Product-Focused Software Process Improvement

15th International Conference, PROFES 2014
Helsinki, Finland, December 10-12, 2014

Proceedings Volume 2

Fabian Fagerholm, Maria Paasivaara, Andreas Jedlitschka,
Pasi Kuvaja, Marco Kuhrmann, Tomi Männistö,
Jürgen Münch, Mikko Raatikainen (editors)

University of Helsinki
Department of Computer Science
Publication series B
Report B-2014-3
ISSN 1458-4786
ISBN 978-951-51-0512-7 (PDF)
Helsinki 2014

Preface

On behalf of the PROFES Organizing Committee, we are proud to present the proceedings volume 2 of the 15th International Conference on Product-Focused Software Process Improvement (PROFES 2014) held in Helsinki, Finland. This volume 2 consists of the tutorial abstracts and the doctoral symposium papers. The volume 1 consisting of the long and short papers is published as Springer's Lecture Notes in Computer Science (LNCS) 8892.

Since 1999, PROFES has established itself as one of the recognized international process improvement conferences. The main theme of PROFES is professional software process improvement (SPI) motivated by product, process, and service quality needs. PROFES 2014 addressed both, quality engineering and management topics, including processes, methods, techniques, tools, organizations, and enabling SPI. Both, solutions found in practice and relevant research results from academia were presented.

We are thankful for the opportunity to have served as Chairs for this conference. We thank the organization committee. The program committee members and reviewers provided excellent support in reviewing the papers. We are also grateful to the authors, presenters, and session chairs for their time and effort in making PROFES 2014 a success.

November 2014

Andreas Jedlitschka
Pasi Kuvaja
Marco Kuhrmann
Tomi Männistö
Jürgen Münch
Mikko Raatikainen

Preface for PROFES 2014 Tutorials

PROFES 2014 hosted tutorials to complement and enhance the main conference program, offering a wider knowledge perspective around the conference topics. These tutorials provided insights into special topics of current and ongoing relevance to the conference focus areas. Three tutorials were presented, covering real-time product innovation planning, software reuse and reusability, and process improvement for QA and testing. Each tutorial was based on research combined with practical application in the software industry. I would like to thank the tutors for sharing their valuable insights, and the tutorial audience for the interesting discussions.

November 2014

Fabian Fagerholm
PROFES 2014 Tutorial Chair

Preface for PROFES 2014 Doctoral Symposium

The PROFES Doctoral Symposium is a forum for PhD students, working on foundations, techniques, methods and tools in the area of software process and product quality improvement, to present their research and receive feedback and advice. In the PROFES 2014 Doctoral Symposium seven PhD students presented and discussed their work. The opening keynote was held by Prof. Torgeir Dingsøyr on "How to make impact with journal publications on software process improvement?" I would like to thank the keynote speaker and all the symposium presenters, participants and discussants. I am also grateful for the Doctoral Symposium Program Committee members, Torgeir Dingsøyr, Martin Höst, Daniel Méndez Fernández, Tommi Mikkonen, Kai Petersen, Kari Smolander, Burak Turhan, Pasi Tyrväinen and Sjaak Brinkkemper, for reviewing the submissions and giving students feedback both before and during the symposium.

November 2014

Maria Paasivaara
PROFES 2014 Doctoral
Symposium Chair

Organization

General Chair

Jürgen Münch	University of Helsinki, Finland
Tomi Männistö	University of Helsinki, Finland

Program Co-Chairs

Andreas Jedlitschka	Fraunhofer IESE, Germany
Pasi Kuvaja	University of Oulu, Finland

Short Papers and Posters Chair

Marco Kuhrmann	University of Southern Denmark, Denmark

Tutorial Chair

Fabian Fagerholm	University of Helsinki, Finland

Doctoral Symposium Chair

Maria Paasivaara	Aalto University, Finland

Proceedings Chair

Mikko Raatikainen	Aalto University, Finland

Local Arrangements Chair

Simo Mäkinen	University of Helsinki, Finland

Publicity Chair

Kari Liukkunen	University of Oulu, Finland

Social Media Chair

Daniel Graziotin	Free University of Bozen-Bolzano, Italy
Daniel Méndez Fernández	Technische Universität München, Germany

