulated jointly and team members agreed to honor those policies.

8.4.4. Second Cycle

Action Planning and Action Taking

Limiting work in process and *managing flow* were introduced to the team during the second Kanban coaching session. Different visualization possibilities were shown and the importance of *limiting WIP* and its effect on transition time, based on Little's Law (Little and Graves, 2008), were explained. It was also discussed how WIP limits could be used to make problems visible and improve flow.

Evaluating and Specified Learning

Limiting WIP put the focus again on the Jira board. This focus uncovered something that should have been uncovered during the first cycle. The cur-

8. Combining FOSS and Kanban: An Action Research

rent board did not model a pull system. It was rather a push system. While team members pulled tasks from the backlog into development, the transition from development to code acceptance was a push process. Developers pushed the task from development to done where senior developers had to take them and move them to *done and accepted*. There was not even a state for being *in acceptance*. It was not possible to determine whether a task was already in the process of being accepted, or if it was still simply marked as *done*. This oversight might be ascribed to the focus on redefining the goal, which was still in progress during the first cycle.

To repair this flaw in the Jira workflow, a new state was introduced into the workflow and the column *in acceptance* was added to the board, now reflecting a real pull system. The columns *in development* and *done* were merged into one column. Initially, the team wanted to create two sub-columns for *in development*, but Jira does not offer this functionality. Therefore different possibilities of visualization were tried out. Figure 8.2 depicts on the one hand the desired visualization and on the other hand the current visualization, which is realizable within the constraints of Jira.

Afterwards, WIP limits were set and monitored. They soon revealed a bottleneck at the acceptance state. The root cause was quickly identified and team members were working hard to resolve this bottleneck. Due to the merge with the super-project the team now depended on the senior developers of the super-project to accept the sub-team's code. The senior members of the sub-team were and still are working hard to familiarize themselves with the slightly different acceptance process of the super-team and thus dissolving the bottleneck as soon as possible and improving flow. Another positive effect of the focus on the board is, the team now uses their Kanban board during each meeting, which it did not do prior to the AR cycles.

8.4.5. Third Cycle

Every team member watched the earlier mentioned Kanban video before starting this cycle. *Implement feedback loops* was introduced to the team during this AR cycle. Different forms of feedback and how the team's work

8.4. Action Research Cycles

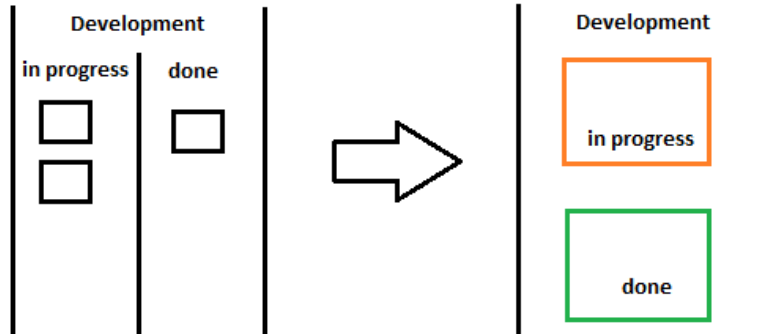


Figure 8.2.: The desired visualization on the left side and the actual visualization, which is possible in Jira, on the right side. The different states are highlighted in different colors.

processes could be improved were discussed. To receive feedback about motivation within the project a questionnaire, see Appendix B, was issued to team members.

Action Planning and Action Taking

For the first feedback cycle focus was put on two aspects. First, information about a new Jira workflow (created during (Harzl, 2016)) was fed back to the super-team and the UX team. Previously the board was partly a push system, this new workflow implemented a pull system. To inform the other teams, an informal meeting with the coordinators of both teams was held and the changes and their rationales were explained. The coordinators were also asked for feedback about the new Jira workflow.

Second, the team wanted to eliminate superfluous activities. Were there any activities in their work process, which did not deliver value to the team, their stakeholders or users? To explore this topic, the following aspects were explored with the team: is there anything in their work process which

8. Combining FOSS and Kanban: An Action Research

they deem unnecessary or hindering. What could be improved in the work process in the next iteration? All topics were collected on a whiteboard and after that every team member voted for his or her most important one. The topic with the most votes was *estimating Jira issues*, therefore, it was selected. First, it was analyzed why estimation was identified as an issue, second, alternatives were determined and third, the initial approach was compared with a possible new method.

Evaluating and Specified Learning

Inter-team Feedback

Both coordinators, from the super-team and the UX team, reacted positively to the new workflow. The super-team coordinator even showed some interest in adopting it for his team. The new workflow also seemed to trigger a thought and discussion process for the coordinators, as both approached the researcher about additional changes to the workflow. The UX coordinator had suggestions to better integrate the UX team into the workflow and the super-team coordinator suggested new workflow states for the code review part. These conversations determined the theme for the fourth cycle, *visualize the workflow*.

Estimating Jira Issues

In ASD effort of issues is usually estimated by the team to acquire some knowledge about the size and complexity of an issue. These estimates are then often used to determine a team's velocity, to predict if a goal can be achieved within the next iteration and to trade work items of similar size in and out of an iteration, if a time-boxed ASD approach is used. Different estimating techniques exist, e.g., story point estimation, Ideal Days (ID) estimation, T-shirt or even dog size estimation.

Kanban teams often do not estimate issues, because estimates do not deliver customer value and therefore do not fulfill a purpose. However, in this project story point estimation is used.

8.4. Action Research Cycles

“A story-point estimate is an amalgamation of the amount of effort involved in developing the feature, the complexity of developing it, the risk inherent in it, and so on.” (Cohn, 2005).

Typical scales are derived from the Fibonacci sequence and contain values between one and ten, 21 or 100.

Advantages of story point estimation are:

- It is more accurate than other estimation methods.
- It is independent of time units.
- One can calculate team velocity directly (without needing to convert sizes to numerical values).
- A more fine-grained scale offers more detailed information.

Disadvantages of story point estimation are:

- Many articles about agile estimating specifically mention that joint experience as a team is a key factor to accurately determine story point estimates.
- New teams often struggle with estimating stories effectively, and it takes some iterations until a team’s velocity becomes stable.
- In the beginning estimates vary too much and velocity is therefore unstable as well.
- If you cannot determine velocity, you cannot derive duration.

The story point scale used in this hybrid student FOSS project contains the values 1, 2, 5, 10, 20, 50, 100, 200, and 500. The story points are not used to determine velocity, to determine the amount of issues for one iteration, or anything of this nature. In fact, right now it is not possible to calculate an average team velocity because iterations are very different in the amount of features implemented and time duration. Hence estimates do not fulfill one of the usual purposes. Team members of the studied team only use estimates to determine, if an issue is small enough to fit into their schedule.

The major problems for the team with this kind of estimating were:

- The concept of story points is unknown to most contributors and this will probably not change, due to transient team members. Estimating by story points has to be explained before every planning meeting,

8. Combining FOSS and Kanban: An Action Research

which consumes quite some time and still this concept stays too abstract for some.

- The scale is too far reaching, making it very difficult to compare the efforts, i.e., what does it mean, if an issue is 200 or 500 times the effort of another issue? Humans have difficulties comparing values over more than one order of magnitude (Miranda, 2001; Saaty, 1996).
- Issues and their effort were only compared within one iteration. The values were assigned more or less arbitrary and there was no consistency between planning meetings. This reduced the informative value of the effort estimation to a minimum and team members were unable to use it to select an issue which fitted into their schedule, because the meaning of the different values changed distinctively between two iterations.
- Thus, story points delivered no value to the team or anyone else. Effort estimation just “had to be done”.
- Effort estimation did not provide the information team members wanted. For them the most important information concerning effort is connected to the time necessary to resolve an issue. However, relative sizing with story points is not intended to convey information about time. The duration of an iteration is only derived from the total amount of story points divided by the team’s velocity.

The team could not determine velocity and therefore could not derive duration and, thus, not determine how long an issue would take. As a result team members saw no meaning in estimating issues and regarded it as unnecessary. To give estimating a meaning for the team alternatives were investigated. Two of them will be shortly explained here, because they motivated the new estimation scale.

First T-shirt sizing, issues are assigned a T-shirt size between, e.g., XS and XXL. Other methods relate size to cars or dogs, but they share the same principle, removing the implied precision of numbers. These sizes (T-shirts, cars, dogs) can be related to story point values, e.g., M = 5, L = 10, which can be used in metrics.

Advantages of T-shirt sizing:

- It can expedite the voting processes, because it provides fewer options.

8.4. Action Research Cycles

- It does not suggest precision because of a non-numerical score.
- It allows to think in a more abstract way about effort. It even allows for creativity and fun during estimation, if more unconventional sizes are used.
- It can also be beneficial to start with a simpler approach than story points for new teams, e.g., T-shirt sizes, dog sizes and to slowly move towards a numerical scale, when a team has some experience with estimating.

Disadvantages of T-shirt sizing:

- The accuracy of velocity estimates might be reduced, because the estimation scale is less detailed.
- There is no clear mathematical correlation between the different sizes.
- If you want to track effort and velocity over time, you need to convert the sizes to numerical values.

Another method is estimating in units of time. If you approximate time, you can either estimate in elapsed days or in ID. Elapsed days contain all interruptions, which might occur, while working on an issue, whereas ID contain only the amount of time an issue will take, without any interruptions, organizational overhead etc. Since one can never anticipate all possible interruptions estimating in ID is easier than in elapsed days. Because ID only consider the time necessary to finish an issue, they are also an estimate of size, but less strictly than story points (Cohn, 2005).

The team liked the idea of estimating in time, because this information is important to them. But they did not want to estimate an exact number of ID, because this would suggest that their estimates are more precise than they actually are. They preferred a less detailed classification and wanted the estimate to convey some information about time duration. Thus, the decision was made to combine both methods and create a proposal for a future scale containing three values, S, M, and L. Similar to story points, for every team these sizes can mean something different, e.g., S = one to three ID, or S = up to one ID. This could lead to misunderstandings between teams, when they collaborate on larger features, but this is true for all team-specific estimation techniques and because there are only three sizes, differences between teams should be negligible.

8. Combining FOSS and Kanban: An Action Research

In the context of this team ID usually are not successional, e.g., if an issue is estimated to take two ID, they can in fact stretch over a few weeks, as people usually do not work full time on the project. One ID can contain several smaller units, which take place on different week days in different weeks, whenever people have time to work on the project.

The team decided to use the following units in the beginning and to adapt them in the future, if necessary: S = up to one ID, M = two to three ID, and L = four to five ID. Larger issues would be divided into smaller issues before working on them. The team decided that this scale should be detailed enough for their purposes. A more in depth classification would not be needed and would have unnecessarily prolonged time expenditure for estimating issues. Although, the direct correlation of issue size to time is not ideal, it is a starting point and provides some information about time duration, which is currently the only information used by team members. This proposal was compared with the story points method. Table 8.1 shows the results of this comparison.

The story points approach was used for several iterations in the past. For future iterations the team switched their estimating scale to T-shirt sizes, because they deemed this approach more appropriate for them, because it reduces the time needed for voting due to the reduced number of options. After one iteration with estimating in T-shirt sizes, including planning meeting, estimating issues and code development, the team filled in a questionnaire, see Appendix B, about their experiences in this iteration with T-shirt sizes, and their experiences with story points in the iterations before the last. The questionnaire was answered by seven people. It was statistically analyzed. In one question contributors were asked to rate how many of their issues they estimated on a scale of 0% to 100% (adapted from Shodan Input Metric Survey (Williams, Krebs, and Layman, 2004) with:

- 0% being never,
- 10% hardly ever,
- 20% rarely,
- 30% sometimes,
- 40% common,
- 50% half & half,
- 60% usually,

8.4. Action Research Cycles

Table 8.1.: Characteristics of story points and T-shirt sizes as estimating units as seen by team members.

Criteria	Story points	T-shirt sizes
Simplicity/Complexity	Estimation depends on the reference issue(s), which has to stay the same over time. Otherwise estimations are not meaningful.	Simpler, intuitive, not interdependent, relates to working time, easier. Everybody can do it.
Level of detail	More exact. If it is not correctly done, it is less meaningful. It is more complicated to assign issues a correct value.	It is less detailed. Estimation and reality are closer to each other.
Usefulness	Useful if it is done correctly. More experience needed to estimate correctly. If there are too many different units, estimating becomes impossible. Usefulness depends on project and team size. Our estimation procedure delivered useless estimates.	Always useful. A rough estimation is always helpful. It is sufficient if a more detailed planning is not necessary.
Informative value	It is more exact but also more error-prone. One needs to know the estimation system. A finish date can be determined more exactly.	Less granular.
Makes use of learning effects	Estimates become more accurate with experience.	No.
Time expenditure	It is much more time-consuming, especially in the beginning.	It is faster.

8. Combining FOSS and Kanban: An Action Research

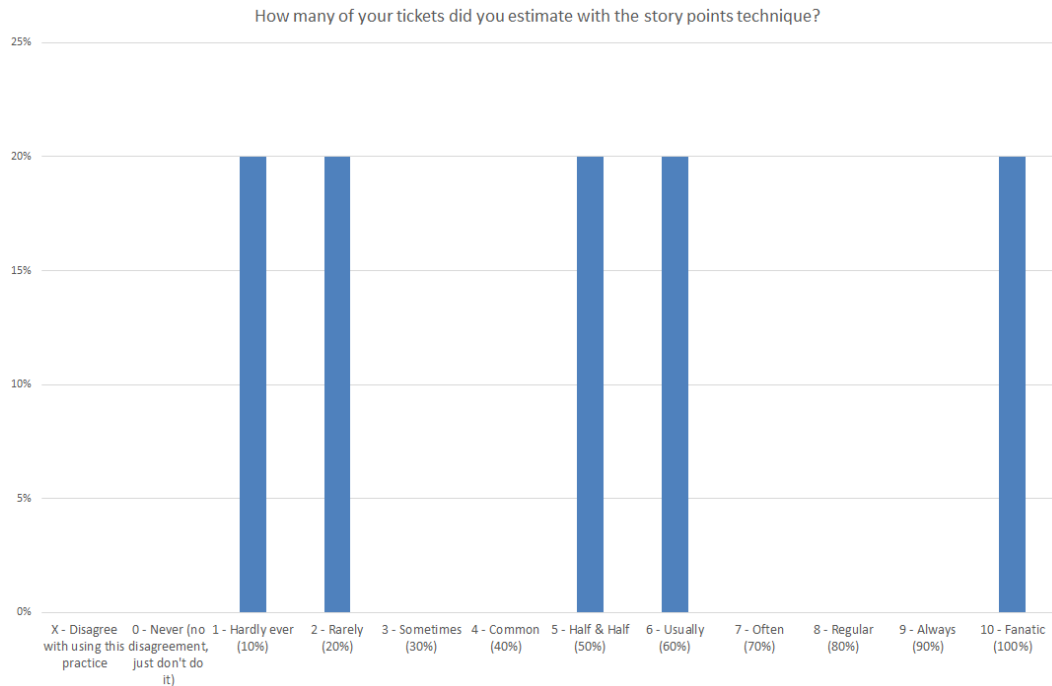


Figure 8.3.: How many of your tickets did you estimate with story points technique?

- 70% often,
- 80% regular,
- 90% always, and
- 100% fanatic

Another question asked how many of their tickets they would like to have estimated (desired value) with the same scale.

For the story points method the actual mean value was 48% (Standard Deviation (SD)=36%), see Figure 8.3, two people did not answer this question. For the desired value the mean value was 64% (SD=32%), see Figure 8.4.

For the T-shirt size method the actual average value was 92% (SD=4%) and the desired value was 93% (SD=5%) for all respondents. Because these questions were answered by all participants, the results are presented together in Figure 8.5.

8.4. Action Research Cycles

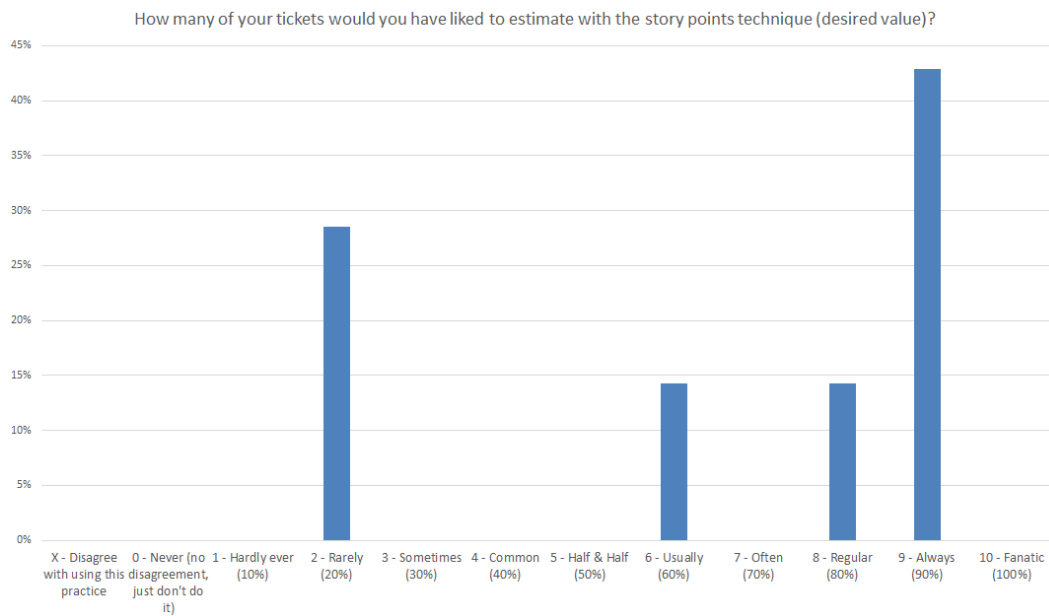


Figure 8.4.: How many of your tickets would you have liked to estimate with story points technique (desired value)?

8. Combining FOSS and Kanban: An Action Research

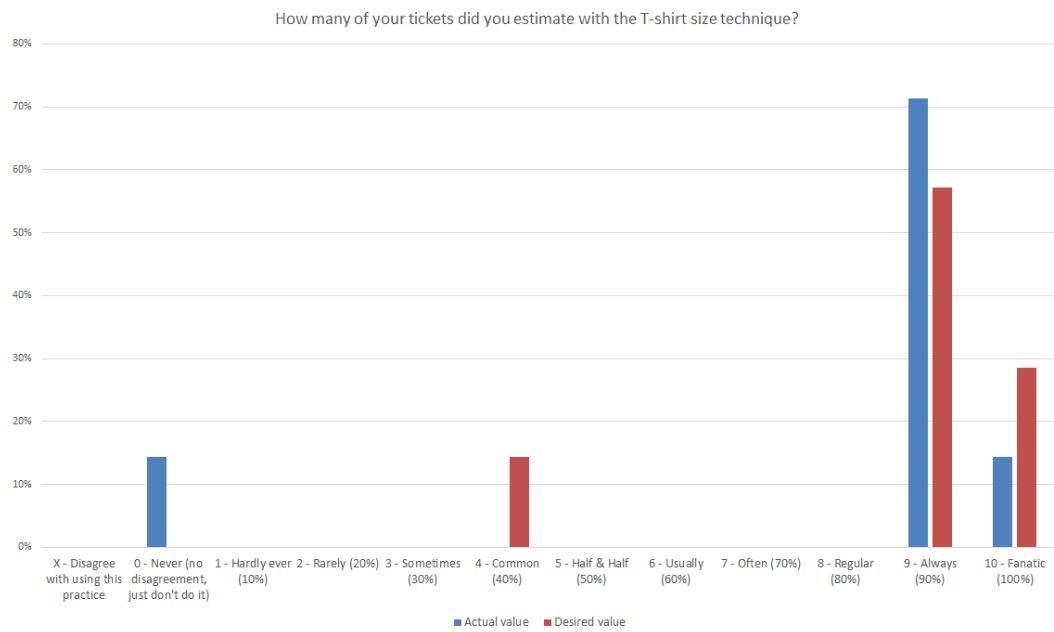


Figure 8.5.: How many of your tickets did you estimate (actual value) and how many would you have liked to estimate (desired value) with the T-shirt size technique?

8.4. Action Research Cycles

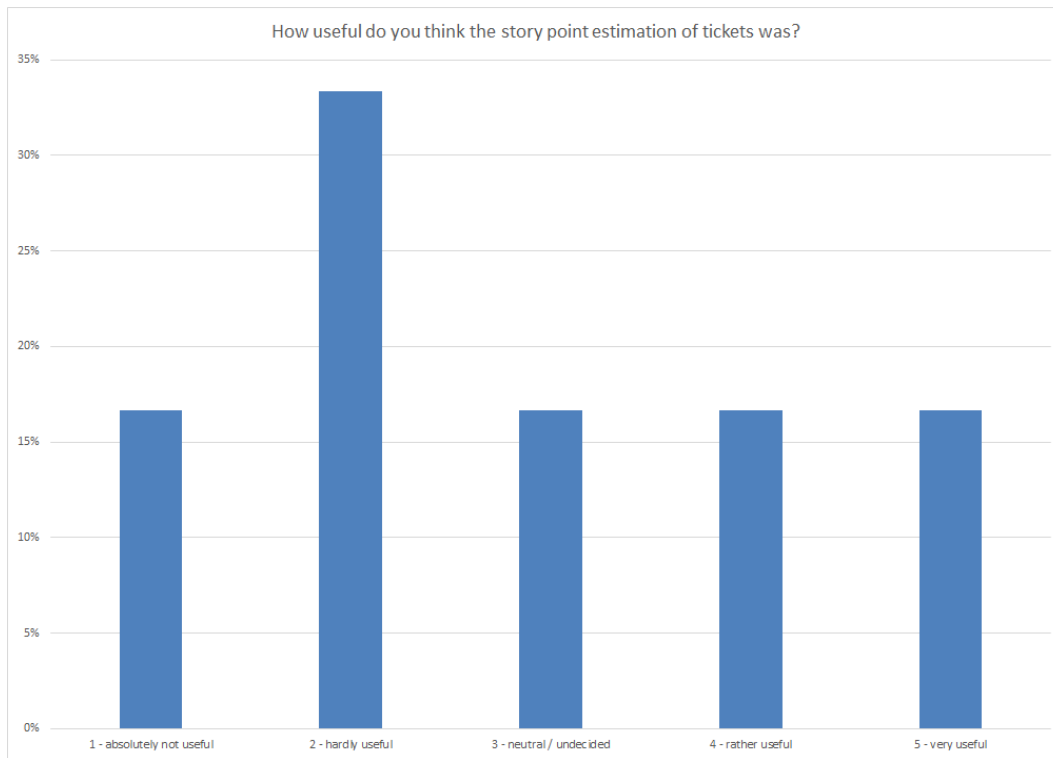


Figure 8.6.: How useful do you think the story point estimation of tickets was?

Regarding the usefulness of both methods team members had to select between:

- *absolutely not useful* = 1
- *hardly useful* = 2
- *neutral / undecided* = 3
- *rather useful* = 4 and
- *very useful*=5

Ratings for the story points method were: *hardly useful* and *absolutely not useful* were selected by 50%, 33% selected *rather useful* and *very useful*, and 17% were *neutral / undecided* (Mean Value (MV)=2.8, SD=1.5), one person did not answer this question. See Figure 8.6.

8. Combining FOSS and Kanban: An Action Research

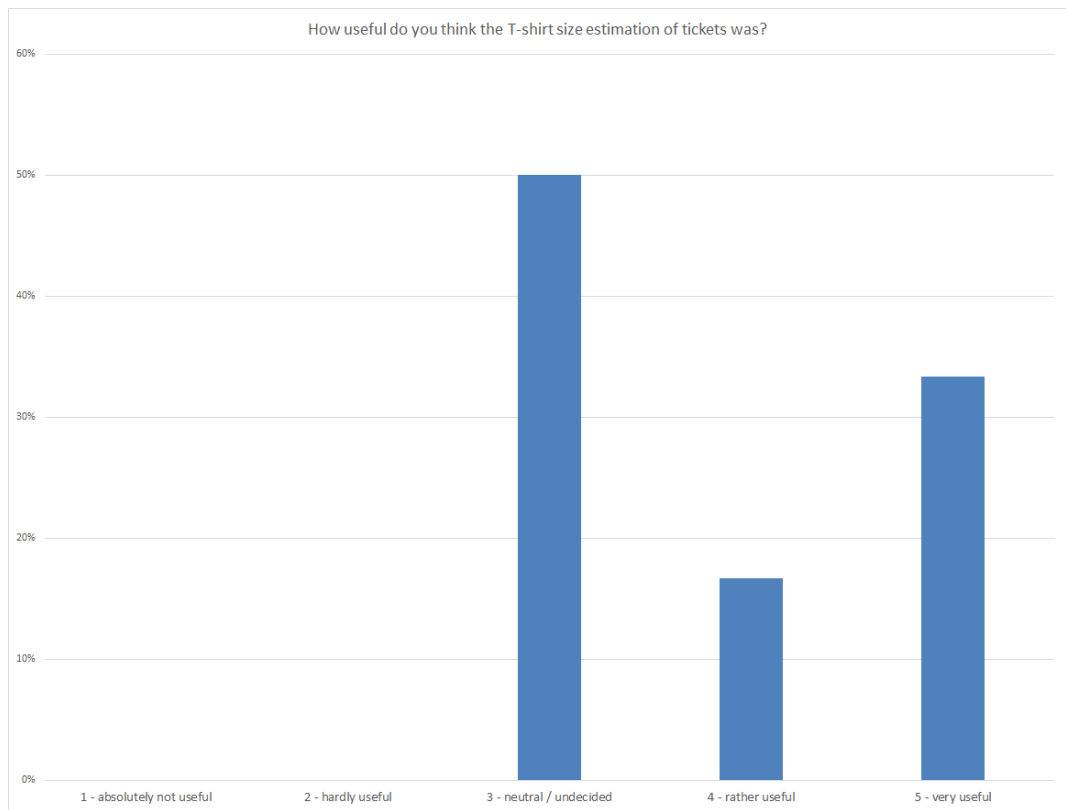


Figure 8.7.: How useful do you think the T-shirt size estimation of tickets was?