Web Site Chair

Max Pagels	University of Helsinki, Finland

Head of Conference Secretariat

Mary-Ann Wikström	Aalto University, Finland

Table of Contents

Tutorials

Doctoral Symposium

Tutorial: Towards Real-time Product Innovation Planning

Guenther Ruhe

Software Engineering Decision Support Laboratory
University of Calgary
Alberta, Canada

Abstract

This tutorial provides hands-on knowledge and skills to perform real-time product innovation planning. As the starting point, current deficits in the product innovation planning process are analyzed. Subsequently, different strategies to overcome the current deficits are studied. The proposed process is accompanied by tool support combining optimized release planning strategies offered by ReleasePlanner 2.0 with real-time issue management operated by Atlassians JIRA. A variety of use cases are studied to (i) illustrate the process and (ii) document the added value for product innovation.

Tutorial: Software Reuse and Reusability based on Business Processes and Requirements

Hermann Kaindl

Vienna University of Technology, ICT
Vienna, Austria

Abstract

Software reuse and reusability is often just addressed at the level of code or low-level design. In contrast, this tutorial explains them based on business processes and requirements. It presents and compares three approaches co-developed by the presenter over more than a decade.

The first of these approaches deals with requirements reuse in the context of product lines. It makes the relations among product line requirements explicit, so that single system requirements in this product line can be derived consistently. A key issue is commonality and variability across different products. This tutorial shows how requirements for a product line can be modeled, selected and reused to engineer the requirements for innovative new products.

The second approach for software reuse involves case-based reasoning. Instead of explicit relations between requirements (or other artifacts), similarity metrics are employed for finding the most similar software case in a repository to a given set of requirements. This even works when a single envisioned usage scenario is specified yet, and it allows reusing also requirements from retrieved cases. The major point, however, is to facilitate reusing software design (including architecture) and code from similar software cases.

The third approach strives for (partly) automating software development for certain business applications through reusing business knowledge and software, where both are tightly connected. It involves automated reuse of business processes, and software (services)executing them, based on ontological knowledge. A key point is closing the representational gap between procedurally represented business processes and declaratively represented concepts and their relations, taxonomies, partonomies, etc. So, this is an ontology-based approach for (partly) automated software development guided by business models.

These approaches have different key properties and trade-offs between costs of making software artefacts reusable and benefits for reusing them. These will be particularly explained in this tutorial.

Tutorial: Practical Process Improvements — Applying the LAPPI Technique in QA and Testing

Anu Raninen[1] and Päivi Brunou[2]

[1]Sogeti Finland Oy
Espoo, Finland

[2] Knowit Oy
Helsinki, Finland

Abstract

Understanding the current state of software development and QA processes and their problem points is important. Without this understanding, software process improvement (SPI) resources may not be properly allocated. This tutorial focuses on identifying and prioritizing SPI activities from a quality improvement point of view. The importance of motivating and monitoring during the improvement is not forgotten either.

This tutorial is organised as an interactive workshop interlaced with real life experiences in SPI. During the tutorials attendees learn about the Light-weight Technique to Practical Process Modelling and Improvement Target Identification (LAPPI). The technique enables process modelling and improvement target identification, is suited to organizations of all sizes, and can be integrated with various SPI initiatives. It was developed through multiple academia-industry collaboration projects and by industry actors themselves.

From Product Roadmapping towards Continuous Planning: Changing the Ways of Planning

Tanja Suomalainen[1]
[1]Kaitoväylä 1, P.O. Box 1100, FI-90571 Oulu, Finland
tanja.suomalainen@vtt.fi

Supervisor:
Professor Jouni Similä[2]
[2]Department of Information Processing Science
P.O. Box 3000, 90014 University of Oulu, Finland
jouni.simila@oulu.fi

Abstract. In recent few years, the adoption of agile and lean development practices has increased remarkably. However, this is not the end, new and innovative approaches that support continuous practices are needed. Continuity is required in all the levels of the organisation; from business strategy and planning to software development and thereafter to operational deployment, as well as between these levels. Continuous planning means implementing planning practices continuously, in rapid parallel cycles, instead of the predefined and regular planning occasions. Continuous planning is not commonly adopted and applied throughout the organisation and currently involves only a certain level of planning, e.g. release planning. Based on the current literature, continuous planning is a relatively new and not yet fully studied field of research. To augment the knowledge relating to continuous planning, this research plan presents how the knowledge relating to continuous planning is going to be increased and disseminated.

Keywords: Continuous planning, product roadmapping, agile, lean

1 Introduction and Background

Market uncertainties, competitive pressures, and the constant need for shortened development cycles call for flexible, responsive and adaptive software development practices [1]. Since the mid-1990s, a variety of agile methods and practices have been designed to enhance development teams' or an organisation's ability to respond to dynamic market changes [1, 2, 3]. Also, in recent years, lean thinking [4] has been introduced in software development companies [5, 6, 7] with the aim of achieving a continuous and smooth flow of production in pursuance of removing waste in processes. The promise of lean development is to create a change-tolerant organisation that can survive and succeed in times of uncertainty, change and complexity [7]. In emphasising the use of iterations and development of small

features, agile practices and lean development have indeed increased the ability for software development companies to accommodate fast changing customer requirements and fluctuating market needs as well as decreasing lead times and improving the quality of their products [1].

Even though many software development companies have succeeded adopting agile practices, in order to improve responsiveness to customers, agile development is not the end, the final step of software development [1]. Therefore, new and innovative approaches that support continuous practices are needed. Thus, Olsson et al. [1] highlight that software development companies need to move beyond the concept of agile development and towards a situation in which software functionality is continuously deployed and where customer feedback is the main driver for innovation. Fizgerald and Stol [8], on the other hand, emphasize continuous integration between software development and its operational deployment, called DevOps. Similarly the link between business strategy and software development should be continuously assessed and improved, called BizDev [8].

Planning in general is seen to consist of two things: actions and forecasts (i.e. expected outcomes). After understanding the aims and directions, it is time to plan how to get there [9]. However, planning is not only about target setting, since it lies behind us, instead, targets should be ambitious and forecasts realistic in order to take the necessary corrective actions [10]. In large organisations there are multiple levels of planning that are performed at time horizons and by different actors [11, 12]. Therefore, the most important aspects of organisational planning are the required planning levels and time horizons [13]. In the software development context, planning is commonly episodic and performed according to a traditional cycle usually triggered by annual financial year-end considerations [8]. In fact, the ongoing planning problem is that time is divided into a number of planning horizons, each lasting a significant period of time [8], and the continuity is not seen.

Already, for some years ago, long-term planning called roadmapping has been proposed as a solution to bridge the gap between different levels of planning [14]. It was realised then that changing planning processes e.g. towards a roadmap-based planning is a major cultural change. Roadmapping requires cross-functional participation to be successful and getting various departments to work together can be hard [15]. Therefore, it was suggested that roadmapping should be seen as part of the company's overall business processes and integrated into the company's planning cycles [15]. But now, as the agile and lean practices are becoming the norm, the transformation towards continuous practices is emphasized in the organisations. Drawing on the lean concept of flow, Fizgerald and Stol [8] identify a number of continuous activities which are important to software development today's context, one of them being continuous planning.

Continuous planning is about implementing planning practices continuously, in rapid parallel cycles, instead of the predefined and regular planning occasions. Thus, planning is not conducted just as part of a top-down annual event [9]. Environmental changes trigger planning instead of the predefined and regular planning cycle (e.g. financial year) and thus, plans are adjusted according to internal and external events [16]. Planning should be done continuously so that at any time, the full scale of the development can be presented [17]. Fizgerald and Stol [8] define continuous planning as a holistic attempt involving multiple stakeholders from business and software

functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment, and thus involve a tighter integration between planning and execution. In software development, continuous planning refers to the organisational capability to conduct planning in rapid parallel cycles, which can be hours, days or very small numbers of weeks or months depending on the level of planning. In order to achieve continuous planning, organisations need to be capable of changing their operations and adapting their mind-set towards continuous planning and transparency throughout the whole organisation.