Ratings for the T-shirt sizes method (one person did not answer this question) were: 50% were *neutral / undecided* and 50% selected *rather useful* and *very useful* ($MV=3.8$, $SD=1$). Thus, the use of estimation has improved and team members regard estimation as more useful than before.

This is also supported by the analysis of the Jira issues. With story point estimation 63% issues were not estimated or had the default value. "Not estimated" was added as default option only shortly before switching from story point estimation. With T-shirt size estimation only 3% of issues are not estimated. The accuracy of the estimations cannot be determined, because team members do not log their working hours for issues.

Despite the advantages of story points and the shortcomings of T-shirt

8.4. Action Research Cycles

sizes, for this hybrid student FOSS team a time related estimation seems to be more desirable. And because story points and velocity are not used in the value chain the team was able to switch to a simpler approach. It is more efficient for them, because the scale is less detailed, more intuitive to them, because they are more used to estimating in units of time, and it provides all the information the team requires, i.e., if an issue fits into their schedule. Hence, in the context of FOSS projects using simpler methods can sometimes be more useful, than intending to use more elaborate methods and have people not use them, because they are perceived as too complicated, too time-consuming, and as to not deliver value.

An additional result of this questionnaire was, that planning meetings were conducted too rarely. Some new team members had not taken part in one up to this point. This finding called for a closer investigation of that topic. Analysis of previous planning meetings showed that they were conducted only twice a year and lasted three hours on average. The long duration made it difficult in the past to set a date, where everybody could participate. Thus, the planning was sometimes conducted in the late evening, after students' working hours in their day jobs and various university courses, making it difficult to concentrate the whole time and making the whole planning meeting strenuous. Therefore, team members were not looking forward to planning meetings. A new approach was proposed to the team. They could consider their average number of finished issues per month and plan iterations accordingly, which in their case would mean to plan fewer issues. It was also proposed to plan smaller iterations, which would yield two positive effects. They would become more experienced regarding planning and the meeting would be shorter, making it easier to fit it in everybody's schedule. The team tried it and the following meeting took only one and a half hours, all necessary information for the next iteration was gathered and afterwards one member said "This was fun, can we do it more often."

Motivation Feedback

The questionnaire about the motivation within the project, see Appendix B, was answered by seven people and shows a diverse picture of motivation for team members. It ranges from up and downs, an increase in motivation, a decrease, and no change in motivation. Some are more motivated in

8. Combining FOSS and Kanban: An Action Research

the beginning, some rather at the end, when they recognize all the possibilities the project offers. Drops in motivation were experienced in phases, when the project stagnated or major changes took place. Work and exam times also have an impact on contributors motivation. In general if the motivation improves, respondents report having more fun, less stress, a clearer view of the goals, they are self-reportedly more productive and creative. Reasons for changes in motivation ranged from better team spirit, general mood of the team, coaching, status of the project, work, exams, to frustration at work. When asked about the specific impact of the Kanban coaching all respondents answered that Kanban coaching and all its results improved their motivation. They have rather more (70%) or much more (30%) motivation than before. Positive comments include: "I knew what has to be done and where problems might come from.", "Transparency and overview with the Kanban board improved the basic motivation.", "I always like structure in my work, therefore I liked this a lot.", "Defining the roles of project members within the team was very insightful for me and for sure improved my motivation. Practices like WIP limits and a meaningful segmentation of the board were also good.", "In my opinion the team grew together, the goal became clear. Work became fun and had a 'meaning'." Other comments included: "For the moment, Kanban coaching did not lead to a change in my motivation", "Kanban coaching did not influence my motivation, because I was at the end of my contributing time. I think the concept is very interesting, but I did not see it in real action." The following reasons for the previous answers were given: "flow was visualized, team work improved", "fits better for the team, especially the change of visualizing techniques", "Plus: transparency, division of work, overview, workflow", "the initial effort is higher, but one can see what is done or has to be done", "it became more fun and motivating". When asked how much they benefited from the newly acquired knowledge all respondents answered that they rather profited (57%) or profited very much (43%). As reasons were given: "positive results in the project, used Kanban knowledge also for work, change of mindset, trying to increase efficiency in general, can be applied to other group work or projects, practices have a positive impact on motivation, no possibility to apply it actively elsewhere, left the project shortly after". A positive influence of the Kanban coaching can be seen in Figure 8.23 as well. It shows the morale of the study participants throughout the AR. In the beginning, during the diagnosing phase,

they enjoyed their work around 60% of the time, at the end of the study this value increased to almost 80%.

8.4.6. Fourth Cycle

Based on the feedback from the super-team and UX team received during the third cycle the Kanban practices *visualize the workflow* and *make policies explicit* were also the theme of the fourth cycle.

Action Planning and Action Taking

During a regular meeting previously created policies were reviewed and discussed by the team, if they were still valid, if some became obsolete or if they should add new ones.

To revise the Jira workflow all coordinators from all teams within the umbrella project were invited. The coordinator of the super-team and the UX team because they are directly connected to the team and the others because they could deliver different points of view on the matter and at the same time were informed about possible workflow changes. A meeting was held and whiteboards were used to draft the new workflow. All possible states within a new workflow were collected and discussed. Afterwards, they were put into an order and it was discussed which transitions were needed, who should be allowed to use the transitions, e.g., code review transitions should only be done by experienced team members. Finally, the whole workflow was sketched on a whiteboard and everyone was asked if they could think of any more improvements or if they approved of it this way, until consensus was reached. The whiteboard drawing can be seen in Figure 8.8. This workflow was then implemented in Jira.

Evaluating and Specified Learning

Only a few policies had to be changed, but reviewing them refreshed everybody's memory. The team realized that they should reconsider them on

8. Combining FOSS and Kanban: An Action Research

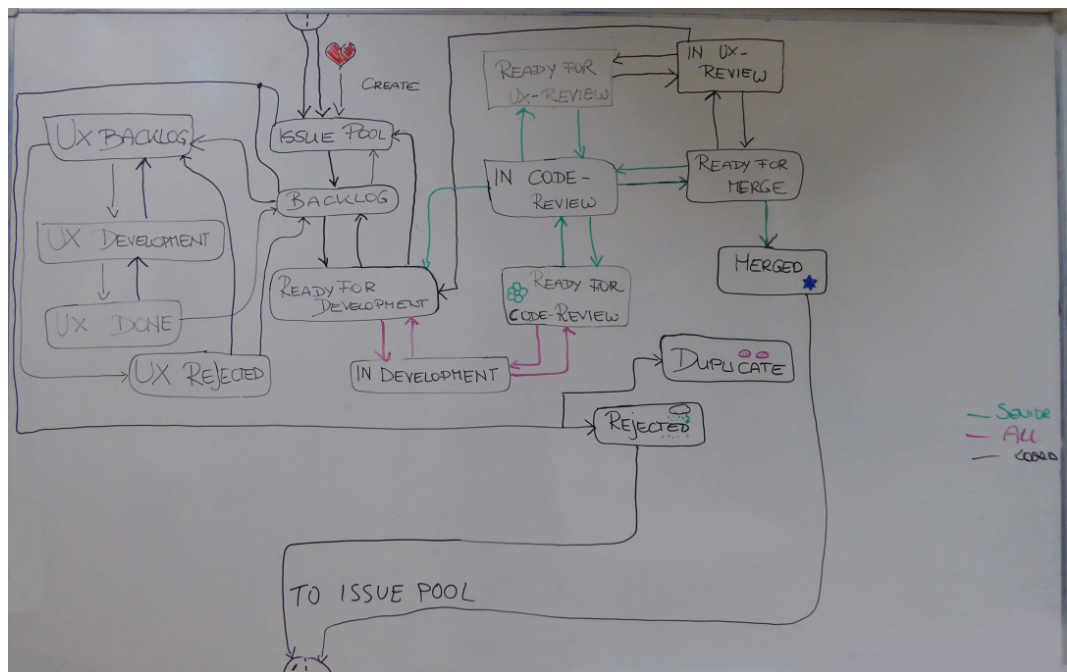


Figure 8.8.: The new workflow as sketched on the whiteboard.

a regular basis to keep the policies alive and present.

The new workflow (see Figure 8.9) is more elaborate than the previous one and allows for a more thorough division of labor and a more detailed view of the current work items. The Issues Pool is a mixture of user feature requests and bug reports, issues created by the project head and the team. Between the Issues Pool and Ready for Development issues are sorted through, e.g. for duplicates, and requirements are defined. Code reviews are split into two parts, one being the actual review and the other one being the actual merge into the master branch. Previously code review and merging code were combined in one state. Because the project requires the code to be tested, before it is merged (ideally a test-driven approach is used), this also means that code *and* tests have to be reviewed. As it turned out contributors sometimes are afraid of the responsibility which comes with merging new code into the master branch, because they are not very experienced in using Git, and because of that, were reluctant to conduct code reviews at all. Now they can start with reviewing code and slowly step up to also merging code. This separation of review and merge also provides the opportunity to include UX reviews of new features neatly into the workflow after the code has been approved technically and before the code is merged into the master branch. This was an important request from the UX team. Before this change, developers often forgot to ask them for feedback before merging code, because the UX review was not represented in the workflow. As a result sometimes new features were merged, that did not meet UX specifications and had to be repaired. Now if an issue received UX input in the beginning, before developing it, it has to undergo UX review as well before it can be merged. Transitions between UX specific states can only be used by UX members. Some transitions are restricted to seniors members, because the actions connected to those transitions require a certain amount of knowledge about the project. Actions necessary for a release are only executed by a handful of persons. These steps are not represented in the Jira workflow.

The new workflow was initially intended for the AR team and only meant to be distributed to other teams if it proved beneficial, but many coordinators wanted to test it with their teams as well, so the new Jira workflow was adopted throughout the whole umbrella project. Some teams are already considering introducing WIP limits and one team even started its

8. Combining FOSS and Kanban: An Action Research

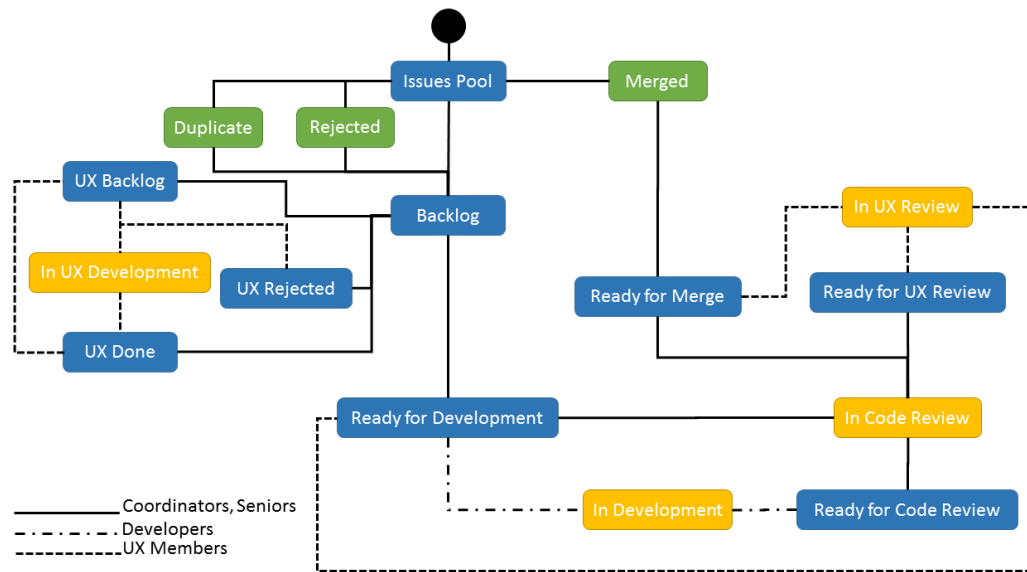


Figure 8.9.: The new workflow for the whole umbrella project in Jira.

own improvement initiative. It is about improving code reviews, not introducing Kanban, but it is a first step into actively improving their work process. Thus, it can be observed that the effects of the Kanban initiative have spread beyond the original team and triggered changes in other teams as well.

8.4.7. Fifth Cycle

The fifth cycle was again about *implement feedback loops* but this time it was similar to a retrospective. What did the team achieve so far? Do they recognize improvements or not? Has something changed for the worse? What could be the next steps?

8.4. Action Research Cycles

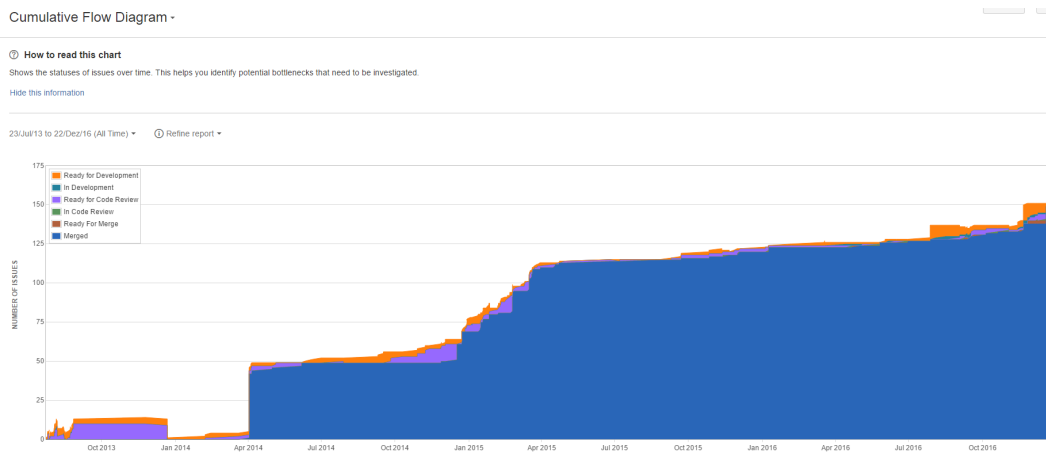


Figure 8.10.: The team's Cumulative Flow Diagram (CFD).

Action Planning and Action Taking

The team's CFD, a questionnaire and a feedback meeting were used to answer these questions. The questionnaire included the topics communication, feedback about Kanban, and the importance of a designated coach role. During the feedback meeting the team should collect issues which they would like to improve in their work process. Arising topics were discussed. Team members gave examples and reasons why things should change. After the major topics were collected, the team decided in consensus on concerns to work on.

Evaluating and Specified Learning

Jira Board and CFD The analysis of the Jira board and CFD (see Figure 8.10) showed, that the team was not developing more issues than before, only the distribution was slightly more even. Before the Kanban coaching the team sometimes binge-worked or updated their Jira board irregularly.

Questionnaires

8. Combining FOSS and Kanban: An Action Research

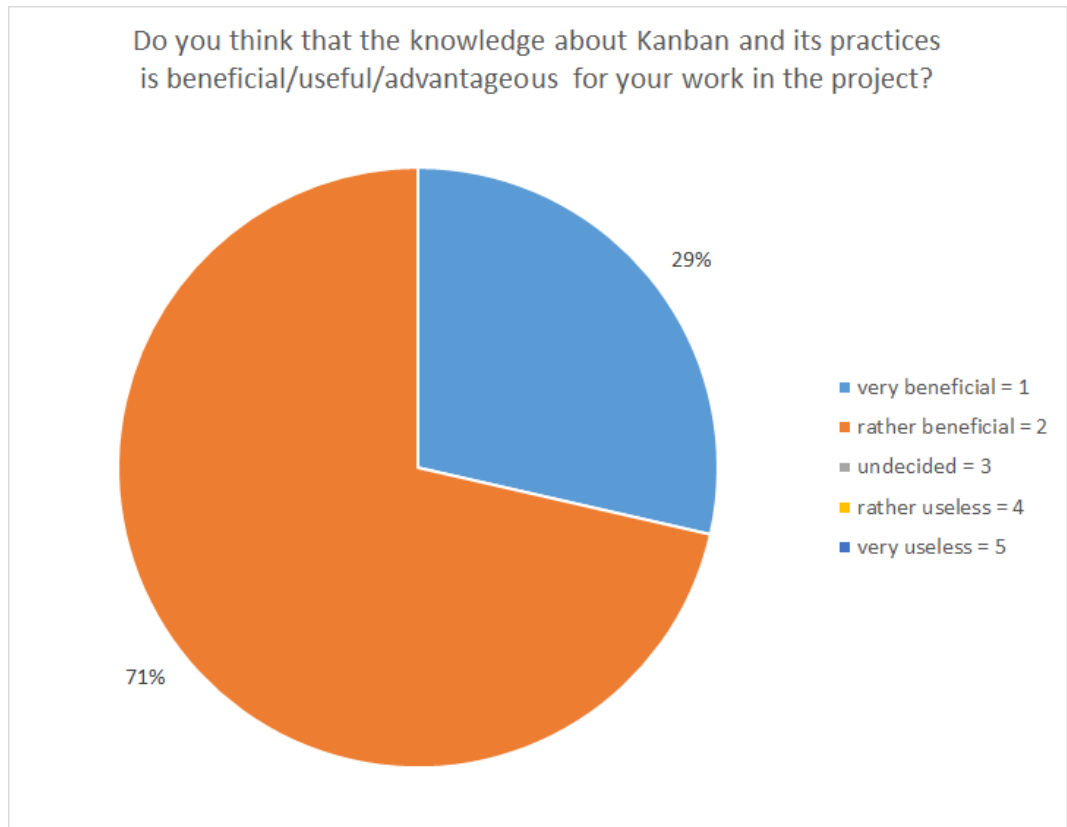


Figure 8.11.: Answers to the question: Do you think that the knowledge about Kanban and its practices is beneficial/useful/advantageous for your work in the project?

8.4. Action Research Cycles

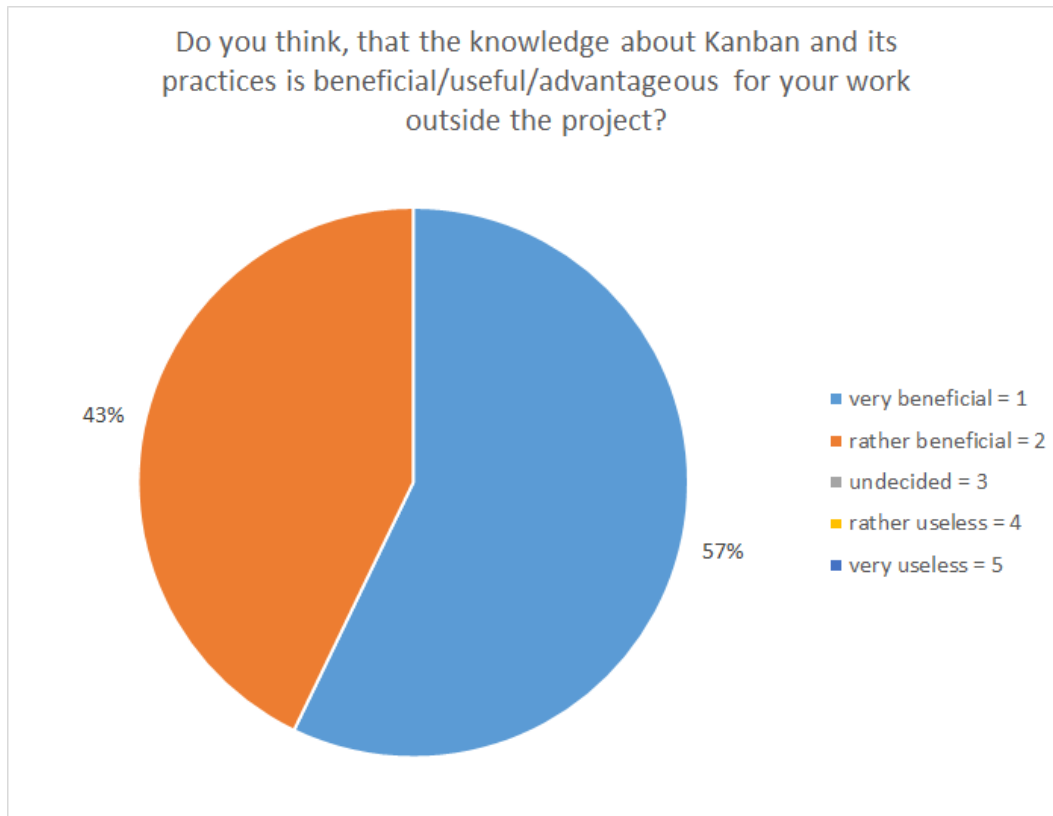


Figure 8.12.: Answers to the question: Do you think, that the knowledge about Kanban and its practices is beneficial/useful/advantageous for your work outside the project?

8. Combining FOSS and Kanban: An Action Research

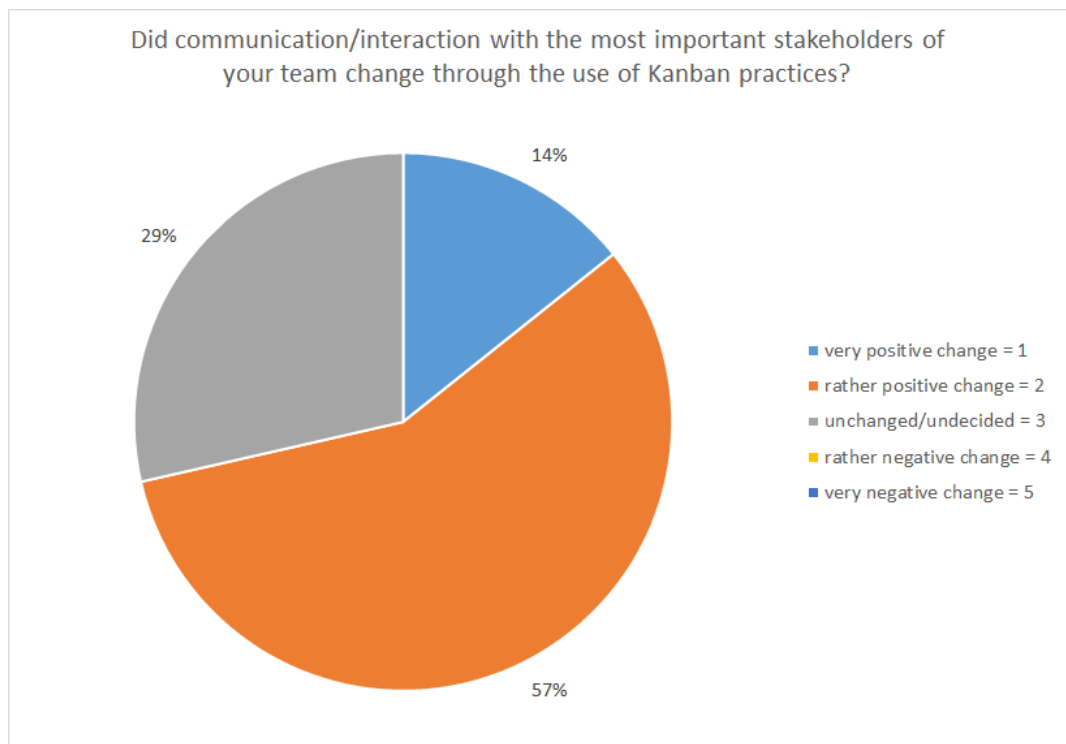


Figure 8.13.: Answers to the question: Did communication/interaction with the most important stakeholders of your team change through the use of Kanban practice?

8.4. Action Research Cycles

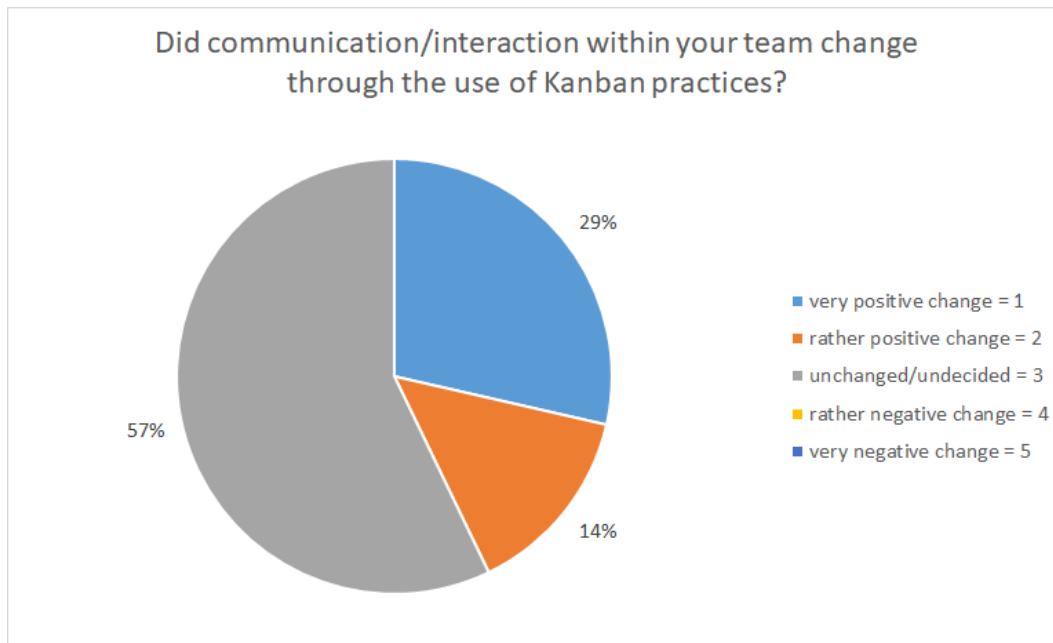


Figure 8.14.: Answers to the question: Did communication/interaction within your team change through the use of Kanban practices?