Based on the current literature, continuous planning is not commonly adopted and applied throughout the organisation and currently involves only a certain level of planning, e.g. release planning (e.g. using Scrum). According to Fizgerald and Stol [8] the only form of continuous planning is what emerges from agile development approaches and is related to sprint iterations or at best, software releases. Similarly they conclude that continuous planning is not widespread throughout the organisation. For example, Heikkilä et al. [18] have adopted a three-level planning model: including strategic planning, release planning, and operational planning. Strategic planning is the interface between business and management and development and it is performed in long-term. Release planning defines the feature content of the next release and on planning how to efficiently create content. Operational planning, on the contrary, concerns implementation of features on day-to-day basis. However, they focus on release planning in large agile organisations. Thus, continuous planning requires wider perspective than currently considered.

Based on the literature review, continuous planning is a relatively new and poorly studied field of research, and hence the literature relating to continuous planning is not adequate. The current literature focuses mainly on some specific or single level of planning in the organisation and a wider perspective of continuous planning is not viewed. For instance, there is very little empirical research literature of continuous planning that describes how continuous planning is conducted at the different levels of planning and at the different levels of organisation, and how the information from the planning and the plans are visible to the other levels of planning. The intention in this research is to increase the current empirical evidence on continuous planning and roadmapping both for industry and science, and based on that evidence companies can develop their own planning and roadmapping activities. Also, the intention is to provide guidelines how to improve and change the planning practices towards continuous ways of working and planning.

2 Research design

The purpose of this research is to study continuous planning and roadmapping in the context of Information and Communication Technology (ICT) companies. Furthermore, the research is done in the context of large organisations that have multiple levels of planning and that are performed at time horizons and by different actors. The research focuses on defining factors related to continuous planning and roadmapping, e.g. level of planning, timeframe, process, and participants. The

background for this research lies on agile and lean development methods and practices that have triggered the need for continuous ways working and planning.

The research is conducted as empirical research and the research is carried out as a case study [19]. Case study research can be used to achieve various research aims, for instance to provide descriptions of the phenomena, to test a theory, and to develop a theory [20]. The case studies can be based on both quantitative and qualitative evidence, because data is collected through such methods as inquiries, interviews, observation, and through the use of documents and artefacts [19, 21]. Also, the case study methodology claims to be well suited for software engineering research as it studies contemporary phenomena in its natural context [22].

A multiple-case studies approach [19] is chosen for this research, since the theory on the continuous planning is not yet well formulated and practical experiences from the field of research are difficult to find. Therefore, the purpose is to fill in the gaps found in the literature, and thereafter, to create a theory relating to continuous planning practices and processes. To verify the theory, the experiences of several companies should be gathered and analysed. Data collection is carried out by conducting narrative and semi-structured interviews to persons from different levels of the organisation namely people that are involved in the strategy and business planning, product planning, and release planning. The case companies are selected based on their size, since the focus is on large organisations that have multiple levels of planning. The duration of the interviews will be around one hour and all the interviews are recorded. After the interviews, the recordings will be transcribed and analysed. The data will be analysed by using e.g. a generic process of data analysis [23]. A brief summary from each of the interview will be written and sent to the interviewees to be read through and validated (e.g. corrected if needed).

2.1 Research Questions

The research problem is addressed through answering the following research questions:

RQ1. What is product roadmapping (process, participants, roles, etc.)? (Paper I and II)
   a) How collaboration affects product roadmapping? (Paper I)

RQ2. How continuous planning is conducted? (Paper III, IV)
   a) How information from the continuous planning and the plans are visible to the other levels of planning, and how they affect to each other?
   b) What are the benefits, challenges and drawbacks of continuous planning? (Paper III)

2.2 Results achieved so far

Following articles relating to the thesis have been already written:

I. Suomalainen, Tanja; Tihinen, Maarit; Parviainen, Päivi. Challenges for product roadmapping in inter-company collaboration. SEAFOOD Proceedings of the Third International Conference on Software Engineering Approaches

for Offshore and Outsourced Development (SEAFOOD). ETH Zurich, Switzerland on July 2-3 2009. Springer (2009), pp. 66 – 80.