8. Combining FOSS and Kanban: An Action Research

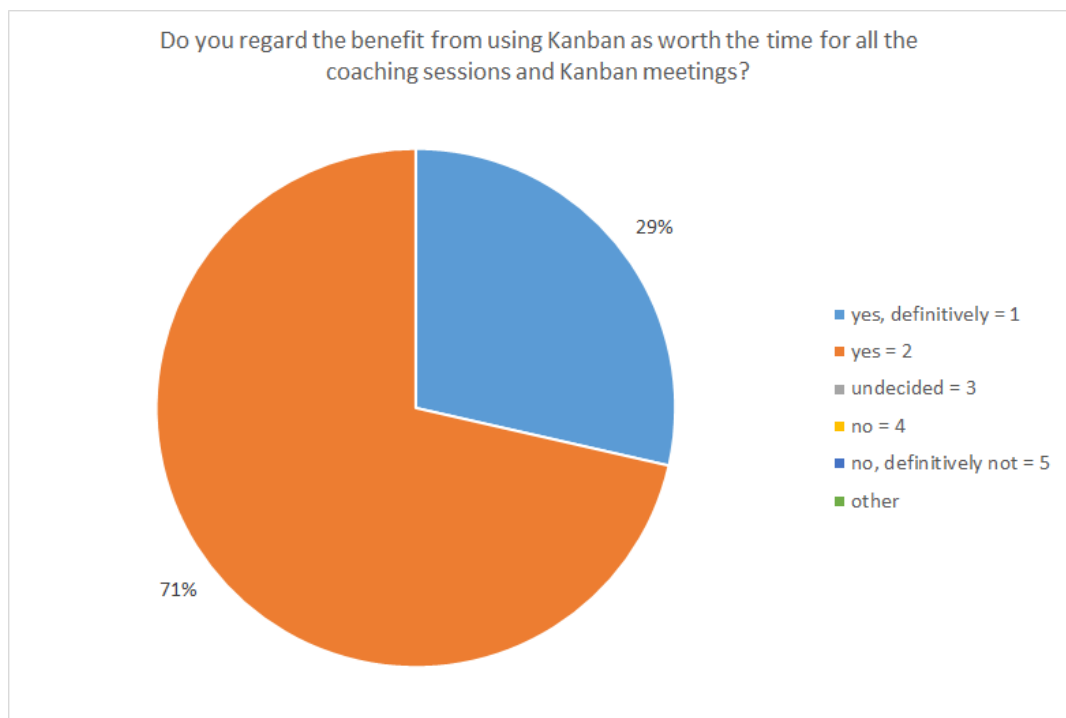


Figure 8.15.: Answers to the question: Do you regard the benefit from using Kanban as worth the time for all the coaching sessions and Kanban meetings?

In the questionnaire about Kanban (see Appendix B) AR participants had to decide if they experience their Kanban knowledge as:

- *very beneficial* = 1
- *rather beneficial* = 2
- *undecided* = 3
- *rather useless* = 4
- *very useless* = 5

to their work within the project and outside the project. The same choices were given to rate the usefulness of a coach. 29% rated Kanban knowledge as *very beneficial* for their work within the project and 71% rated it *beneficial* (MV=1.7, SD=0.48). 57% rated it as *very beneficial* and 43% as *beneficial* for their work outside the project (MV=1.4, SD=0.53). See Figures 8.11, 8.12. Interestingly, more people rated Kanban knowledge as *very beneficial* for their work outside the project (57%) than within the project (29%). The following answers may explain this discrepancy. 28% created a personal Kanban board (Benson and DeMaria, 2011), which contains tasks concerning their whole life and not only the project, which makes Kanban outside the project more important for them than within the project. Another reason was that sometimes the time spent on Kanban practices reduced the available time for writing code. Team members often work in companies while studying and therefore have only a very limited amount of hours to work on the project and sometimes spent more time on Kanban practices than coding. One team member responded: “I think Kanban is generally very beneficial to the team. However, I think the programming vs. Kanban work ratio is not perfect right now”. Additionally, he talked to his boss at his workplace to introduce Kanban to the company. He spends more time working at the company than in the FOSS project, so he rated the usefulness of Kanban knowledge outside the project higher than within the project.

The presence of a Kanban coach was rated *very beneficial* by all team members. On the one hand team members appreciate the role of a coach and on the other hand are aware of the risks a (missing) coach entails: “The board is now up-to-date but in my opinion the reason is more the presence of the coach than Kanban”, “Without a coach this could end in chaos”, “Without a coach there is a greater risk of leaving important things out or to ignore

8. Combining FOSS and Kanban: An Action Research

them because it's easier", "You can become dependent on the coach, so you don't achieve anything without him or the coach could take on a leading role instead of a coaching role or he might be seen that way by some team members", "The team would have never dealt this intensely with the topic without a coach", "Feedback from outside the team is great, because inside the team you often develop tunnel vision", "The coach points out possibilities for improvement we may not have discovered otherwise", "Whenever you try to change something, you are easily tempted to fall back into old habits. In these moments someone who puts you back on the right path is very precious".

In general, team members seem to experience the use of the Kanban method as beneficial to their work inside and even outside the project, which answers RQ 2.1 "Do FOSS contributors, who are coached in the Kanban Method, experience this knowledge as beneficial to their work or not?".

Regarding the communication, AR participants had to rate their communication with stakeholders and within the team on the following scale:

- *very positive change* = 1
- *rather positive change* = 2
- *unchanged/undecided* = 3
- *rather negative change* = 4
- *very negative change* = 5

The communication to stakeholders outside the team changed *positively* or *very positively* according to 71%, and 29% said it *did not change* or they are *undecided* (MV=2.1, SD=0.69). As most important changes, communication with the super-team, the project head, the UX team and users were mentioned. Team members are aware that there is still plenty of room for improvement, but thanks to the stakeholder analysis in cycle zero they now know who they need to talk to and which relations they need to cultivate more. Within the team itself changes were not that distinct. 43% said it changed *positively* or *very positively* and 57% said it *did not change* or they were *undecided* (MV=2.3, SD=0.95). Text comments showed that team members consider the communication within the team generally as very good, so it is not so surprising, that changes were not as evident. Some team members noticed that meetings were held more regularly and attendance improved, although they could not say if it was due to Kanban

8.4. Action Research Cycles

or the coach's presence. In general, team members recognized a positive change regarding communication and interaction within the team and to other teams. This is a strong indication that the communications behavior changed and RQ 2.2 Do interaction or communication during meetings change with the use of the Kanban Method? can be confirmed. Concerning the interaction during meetings, the most visible change is, that the team now almost always uses their Jira board, when discussing issues. Previously they only seldom looked at it.

Regarding RQ 2.3 Do FOSS contributors regard their time acquiring Kanban knowledge well spent? Team members had to answer if they regard the benefit from using Kanban, as worth the time for all the coaching sessions and Kanban meetings. Possible answers were:

- *yes, definitively* = 1
- *yes* = 2
- *undecided* = 3
- *no* = 4
- *no, definitively not* = 5
- *other*

29% answered *yes, definitively* and 71% answered *yes* (MV=1.7, SD=0.48), so this team regards their time well spent learning about Kanban.

Feedback Meeting

During this meeting several concerns arose, e.g., scope of issues is too extensive, the most important ones which were selected to be worked on will be discussed here. A major issue was that team members wanted to be more proactive when dealing with problems and bottlenecks. Although they identified a major bottleneck in their workflow, they ignored it for a while and waited for someone else to resolve it for them. This was mentioned by team members as both an advantage and disadvantage of Kanban. On the pro side problems and bottlenecks become visible quite early, but on the con side, if nobody feels responsible and the bottleneck is not resolved, the WIP limit stops the whole development process. We talked about this and it was explained to them, that this is one purpose of WIP limits, to make it impossible to ignore problems and bottlenecks so they have to be resolved and not linger on for all eternity. If they still choose

8. Combining FOSS and Kanban: An Action Research

to ignore it, they will soon see the consequences. They understood this explanation and consequently they made a time-phased plan how to resolve this bottleneck and immediately took the first steps. Another effect of this situation was, that they want to give the work on the project more priority, so this “Waiting for Godot” will not happen again. As this is a more vague resolution this strategy should be examined in the future.

8.5. Analysis of Usage of Agile Practices

Over the course of the AR participants filled in the Usage of Agile Practices (UAP) questionnaire. It is based on the Shodan 2.0 Input Metric Survey from Williams, Krebs, and Layman (2004). Study participants filled in the UAP questionnaire several times over the course of the study. First in the diagnosing phase, second after Cycle Zero, third after the second cycle, fourth after the fifth cycle and fifth around three month after the fifth cycle. The first UAP questionnaire was answered by six people, the second by only three people, therefore, it is not included in the analysis, the third by eight, the fourth by seven, and the fifth by six people. The number of respondents is included in square brackets for each UAP in all the figures. For every XP and Kanban practice there was an initial question how much the person knows about the practice. If the answer was “Nothing” subsequent questions about the practice were not shown to the user. If more than half of the respondents did not know the practice the usage was not evaluated. As a consequence some practices are only evaluated for a part of the questionnaires.

According to the questionnaires usage of the following practices shows a change (more than 10 points):

- Customer acceptance tests, see Figure 8.16: Possible reasons for the change could be: Increased user focus due to the user analysis in cycle zero. Introduction of acceptance criteria for tickets as new team policy.
- TDD, see Figure 8.17: Possible reasons for the change could be: Introduction of a team policy that every ticket needs to be tested. More focus on code reviews due to workflow changes in the fourth cycle.

8.5. Analysis of Usage of Agile Practices

- Pair programming, see Figure 8.18: Possible reasons for the change could be: More focus on the team and joint work, due to regular meetings, focus on team spirit with team rules, joint decisions regarding workflow and so on. Interestingly this is the only practice for which the desired value is below the actual value at one point. This could be due to programming days where the whole group worked together on one topic.
- Small / Short releases, see Figure 8.19: Possible reasons for the change could be: A new GUI was necessary, which meant a longer time span without an internal release.
- Meeting (taken from Williams, Krebs, and Layman (2004)), see Figure 8.20: Possible reasons for the change could be: Meetings were held more regularly and the Jira board was used. Hence, meetings were shorter and focused on the current topics.
- Artifact reduction, see Figure 8.22: Possible reasons for the change could be: Usage of pair programming and TDD increased, as a consequence artifact reduction improved as well.
- Morale (taken from Williams, Krebs, and Layman (2004)), see Figure 8.23: Morale increased by around 20 points.: Possible reasons for the change could be: Probably all larger and smaller improvements contributed to the improved morale. Improving team spirit and mood was an important issue for the team, so this is a very positive outcome.

Growth, see Figure 8.21, (taken from Williams, Krebs, and Layman (2004)) and measure and manage flow, see Figure 8.24 show a smaller change, around 10 points increase. This could be due to either the general focus, which was put on improving the way of work or on “natural fluctuation”, for example, if a participant thinks a practice is used between 40% and 50% of the time the selected answer might be alternating between both options between questionnaires.

According to the questionnaires usage of the following practices seems to be stable. Fluctuations are around or smaller than 10 points and probably due to change in respondents or “natural fluctuations” in answers as well. Therefore, these practices will not be discussed and the according figures can be found in the Appendix C:

8. Combining FOSS and Kanban: An Action Research

- Unit tests, see Figure C.1
- CI, see Figure C.2
- Refactoring, see Figure C.3
- Release planning / Planning game, see Figure C.4
- On-site customer, see Figure C.5
- Coding standards, see Figure C.6
- Collective code ownership, see Figure C.7
- Sustainable pace, see Figure C.8
- Simple design, see Figure C.9
- Metaphor / System of names, see Figure C.10
- Lessons learned (taken from Williams, Krebs, and Layman (2004)), see Figure C.11
- Visualize the workflow, see Figure C.12
- Limit WIP, see Figure C.13
- Make process policies explicit, see Figure C.14

Interestingly the usage of Kanban practices did not change drastically over time. The reason could be that first team members did not know about them, resulting in no data points for the first and second UAP questionnaire. Afterwards, as soon as a practice was introduced it was used on a relatively constant level.

8.5. Analysis of Usage of Agile Practices

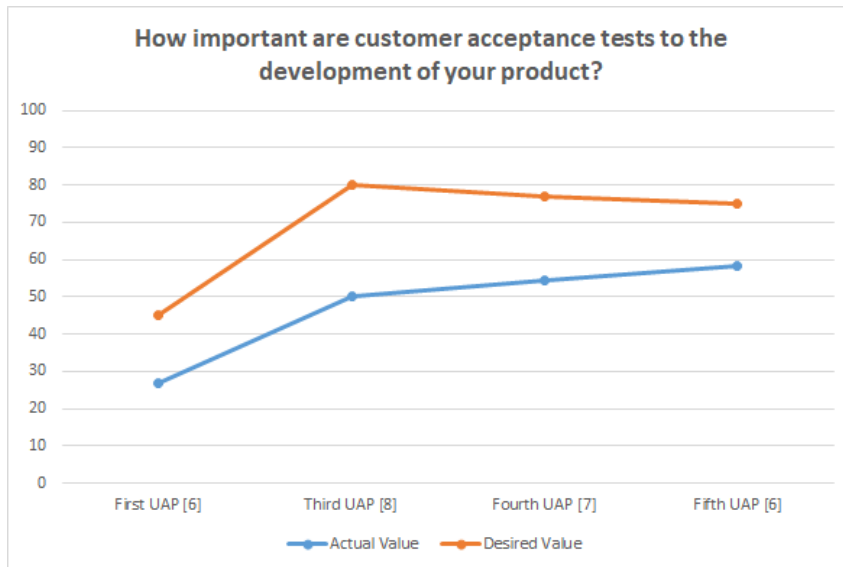


Figure 8.16.: How important are customer acceptance tests to the development of your product?

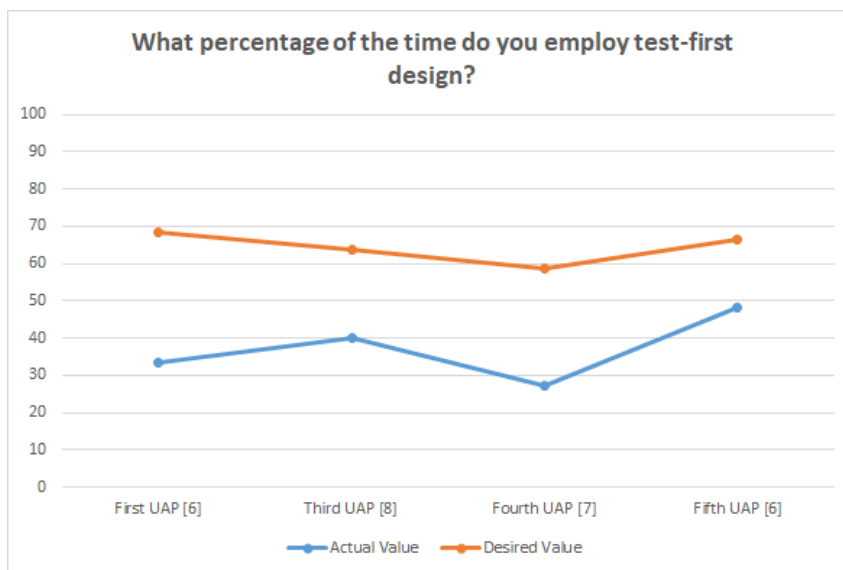


Figure 8.17.: What percentage of the time do you employ test-first design?

8. Combining FOSS and Kanban: An Action Research

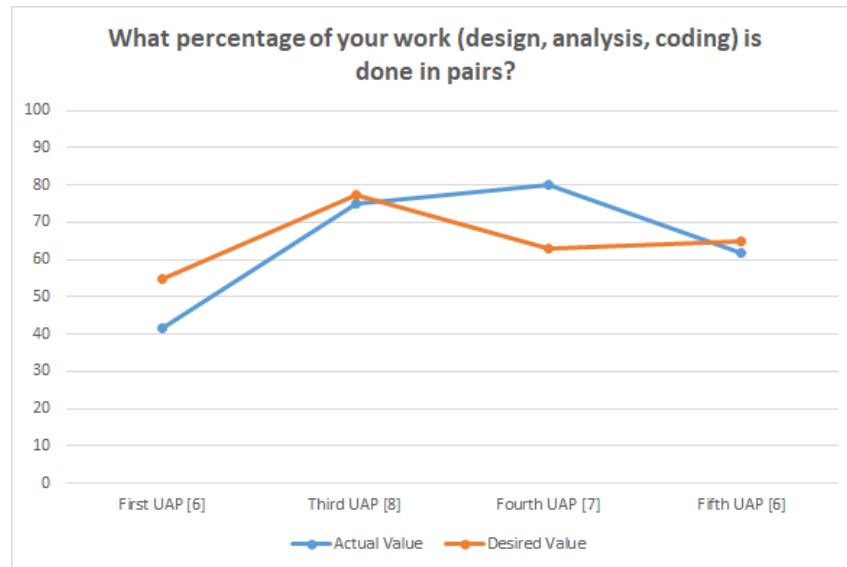


Figure 8.18.: What percentage of your work (design, analysis, coding) is done in pairs?

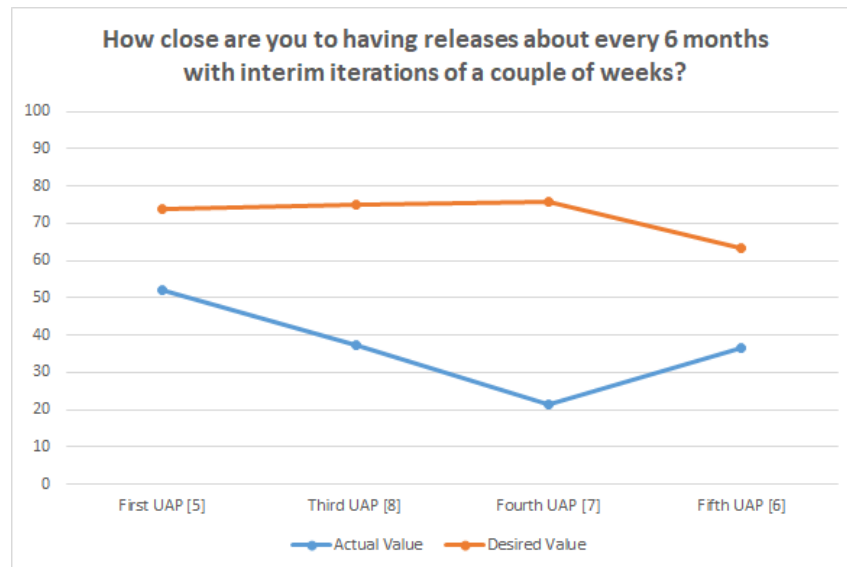


Figure 8.19.: How close are you to having releases about every 6 months with interim iterations of a couple of weeks?

8.5. Analysis of Usage of Agile Practices

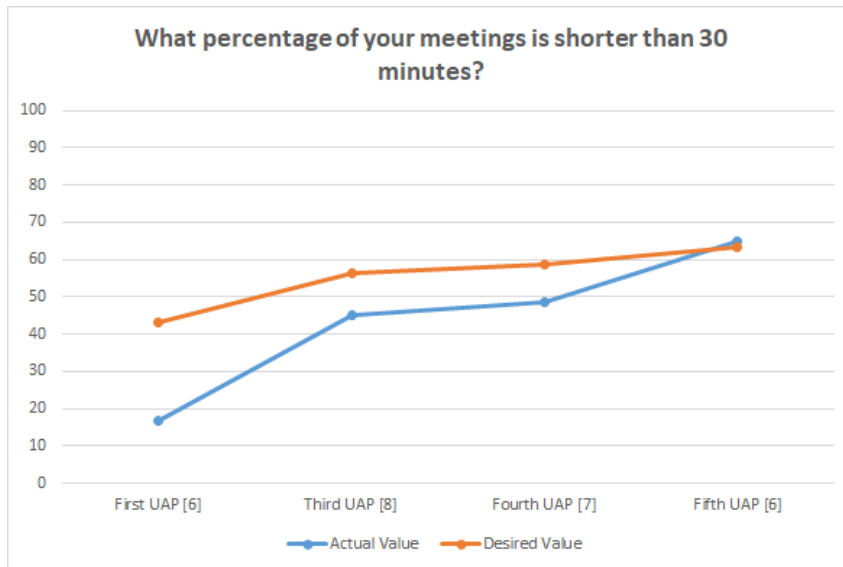


Figure 8.20.: What percentage of your meetings is shorter than 30 minutes?

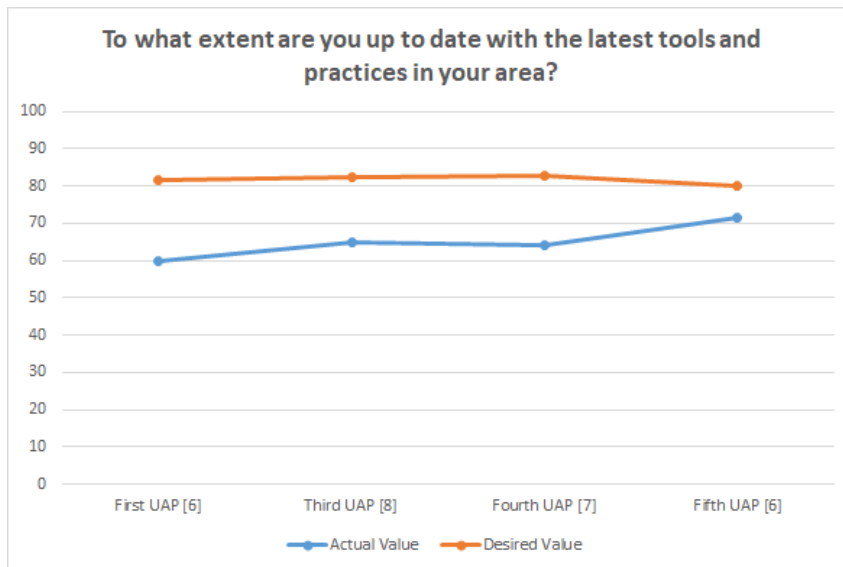


Figure 8.21.: To what extent are you up to date with the latest tools and practices in your area?

8. Combining FOSS and Kanban: An Action Research

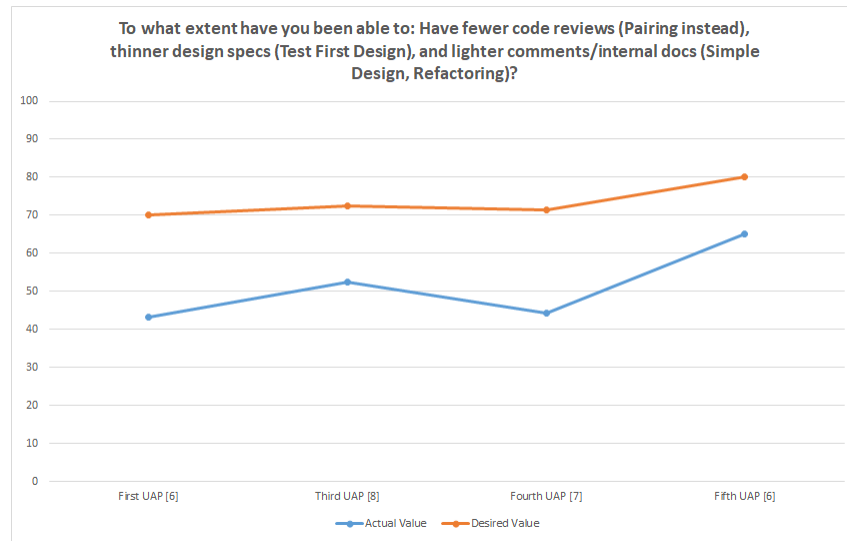


Figure 8.22.: To what extent have you been able to: Have fewer code reviews (Pairing instead), thinner design specs (Test First Design), and lighter comments/internal docs (Simple Design, Refactoring)?

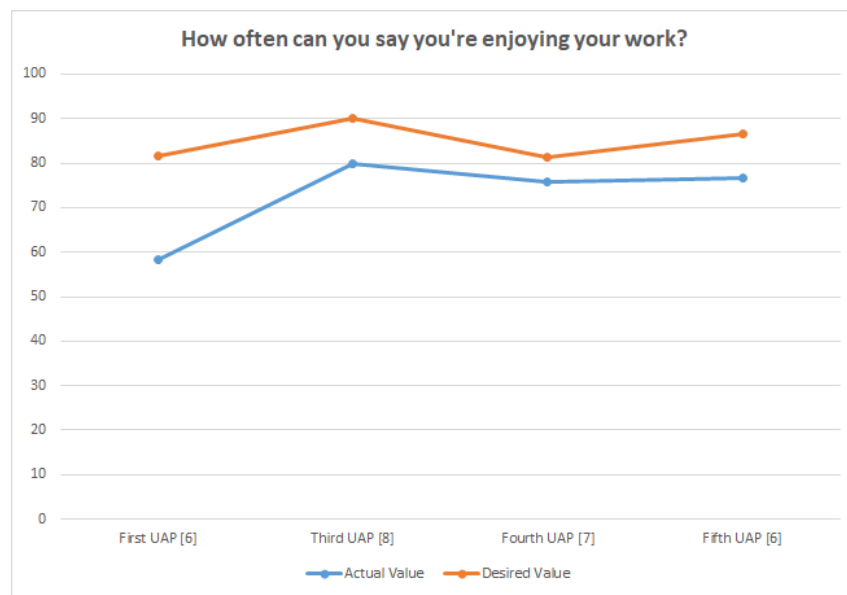


Figure 8.23.: How often can you say you're enjoying your work?

8.5. Analysis of Usage of Agile Practices

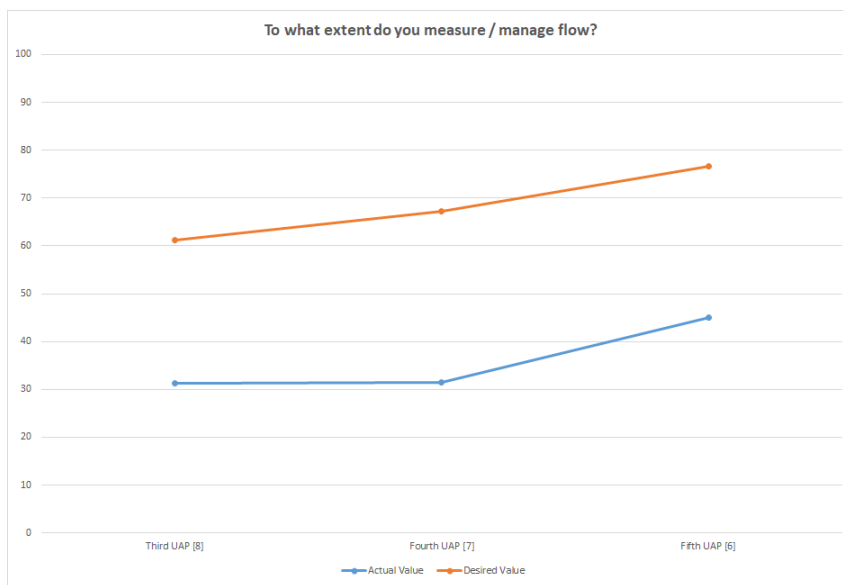


Figure 8.24.: To what extent do you measure / manage flow?

9. Results and Discussion

This chapter is based on Harzl (2016) and Harzl (2017).

9.1. Results

Research results show some promising outcomes regarding the possible benefits of integrating Kanban practices into FOSS development and also some possible challenges. While this research cannot conclusively answer the research question RQ₁ “Can FOSS and ASD be comprehensively combined?” for all FOSS projects, it shows some promising outcomes. Integrating Kanban and FOSS has so far been successful and beneficial for the research participants. New insights, e.g. on the target user group, have been gained and the team’s workflow has become more effective. Also motivation of team members improved due to the positive effects the Kanban coaching had on the team and its processes.

RQ_{2.1} asks if FOSS contributors experience Kanban knowledge as beneficial to their work and all study participants answered yes. Interestingly, some study participants even use personal Kanban after the AR and one person wants to introduce it to the work place environment. In this case the use of Kanban transferred over into other parts of the participants’ lives.

RQ_{2.2} raises the question whether interaction or communication during meetings change with the use of the Kanban method. The majority observed a positive change in interaction and communication especially with other teams and stakeholders. Changes within the team were not experienced as that distinctive.

9. Results and Discussion

RQ2.3 asks if contributors see their time learning about Kanban as well spent or not. All of them regard their time as well spent.

Interesting to note is the role of the coach. It was rated as *very beneficial* by all study participants, which begs the question, if integration of an ASD method can be accomplished in other FOSS settings where a coach is not available to the team and if the observed benefits are stemming more from the coach's presence than the use of Kanban practices. Reliance on a coach could also become a problem, if a team depends on the coach too much, as one participant stated in Section 8.4.7. Another challenge may be, that a team decides to ignore problems, despite all Kanban practices, and one has to figure out how to overcome such blockades. Adding a new role to already established FOSS roles (Tatham, 2010) could be one way to solve both problems. Someone with interest in the topic could gather teaching materials, e.g., videos, which are already available online, and could remind contributors to pay attention to the WIP limits and flow, very similar to inspecting code and giving feedback on the code. This role could also be mindful of possible blockades and speak out, if one is discovered.

Although these results do not provide comprehensive proof that all FOSS projects can profit from using agile methods, they show a case where a project profited from integrating agile methods. The introduction of new methods most probably takes longer than in companies due to the limited amount of time contributors can spend on a project per week, so one needs patience and endurance to introduce Kanban. The team will probably experience some problems with fall-backs into old habits, e.g., trying to sit problems out instead of resolving them quickly, before people develop a sense of kaizen.

Extending the Kanban practices Based on the experiences described in Section 8.4, two additional Kanban practices for FOSS projects are proposed.

- Conduct regular user interviews or feedback sessions with your users
- Review your assumptions about your current development practices

9.2. Threats to Validity

These recommendations are of course based on a hybrid student FOSS project, but usual FOSS projects could benefit from these additional practices as well. While most companies applying agile and lean practices have a marketing and sales team or even a user focus group, FOSS projects tend not to have this kind of resources. Nowadays, many FOSS solutions are employed by a large number of people, who do not contribute to the code, e.g. Mozilla Firefox or Linux. Thus, FOSS developers are not simply “scratching their own itch” anymore, they serve many people all over the world, who must not share the developer’s requirements and domain knowledge. Therefore, it could be beneficial for FOSS projects to investigate their users’ needs. As for the second recommendation: Although, the Kanban principles allow to “start where you are”, it could be beneficial for FOSS projects to review their current development practices before embarking on the endeavor of integrating Kanban, or any other agile or lean method, into their development process. FOSS projects usually do not have Scrum masters, process experts or in general someone, who controls whether software development practices are exercised correctly. An honest and critical reflection about the current practices can clear some misconceptions, can further a joint understanding of the current situation and it is a first step into the direction of kaizen.

9.2. Threats to Validity

The main threats to the validity of this research will be discussed in this section.

Internal validity Response bias, might have led to a more positive feedback about Kanban and its possible benefits, because team members know the researcher personally. In an attempt to counterbalance this response bias, study participants were assured, that negative feedback is valuable feedback and that nobody is looking for unwarranted praise.

Another limitation could be the researchers positionality (Herr and Anderson, 2015) in the setting. Herr and Anderson (Herr and Anderson, 2015) describe positionality as asking the question, “Who am I in relation to

9. Results and Discussion

my participants and my setting?”. In Subsection 6.4 power distance was already identified as a possible limitation. Participants perceiving the researcher as someone with informal power, may result in research bias, since suggestions the researcher makes could be accepted due to this perceived power distance and not only because team members agree with suggestions. In an attempt to counterbalance this bias somewhat, for all necessary decisions at least two alternatives were proposed whenever possible and the final decision was made by the team.

External validity The research is limited to a single case of a hybrid student FOSS project, so the external validity is very limited. Further research is needed to achieve more generalizable results, which are applicable to other teams and other FOSS projects.

Although a

“frequent misconception is that empirical research within one company or one project is not good enough, provides little value for the academic community, and does not contribute to scientific development” (Shihab, Bird, and Zimmermann, 2012),

the authors of Shihab, Bird, and Zimmermann (2012) cite publications, which show a different picture. Flyvbjerg (2006) shows examples from social sciences, physics and economics and Basili, Shull, and Lanubile (1999) argue that both kinds of studies, those from single cases and those of large samples are essential (Teixeira, Robles, and González-Barahona, 2015).

Setting The special setting of a hybrid student FOSS project may be seen as a limiting factor as well, because some characteristics differ from traditional FOSS projects. Developers want to earn course credits and not only earn reputation among other developers, contribute to a bigger cause or “scratch a personal itch” (Raymond, 2001). There exists no group of core developers, which in typical FOSS projects consists of 10% to 20% of a team, and which creates around 80% of the source code (Koch, 2004), and student members change regularly. This could be a future area of research, determining if and how these different characteristics impact a FOSS project.

9.2. Threats to Validity

Students as main developers of this project may also be considered as a limiting factor, because they have not finished their studies. However, many people without a formal education in software engineering and from various backgrounds are FOSS contributors, regardless of their formal education. IT students work as normal developers on many FOSS projects. In 2000 Höst, Regnell, and Wohlin (2000) already concluded that last-year software engineering students are relevant when considering experimentation in software engineering. The use of students as study objects for establishing a trend is quite acceptable, e.g. Madeyski (2010) describes the benefit of students as study objects. And according to Salman et al. (Salman, Misirli, and Juzgado, 2015) when a development approach is new to both groups, students and professionals, show similar performances in carefully scoped software engineering experiments.

The setting with changing participants is also a limitation of the study. Conditions and questionnaire outcomes might have changed due to people leaving and joining the project. Also people not answering all questionnaires while working on the project, might impact the outcomes. That is why for UAP evaluations only changes of more than 10 points, were regarded as actual changes. Values below that are regarded as “natural fluctuation”. Apart from reminding people on a regular basis to answer questionnaires, there is no good way of keeping the number of questionnaire respondents and team members constant. Not taking on new members would negatively impact the team and stopping people from leaving the project is not possible at all. One could only limit the length of the study to a few weeks to minimize the risk of losing members. However, this would probably not yield meaningful insights.

10. Conclusion and Future Work

This chapter is based on Harzl (2016) and Harzl (2017).

10.1. Conclusion

This thesis describes a practical integration of Kanban through AR in the context of a hybrid student FOSS project. Studying the integration of ASD and FOSS furthered the understanding of both worlds. It could be shown that Kanban and the hybrid student FOSS could be combined and that the project members benefited from it. Based on the findings of this work additional Kanban practices for FOSS projects were proposed, which could support FOSS projects on their way of adopting agile practices. There is a lack of research regarding integration of ASD in the FOSS development context (Gandomani et al., 2013). Thus, this work contributes by offering some insights on the matter.

10.2. Future Work

As this AR was done with only one team of a hybrid student FOSS project, further steps are required to strengthen the results and to increase the validity of the contributions. Possible future work could include one or more of the following topics.

- Monitor the studied team to see if the changes are permanent.
- Extend AR to more teams of the same project, to evaluate if the outcome depended on the specific team.

10. Conclusion and Future Work

- Use unfamiliar and separate persons as agile coach and researcher to reduce response bias.
- Repeat the AR with various other FOSS projects to assure the validity of the findings so far.
- Research community-initiated projects and sponsored or spinout projects, if they respond differently to the introduction of Kanban.
- Investigate the role of the agile coach. Is this role necessary to integrate an ASD method into a FOSS project? What could be alternatives to convey the knowledge and to support the use of the practices?
- Investigate if an additional FOSS role could replace an on-site coach.
- Investigate if the observed benefits stem from the use of Kanban or the coach.
- Investigate if or how the (non) existence of a core group and ever changing (student) members impact a FOSS project.

Finally I would like to conclude with a quote:

“As with all qualitative research, we do not intend to portray a generalized view of all free software development projects.” (Elliott and Scacchi, 2005)

Appendix

Appendix A.

Papers

This chapter summarizes all peer-reviewed and published papers. Contributions of co-authors will be explicitly listed, everything else was the contribution of the author of this thesis.

- Harzl, Krnjic, et al. (2013a). “Comparing Purely Visual with Hybrid Visual/Textual Manipulation of Complex Formula on Smartphones.” In: *Proceedings of the 19th International Conference on Distributed Multimedia Systems, DMS 2013, August 8-10, 2013, Holiday Inn, Brighton, UK*. Knowledge Systems Institute, pp. 198–201. ISBN: 1-891706-34-9
Wolfgang Slany motivated the paper. Franz Schreiner implemented the formula editor. Vesna Krnjic wrote major parts of the paper. I helped with writing and provided major contributions in terms of literature research and proof-reading the paper.
Referring to Chapter 4.
- Harzl, Neidhoefer, et al. (2013). “A Scratch-like visual programming system for Microsoft Windows Phone 8.” In: *CoRR abs/1310.1390*. URL: <http://arxiv.org/abs/1310.1390>
Wolfgang Slany motivated the paper. Philipp Neidhoefer, Valentin Rock and Maximilian Schafzahl implemented the application and wrote about the application and typical usage example. I wrote about Pocket Code and provided major contributions in terms of literature research and proof-reading the paper.
Referring to Chapter 4.

Appendix A. Papers

- Harzl, Krnjic, et al. (2013b). “Purely Visual and Hybrid Visual/Textual Formula Composition: A Usability Study Plan.” In: *Proceedings of Programming for Mobile and Touch PProMoTo 2013*
Wolfgang Slany motivated the paper. Franz Schreiner implemented the formula editor. Vesna Krnjic wrote major parts of the paper. I helped with writing and provided major contributions in terms of literature research and proof-reading the paper.
Referring to Chapter 4.
- Fellhofer, Harzl, and Slany (2015). “Scaling and Internationalizing an Agile FOSS Project: Lessons Learned.” In: *Open Source Systems: Adoption and Impact - 11th IFIP WG 2.13 International Conference, OSS 2015, Florence, Italy, May 16-17, 2015, Proceedings*. Ed. by Damiani et al. Vol. 451. IFIP Advances in Information and Communication Technology. Springer, pp. 13–22. ISBN: 978-3-319-17836-3. DOI: [10.1007/978-3-319-17837-0_2](https://doi.org/10.1007/978-3-319-17837-0_2). URL: http://dx.doi.org/10.1007/978-3-319-17837-0_2
I had the idea for and motivated the paper. Stephan Fellhofer realized the technical implementation. Stephan Fellhofer and I equally contributed to writing the paper. Additionally I provided major contributions in terms of literature research and proof-reading the paper. Wolfgang Slany helped in proof-reading the paper.
Referring to Chapters 4 and 7.
- Harzl (2015). “Combining Kanban and FOSS: Can it work?” In: *Agile Processes, in Software Engineering, and Extreme Programming*. , pp. 352–353
Wolfgang Slany helped in proof-reading the paper.
Referring to Chapter 1.
- Harzl (2016). “Combining FOSS and Kanban: An Action Research.” In: *Open Source Systems: Integrating Communities - 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30 - June 2, 2016, Proceedings*. Ed. by Kevin Crowston et al. Vol. 472. IFIP Advances in Information and Communication Technology. Springer, pp. 71–84. ISBN: 978-3-319-39224-0. DOI: [10.1007/978-3-319-39225-7_6](https://doi.org/10.1007/978-3-319-39225-7_6). URL: http://dx.doi.org/10.1007/978-3-319-39225-7_6
Sole author.
Referring to Chapter 8.
- Harzl (2017). “Can FOSS projects benefit from integrating Kanban: a

case study." In: *Journal of Internet Services and Applications* 8.1, p. 7.
ISSN: 1869-0238. DOI: [10.1186/s13174-017-0058-z](https://doi.org/10.1186/s13174-017-0058-z). URL: <http://dx.doi.org/10.1186/s13174-017-0058-z>
Sole author.
Referring to Chapter 8.

Appendix B.

Questionnaires

TWiki Survey

1. Have you ever used some kind of knowledge management system?
 - Yes
 - No
2. If yes, which topics, kind of data were in the system?
3. If yes, which technologies were used (multiple answers possible)?
 - Collection of documents
 - Wiki
 - SharePoint
 - Other:
4. If yes, how helpful do you think the knowledge management was?
 - Very helpful
 - Helpful
 - Not very helpful
 - Not helpful
5. How often do you use the Catrobat TWiki?
 - Seldom or never
 - Once a month
 - Several times a month
 - Once a week
 - Several times a week
 - Daily
 - Other
6. How helpful do you think the TWiki is?
 - Very helpful (I always find what I'm looking for)
 - Helpful (I often find what I'm looking for)
 - Not very helpful (I seldom find what I'm looking for or the information is hardly in the TWiki/up-to-date/helpful)
 - Not helpful (I almost never find what I'm looking for or the information is not in the TWiki/up-to-date/helpful)
 - Other:

-
7. What areas could be improved in the TWiki (multiple selections possible)?
- Clear arrangement
 - Up-to-dateness of the information
 - Completeness of the information
 - Correctness of the information
 - In all areas
 - In no areas
 - Other:
8. Please describe in detail what should be improved, so we can derive concrete improvements. E.g. the structure of the start page should look like this.... I would like.... This & that should be done this way....
9. Which topics / information are missing in the TWiki from your point of view?
10. Which topics / information are overrepresented in the TWiki or should be removed from your point of view?
11. Would you use the TWiki more often in the future, if your aforementioned topics would be taken into consideration and implemented?
- Yes
 - Rather yes
 - Rather no
 - No
12. If rather no or no, why?
13. What other organizational changes / improvements would you like to see in the project (e.g. to spend as little time as possible on administration)?
14. What other wishes do you have towards the project? What do you expect from the project, teammates, supervision...?

Kanban Questionnaire

1. Do you think, that knowledge about Kanban and its practices is beneficial to your work in the project?

- 1 = very beneficial
- 2 = rather beneficial
- 3 = undecided
- 4 = rather useless
- 5 = very useless

Please only select one of the following options:

- 1
- 2
- 3
- 4
- 5

2. In what way? Please give an example.

In which case did you benefit from your knowledge about Kanban or from applying Kanban practices, in which case did you not benefit? Was there a practical benefit?

If you benefited from Kanban, please reflect on your example, if the experienced benefit was due to the use of Kanban practices or due to having a coach. The role of the coach will be examined later in the questionnaire.

3. Do you think the benefit, you may have gained from using Kanban, is worth the time you put into the Kanban training and meeting sessions or not? (Please do not take into account the time you needed to fill in the questionnaires.)

Please only select one of the following options:

- yes, definitively
- yes
- marginally / undecided
- no
- no, definitively not
- other:

4. Do you think, that knowledge about Kanban and its practices is beneficial to your work outside the project?

- 1 = very beneficial
- 2 = rather beneficial
- 3 = undecided
- 4 = rather useless
- 5 = very useless

Please only select one of the following options:

- 1
- 2
- 3
- 4
- 5

5. In what way? Please give examples.

Appendix B. Questionnaires

6. Do you think, it is beneficial to have a coach in your team?

Please do not rate the coach personally (or his abilities). Please rate if you think it is beneficial to have someone at team meetings who keeps an eye on work processes and how to improve them.

- 1 = very beneficial
- 2 = rather beneficial
- 3 = undecided
- 4 = rather useless
- 5 = very useless

Please only select one of the following options:

- 1
- 2
- 3
- 4
- 5

7. In what way? Please give examples.

Which suggestions, ideas, actions were helpful, which were not?

What did you experience as supportive, what was rather impedimentary?

Please try to distinguish between the person and the role.

8. Did the team's communication/interaction with the most important stakeholders change due to the use of Kanban practices?

1 = very positive change

2 = rather positive change

3 = unchanged / undecided

4 = rather negative change

5 = very negative change

Please only select one of the following options:

1

2

3

4

5

9. What did change? For better? For worse? What are the consequences of the change?

Appendix B. Questionnaires

10. Did the team's communication/interaction within the team change due to the use of Kanban practices?

1 = very positive change

2 = rather positive change

3 = unchanged / undecided

4 = rather negative change

5 = very negative change

Please only select one of the following options:

1

2

3

4

5

11. What did change? For better? For worse? What are the consequences of the change?

12. Would you like to add something, about Kanban, the team, work, the project, anything else?

Feedback Effort Estimation

1. What is effort estimation meant to achieve? Why does one need it **in general**?

Agile Estimating – Story Points Method with 1, 2, 5, 10, 20, 50, 100, 200, 500

2. How many of your tickets did you actually estimate? (no quasi-estimating, using default value)

X - Disagree with using this practice
0 - Never (no disagreement, just don't do it)
1 - Hardly ever (10%)
2 - Rarely (20%)
3 - Sometimes (30%)
4 - Common (40%)
5 - Half & Half (50%)
6 - Usually (60%)
7 - Often (70%)
8 - Regular (80%)
9 - Always (90%)
10 - Fanatic (100%)

3. What is your desired value for **Agile Estimating – Story Points Method**?

X - Disagree with using this practice
0 - Never (no disagreement, just don't do it)
1 - Hardly ever (10%)
2 - Rarely (20%)
3 - Sometimes (30%)
4 - Common (40%)
5 - Half & Half (50%)
6 - Usually (60%)
7 - Often (70%)
8 - Regular (80%)
9 - Always (90%)
10 - Fanatic (100%)

Appendix B. Questionnaires

4. Did you use the effort estimation for e.g. planning, deciding to take a ticket? Please describe why and how you used it, respectively, did not use it?

Agile Estimating – Story Points Method

5. How did you conduct effort estimation? **Agile Estimating – Story Points Method**

6. How did you use the ratios? **Agile Estimating – Story Points Method**

7. Did you use a consistent base item or baseline for estimating over time?
Agile Estimating – Story Points Method

8. How much time did you spend on average to estimate a ticket? **Agile Estimating – Story Points Method**

9. In your opinion, how useful was estimating the tickets? **Agile Estimating – Story Points Method**

Absolutely not useful

Hardly useful

Neutral/Undecided

Rather useful

Very useful

10. Please give a reason for your previous answer (9) **Agile Estimating – Story Points Method**

Agile Estimating – T-Shirt Size Method with sizes S, M, L

S: You can do it at once

M: You will need 2-3 days

L: You will need one week or more

1. How many of your tickets do you actually estimate? (no quasi-estimating, using default value) **Agile Estimating – T-Shirt Size Method**

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

2. What is your desired value for **Agile Estimating – T-Shirt Size Method?**

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

3. Do you use the effort estimation for e.g. planning, deciding to take a ticket? Please describe why and how you used it, respectively, did not use it? **Agile Estimating – T-Shirt Size Method**

4. How do you conduct effort estimation? **Agile Estimating – T-Shirt Size Method**

5. How do you use the ratios? **Agile Estimating – T-Shirt Size Method**

Appendix B. Questionnaires

6. Do you use a consistent base item or baseline for estimating over time? **Agile Estimating – T-Shirt Size Method**

7. How much time do you spend on average to estimate a ticket? **Agile Estimating – T-Shirt Size Method**

8. In your opinion, how useful is estimating the tickets? **Agile Estimating – T-Shirt Size Method**

Absolutely not useful

Hardly useful

Neutral/Undecided

Rather useful

Very useful

9. Please give a reason for your previous answer (8) **Agile Estimating – T-Shirt Size Method**

Feedback Motivation

1. (How) Did your motivation/mood within the Catrobat project change over time?
much less motivation
rather lower motivation
unchanged
rather more motivation
much more motivation
2. If there have been different phases, please describe them chronologically, what characterized them and what were triggering circumstances (if there were any)?
3. How do changes in your motivation show? E.g. more fun, you take more time for the project, in general more content, less stress...
4. What do you think could be the reasons for the changes in motivation?
E.g. less exam stress, workload outside of work, private circumstances...
5. Did the Kanban coaching contribute to changes in your motivation? If yes, how?
6. How would you assess the changes due to the Kanban coaching and its consequences?
much less motivation
rather lower motivation
unchanged
rather more motivation
much more motivation
7. Please give reasons for your answer
8. To what extent do you benefit from your knowledge about Kanban practices?
influenced me negatively / did not want to know
not at all
indifferent – no advantage or disadvantage
rather benefitted
benefitted very much
9. Please give reasons for your answer

Usage of Agile Practices

adapted from Shodan 2.0 Input Metric Survey

This survey is designed to evaluate to what extent you are using eXtreme Programming and other practices in your Catrobat team (not at work). If a question is aimed at the general use of a practice and not restricted to Catrobat, it is stated in the question. Knowledge questions are of course aimed at your general knowledge and not restricted to knowledge you acquired through Catrobat.

The survey will also help teach you a bit about XP and Kanban practices and hopefully provide insights into what the various practices may involve. Even though you may not be fully into XP and Kanban yet, you may use a few of these practices to some extent as a part of your personal software process. Furthermore, it is very common for XP teams not to score perfectly on all of the practices, and a perfect score is by no means a prerequisite for the successful use of XP or Kanban.

In the survey you will find a list of XP and Kanban practices. Next to each question will be a dropdown for selecting a score from 1-10 which will be used in the computation of XPness. Alongside each practice is a question and several items to help you consider and gauge the score you select for that practice. Please put down your scores according to your personal programming practices; not the team as a whole.

P.S. Low scores are okay!

1. **Automated Unit Tests**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

2. **Automated Unit Tests** (only shown if previous answer is not "Nothing")

Automated unit tests (such as JUnit) are an essential part of the development process. They provide a testbed for verifying the correctness of software and allow code to be safely integrated and disseminated among team members.

To what extent do you employ automated unit tests?

- Automated unit tests exist for production code.

- A tool is used to measure test coverage.

- There is an automated way to run the entire suite of unit tests for an entire program.

- All unit tests are run and passed when a task is finished and before checking in/integrating.

- When fixing bugs, unit tests are used to capture the bug before fixing.

- Unit tests are refactored.

- Unit tests are fast enough to be run all the time.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

3. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

4. Customer Acceptance Tests

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

5. Customer Acceptance Tests (only shown if previous answer is not "Nothing")

Customer acceptance tests exist to ensure both the developers and the customer know what they want. All acceptance tests must be passed before the product can be delivered to the customer.

How important are customer acceptance tests to the development of your product?

- Acceptance tests are used to verify system functionality and customer requirements.

- Customer provides acceptance criteria.

- Customer uses acceptance test to determine what has been accomplished at the end of an iteration.

- Acceptance testing is automated.

- A User Story is not finished until its acceptance tests pass.

- Acceptance tests are run automatically every night.

- A test environment that matches our end-user's environment is used to test.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

6. What is your desired value for this specific practice mentioned directly above?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

7. **Test-First Design/Test-Driven Development**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

8. **Test-First Design** (only shown if previous answer is not "Nothing")

Test-first design is the practice by which a test case is written before the code is implemented. The implemented code is written to pass the test case. This practice produces higher quality code and higher programmer confidence in their code.

What percentage of the time do you employ test-first design?

- Code is only written after a unit test (that fails) has been written first.