II. Suomalainen, Tanja; Salo, Outi; Abrahamsson, Pekka; Similä, Jouni. Software product roadmapping in a volatile business environment. The Journal of Systems and Software, volume 84 issue 6, (2011), pp. 958–975.

III. Suomalainen, Tanja; Kuusela, Raija; Tihinen, Maarit. Continuous Planning – An important Aspect of Agile and Lean Development. **SUBMITTED** to Journal of Agile Systems and Management

IV. Suomalainen, Tanja; Kuusela, Raija; Teppola, Susanna; Huomo, Tua. Challenges of ICT Companies in Lean Transformation. **ToBeSubmitted**

The first two articles (I and II) create ground understanding about product roadmapping by describing the theory and practice behind it. The intention was to define the process, participants, and roles of product roadmapping (II), and how collaboration affects to it (I). When the research relating to these articles was done, there was clear need for this research, since only a few scientific articles could be found about product roadmapping. Since then, the scientific literature about product roadmapping has somewhat increased, but the major change is that the world has moved more towards continuous planning mode, and it has been even questioned how vital the long-term plans are. Thus, article III, focuses on defining continuous planning and how it is conducted as well as identifying the main benefits and challenges of continuous planning. The research also defines what the main elements of continuous planning are: organisational planning, strategic planning and business planning. Accordingly, all of these elements are tightly related to each other; organisational planning defining organisational level and the time frames of the plan; strategic planning forming the overall plan of the organisation and business planning giving the budgeting frame to the plans. Article IV, defined lean transformation framework including three cycles, i.e. strategic alignment cycle, organisational and business alignment cycle, and lean implementation cycle. Continuous planning is mainly discussed at the strategic alignment cycle in the article.

2.3 Future Research Agenda

The continuous planning research is going to be continued in the Need for Speed (N4S) program (http://www.n4s.fi/en/), which is a Tekes funded program of Digile (2014-2017). First, the future case studies are planned and the case companies are selected (during autumn 2014), and thereafter the case studies are conducted, i.e. interviews are held (during spring 2015). Finally, the case study results are analysed and results are reported (during autumn 2015), including writing both journal and conference articles based on the results. The dissertation will be written during 2016.

References

1. Olsson, H. H., Bosch, J., Alahyari, H.: Towards R&D as Innovation Experiment Systems: A Framework for Moving Beyond Agile Software Development. IASTED