- All production code is written using test first design.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

9. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

Appendix B. Questionnaires

10. **Pair programming**
How much do you know about this Practice? *
Please choose only one of the following:
Nothing
Heard about
Read about
Done it
Everything
11. **Pair programming** (only shown if previous answers is not "Nothing")
Two people, one computer. One thinks strategy, the other thinks tactics. This practice produces higher quality code at the same level of productivity.
What percentage of your work (design, analysis, coding) is done in pairs?
- People can go on vacation without regard to what work needs to be done.
- Drivers keep Navigators engaged.
- Drivers and Navigators switch roles often.
- Navigators keep a to do list.
- People switch pairing partners regularly.
- The team has work stations conducive to pair programming.
- Practices are enforced by peer pressure.
- Production code is not written without a pair.
- Repetitive and dull tasks that would not gain from pair programming are automated.
Please choose only one of the following:
X - Disagree with using this practice
0 - Never (no disagreement, just don't do it)
1 - Hardly ever (10%)
2 - Rarely (20%)
3 - Sometimes (30%)
4 - Common (40%)
5 - Half & Half (50%)
6 - Usually (60%)
7 - Often (70%)
8 - Regular (80%)
9 - Always (90%)
10 - Fanatic (100%)
12. **What is your desired value for this specific practice mentioned directly above?**
Please choose only one of the following:
X - Disagree with using this practice
0 - Never (no disagreement, just don't do it)
1 - Hardly ever (10%)
2 - Rarely (20%)
3 - Sometimes (30%)
4 - Common (40%)
5 - Half & Half (50%)
6 - Usually (60%)
7 - Often (70%)
8 - Regular (80%)
9 - Always (90%)
10 - Fanatic (100%)

13. **Refactoring**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

14. **Refactoring** (only shown if previous answers is not "Nothing")

Rewrite code that 'smells bad' to improve future maintenance and flexibility without changing its behavior.

How often do you stop to cleanup code that has already been implemented without changing functionality?

- Code contains minimal or no duplication.

- Team refactors often or when applicable.

- There are enough unit tests and/or automated acceptance tests to allow merciless refactoring.

- Any code is open to refactoring.

- Refactoring is done only to improve existing code and not to anticipate future tasks.

- Future refactorings have been identified.

- Long term refactorings are going on now.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

15. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

Appendix B. Questionnaires

16. **Release Planning / Planning Game**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

17. **Release Planning / Planning Game** (only shown if previous answers is not "Nothing")

The planning game is a highly interactive process between all stakeholders wherein customers and developers trade items in and out of the plan based on current priorities and costs. Adaptation is favored over following a plan. **Do you allow for changes in release plans/requirements after each iteration based on customer feedback and current implementation? How well does planning correspond to the criteria below?**

- There is a release plan.
- The whole team including coach, customer, developer, etc. is present during release planning.
- The customer picks the order of the User Stories in the release plan.
- When stories are added to a release, stories of equal value may be re-prioritized.
- Developers estimate the time needed to complete the User Stories.
- Developers break down User Stories into tasks. Each developer signs up for tasks and estimates the ones he/she owns.
- The release plan is used to determine how much can be done by a certain time.
- Past User Story experience aids in determining how much can be done by a certain time.
- Release points have been identified and communicated to all stakeholders
- At least one User Story is created for automating acceptance tests.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

18. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

19. **On-Site Customer**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

20. **On-Site Customer & Small Releases** (only shown if previous answer is not "Nothing")

The customer is the body for whom the product is being developed and may be either internal or external. Customer access is imperative to developing a product that satisfies the customers' needs as well as clear up requirement ambiguity/incompleteness. On-Site Customer is best, but you can use chat, e-mail, telephone, etc., to quickly verify requirements and get feedback. Ideally, the customer is always available.

What percentage of the time do you get quick interaction with your customers when needed?

- Customer is involved in release planning.
- The developers have direct access (telephone/email/video conference) to the customer.
- The developers have same day responses from customer.
- The customer is on-site.
- Fast and consistent feedback between customer and developer.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

21. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

22. **Short Releases**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

23. **Short Releases** (only shown if previous answer is not "Nothing")

You have frequent smaller releases instead of larger less frequent ones. This lets the customer see the progress of the project and allows the developer to get feedback.

How close are you to having releases about every 6 months with interim iterations of a couple of weeks?

- The customer has identified release points.

- The product is releasable (internally or externally) every six months or less.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

24. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

25. **Continuous Integration**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

26. **Continuous Integration** (only shown if previous answers is not "Nothing")

Continuous integration works in concert with collective code ownership to ensure that developers have the most recent version of code available. Code is checked in quickly to avoid code synchronization/integration hassles.

How often to you synchronize and check in your code on average?

- Source control/VCS is used.
- A build machine automatically builds at least once per day.
- Unit tests and acceptance tests are run as a part of each build.
- The build machine informs developers when the build fails.
- The build process is fast enough to support continuous integration.
- The team integrates at least once per day, preferably several times per day (on days they are working).

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

27. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

28. **Coding Standards**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

29. **Coding Standards** (only shown if previous answer is not "Nothing")

Do you have and adhere to team coding standards? Besides brace placement, this may include things like logging and performance idioms. Strong standards make collaboration, refactoring, and collective ownership an easier process.

Is there a coding standard in place and how often is it followed?

- There is a coding standard for all used languages.
- The coding standard is known and used by the whole team.
- A tool exists to format the code in accordance with the coding standards.
- The coding standard is short and covers readability issues and not design issues.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

30. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

31. **Collective Code Ownership**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

32. **Collective Code Ownership** (only shown if previous answer is not "Nothing")

You can change anyone's code and they can change yours. This allows for stronger knowledge transfer amongst the team and ensures that you don't get stuck when the expert is busy or on vacation. People know many parts of the system.

Can people change code they did not originally write, and how often do they do so?

- Some form of VCS is used that allows multiple people to work on the same file at the same time.
- Everyone is allowed to change any code.
- Everyone has the knowledge to change any code or can pair with a person that has the knowledge.
- There are enough unit tests and/or automated acceptance tests to allow people to safely change any code.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

33. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

34. **Sustainable Pace**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

35. **Sustainable Pace** (only shown if previous answers is not "Nothing")

People need to be effective over the long haul. Overworking has negative impacts on productivity, morale, and home life.

How well do you pace yourself? What percentage of the development time is evenly paced (for example working the same number of hours each week)

- High-rate of productivity is maintained without being overworked.
- Vacations or classes are never postponed or canceled due to work.

Examples Scores: 10- I maintain a sustainable pace and the same high rate of output. 5 - I work longer than what I consider a sustainable pace, but still produce at a high rate and feel only a little burnt out. 2 - I work beyond a sustainable pace and feel burnt out. My code isn't at its usual high quality.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

36. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

37. **Simple Design**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

38. **Simple Design** (only shown if previous answers is not "Nothing")

Keep it simple at first; do the simplest thing that could possibly work. You don't follow the philosophy of "I'll include this because the customer might possibly need it later" even though the feature isn't in the requirements. Also, you do not spend a lot of time on design documents.

How often do you succeed in 'Keeping it Simple'?

- Always do "the simplest thing that can possibly work".
- Follow the principle of YAGNI, "You aren't going to need it!" --Only build what is currently necessary.
- Refactoring is used to keep design clean.
- There is no unused or commented-out code.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

39. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

Appendix B. Questionnaires

40. **Metaphor & System of Names**

How much do you know about this Practice? *

Please choose only one of the following:

- Nothing
- Heard about
- Read about
- Done it
- Everything

41. **Metaphor & System of Names** (only shown if previous answer is not "Nothing")

A single, overarching metaphor is used to describe the system, such as an "assembly line." In the event that this is not possible, the team may use a "system of names" to describe the various components of the project in a consistent manner. For example, all items related to the database are prefaced by Database. The metaphor/system of names is used by developers to help communicate ideas and to explain concepts to customers.

How often do you feel this is true of the systems you develop?

- Classes and methods have good, descriptive names.
- Classes and methods have names relative to one another.
- New members of the team do not need to often ask or refer to a document to understand the architecture.
- The customer understands/can explain the metaphor.
- The developer understands/can explain the metaphor.

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

42. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

- X - Disagree with using this practice
- 0 - Never (no disagreement, just don't do it)
- 1 - Hardly ever (10%)
- 2 - Rarely (20%)
- 3 - Sometimes (30%)
- 4 - Common (40%)
- 5 - Half & Half (50%)
- 6 - Usually (60%)
- 7 - Often (70%)
- 8 - Regular (80%)
- 9 - Always (90%)
- 10 - Fanatic (100%)

43. **Meeting**

What percentage of your meetings is shorter than 30 minutes?

- Meetings take place regularly.
- Meetings are short and to the point, focusing only on what has been done and needs to be done over the next days.
- Team members exhibit courage in discussing concerns and successes.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

44. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

45. **Lessons Learned**

To what extent does the team review how to get better after every release?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

46. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

47. Growth

Consider the latest tools and practices in addition to skills. If you're not learning, you're falling behind!

To what extent are you up to date with the latest tools and practices in your area?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

48. What is your desired value for this specific practice mentioned directly above?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

Appendix B. Questionnaires

49. **Artifact Reduction**

With agile methods you have fewer/thinner versions of artifacts from classic techniques. This saves time, which can be invested in better tests, new code, refactoring, etc.

To what extent have you been able to: Have fewer code reviews (Pairing instead), Thinner design specs (Test First Design), and Lighter comments/internal docs (Simple Design, Refactoring)

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

50. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

51. **Morale**

How often can you say you're enjoying your work? Ok, this is not really an output of your process, but it's collected here for validation and to see if XP is enjoyable.

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

52. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

Appendix B. Questionnaires

53. **Visualize the workflow**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

54. **Visualize the workflow** (only shown if previous answer is not "Nothing")

You cannot improve what you cannot see. Knowledge work needs a way to show progress. Kanban boards are one of the ways to display progress.

What percentage of your work items is displayed on your Kanban board?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

55. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

56. **Work in progress (WIP) Limit**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

57. **Work in progress (WIP) Limit** (only shown if previous answers is not "Nothing")

The number of issues, which can be in one state (column) is limited. That leads to a pull system where every subsequent station pulls work from the previous station, in contrast to pushing finished work to the next station.

To which extent do you limit WIP?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

58. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

Appendix B. Questionnaires

59. **Measuring the flow**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

60. **Measure and manage flow** (only shown if previous answers is not "Nothing")

Things like lead time, cycle time and throughput are measured and used to determine how well work is organized and where is room for improvement. This facilitates planning and improves the reliability.

To what extent do you measure these values?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

61. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

62. **Make policies explicit**

How much do you know about this Practice? *

Please choose only one of the following:

Nothing

Heard about

Read about

Done it

Everything

63. **Make policies explicit** (only shown if previous answer is not “Nothing”)

To ensure everybody knows with which rules and assumptions work is done, all rules are made explicit. For example the term 'done' is defined, similar to 'Definition of Done in Scrum', meaning of the different columns on the board etc.

What percentage of your rules are explicit stated, so everybody knows them?

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

64. **What is your desired value for this specific practice mentioned directly above?**

Please choose only one of the following:

X - Disagree with using this practice

0 - Never (no disagreement, just don't do it)

1 - Hardly ever (10%)

2 - Rarely (20%)

3 - Sometimes (30%)

4 - Common (40%)

5 - Half & Half (50%)

6 - Usually (60%)

7 - Often (70%)

8 - Regular (80%)

9 - Always (90%)

10 - Fanatic (100%)

*Compulsory question

Questions “How much do you know about this Practice” were removed as soon as nobody selected “Nothing” in the previous survey.

Appendix C.

Supplementary Material

Appendix C. Supplementary Material

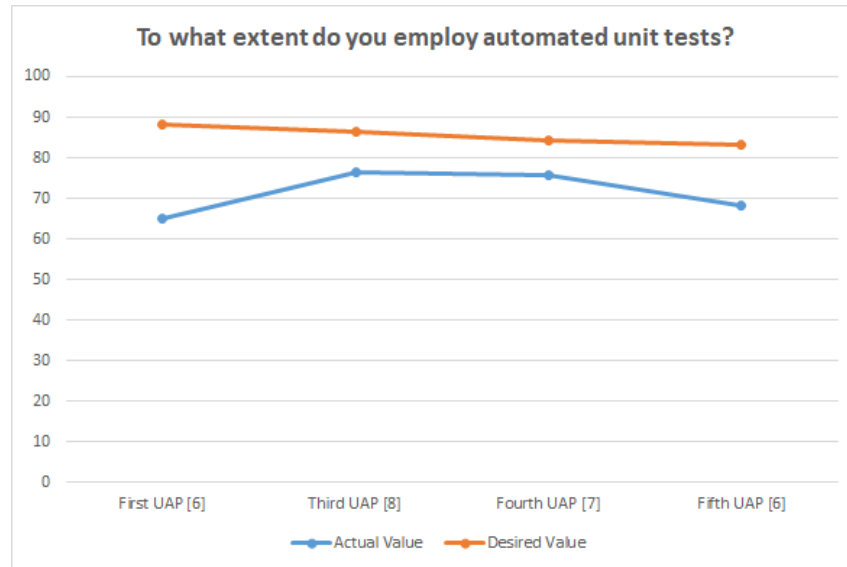


Figure C.1.: To what extent do you employ automated unit tests?

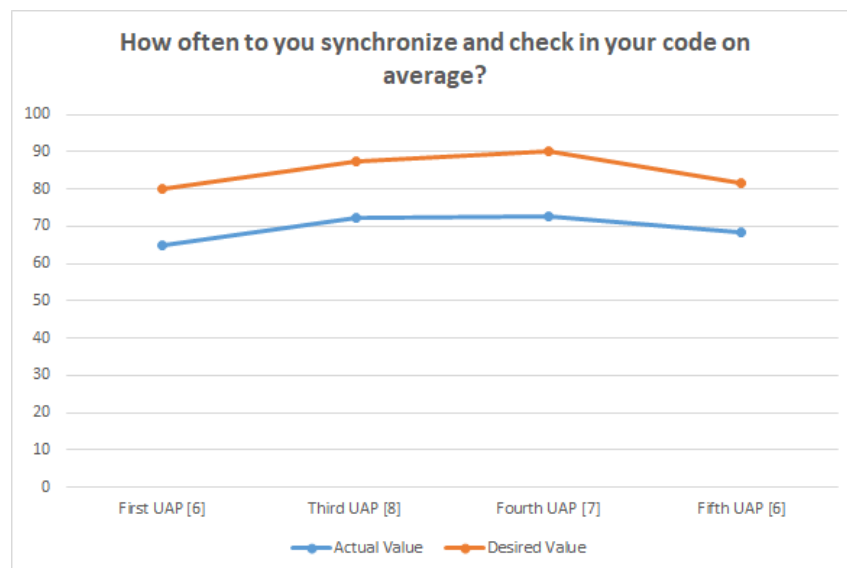


Figure C.2.: How often to you synchronize and check in your code on average?

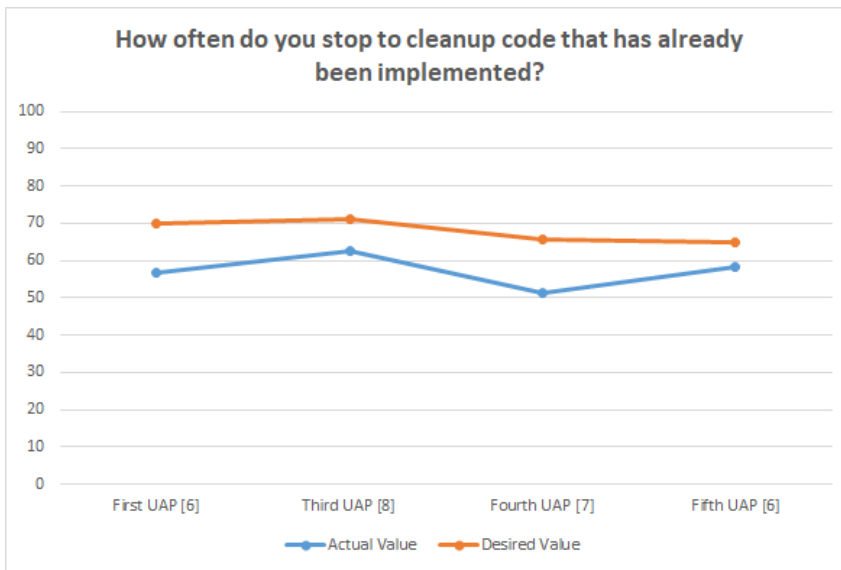


Figure C.3.: How often do you stop to cleanup code that has already been implemented?

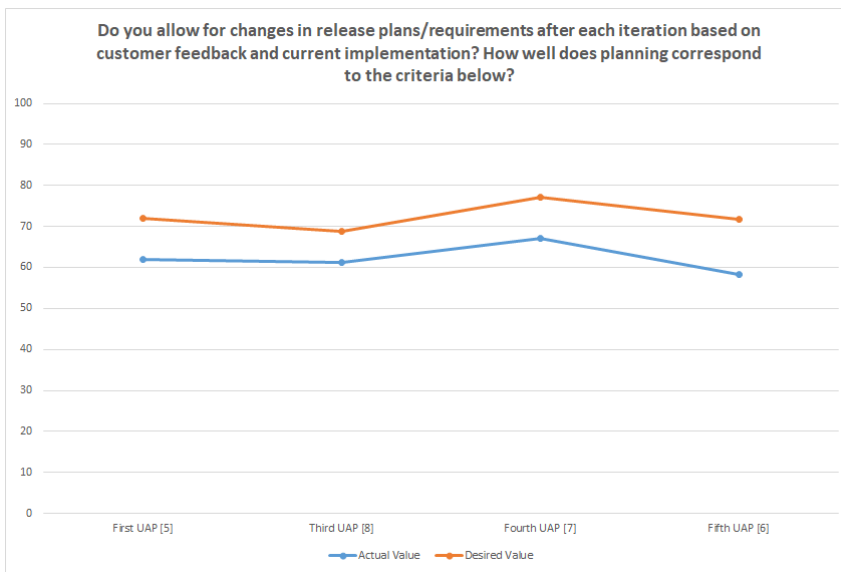


Figure C.4.: Do you allow for changes in release plans/requirements after each iteration based on customer feedback and current implementation? How well does planning correspond to the criteria below?

Appendix C. Supplementary Material

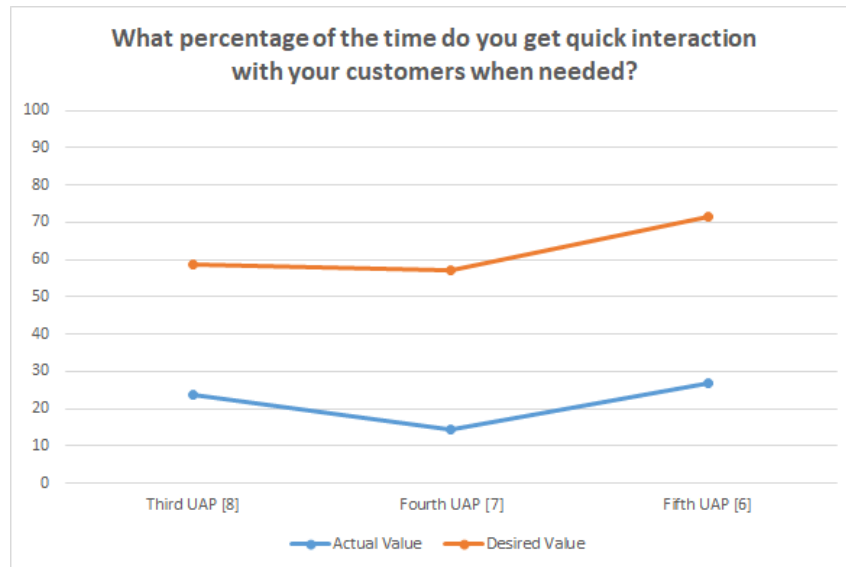


Figure C.5.: What percentage of the time do you get quick interaction with your customers when needed?

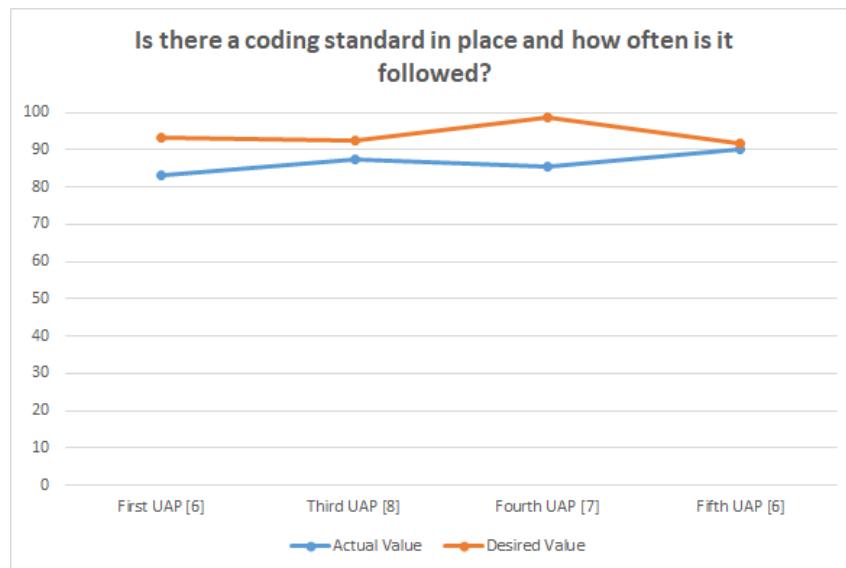


Figure C.6.: Is there a coding standard in place and how often is it followed?

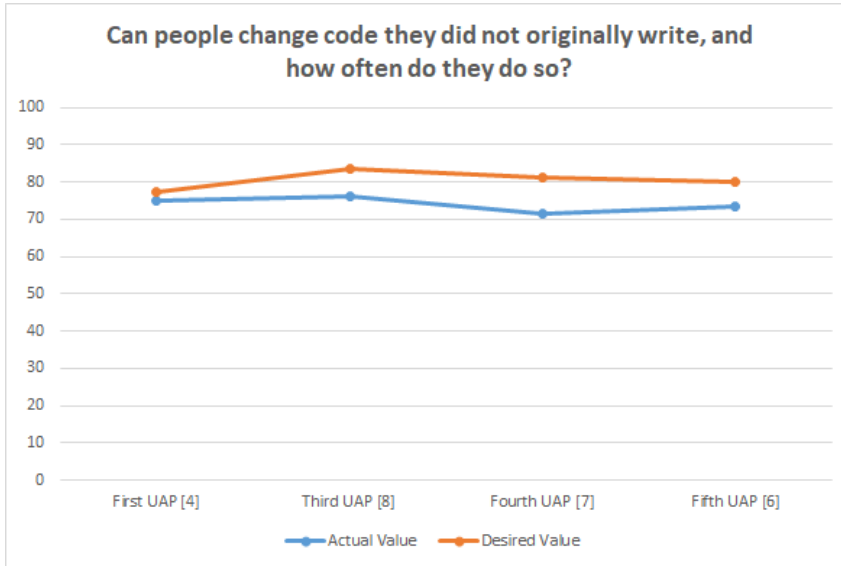


Figure C.7.: Can people change code they did not originally write, and how often do they do so?

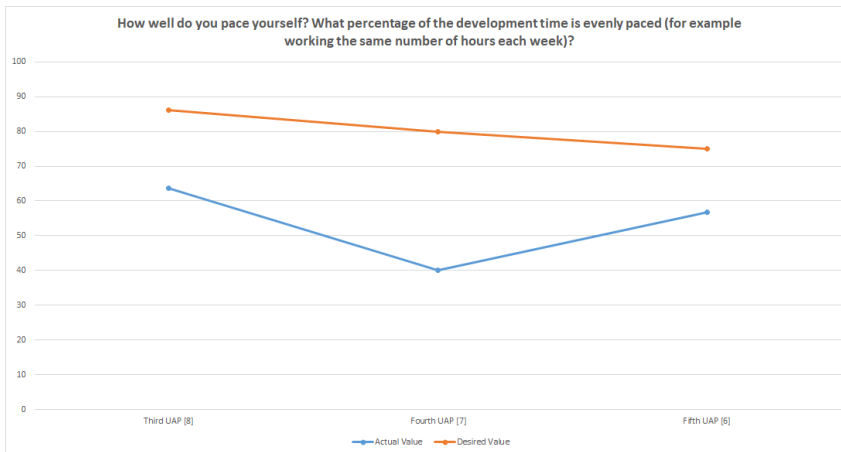


Figure C.8.: How well do you pace yourself? What percentage of the development time is evenly paced (for example working the same number of hours each week)?

Appendix C. Supplementary Material

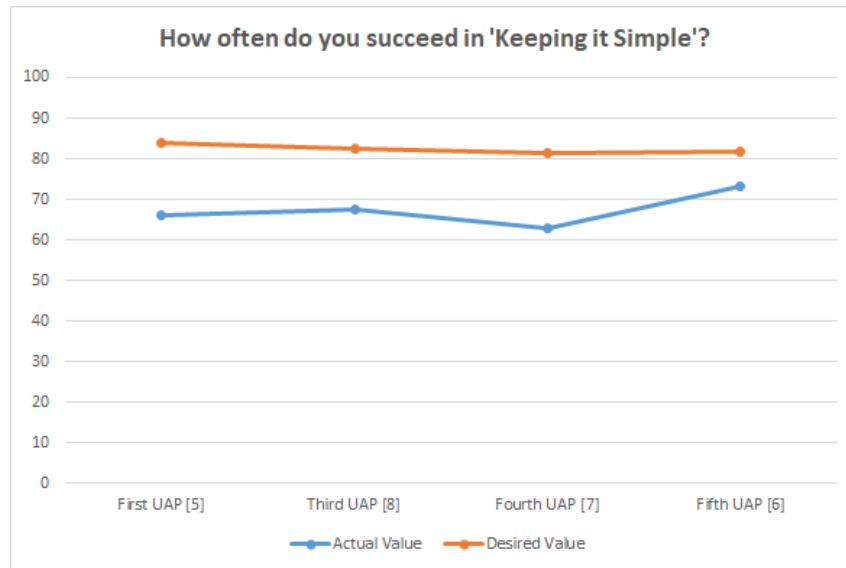


Figure C.9.: How often do you succeed in 'Keeping it Simple'?

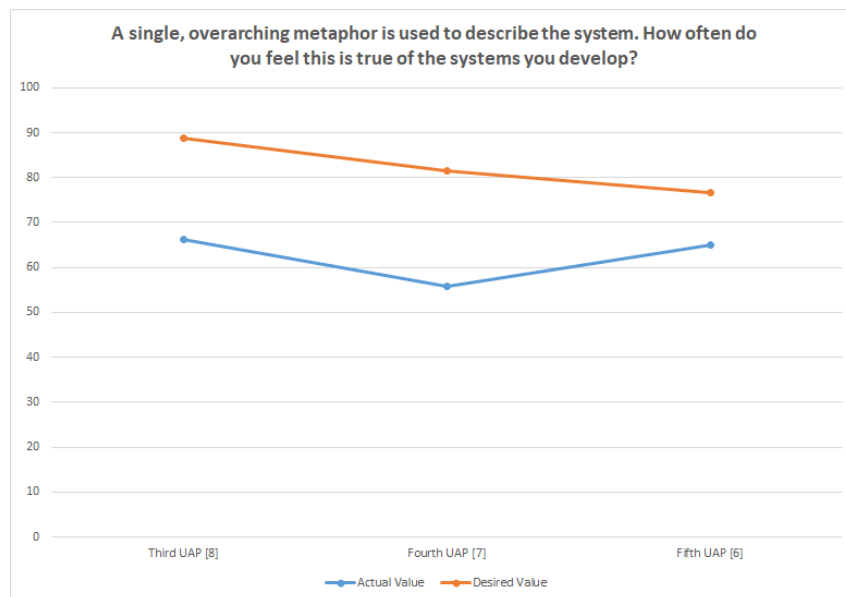


Figure C.10.: A single, overarching metaphor is used to describe the system. How often do you feel this is true of the systems you develop?

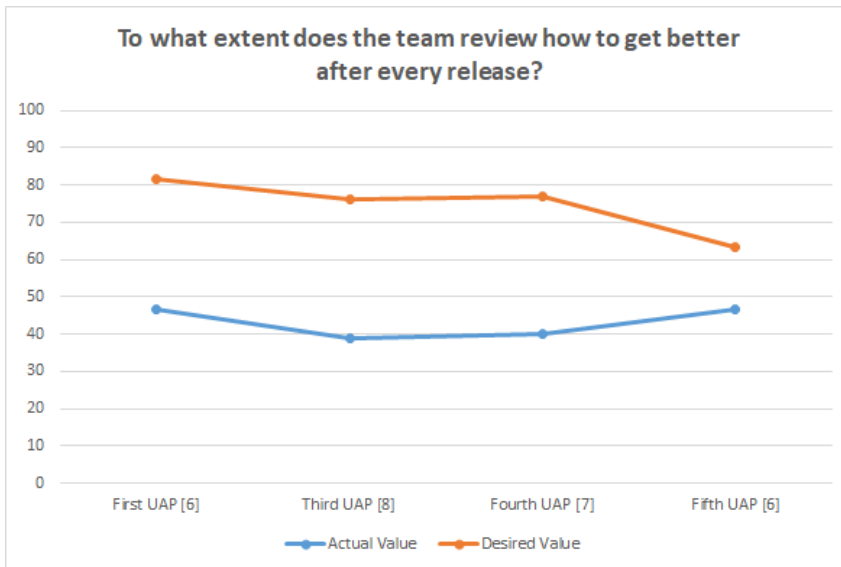


Figure C.11.: To what extent does the team review how to get better after every release?

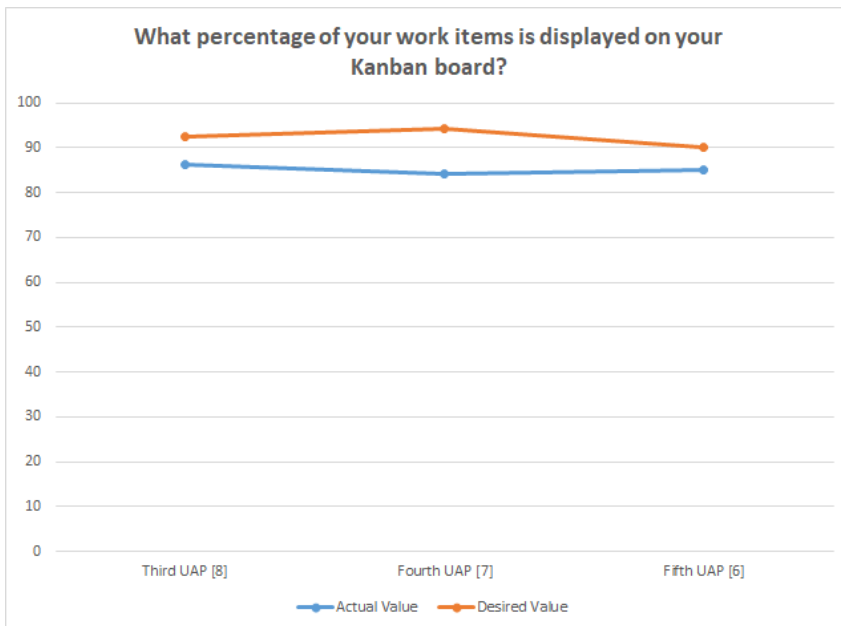


Figure C.12.: What percentage of your work items is displayed on your Kanban board?

Appendix C. Supplementary Material

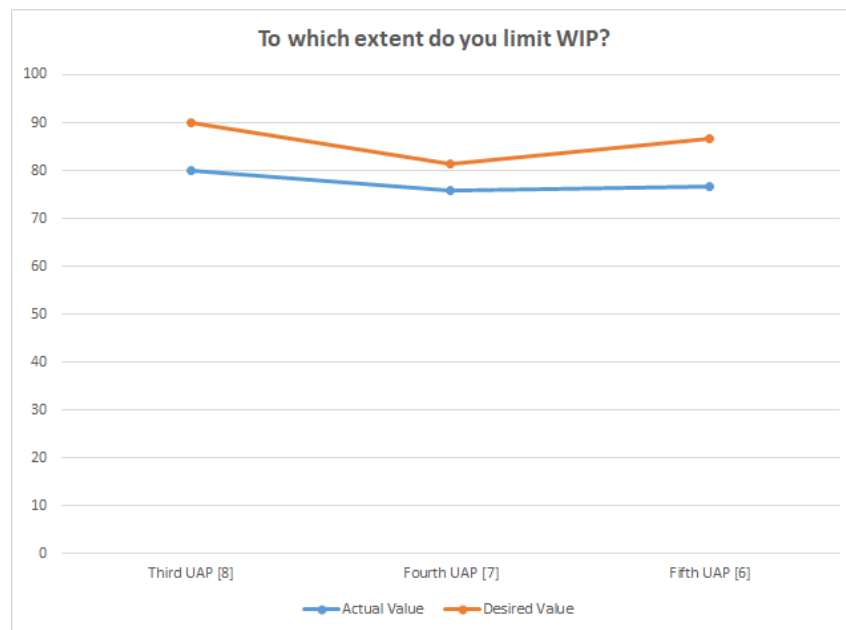


Figure C.13.: To which extent do you limit Work in Progress?



Figure C.14.: What percentage of your rules are explicit stated, so everybody knows them?

Bibliography

- Abrahamsson, Warsta, Siponen, and Ronkainen (2003). "New directions on agile methods: a comparative analysis." In: *2003 Proceedings. 25th International Conference on Software Engineering*. Ieee, pp. 244–254 (cit. on p. 9).
- Acuña, Castro, Dieste, and Juristo Juzgado (2012). "A systematic mapping study on the open source software development process." In: *16th International Conference on Evaluation & Assessment in Software Engineering, EASE 2012, Ciudad Real, Spain, May 14-15, 2012. Proceedings*. Ed. by Maria Teresa Baldassarre, Marcela Genero, Emilia Mendes, and Mario Piattini. IET - The Institute of Engineering and Technology / IEEE Xplore, pp. 42–46. ISBN: 978-1-84919-541-6. DOI: [10.1049/ic.2012.0005](https://doi.org/10.1049/ic.2012.0005). URL: <https://doi.org/10.1049/ic.2012.0005> (cit. on p. 36).
- Adams and Capiluppi (2009). "Bridging the Gap between Agile and Free Software Approaches: The Impact of Sprinting." In: *IJOSSP 1.1*, pp. 58–71. DOI: [10.4018/jossp.2009010104](https://doi.org/10.4018/jossp.2009010104). URL: <https://doi.org/10.4018/jossp.2009010104> (cit. on p. 81).
- Ågerfalk, Fitzgerald, and Slaughter (2009). "Introduction to the Special Issue - Flexible and Distributed Information Systems Development: State of the Art and Research Challenges." In: *Information Systems Research* 20.3, pp. 317–328. DOI: [10.1287/isre.1090.0244](https://doi.org/10.1287/isre.1090.0244). URL: <http://dx.doi.org/10.1287/isre.1090.0244> (cit. on p. 1).
- AgileModeling (2017). URL: <http://agilemodeling.com/> (cit. on p. 8).
- AgileUnifiedProcess (2006). URL: <http://www.ambyssoft.com/unifiedprocess/agileUP.html> (cit. on p. 8).
- Ahmad, Liukkunen, and Markkula (2014). *Student perceptions and attitudes towards the software factory as a learning environment*. Undetermined (cit. on pp. 1, 2, 71, 73, 75).

Bibliography

- Ahmad, Markkula, and Oivo (Jan. 2014). "Kanban for software engineering teaching in a software factory learning environment." In: 12, pp. 338–343 (cit. on pp. 73–75).
- Aladwani (2001). "Change management strategies for successful ERP implementation." In: *Business Process management journal* 7.3, pp. 266–275 (cit. on pp. 102, 103).
- Ambler and Lines (2012). *Disciplined Agile Delivery: A Practitioner's Guide to Agile Software Delivery in the Enterprise*. 1st ed. IBM Press (cit. on p. 8).
- Anderson (2010). *Kanban - Successful Evolutionary Change for Your Technology Business*. Seattle, USA: Blue Hole Press (cit. on pp. 2, 3, 9, 20, 21, 52, 92, 108).
- Andres and Beck (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley (cit. on pp. 8–19, 52, 62, 92).
- Auer and Miller (2002). *Extreme Programming Applied: Playing to Win*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0-201-61640-8 (cit. on p. 8).
- Basili, Shull, and Lanubile (1999). "Building Knowledge through Families of Experiments." In: *IEEE Trans. Software Eng.* 25.4, pp. 456–473. DOI: 10.1109/32.799939. URL: <https://doi.org/10.1109/32.799939> (cit. on p. 156).
- Beck (2003). *Test-Driven Development: By Example*. Addison-Wesley Professional (cit. on pp. 16, 62, 92).
- Beck and Andres (2004). *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional. ISBN: 0321278658 (cit. on pp. 8, 10, 11, 13, 19).
- Beck, Beedle, Van Bennekum, Cockburn, Cunningham, Fowler, Grenning, Highsmith, Hunt, Jeffries, et al. (2001). *The agile manifesto*. URL: <http://agilemanifesto.org/> (cit. on p. 9).
- Becking, Course, van Enk, Hangyi, Lahaye, Ockeloen, Peters, Rosbergen, and van Wendel de Joode (2005). "MMBase: An open-source content management system." In: *IBM Systems Journal* 44.2, pp. 381–398. DOI: 10.1147/sj.442.0381. URL: <https://doi.org/10.1147/sj.442.0381> (cit. on p. 91).
- Benson and DeMaria (2011). *Personal Kanban: Mapping Work, Navigating Life*. Seattle, USA: Modus Cooperandi Press (cit. on pp. 9, 20, 141).

- Boehm (2002). "Get Ready for Agile Methods, with Care." In: *IEEE Computer* 35.1, pp. 64–69. DOI: [10.1109/2.976920](https://doi.org/10.1109/2.976920). URL: <http://doi.ieeecomputersociety.org/10.1109/2.976920> (cit. on pp. 1, 6).
- Boehm and Turner (2003a). *Balancing Agility and Discipline: A Guide for the Perplexed*. 7th ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0321186125 (cit. on p. 52).
- Boehm and Turner (2003b). "Observations on Balancing Discipline and Agility." In: *2003 Agile Development Conference (ADC 2003), 25-28 June 2003, Salt Lake City, UT, USA*. IEEE Computer Society, pp. 32–39. ISBN: 0-7695-2013-8. DOI: [10.1109/ADC.2003.1231450](https://doi.org/10.1109/ADC.2003.1231450). URL: <https://doi.org/10.1109/ADC.2003.1231450> (cit. on p. 80).
- Boland and Fitzgerald (May 2004). "Transitioning from a co-located to a globally-distributed software development team: A case study at Analog Devices, Inc." In: *Proceedings of 3rd Workshop on Global Software Development*. ICSE '04. Edinburgh, UK, pp. 4–7. DOI: <http://dx.doi.org/10.1049/ic:20040303> (cit. on p. 1).
- Bonaccorsi and Rossi (Dec. 2006). "Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business." In: *Knowledge, Technology & Policy* 18.4, pp. 40–64. ISSN: 0897-1986. DOI: [10.1007/s12130-006-1003-9](http://dx.doi.org/10.1007/s12130-006-1003-9). URL: <http://dx.doi.org/10.1007/s12130-006-1003-9> (cit. on p. 1).
- Bowen and Stavridou (1993). "Safety-critical systems, formal methods and standards." In: *Software Engineering Journal* 8.4, pp. 189–209. DOI: [10.1049/sej.1993.0025](https://doi.org/10.1049/sej.1993.0025). URL: <https://doi.org/10.1049/sej.1993.0025> (cit. on p. 83).
- Bryant and Jones (2012). "Responsive Web Design." In: *Pro HTML5 Performance*. Berkeley, CA: Apress, pp. 37–49. ISBN: 978-1-4302-4525-4. DOI: [10.1007/978-1-4302-4525-4_4](http://dx.doi.org/10.1007/978-1-4302-4525-4_4). URL: http://dx.doi.org/10.1007/978-1-4302-4525-4_4 (cit. on p. 10).
- Casson and Hawthorn (2011). "Introducing the Oregon State University Open Source Lab." In: *Open Source Business Resource* August 2011 (cit. on p. 2).
- Chein, Cook, and Harding (1948). "The field of action research." In: *American Psychologist* 3.2, p. 43 (cit. on pp. 87, 88).
- Ciborra and Andreu (2001). "Sharing knowledge across boundaries." In: *JIT* 16.2, pp. 73–81. DOI: [10.1080/02683960110055103](https://doi.org/10.1080/02683960110055103). URL: <https://doi.org/10.1080/02683960110055103> (cit. on p. 91).

Bibliography

- Cockburn (2004). *Crystal Clear a Human-powered Methodology for Small Teams*. First. Addison-Wesley Professional. ISBN: 0201699478 (cit. on p. 8).
- Cohn (2005). *Agile Estimating and Planning*. Upper Saddle River, USA: Pearson Education. ISBN: 9780132703109 (cit. on pp. 119, 121).
- Corbin and Strauss (1998). *Basics of Qualitative Research: Grounded Theory Procedures and Techniques (2nd Edition)*. Thousand Oaks, USA: Sage publications (cit. on p. 109).
- Crisà, Del Bianco, and Lavazza (2006). "A tool for the measurement, storage, and pre-elaboration of data supporting the release of public datasets." In: *Workshop on Public Data about Software Development (WoPDaSD 2006)*, Como (cit. on p. 82).
- Crispin and Gregory (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. 1st ed. Addison-Wesley Professional (cit. on pp. 8, 12, 27).
- Crowston, Kevin, Imed Hammouda, Björn Lundell, Gregorio Robles, Jonas Gamalielsson, and Juho Lindman, eds. (2016). *Open Source Systems: Integrating Communities - 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30 - June 2, 2016, Proceedings*. Vol. 472. IFIP Advances in Information and Communication Technology. Springer. ISBN: 978-3-319-39224-0. DOI: 10.1007/978-3-319-39225-7. URL: <http://dx.doi.org/10.1007/978-3-319-39225-7>.
- Crowston and Howison (2005). "The social structure of free and open source software development." In: *First Monday* 10.2. URL: <http://firstmonday.org/ojs/index.php/fm/article/view/1207> (cit. on pp. 41, 42, 44, 54, 55).
- Crowston, Howison, Masango, and Eseryel (2005). "Face-to-face interactions in self-organizing distributed teams." In: *Academy of Management Conference, Honolulu, HI* (cit. on p. 43).
- Crowston, Li, Wei, Eseryel, and Howison (2007). "Self-organization of teams for free/libre open source software development." In: *Information and software technology* 49.6, pp. 564–575 (cit. on p. 43).
- Crowston and Scozzi (2002). "Open source software projects as virtual organisations: competency rallying for software development." In: *IEE Proceedings-Software* 149.1, pp. 3–17 (cit. on p. 44).
- Crowston and Scozzi (2004). "Coordination practices for bug fixing within FLOSS development teams." In: *1st International Workshop on Computer Supported Activity Coordination (CSAC)* (cit. on p. 41).

- Crowston and Scozzi (2008). "Coordination practices within free/libre open source software development teams: The bug fixing process." In: 19.2, pp. 1–30 (cit. on p. 43).
- Crowston, Wei, Howison, and Wiggins (2012). "Free/Libre open-source software development: What we know and what we do not know." In: *ACM Comput. Surv.* 44.2, p. 7. DOI: [10.1145/2089125.2089127](https://doi.org/10.1145/2089125.2089127). URL: <http://doi.acm.org/10.1145/2089125.2089127> (cit. on pp. 1, 40, 43, 91).
- David, Waterman, and Arora (2003). "FLOSS-US the free/libre/open source software survey for 2003." In: *Stanford Institute for Economic Policy Research, Stanford University, Stanford, CA* (<http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf>) (cit. on pp. 41, 52).
- Davoli, Renzo, Michael Goldweber, Guido Rößling, and Irene Polycarpou, eds. (2017). *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2017, Bologna, Italy, July 3-5, 2017*. ACM. ISBN: 978-1-4503-4704-4. DOI: [10.1145/3059009](https://doi.org/10.1145/3059009). URL: <http://doi.acm.org/10.1145/3059009>.
- DeKoenigsberg (2008). "How Successful Open Source Projects Work, and How and Why to Introduce Students to the Open Source World." In: *Proceedings 21st Conference on Software Engineering Education and Training, CSEET 2008, 14-17 April 2008, Charleston, South Carolina, USA*. Ed. by Hossein Saiedian and Laurie A. Williams. IEEE Computer Society, pp. 274–276. ISBN: 978-0-7695-3144-1. DOI: [10.1109/CSEET.2008.42](https://doi.org/10.1109/CSEET.2008.42). URL: <https://doi.org/10.1109/CSEET.2008.42> (cit. on p. 2).
- Deming (2000). *Out of the Crisis*. Massachusetts Institute of Technology, Center for Advanced Engineering Study. ISBN: 9780262541152. URL: <https://books.google.at/books?id=LA15eD10PgoC> (cit. on p. 20).
- Deshpande and Riehle (2008). "Continuous Integration in Open Source Software Development." In: *Open Source Development, Communities and Quality, IFIP 20th World Computer Congress, Working Group 2.3 on Open Source Software, OSS 2008, September 7-10, 2008, Milano, Italy*. Ed. by Barbara Russo, Ernesto Damiani, Scott A. Hissam, Björn Lundell, and Giancarlo Succi. Vol. 275. IFIP. Cham, Switzerland: Springer, pp. 273–280. ISBN: 978-0-387-09683-4 (cit. on pp. 1, 81).
- Dick (2000). *A beginner's guide to action research*. URL: <http://www.aral.com.au/resources/guide.html> (cit. on p. 89).

Bibliography

- Dingsøy, Torgeir, Sridhar P. Nerur, Venugopal Balijepally, and Nils Brede Moe (2012). "A decade of agile methodologies: Towards explaining agile software development." In: *Journal of Systems and Software* 85.6, pp. 1213–1221. DOI: [10.1016/j.jss.2012.02.033](https://doi.org/10.1016/j.jss.2012.02.033). URL: <http://dx.doi.org/10.1016/j.jss.2012.02.033> (cit. on p. 1).
- Dinh-Trong and Bieman (2005). "The FreeBSD Project: A Replication Case Study of Open Source Development." In: *IEEE Trans. Software Eng.* 31.6, pp. 481–494. DOI: [10.1109/TSE.2005.73](https://doi.org/10.1109/TSE.2005.73). URL: <https://doi.org/10.1109/TSE.2005.73> (cit. on p. 41).
- DSDMConsortium (2008). *DSDM Atern: the Handbook* (cit. on p. 8).
- Düring (2006a). "Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project (PyPy)." In: *Extreme Programming and Agile Processes in Software Engineering, 7th International Conference, XP 2006, Oulu, Finland, June 17-22, 2006, Proceedings*. Ed. by Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi. Vol. 4044. Lecture Notes in Computer Science. Berlin Heidelberg, Germany: Springer-Verlag, pp. 191–195. ISBN: 3-540-35094-2 (cit. on pp. 1, 65).
- Düring (2006b). "Sprint Driven Development: Agile Methodologies in a Distributed Open Source Project (PyPy)." English. In: *Extreme Programming and Agile Processes in Software Engineering*. Ed. by Pekka Abrahamsson, Michele Marchesi, and Giancarlo Succi. Vol. 4044. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 191–195. ISBN: 978-3-540-35094-1. DOI: [10.1007/11774129_22](https://doi.org/10.1007/11774129_22). URL: http://dx.doi.org/10.1007/11774129_22 (cit. on p. 80).
- Düring (July 2006c). "Trouble in paradise: the open source project PyPy, EU-funding and agile practices." In: *Agile Conference, 2006*. DOI: [10.1109/AGILE.2006.58](https://doi.org/10.1109/AGILE.2006.58) (cit. on p. 80).
- Edmondson and McManus (2007). "Methodological Fit in Management Field Research." In: *Academy of Management Review* 32, no. 4 (cit. on p. 89).
- Edwards (2001). "Epistemic communities, situated learning and open source software development." In: *Epistemic Cultures and the Practice of Interdisciplinarity* (cit. on p. 91).
- Elferink, Griffiths, and Zondergeld (2016). "Comparing Software Development Models: Structural Problems in the Cathedral and Bazaar metaphors." In: *Open Source for Education in Europe, Research & Practise Conference*

- Proceedings*. Open University of the Netherlands, Heerlen, Educational Technology Expertise Centre (cit. on p. 45).
- Elliott and Scacchi (2005). "Free software development: cooperation and conflict in a virtual organization." In: *Free/open source software development* 152 (cit. on pp. 43, 44, 160).
- Ellis and Hislop (2016). "Pathways to Student Learning within HFOSS." In: *Proceedings of the 17th Annual Conference on Information Technology Education and the 5th Annual Conference on Research in Information Technology, SIGITE/RIIT 2016, Boston, MA, USA, September 28 - October 1, 2016*. Ed. by Deborah Boisvert and Stephen J. Zilora. ACM, p. 168. ISBN: 978-1-4503-4452-4. DOI: [10.1145/2978192.2978242](https://doi.org/10.1145/2978192.2978242). URL: <http://doi.acm.org/10.1145/2978192.2978242> (cit. on p. 78).
- Ellis and Hislop (2017). "A Course Based on Open Organization Principles." In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2017, Bologna, Italy, July 3-5, 2017*. Ed. by Renzo Davoli, Michael Goldweber, Guido Rößling, and Irene Polycarpou. ACM, p. 378. ISBN: 978-1-4503-4704-4. DOI: [10.1145/3059009.3072998](https://doi.org/10.1145/3059009.3072998). URL: <http://doi.acm.org/10.1145/3059009.3072998> (cit. on p. 78).
- Ellis, Hislop, and Burdge (2017). "Courseware: HFOSS Project Evaluation." In: *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education, ITiCSE 2017, Bologna, Italy, July 3-5, 2017*. Ed. by Renzo Davoli, Michael Goldweber, Guido Rößling, and Irene Polycarpou. ACM, pp. 90–91. ISBN: 978-1-4503-4704-4. DOI: [10.1145/3059009.3072975](https://doi.org/10.1145/3059009.3072975). URL: <http://doi.acm.org/10.1145/3059009.3072975> (cit. on p. 78).
- Ellis, Hislop, and Purcell (2013). "Project selection for student involvement in humanitarian FOSS." In: *26th International Conference on Software Engineering Education and Training, CSEE&T 2013, San Francisco, CA, USA, May 19-21, 2013*. Ed. by Tony Cowling, Shawn Bohner, and Mark A. Ardis. IEEE, pp. 359–361. DOI: [10.1109/CSEET.2013.6595279](https://doi.org/10.1109/CSEET.2013.6595279). URL: <https://doi.org/10.1109/CSEET.2013.6595279> (cit. on p. 78).
- Ellis, Morelli, de Lanerolle, and Hislop (2007). "Holistic Software Engineering Education Based on a Humanitarian Open Source Project." In: *20th Conference on Software Engineering Education and Training (CSEE&T 2007), 3-5 July 2007, Dublin, Ireland*. IEEE Computer Society, pp. 327–

Bibliography

335. ISBN: 0-7695-2893-7. DOI: [10.1109/CSEET.2007.26](https://doi.org/10.1109/CSEET.2007.26). URL: <https://doi.org/10.1109/CSEET.2007.26> (cit. on p. 78).
- Ellis, Purcell, and Hislop (2012). "An approach for evaluating FOSS projects for student participation." In: *Proceedings of the 43rd ACM technical symposium on Computer science education, SIGCSE 2012, Raleigh, NC, USA, February 29 - March 3, 2012*. Ed. by Laurie A. Smith King, David R. Musicant, Tracy Camp, and Paul T. Tymann. ACM, pp. 415–420. ISBN: 978-1-4503-1098-7. DOI: [10.1145/2157136.2157260](https://doi.org/10.1145/2157136.2157260). URL: <http://doi.acm.org/10.1145/2157136.2157260> (cit. on p. 78).
- Espinosa, Kraut, Slaughter, Lerch, Herbsleb, and Mockus (2002). "Shared mental models, familiarity, and coordination: A multi-method study of distributed software teams." In: *ICIS 2002 Proceedings*, p. 39 (cit. on p. 44).
- Fagerholm, Oza, and Münch (2013). "A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory." In: *3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development, CTGDSD 2013, San Francisco, CA, USA, May 25, 2013*. IEEE Computer Society, pp. 1–5. ISBN: 978-1-4673-6294-8. DOI: [10.1109/CTGDSD.2013.6635237](https://doi.org/10.1109/CTGDSD.2013.6635237). URL: <https://doi.org/10.1109/CTGDSD.2013.6635237> (cit. on p. 73).
- Feller and Fitzgerald (2000). "A framework analysis of the open source software development paradigm." In: *Proceedings of the Twenty-First International Conference on Information Systems, ICIS 2000, Brisbane, Australia, December 10-13, 2000*. Ed. by Soon Ang, Helmut Krcmar, Wanda J. Orlikowski, Peter Weill, and Janice I. DeGross. Association for Information Systems, pp. 58–69. URL: <http://aisel.aisnet.org/icis2000/7> (cit. on p. 39).
- Fellhofer, Harzl, and Slany (2015). "Scaling and Internationalizing an Agile FOSS Project: Lessons Learned." In: *Open Source Systems: Adoption and Impact - 11th IFIP WG 2.13 International Conference, OSS 2015, Florence, Italy, May 16-17, 2015, Proceedings*. Ed. by Damiani, Frati, Riehle, and Wasserman. Vol. 451. IFIP Advances in Information and Communication Technology. Springer, pp. 13–22. ISBN: 978-3-319-17836-3. DOI: [10.1007/978-3-319-17837-0_2](https://doi.org/10.1007/978-3-319-17837-0_2). URL: http://dx.doi.org/10.1007/978-3-319-17837-0_2 (cit. on pp. 91, 164).