Multiconferences-Proceedings of the IASTED International Conference on Software Engineering, SE 2013, pp. 798-805 (2013)
2. Highsmith, J.: Agile software development ecosystems. Addison-Wesley, Boston (2002)
3. Kettunen, P.: Adopting Key Lessons from Agile Manufacturing to Agile Software Product development—A Comparative Study. Technovation 29, pp. 408-422 (2009)
4. Womack, J. P., Jones, D. T.: Lean thinking: Banish waste and create wealth in your corporation, revised and updated. Free Press (2003)
5. Middleton, P., Flaxel, A., Cookson, A.: Lean Software Management Case Study: Timberline Inc. Extreme Programming and Agile Processes in Software Engineering, pp. 1-9-9 Springer, (2005)
6. Poppendieck, M., Poppendieck, T.: Lean software development: An agile toolkit. Addison-Wesley, Boston, MA (2003)
7. Charette, R. N.: Challenging the Fundamental Notions of Software Development. Cutter Consortium, Executive Rep 4, (2003)
8. Fitzgerald, B., Stol, K.: Continuous Software Engineering and Beyond: Trends and Challenges. Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering, pp. 1-9-9 ACM, (2014)
9. Hope, J., Fraser, R.: Beyond budgeting: How managers can break free from the annual performance trap. Harvard Business School Press, Boston, Massachusetts (2003)
10. Bogsnes, B.: Implementing beyond budgeting: Unlocking the performance potential. John Wiley & Sons, Hoboken, New Jersey (2008)
11. Cohn, M.: Agile estimation and planning. Prentice Hall, NJ; US (2006)
12. Leffingwell, D.: Agile software requirements: Lean requirements practices for teams, programs, and the enterprise. Pearson Education, Inc., Boston, MA, USA (2011)
13. Lehtola, L., Kauppinen, M., Vähäniitty, J.: Strengthening the Link between Business Decisions and RE: Long-Term Product Planning in Software Product Companies. 15th IEEE International Requirements Engineering Conference (RE' 07), pp. 153-162 IEEE Computer Society, (2007)
14. Phaal, R., Muller, G.: An Architectural Framework for Roadmapping: Towards Visual Strategy. Technological Forecasting and Social Change 76, pp. 39-49 (2009)
15. Cosner, R. R., Hynds, E. J., Fusfeld, A. R. et al.: Integrating Roadmapping into Technical Planning. Research-Technology Management 50, pp. 31-48 (2007)
16. Rickards, R. C., Ritsert, R.: Rediscovering Rolling Planning: Controller's Roadmap for Implementing Rolling Instruments in SMEs. Procedia Economics and Finance 2, pp. 135-144 (2012)
17. Westkamper, E., Von Briel, R.: Continuous Improvement and Participative Factory Planning by Computer Systems. CIRP Annals-Manufacturing Technology 50, pp. 347-352 (2001)
18. Heikkilä, V. T., Paasivaara, M., Lassenius, C. et al.: Continuous release planning in a large-scale scrum development organization at ericsson. Springer, Berlin Heidelberg (2013)
19. Yin, R. K. Applied social research methods series vol 5; case study research: Design and methods. 2nd ed. Sage Publications, Inc.Yin, R.K, Thousand Oaks, California (1994)
20. Darke, P., Shanks, G., Broadbent, M.: Successfully Completing Case Study Research: Combining Rigour, Relevance and Pragmatism. Information Systems Journal 8, pp. 273-289 (1998)
21. Patton, M. Q.: Qualitative research & evaluation methods. Sage Publications, Inc.Patton, M.Q, Thousand Oaks, California (2002)
22. Runeson, P., Höst, M.: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical Software Engineering 14, pp. 131-164 (2009)
23. Creswell, J. W. Research design: Qualitative, quantitative, and mixed method approaches. 2nd ed. Sage Publications, Thousand Oaks, California (2003)

Understanding Continuous Delivery – A Research Plan

Eero Laukkanen and Casper Lassenius (supervisor)

Aalto University, P.O. BOX 19210, FI-00076, Aalto, Finland,
{eero.laukkanen,casper.lassenius}@aalto.fi

Abstract. Continuous delivery is a software development discipline in which software can be released to production at any time. While many organizations report adopting continuous delivery or at least pursuing it, there exist little scientific knowledge about it. A research plan for doctoral dissertation is introduced which seeks to create a base of scientific knowledge for understanding and further studying continuous delivery. The dissertation aims to understand the problems that emerge when adopting continuous delivery and find solutions to those problems.

Keywords: continuous integration, continuous delivery, continuous deployment, software project

1 Introduction

Continuous delivery (CD) is a software development discipline in which software can be released to production at any time [12]. The discipline is an extension of continuous integration (CI) [11] where software is integrated multiple times per day. CI gained most attention as a part of extreme programming (XP) [1] and is broadly adopted today [21], independently of other XP practices. Contrasted to CD, CI does not mandate that software is always in a releasable state.

Until recently, little empirical research has been done addressing continuous integration individually. At the same time, continuous delivery and its more ambitious form, continuous deployment, are gaining much attention amongst practitioners. This research plan for doctoral dissertation aims to bridge the gap between research and practice.

2 Research area and problem

The dissertation will be conducted in the field of empirical software engineering. Continuous delivery describes how software is developed, tested and deployed. It does not mandate how requirements are elicited or how architecture is designed, except that architecture needs to be structured so that software can be developed in a continuous way, and requirements need to be splittable to small increments.

More ambitious form of continuous delivery is continuous deployment [9], where the software is deployed to its users automatically after a change has

passed all automated tests. This makes the release of the software a non-event, thus affects release planning part of software development. However, this release strategy might not be feasible for software projects with certain business reasons [14]. In some contexts, it might be feasible to release only at specific times and some testing might need to be done manually. Continuous delivery allows these, while continuous deployment does not. Therefore, the initial scope of the dissertation is limited to continuous delivery, so that the field of software engineering can be covered more broadly.