- Fielding (1999). "Shared Leadership in the Apache Project." In: *Commun. ACM* 42.4, pp. 42–43. DOI: [10.1145/299157.299167](https://doi.org/10.1145/299157.299167). URL: <http://doi.acm.org/10.1145/299157.299167> (cit. on p. 40).
- Fink (2003). *The business and economics of Linux and open source*. Prentice Hall Professional (cit. on p. 33).
- Fitzgerald (2006). "The Transformation of Open Source Software." In: *MIS Quarterly* 30.3, pp. 587–598. URL: <http://misq.org/the-transformation-of-open-source-software.html> (cit. on p. 40).
- Fitzgerald (2011). "Open Source Software: Lessons from and for Software Engineering." In: *IEEE Computer* 44.10, pp. 25–30. DOI: [10.1109/MC.2011.266](https://doi.org/10.1109/MC.2011.266). URL: <https://doi.org/10.1109/MC.2011.266> (cit. on pp. 29, 39–41, 43–45).
- Fitzgerald, Musial, and Stol (2014). "Evidence-based decision making in lean software project management." In: *36th International Conference on Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31 - June 07, 2014*. Ed. by Pankaj Jalote, Lionel C. Briand, and André van der Hoek. ACM, pp. 93–102. ISBN: 978-1-4503-2768-8. DOI: [10.1145/2591062.2591190](https://doi.org/10.1145/2591062.2591190). URL: <http://doi.acm.org/10.1145/2591062.2591190> (cit. on p. 3).
- Flyvbjerg (2006). "Five Misunderstandings About Case-Study Research." In: *Qualitative Inquiry* 12.2, pp. 219–245. DOI: [10.1177/1077800405284363](https://doi.org/10.1177/1077800405284363). eprint: <http://dx.doi.org/10.1177/1077800405284363>. URL: <http://dx.doi.org/10.1177/1077800405284363> (cit. on p. 156).
- Fogel (2005). *Producing open source software - how to run a successful free software project*. O'Reilly. ISBN: 978-0-596-00759-1. URL: <http://www.oreilly.de/catalog/producingoss/index.html> (cit. on p. 44).
- FSFE (2017a). *We speak about Free Software*. English. URL: <https://fsfe.org/documents/whyfs.en.html> (cit. on pp. 31, 32, 35).
- FSFE (2017b). *What is Free Software?* English. URL: <https://fsfe.org/about/basics/freesoftware.en.html> (cit. on pp. 32, 35).
- Fucci and Turhan (2013). "A Replicated Experiment on the Effectiveness of Test-First Development." In: *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Baltimore, Maryland, USA, October 10-11, 2013*. IEEE Computer Society, pp. 103–112. ISBN: 978-0-7695-5056-5. DOI: [10.1109/ESEM.2013.15](https://doi.org/10.1109/ESEM.2013.15). URL: <http://dx.doi.org/10.1109/ESEM.2013.15> (cit. on p. 1).

Bibliography

- Gacek and Arief (Jan. 2004). "The many meanings of open source." In: *IEEE Software* 21. "We determined a set of characteristics that are almost always present and others that vary among open source projects, and this serves as the core of this work" "Section 3 describes some open source characteristics that can be used in determining whether a project is or not open source", pp. 34–40. ISSN: 0740-7459. DOI: [10.1109/MS.2004.1259206](https://doi.org/10.1109/MS.2004.1259206) (cit. on pp. 38, 43).
- Gandomani, Taghi Javdani, Hazura Zulzalil, Abdul Azim Abdul Ghani, and Abu Bakar Md Sultan (2013). "A Systematic Literature Review on relationship between agile methods and Open Source Software Development methodology." In: *CoRR abs/1302.2748*, pp. 1602–1607. URL: <http://arxiv.org/abs/1302.2748> (cit. on pp. 1, 79, 84, 159).
- Gary, Enquobahrie, Ibáñez, Cheng, Yaniv, Cleary, Kokoori, Muffih, and Heidenreich (2011). "Agile methods for open source safety-critical software." In: *Softw., Pract. Exper.* 41.9, pp. 945–962. DOI: [10.1002/spe.1075](https://doi.org/10.1002/spe.1075). URL: <https://doi.org/10.1002/spe.1075> (cit. on p. 83).
- Gary, Ibáñez, Aylward, Gobbi, Blake, and Cleary (2006). "IGSTK: An Open Source Software Toolkit for Image-Guided Surgery." In: *IEEE Computer* 39.4, pp. 46–53. DOI: [10.1109/MC.2006.130](https://doi.org/10.1109/MC.2006.130). URL: <https://doi.org/10.1109/MC.2006.130> (cit. on p. 83).
- Gaughan, Fitzgerald, and Shaikh (2009). "An Examination of the Use of Open Source Software Processes as a Global Software Development Solution for Commercial Software Engineering." In: *35th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2009, Patras, Greece, August 27-29, 2009, Proceedings*. IEEE Computer Society, pp. 20–27. ISBN: 978-0-7695-3784-9. DOI: [10.1109/SEAA.2009.86](https://doi.org/10.1109/SEAA.2009.86). URL: <https://doi.org/10.1109/SEAA.2009.86> (cit. on p. 43).
- Gehring (2011). "From the manager's perspective: Classroom contributions to open-source projects." In: *2011 Frontiers in Education Conference, FIE 2011, Rapid City, SD, USA, October 12-15, 2011*. IEEE, p. 1. ISBN: 978-1-61284-468-8. DOI: [10.1109/FIE.2011.6143028](https://doi.org/10.1109/FIE.2011.6143028). URL: <https://doi.org/10.1109/FIE.2011.6143028> (cit. on p. 76).
- Germán (2003). "The GNOME project: a case study of open source, global software development." In: *Software Process: Improvement and Practice* 8.4, pp. 201–215. DOI: [10.1002/spip.189](https://doi.org/10.1002/spip.189). URL: <https://doi.org/10.1002/spip.189> (cit. on p. 40).

- Ghosh (1998). "FM Interviews: Interview with Linus Torvalds: What motivates software developers." In: *First Monday* 3.3 (cit. on p. 39).
- Ghosh (2005). "Understanding free software developers: Findings from the FLOSS study." In: *Perspectives on free and open source software*, pp. 23–46 (cit. on p. 41).
- GNU and FSF (2017). *Categories of free and nonfree software*. URL: <https://www.gnu.org/philosophy/categories.html.en> (cit. on p. 34).
- Goldratt and Cox (1992). *The Goal: A Process of Ongoing Improvement*. North River Press. ISBN: 9780884270614. URL: <https://books.google.at/books?id=0T5EAAAAMAAJ> (cit. on p. 20).
- Goth (2007). "Sprinting toward Open Source Development." In: *IEEE Software* 24.1, pp. 88–91. DOI: 10.1109/MS.2007.28. URL: <https://doi.org/10.1109/MS.2007.28> (cit. on p. 81).
- Greenwood and Levin (2007). *Introduction to Action Research: Social Research for Social Change*. SAGE Publications (cit. on p. 85).
- Hann, Roberts, and Slaughter (2004). "Why developers participate in open source software projects: An empirical investigation." In: *ICIS 2004 Proceedings*, p. 66 (cit. on p. 39).
- Hann, Roberts, Slaughter, and Fielding (2002). "Economic incentives for participating in open source software projects." In: *ICIS 2002 Proceedings*, p. 33 (cit. on p. 39).
- Hars and Ou (2001). "Working for Free? - Motivations of Participating in Open Source Projects." In: *34th Annual Hawaii International Conference on System Sciences (HICSS-34), January 3-6, 2001, Maui, Hawaii, USA*. IEEE Computer Society. ISBN: 0-7695-0981-9. DOI: 10.1109/HICSS.2001.927045. URL: <http://dx.doi.org/10.1109/HICSS.2001.927045> (cit. on pp. 39, 43, 45).
- Harzl (2015). "Combining Kanban and FOSS: Can it work?" In: *Agile Processes, in Software Engineering, and Extreme Programming*. , pp. 352–353 (cit. on p. 164).
- Harzl (2016). "Combining FOSS and Kanban: An Action Research." In: *Open Source Systems: Integrating Communities - 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30 - June 2, 2016, Proceedings*. Ed. by Kevin Crowston, Imed Hammouda, Björn Lundell, Gregorio Robles, Jonas Gamalielsson, and Juho Lindman. Vol. 472. IFIP Advances in Information and Communication Technology. Springer, pp. 71–84. ISBN: 978-3-319-39224-0. DOI: 10.1007/978-3-319-39225-

Bibliography

- 7_6. URL: http://dx.doi.org/10.1007/978-3-319-39225-7_6 (cit. on pp. 65, 84, 85, 88, 107, 117, 153, 159, 164).
- Harzl (2017). "Can FOSS projects benefit from integrating Kanban: a case study." In: *Journal of Internet Services and Applications* 8.1, p. 7. ISSN: 1869-0238. DOI: 10.1186/s13174-017-0058-z. URL: <http://dx.doi.org/10.1186/s13174-017-0058-z> (cit. on pp. 65, 84, 88, 107, 153, 159, 164).
- Harzl, Krnjic, Schreiner, and Slany (2013a). "Comparing Purely Visual with Hybrid Visual/Textual Manipulation of Complex Formula on Smartphones." In: *Proceedings of the 19th International Conference on Distributed Multimedia Systems, DMS 2013, August 8-10, 2013, Holiday Inn, Brighton, UK*. Knowledge Systems Institute, pp. 198–201. ISBN: 1-891706-34-9 (cit. on pp. 47, 163).
- Harzl, Krnjic, Schreiner, and Slany (2013b). "Purely Visual and Hybrid Visual/Textual Formula Composition: A Usability Study Plan." In: *Proceedings of Programming for Mobile and Touch PProMoTo 2013* (cit. on p. 164).
- Harzl, Neidhoefer, Rock, Schafzahl, and Slany (2013). "A Scratch-like visual programming system for Microsoft Windows Phone 8." In: *CoRR* abs/1310.1390. URL: <http://arxiv.org/abs/1310.1390> (cit. on pp. 47, 163).
- Herr and Anderson (2015). *The Action Research Dissertation - A Guide for Students and Faculty 2nd Edition*. Thousand Oaks, USA: SAGE Publications (cit. on p. 155).
- Hertel, Niedner, and Herrmann (2003). "Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel." In: *Research policy* 32.7, pp. 1159–1177 (cit. on p. 39).
- Highsmith (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. New York, NY, USA: Dorset House Publishing Co., Inc. ISBN: 0-932633-40-4 (cit. on p. 8).
- Highsmith, Jim and Alistair Cockburn (2001). "Agile Software Development: The Business of Innovation." In: *IEEE Computer* 34.9, pp. 120–122. DOI: 10.1109/2.947100. URL: <https://doi.org/10.1109/2.947100> (cit. on p. 9).
- Hippel (July 2001). "Innovation by User Communities: Learning from Open-Source Software." In: *MIT Sloan management review* 42.4, p. 82 (cit. on p. 44).

- Hippel and Krogh (2003). "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science." In: *Organization Science* 14.2, pp. 209–223. DOI: [10.1287/orsc.14.2.209.14992](https://doi.org/10.1287/orsc.14.2.209.14992). URL: <http://dx.doi.org/10.1287/orsc.14.2.209.14992> (cit. on p. 44).
- Hiranabe (2008). *Kanban Applied to Software Development: from Agile to Lean*. [Online; accessed 15-December-2014]. URL: <http://www.infoq.com/articles/hiranabe-lean-agile-kanban> (cit. on p. 92).
- Hislop and Ellis (2015). "Practical Experiences for IT Students in Humanitarian Free and Open Source Software Projects." In: *Proceedings of the 16th Annual Conference on Information Technology Education, SIGITE 2015, Chicago, Illinois, USA, September 30 - October 3, 2015*. Ed. by Amber Settle, Terry Steinbach, and Deborah Boisvert. ACM, p. 99. ISBN: 978-1-4503-3835-6. DOI: [10.1145/2808006.2808042](https://doi.org/10.1145/2808006.2808042). URL: <http://doi.acm.org/10.1145/2808006.2808042> (cit. on p. 78).
- Hopp and Spearman (2000). *Factory Physics: Foundations of Manufacturing Management*. Second. Irwin/McGraw Hill (cit. on pp. 22, 23).
- Höst, Regnell, and Wohlin (2000). "Using Students as Subjects-A Comparative Study of Students and Professionals in Lead-Time Impact Assessment." In: *Empirical Software Engineering* 5.3, pp. 201–214 (cit. on p. 157).
- Howison (2009). *Alone Together: A socio-technical theory of motivation, coordination and collaboration technologies in organizing for free and open source software development*. Syracuse University (cit. on p. 40).
- Hussain, Lechner, Milchrahm, Shahzad, Slany, Umgeher, Vlk, Koeffel, Tschelligi, and Wolkerstorfer (2012). "Practical Usability in XP Software Development Processes." In: *ACHI 2012, The Fifth International Conference on Advances in Computer-Human Interactions*, pp. 208–217 (cit. on p. 92).
- Jeffries, Anderson, and Hendrickson (2000). *Extreme Programming Installed*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN: 0201708426 (cit. on p. 8).
- Jensen and Scacchi (2005). "Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community." In: *38th Hawaii International Conference on System Sciences (HICSS-38 2005), CD-ROM / Abstracts Proceedings, 3-6 January 2005, Big Island, HI, USA*. IEEE Computer Society. ISBN: 0-7695-2268-8. DOI: [10.1109/HICSS.2005.147](https://doi.org/10.1109/HICSS.2005.147). URL: <https://doi.org/10.1109/HICSS.2005.147> (cit. on p. 40).

Bibliography

- Jørgensen (2001). "Putting it all in the trunk: incremental software development in the FreeBSD open source project." In: *Inf. Syst. J.* 11.4, p. 321. DOI: [10.1046/j.1365-2575.2001.00113.x](https://doi.org/10.1046/j.1365-2575.2001.00113.x). URL: <https://doi.org/10.1046/j.1365-2575.2001.00113.x> (cit. on pp. 44, 45).
- Kagdi, Hammad, and Maletic (2008). "Who can help me with this source code change?" In: *24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28 - October 4, 2008, Beijing, China*. IEEE Computer Society, pp. 157–166. ISBN: 978-1-4244-2613-3. DOI: [10.1109/ICSM.2008.4658064](https://doi.org/10.1109/ICSM.2008.4658064). URL: <http://dx.doi.org/10.1109/ICSM.2008.4658064> (cit. on p. 56).
- Kampenes, Anda, and Dybå (2008). "Flexibility in Research Designs in Empirical Software Engineering." In: *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008, University of Bari, Italy, 26-27 June 2008*. Ed. by Giuseppe Visaggio, Maria Teresa Baldassarre, Stephen G. Linkman, and Mark Turner. Workshops in Computing. BCS. URL: <http://ewic.bcs.org/category/16334> (cit. on p. 89).
- Kelty (2008). *Two bits: The cultural significance of free software*. Duke University Press (cit. on p. 30).
- Kemmis, McTaggart, and Nixon (2014). *The Action Research Planner - Doing Critical Participatory Action Research*. Springer Singapore. DOI: [10.1007/978-981-4560-67-2](https://doi.org/10.1007/978-981-4560-67-2) (cit. on pp. 87–89).
- Kniberg (2011). *Lean from the Trenches: Managing Large-Scale Projects with Kanban*. Pragmatic Bookshelf (cit. on p. 9).
- Kniberg and Skarin (2010). *Kanban and Scrum - making the most of both*. USA: C4Media (cit. on pp. 3, 9, 71).
- Knuth (1979). "Classics in Software Engineering." In: ed. by Edward Nash Yourdon. Upper Saddle River, NJ, USA: Yourdon Press. Chap. Structured Programming with Go to Statements, pp. 257–321. ISBN: 0-917072-14-6. URL: <http://dl.acm.org/citation.cfm?id=1241515.1241535> (cit. on pp. 14, 15).
- Koch (2004). "Agile Principles and Open Source Software Development: A Theoretical and Empirical Discussion." In: *Extreme Programming and Agile Processes in Software Engineering, 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004, Proceedings*. Springer-Verlag, pp. 85–93. DOI: [10.1007/978-3-540-24853-8_10](https://doi.org/10.1007/978-3-540-24853-8_10). URL: <http://>

- [//dx.doi.org/10.1007/978-3-540-24853-8_10](https://dx.doi.org/10.1007/978-3-540-24853-8_10) (cit. on pp. 1, 41, 78, 156).
- Koch and Schneider (2002). "Effort, co-operation and co-ordination in an open source software project: GNOME." In: *Information Systems Journal* 12.1, pp. 27–42 (cit. on pp. 41, 78).
- Korkala and Abrahamsson (2007). "Communication in distributed agile development: A case study." In: *33rd EUROMICRO Conference on Software Engineering and Advanced Applications, EUROMICRO-SEAA 2007*. IEEE, pp. 203–210 (cit. on p. 94).
- Kotter and Schlesinger (1979). *Choosing strategies for change*. Harvard Business Review (cit. on pp. 102, 103).
- Krishna and Srinivasa (2011). "Analysis of projects and volunteer participation in large scale free and open source software ecosystem." In: *ACM SIGSOFT Software Engineering Notes* 36.2, pp. 1–5. DOI: [10.1145/1943371.1943389](https://doi.org/10.1145/1943371.1943389). URL: <http://doi.acm.org/10.1145/1943371.1943389> (cit. on p. 41).
- Krishnamurthy (2002). "Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects." In: *First Mon*. URL: <http://firstmonday.org/ojs/index.php/fm/article/view/1477/1392> (cit. on p. 41).
- Kruchten (2004). *The rational unified process: an introduction*. Addison-Wesley Professional (cit. on p. 9).
- Ladas (2008). *Srumban Essays on Kanban Systems for Lean Software Developm*. Modus Cooperandi Press (cit. on p. 9).
- Lakhani and Von Hippel (2003). "How open source software works: 'free' user-to-user assistance." In: *Research policy* 32.6, pp. 923–943 (cit. on p. 39).
- Lakhani and Wolf (2003). "Why hackers do what they do: Understanding motivation and effort in free/open source software projects." In: (cit. on p. 45).
- Lakhani, Wolf, Bates, and DiBona (July 2002). *The Boston Consulting Group Hacker Survey*. URL: <http://ftp3.au.freebsd.org/pub/linux.conf.au/2003/papers/Hemos/Hemos.pdf> (cit. on p. 39).
- Lavazza, Morasca, Taibi, and Tosi (2010). "Applying SCRUM in an OSS Development Process: An Empirical Evaluation." In: *Agile Processes in Software Engineering and Extreme Programming, 11th International Conference, XP 2010, Trondheim, Norway, June 1-4, 2010. Proceedings*, pp. 147–

Bibliography

159. DOI: [10.1007/978-3-642-13054-0_11](https://doi.org/10.1007/978-3-642-13054-0_11). URL: https://doi.org/10.1007/978-3-642-13054-0_11 (cit. on pp. 82, 83).
- Layman, Williams, Damian, and Bures (2006). "Essential communication practices for Extreme Programming in a global software development team." In: *Information and software technology* 48.9, pp. 781–794 (cit. on pp. 94, 96, 100).
- Leopold (Apr. 2012). *Kanban im Schnelldurchlauf*. URL: <https://youtu.be/6n0Ua6E0250> (cit. on pp. 22, 24–27, 109).
- Leopold (Apr. 2017). *Flight Levels: Die Verbesserungsebenen der Organisation*. German. URL: <https://www.leanability.com/de/blog-de/2017/04/flight-levels-die-verbesserungsebenen-der-organisation/> (cit. on p. 21).
- Leopold and Kaltenecker (2013). *Kanban in der IT - Eine Kultur der kontinuierlichen Verbesserung schaffen*. Hanser (cit. on pp. 3, 9, 21, 22, 24–27, 108, 112–114).
- Lerner and Tirole (2002). "Some Simple Economics of Open Source." In: *Journal of Industrial Economics* 50, pp. 197–234. URL: <http://dx.doi.org/10.1111/1467-6451.00174> (cit. on pp. 2, 39).
- Lerner and Tirole (2005). "The scope of open source licensing." In: *Journal of Law, Economics, and Organization* 21.1, pp. 20–56 (cit. on p. 39).
- Levesque (2004). "Fundamental issues with open source software development." In: *First Monday* 2. URL: <http://firstmonday.org/ojs/index.php/fm/article/view/1484/1399> (cit. on pp. 29, 44).
- Lewin (1946). "Action research and minority problems." In: *Journal of social issues* 2.4, pp. 34–46 (cit. on p. 85).
- Lewin (1948). *Resolving social conflicts*. Harper and Rowe (cit. on p. 87).
- Little and Graves (2008). "Little's Law." English. In: *Building Intuition*. Ed. by Dilip Chhajed and TimothyJ. Lowe. Vol. 115. International Series in Operations Research & Management Science. Springer US, pp. 81–100. ISBN: 978-0-387-73698-3. DOI: [10.1007/978-0-387-73699-0_5](https://doi.org/10.1007/978-0-387-73699-0_5). URL: http://dx.doi.org/10.1007/978-0-387-73699-0_5 (cit. on pp. 22, 23, 115).
- Liu (2005). "Enriching software engineering courses with service-learning projects and the open-source approach." In: *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*. Ed. by Gruia-Catalin Roman, William G. Griswold, and Bashar

- Nuseibeh. ACM, pp. 613–614. DOI: [10.1145/1062455.1062566](https://doi.org/10.1145/1062455.1062566). URL: <http://doi.acm.org/10.1145/1062455.1062566> (cit. on p. 75).
- MacKellar, Sabin, and Tucker (2015). “Bridging the Academia-Industry Gap in Software Engineering: A Client-Oriented Open Source Software Projects Course.” In: *Open Source Technology: Concepts, Methodologies, Tools, and Applications*. Hershey USA: IGI Global. Chap. 99, pp. 1927–1950. DOI: [10.4018/978-1-4666-7230-7.ch099](https://doi.org/10.4018/978-1-4666-7230-7.ch099) (cit. on pp. 1, 2, 77).
- Madey, Freeh, and Tynan (2005). “Modeling the Free/Open Source software community: A quantitative investigation.” In: *Free/Open Source Software Development*, pp. 203–221 (cit. on p. 41).
- Madeyski (2010). *Test-Driven Development - An Empirical Evaluation of Agile Practice*. Springer. ISBN: 978-3-642-04287-4. DOI: [10.1007/978-3-642-04288-1](https://doi.org/10.1007/978-3-642-04288-1). URL: <https://doi.org/10.1007/978-3-642-04288-1> (cit. on p. 157).
- Mäenpää, Kilamo, and Männistö (2016). “In-between Open and Closed - Drawing the Fine Line in Hybrid Communities.” In: *Open Source Systems: Integrating Communities - 12th IFIP WG 2.13 International Conference, OSS 2016, Gothenburg, Sweden, May 30 - June 2, 2016, Proceedings*. Ed. by Kevin Crowston, Imed Hammouda, Björn Lundell, Gregorio Robles, Jonas Gamalielsson, and Juho Lindman. Vol. 472. IFIP Advances in Information and Communication Technology. Springer, pp. 134–146. ISBN: 978-3-319-39224-0. DOI: [10.1007/978-3-319-39225-7_11](https://doi.org/10.1007/978-3-319-39225-7_11). URL: https://doi.org/10.1007/978-3-319-39225-7_11 (cit. on pp. 37, 38).
- Magdaleno, Lima Werner, and Mendes de Araujo (2012). “Reconciling software development models: A quasi-systematic review.” In: *Journal of Systems and Software* 85.2. Special issue with selected papers from the 23rd Brazilian Symposium on Software Engineering, pp. 351–369. ISSN: 0164-1212. DOI: <http://dx.doi.org/10.1016/j.jss.2011.08.028>. URL: <http://www.sciencedirect.com/science/article/pii/S0164121211002287> (cit. on pp. 36, 83, 84).
- Marmorstein (2011). “Open source contribution as an effective software engineering class project.” In: *Proceedings of the 16th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, ITiCSE 2011, Darmstadt, Germany, June 27-29, 2011*. Ed. by Guido Rößling, Thomas L. Naps, and Christian Spannagel. ACM, pp. 268–272. ISBN: 978-1-4503-0697-3. DOI: [10.1145/1999747.1999823](https://doi.org/10.1145/1999747.1999823). URL: <http://doi.acm.org/10.1145/1999747.1999823> (cit. on p. 76).

Bibliography

- Masmoudi, Héla, Matthijs den Besten, Claude de Loupy, and Jean-Michel Dalle (2009). “Peeling the Onion.” In: *Open Source Ecosystems: Diverse Communities Interacting, 5th IFIP WG 2.13 International Conference on Open Source Systems, OSS 2009, Skövde, Sweden, June 3-6, 2009. Proceedings*. Ed. by Boldyreff, Crowston, Lundell, and Wasserman. Vol. 299. IFIP Advances in Information and Communication Technology. Springer-Verlag, pp. 284–297. ISBN: 978-3-642-02031-5. DOI: [10.1007/978-3-642-02032-2_25](https://doi.org/10.1007/978-3-642-02032-2_25). URL: http://dx.doi.org/10.1007/978-3-642-02032-2_25 (cit. on pp. 41, 54).
- Melian (2007). “Progressive Open Source: The Construction of a Development Project at Hewlett-Packard.” PhD thesis. Stockholm School of Economics (cit. on p. 39).
- Miranda (Jan. 2001). “Improving Subjective Estimates Using Paired Comparisons.” In: *IEEE Softw.* 18.1, pp. 87–91. ISSN: 0740-7459. DOI: [10.1109/52.903173](https://doi.org/10.1109/52.903173). URL: <http://dx.doi.org/10.1109/52.903173> (cit. on p. 120).
- Mockus, Fielding, and Herbsleb (2000). “A case study of open source software development: the Apache server.” In: *Proceedings of the 22nd International Conference on Software Engineering, ICSE 2000, Limerick Ireland, June 4-11, 2000*. Ed. by Carlo Ghezzi, Mehdi Jazayeri, and Alexander L. Wolf. ACM, pp. 263–272. ISBN: 1-58113-206-9. DOI: [10.1145/337180.337209](https://doi.org/10.1145/337180.337209). URL: <http://doi.acm.org/10.1145/337180.337209> (cit. on p. 43).
- Mockus, Fielding, and Herbsleb (2002). “Two case studies of open source software development: Apache and Mozilla.” In: *ACM Trans. Softw. Eng. Methodol.* 11.3, pp. 309–346. DOI: [10.1145/567793.567795](https://doi.org/10.1145/567793.567795). URL: <http://doi.acm.org/10.1145/567793.567795> (cit. on pp. 40, 41, 43, 54, 78).
- Monge, Fulk, Kalman, Flanagan, Parnassa, and Rumsey (1998). “Production of collective action in alliance-based interorganizational communication and information systems.” In: *Organization Science* 9.3, pp. 411–433 (cit. on p. 44).
- Moon and Sproull (2000). “Essence of Distributed Work: The Case of the Linux Kernel.” In: *First Monday* 5.11. URL: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/801> (cit. on p. 40).
- Morelli, Lanerolle, and Tucker (2012). “The Humanitarian Free and Open-Source Software Project: Engaging Students in Service-Learning through

- Building Software." In: *Service-Learning in the Computer and Information Sciences*. John Wiley & Sons, Inc., pp. 117–136. ISBN: 9781118319130. DOI: 10.1002/9781118319130.ch5. URL: <http://dx.doi.org/10.1002/9781118319130.ch5> (cit. on p. 77).
- Murgia, Concas, Pinna, Tonelli, and Turnu (2009). "Empirical study of software quality evolution in open source projects using agile practices." In: *CoRR* abs/0905.3287. URL: <http://arxiv.org/abs/0905.3287> (cit. on p. 84).
- O'Reilly (1999). "Lessons from Open-Source Software Development - Introduction." In: *Commun. ACM* 42.4, pp. 32–37. DOI: 10.1145/299157.299164. URL: <http://doi.acm.org/10.1145/299157.299164> (cit. on p. 43).
- Ohno (1988). *Toyota Production System: Beyond Large-Scale Production*. Portland, OR: Productivity. ISBN: 0-915299-14-3 (cit. on pp. 9, 20).
- Okoli and Carillo (2012). "The best of adaptive and predictive methodologies: open source software development, a balance between agility and discipline." In: *IJITM* 11.1/2, pp. 153–166. DOI: 10.1504/IJITM.2012.044071. URL: <https://doi.org/10.1504/IJITM.2012.044071> (cit. on p. 84).
- OpenSourceInitiative (Mar. 2007). *The Open Source Definition Version 1.9*. English. Open Source Initiative. URL: <https://opensource.org/osd-annotated> (cit. on pp. 33, 34).
- Orman (2008). "Giving it away for free? The nature of job-market signaling by open-source software developers." In: *The BE Journal of Economic Analysis & Policy* 8.1, pp. 1–33 (cit. on p. 39).
- Palmer and Felsing (2001). *A Practical Guide to Feature-Driven Development*. 1st. Pearson Education. ISBN: 0130676152 (cit. on p. 8).
- Pedroni, Bay, Oriol, and Pedroni (2007). "Open source projects in programming courses." In: *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2007, Covington, Kentucky, USA, March 7-11, 2007*. Ed. by Ingrid Russell, Susan M. Haller, J. D. Dougherty, and Susan H. Rodger. ACM, pp. 454–458. ISBN: 1-59593-361-1. DOI: 10.1145/1227310.1227465. URL: <http://doi.acm.org/10.1145/1227310.1227465> (cit. on p. 76).
- Perens (1999). "The open source definition." In: *Open sources: voices from the open source revolution* 1, pp. 171–188 (cit. on pp. 33, 34).

Bibliography

- Pinto, Lima, Figueira Filho, Steinmacher, and Gerosa (2017). "Training Software Engineers Using Open-Source Software: The Professors' Perspective." In: *30th IEEE Conference on Software Engineering Education and Training, CSEE&T 2017, Savannah, GA, USA, November 7-9, 2017*. Ed. by Hironori Washizaki and Nancy Mead. IEEE, pp. 117–121. ISBN: 978-1-5386-2536-1. DOI: [10.1109/CSEET.2017.27](https://doi.org/10.1109/CSEET.2017.27). URL: <https://doi.org/10.1109/CSEET.2017.27> (cit. on p. 2).
- Poole (2004). "Distributed Product Development Using Extreme Programming." In: *Extreme Programming and Agile Processes in Software Engineering, 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, June 6-10, 2004, Proceedings*, pp. 60–67. DOI: [10.1007/978-3-540-24853-8_7](https://doi.org/10.1007/978-3-540-24853-8_7). URL: https://doi.org/10.1007/978-3-540-24853-8_7 (cit. on p. 94).
- Poppendieck (2007). "Lean Software Development." In: *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007, Companion Volume*. IEEE Computer Society, pp. 165–166. DOI: [10.1109/ICSECOMPANION.2007.46](http://doi.ieeecomputersociety.org/10.1109/ICSECOMPANION.2007.46). URL: <http://doi.ieeecomputersociety.org/10.1109/ICSECOMPANION.2007.46> (cit. on p. 9).
- Porruvecchio, Concas, Palmas, and Quaresima (2007). "An Agile Approach for Integration of an Open Source Health Information System." In: *Agile Processes in Software Engineering and Extreme Programming, 8th International Conference, XP 2007, Como, Italy, June 18-22, 2007, Proceedings*. Ed. by Giulio Concas, Ernesto Damiani, Marco Scotto, and Giancarlo Succi. Vol. 4536. Lecture Notes in Computer Science. Springer, pp. 213–218. ISBN: 978-3-540-73100-9. DOI: [10.1007/978-3-540-73101-6_39](http://dx.doi.org/10.1007/978-3-540-73101-6_39). URL: http://dx.doi.org/10.1007/978-3-540-73101-6_39 (cit. on p. 81).
- Public Interest, Software in the (Apr. 2004). *Debian Free Software Guidelines Version 1.1*. English. Software in the Public Interest. URL: https://www.debian.org/social_contract.en.html (cit. on p. 33).
- Ramesh, Cao, Mohan, and Xu (Oct. 2006). "Can Distributed Software Development Be Agile?" In: *Commun. ACM* 49.10, pp. 41–46. ISSN: 0001-0782. DOI: [10.1145/1164394.1164418](http://doi.acm.org/10.1145/1164394.1164418). URL: <http://doi.acm.org/10.1145/1164394.1164418> (cit. on p. 1).
- Rapoport (1970). "Three dilemmas in action research: with special reference to the Tavistock experience." In: *Human relations* 23.6, pp. 499–513 (cit. on p. 85).

- Raymond (1998a). *Goodbye, "free software"; hello, "open source"*. URL: <http://www.catb.org/~esr/open-source.html> (cit. on pp. 33, 44).
- Raymond (1998b). "Homesteading the Noosphere." In: *First Monday* 3.10. URL: <http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/621> (cit. on p. 40).
- Raymond (2001). *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. Sebastopol, CA, USA: O'Reilly & Associates, Inc. ISBN: 0596001088. URL: <http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/> (cit. on pp. 36–38, 78, 156).
- Reddy (2015). *The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban*. 1st. Addison-Wesley Professional (cit. on p. 9).
- Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. (2009). "Scratch: programming for all." In: *Communications of the ACM* 52.11, pp. 60–67 (cit. on p. 47).
- Riehle, Riemer, Kolassa, and Schmidt (2014). "Paid vs. Volunteer Work in Open Source." In: *47th Hawaii International Conference on System Sciences, HICSS 2014, Waikoloa, HI, USA, January 6-9, 2014*. IEEE Computer Society, pp. 3286–3295. ISBN: 978-1-4799-2504-9. DOI: [10.1109/HICSS.2014.407](https://doi.org/10.1109/HICSS.2014.407). URL: <https://doi.org/10.1109/HICSS.2014.407> (cit. on pp. 39, 42, 45).
- Royce (1970). "Managing the development of large software systems." In: *proceedings of IEEE WESCON*. Vol. 26. 8. Los Angeles, pp. 1–9 (cit. on pp. 6, 8).
- Rusovan, Lawford, and Parnas (2005). "Open source software development: future or fad." In: *Perspectives on free and open source software*, pp. 107–122 (cit. on pp. 29, 44).
- Saaty (1996). *Multicriteria Decision Making: The Analytic Hierarchy Process*. Analytic Hierarchy Process Series. R W S Publications. ISBN: 9780962031717 (cit. on p. 120).
- Salman, Tosun Misirli, and Juristo Juzgado (2015). "Are Students Representatives of Professionals in Software Engineering Experiments?" In: *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. Piscataway, NJ, USA: IEEE Press, pp. 666–676. ISBN: 978-1-4799-1934-5. DOI: [10.1109/ICSE.2015.82](https://doi.org/10.1109/ICSE.2015.82). URL: <http://dx.doi.org/10.1109/ICSE.2015.82> (cit. on p. 157).

Bibliography

- Scacchi (2002). "Understanding the requirements for developing open source software systems." In: *Software, IEE Proceedings-*. Vol. 149. IET, pp. 24–39 (cit. on p. 94).
- Scacchi (2007). "Free/open source software development: recent research results and emerging opportunities." In: *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIG-SOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007, Companion Papers*. Ed. by Ivica Crnkovic and Antonia Bertolino. ACM, pp. 459–468. ISBN: 978-1-59593-812-1. DOI: [10.1145/1295014.1295019](https://doi.org/10.1145/1295014.1295019). URL: <http://doi.acm.org/10.1145/1295014.1295019> (cit. on pp. 35, 41–43).
- Scacchi (2010). "Collaboration practices and affordances in free/open source software development." In: *Collaborative software engineering*. Springer, pp. 307–327 (cit. on pp. 94, 101, 103).
- Scacchi, Feller, Fitzgerald, Hissam, and Lakhani (2006). "Understanding Free/Open Source Software Development Processes." In: *Software Process: Improvement and Practice* 11.2, pp. 95–105. DOI: [10.1002/spip.255](https://doi.org/10.1002/spip.255). URL: <https://doi.org/10.1002/spip.255> (cit. on pp. 29, 34, 36–42, 44, 45).
- Schafer (2000). "Office E-Mail: It's Fast, Easy and All Too Often Misunderstood." In: *International Herald Tribune* (cit. on p. 65).
- Schiessle (2017). *Free Software, Open Source, FOSS, FLOSS - same same but different*. English. URL: <https://fsfe.org/freesoftware/basics/comparison.en.html> (cit. on p. 35).
- Schümmer and Schümmer (2000). "Support for Distributed Teams in eXtreme Programming." In: *Proceedings of eXtreme Programming and Flexible Processes Software Engineering - XP2000*. Addison Wesley, pp. 355–377 (cit. on p. 94).
- Schwaber and Beedle (2001). *Agile Software Development with Scrum*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR. ISBN: 0130676349 (cit. on pp. 9, 82).
- Schwaber and Sutherland (2016). *The Scrum Guide*. URL: <http://www.scrumguides.org/> (cit. on p. 9).
- Shah (2006). "Motivation, governance, and the viability of hybrid forms in open source software development." In: *Management Science* 52.7, pp. 1000–1014 (cit. on pp. 39, 40, 52).

- Shibuya and Tamai (2009). "Understanding the process of participating in open source communities." In: *Emerging Trends in Free/Libre/Open Source Software Research and Development, 2009. FLOSS'09. ICSE Workshop on. IEEE*, pp. 1–6 (cit. on pp. 94, 97, 101, 103).
- Shihab, Bird, and Zimmermann (2012). "The effect of branching strategies on software quality." In: *2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '12, Lund, Sweden - September 19 - 20, 2012*. Ed. by Per Runeson, Martin Höst, Emilia Mendes, Anneliese Amschler Andrews, and Rachel Harrison. ACM, pp. 301–310. ISBN: 978-1-4503-1056-7. DOI: 10.1145/2372251.2372305. URL: <http://doi.acm.org/10.1145/2372251.2372305> (cit. on p. 156).
- Sigfridsson, Avram, Sheehan, and Sullivan (2007). "Sprint-driven development: working, learning and the process of enculturation in the PyPy community." In: *Open Source Development, Adoption and Innovation, IFIP Working Group 2.13 on Open Source Software, June 11-14, 2007, Limerick, Ireland*. Ed. by Joseph Feller, Brian Fitzgerald, Walt Scacchi, and Alberto Sillitti. Vol. 234. IFIP. Springer, pp. 133–146. ISBN: 978-0-387-72485-0. DOI: 10.1007/978-0-387-72486-7_11. URL: https://doi.org/10.1007/978-0-387-72486-7_11 (cit. on pp. 80, 81).
- Slany (2012). "A mobile visual programming system for Android smartphones and tablets." In: *2012 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2012, Innsbruck, Austria, September 30 - October 4, 2012*. Ed. by Martin Erwig, Gem Stapleton, and Gennaro Costagliola. IEEE, pp. 265–266. ISBN: 978-1-4673-0852-6. DOI: 10.1109/VLHCC.2012.6344546. URL: <https://doi.org/10.1109/VLHCC.2012.6344546> (cit. on p. 47).
- Slany (2014). "Tinkering with Pocket Code, a Scratch-like programming app for your smartphone." In: *Proc. of Constructionism* (cit. on p. 47).
- Sommer (2016). "Motivation in an Agile, Educational, Free and Open Source Software Project." MA thesis. Graz University of Technology (cit. on pp. 42, 52, 54–56, 58–61, 63, 64, 67–70).
- Stallman (Feb. 1986). *The Free Software Definition*. URL: <https://web.archive.org/web/20070428013835/https://www.gnu.org/bulletins/bull1.txt> (cit. on p. 31).
- Stallman (2010). *Software, Free Society: Selected Essays of Richard M. Stallman*. Second Edition. Free Software Foundation (cit. on p. 31).

Bibliography

- Stallman and Gay (2009). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. Paramount, CA: CreateSpace (cit. on p. 32).
- Stamelos, Angelis, Oikonomou, and Bleris (2002). "Code quality analysis in open source software development." In: *Inf. Syst. J.* 12.1, pp. 43–60. DOI: 10.1046/j.1365-2575.2002.00117.x. URL: <https://doi.org/10.1046/j.1365-2575.2002.00117.x> (cit. on pp. 29, 44).
- Stewart and Gosain (2001). "An Exploratory Study of Ideology and Trust in Open Source Development Groups." In: *Proceedings of the International Conference on Information Systems, ICIS 2001, December 16-19, 2001, New Orleans, Louisiana, USA*. Ed. by Veda C. Storey, Sumit Sarkar, and Janice I. DeGross. Association for Information Systems, pp. 507–512. URL: <http://aisel.aisnet.org/icis2001/63> (cit. on p. 39).
- Susman and Evered (Dec. 1978). "An Assessment of the Scientific Merits of Action Research." In: *Administrative Science Quarterly* 23.4, pp. 582–603. ISSN: 00018392. DOI: 10.2307/2392581. URL: <http://dx.doi.org/10.2307/2392581> (cit. on pp. 85–88, 108).
- Tatham (2010). *Roles In Open Source Projects*. URL: <http://oss-watch.ac.uk/resources/rolesinopensource> (cit. on p. 154).
- Teixeira, Robles, and González-Barahona (2015). "Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem." In: *J. Internet Services and Applications* 6.1, 14:1–14:27. DOI: 10.1186/s13174-015-0028-2. URL: <http://dx.doi.org/10.1186/s13174-015-0028-2> (cit. on pp. 41, 54, 156).
- Theunissen, Boake, and Kourie (2005). "In Search of the Sweet Spot: Agile Open Collaborative Corporate Software Development." In: *Proceedings of the 2005 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries*. SAICSIT '05. White River, South Africa: South African Institute for Computer Scientists and Information Technologists, pp. 268–277. ISBN: 1-59593-258-5. URL: <http://dl.acm.org/citation.cfm?id=1145675.1145705> (cit. on p. 79).
- Theunissen, Kourie, and Boake (2005). "Open Source and Agile Software Development in Corporates: A Contradiction or An Opportunity?" In: Zeist, Holland: Jacquard Conference. URL: http://espresso.cs.up.ac.za/publications/mtheunissen_etal_jacquard2005_paper.pdf (cit. on p. 79).

- Theunissen, Kourie, and Boake (2007). "Corporate-, Agile- and Open Source Software Development: A Witch's Brew or An Elixir of Life?" In: *Balancing Agility and Formalism in Software Engineering, Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007, Poznan, Poland, October 10-12, 2007, Revised Selected Papers*. Ed. by Bertrand Meyer, Jerzy R. Nawrocki, and Bartosz Walter. Vol. 5082. Lecture Notes in Computer Science. Springer, pp. 84–95. ISBN: 978-3-540-85278-0. DOI: [10.1007/978-3-540-85279-7_7](https://doi.org/10.1007/978-3-540-85279-7_7). URL: http://dx.doi.org/10.1007/978-3-540-85279-7_7 (cit. on pp. 41, 79).
- Torres, Toral, Perales, and Barrero (2011). "Analysis of the Core Team Role in Open Source Communities." In: *International Conference on Complex, Intelligent and Software Intensive Systems, CISIS 2011, June 30 - July 2, 2011, Korean Bible University, Seoul, Korea*. IEEE Computer Society, pp. 109–114. ISBN: 978-0-7695-4373-4. DOI: [10.1109/CISIS.2011.25](https://doi.org/10.1109/CISIS.2011.25). URL: <https://doi.org/10.1109/CISIS.2011.25> (cit. on p. 56).
- Truex, Baskerville, and Travis (2000). "Amethodical systems development: the deferred meaning of systems development methods." In: *Accounting, management and information technologies* 10.1, pp. 53–79 (cit. on p. 6).
- Tsirakidis, Koebler, and Krcmar (July 2009). "Identification of Success and Failure Factors of Two Agile Software Development Teams in an Open Source Organization." In: *ICGSE 2009. Fourth IEEE International Conference on Global Software Engineering*, pp. 295–296. DOI: [10.1109/ICGSE.2009.42](https://doi.org/10.1109/ICGSE.2009.42) (cit. on p. 82).
- Turnu, Melis, Cau, Marchesi, and Setzu (2004). "Introducing TDD on a Free Libre Open Source Software Project: A Simulation Experiment." In: *Proceedings of the 2004 Workshop on Quantitative Techniques for Software Agile Process*. QUTE-SWAP '04. Newport Beach, California: ACM, pp. 59–65. DOI: [10.1145/1151433.1151442](https://doi.org/10.1145/1151433.1151442). URL: <http://doi.acm.org/10.1145/1151433.1151442> (cit. on p. 79).
- Turnu, Melis, Cau, Setzu, Concas, and Mannaro (2006). "Modeling and simulation of open source development using an agile practice." In: *Journal of Systems Architecture* 52.11. Agile Methodologies for Software Production, pp. 610–618. ISSN: 1383-7621. DOI: [http://dx.doi.org/10.1016/j.sysarc.2006.06.005](https://doi.org/10.1016/j.sysarc.2006.06.005). URL: <http://www.sciencedirect.com/science/article/pii/S1383762106000634> (cit. on p. 79).

Bibliography

- VersionOne (2011). *6th Annual State of Agile Survey*. URL: http://www.versionone.com/pdf/2011%5C_State%5C_of%5C_Agile%5C_Development%5C_Survey%5C_Results.pdf (cit. on p. 10).
- VersionOne (2012). *7th Annual State of Agile Survey*. URL: <http://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf> (cit. on p. 10).
- VersionOne (2013). *8th Annual State of Agile Survey*. URL: <http://www.versionone.com/pdf/2013-state-of-agile-survey.pdf> (cit. on p. 10).
- VersionOne (2014). *9th Annual State of Agile Survey* (cit. on p. 10).
- VersionOne (2015). *10th Annual State of Agile Survey*. URL: <https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf> (cit. on pp. 2, 10).
- Warsta and Abrahamsson (2003). "Is Open Source Software Development Essentially an Agile Method?" In: *Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering*. Portland, Oregon: IEEE Computer Society, pp. 143–147 (cit. on pp. 1, 43, 79).
- Wayner (2000). *Free for all: How Linux and the free software movement undercut the high-tech titans*. Harper Business New York (cit. on p. 44).
- Weber (2004). *The success of open source*. Harvard University Press (cit. on p. 30).
- West and O'Mahony (Jan. 2005). "Contrasting Community Building in Sponsored and Community Founded Open Source Projects." In: *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pp. 196c–196c. DOI: [10.1109/HICSS.2005.166](https://doi.org/10.1109/HICSS.2005.166) (cit. on p. 37).
- Williams (2000). "The Collaborative Software Process." PhD thesis. University of Utah, Department of Computer Science (cit. on p. 16).
- Williams, Krebs, and Layman (June 2004). *Extreme Programming Evaluation Framework for Object-Oriented Languages – Version 1.4* (cit. on pp. 10, 14–18, 52, 62, 67, 122, 144–146).
- Wusteman (2009). "OJAX: a case study in agile Web 2.0 open source development." In: *Aslib Proceedings* 61.3, pp. 212–231. DOI: [10.1108/00012530910959781](https://doi.org/10.1108/00012530910959781). eprint: <http://dx.doi.org/10.1108/00012530910959781>. URL: <http://dx.doi.org/10.1108/00012530910959781> (cit. on p. 82).
- Yamauchi, Yokozawa, Shinohara, and Ishida (2000). "Collaboration with Lean Media: how open-source software succeeds." In: *CSCW 2000, Pro-*

- ceeding on the ACM 2000 Conference on Computer Supported Cooperative Work, Philadelphia, PA, USA, December 2-6, 2000*. Ed. by Wendy A. Kellogg and Steve Whittaker. ACM, pp. 329–338. ISBN: 1-58113-222-0. DOI: [10.1145/358916.359004](https://doi.org/10.1145/358916.359004). URL: <http://doi.acm.org/10.1145/358916.359004> (cit. on pp. 44, 94).
- Ye and Kishida (2003). “Toward an Understanding of the Motivation of Open Source Software Developers.” In: *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*. Ed. by Lori A. Clarke, Laurie Dillon, and Walter F. Tichy. IEEE Computer Society, pp. 419–429. ISBN: 0-7695-1877-X. DOI: [10.1109/ICSE.2003.1201220](https://doi.org/10.1109/ICSE.2003.1201220). URL: <http://dx.doi.org/10.1109/ICSE.2003.1201220> (cit. on pp. 39, 42, 43, 55).