While success with continuous delivery has been reported in web application context, e.g. at IMVU [8], Facebook [7] and Rally Software [16], some reports still note that in other contexts companies are still moving towards the state of the art [10, 19]. Therefore, a question arises: why is continuous delivery not adopted more widely outside the web application context? Are organizations just adopting continuous delivery slowly, or does there exist some constraints that inhibit practicing continuous delivery? Might it be that continuous delivery is not useful in some contexts? The main purpose of the dissertation is to provide empirical evidence to answer these questions.

To guide individual parts of the dissertation more specifically, the following research questions are asked:

1. What problems software projects face when adopting continuous integration and delivery? The outcome of this research question is a list of problems that should be studied further.
2. What are the causes for those problems? The outcome of this research question is a causal network of causes and problems.
3. What solutions can be used for solving those problems? The outcome of this research question is a list of solutions to the found problems.

Answering these questions contributes to the knowledge of continuous delivery and is beneficial for practitioners who wish to introduce continuous delivery in their organization or projects. The focus of the research questions is on software projects instead of organizations, because it is possible that suitability of continuous delivery can vary between projects.

3 Literature review

Previously, multiple studies of continuous integration have been published that introduce a technical concept that solves a certain problem [20, 13, 4]. This dissertation takes a different route and empirically studies real-life software projects.

There has been increased interest in empirical research on continuous integration. Downs et al. [5] studied how continuous integration affects awareness and communication. Their research shows that communication and awareness are an important part of continuous integration practice, which is useful background information for the dissertation. Ståhl and Bosch studied what positive effects does continuous integration have and how differences in continuous integration implementations can be modeled [19]. They argue that implementations

and effects of continuous integration vary between projects. This dissertation continues their research by asking why and when do the continuous integration implementations differ. Eck et al. [6] studied how organizations assimilate continuous integration practice. The focus on this dissertation will be on individual software projects instead of organizations.

To our knowledge, there has been no empirical research done that addresses continuous delivery directly. On the other hand, continuous deployment has been addressed in research recently. Olsson et al. [17] created a stairway model through which organizations can transform to accomplish continuous deployment. Claps et al. [2] examined challenges and mitigation strategies when adopting continuous deployment in an organization. Again, our focus is on software projects and not on organizations.

4 Research methodology

Instead of relying on a previous theoretical framework, an inductive approach is taken and the theory is built based on existing literature on the subject. First part of the dissertation will be a systematic literature review (SLR). Guidelines by Kitchenham et al. [15] will be followed to establish and validate a reliable method for including relevant studies. Data will be extracted with qualitative coding and thematic synthesis [3] will be used to synthesize the studies. There are not many scientific studies of continuous delivery or even of continuous integration, but multiple experience reports about the subjects exist. However, the reliability of experience reports is not good enough for scientific knowledge. Therefore the goal of the SLR is only to form initial hypotheses for the research questions which are tested in the later studies.

The results of the SLR will be compared to the results of a survey which has been done to understand continuous delivery within Finnish IT companies. To further study individual challenges, case studies (CS) will be performed. Case study is a suitable research strategy if the form of the research question requires deep understanding about the subject, which is the case in the dissertation. Methodology provided by Yin [22] and guidelines for software engineering context by Runeson and Höst [18] are followed. Case study is an appropriate research strategy when a contemporary phenomenon is studied in its context and there is no distinct boundary between the phenomenon and the context [18]. In our study, the unit of analysis is continuous delivery discipline and the context for it is a distinct software project. Since software development practices are often adapted, it would be infeasible to study continuous delivery without taking the software project itself into account.

For the case studies, data collection methods can include surveys, interviews, observation and documents. Multiple methods are needed to triangulate findings in a single case, and later on multiple-case studies are conducted to generalize results. Both quantitative and qualitative data are expected from the case studies, because quantitative data is available from continuous integration systems while interviews and observation methods produce qualitative data. Therefore mixed

methods are used for data analysis. Data sources are Finnish software companies in the Digile Need for Speed program that continues until 2017. Thus, there is an opportunity to conduct long-term longitudinal case studies of continuous delivery adoption. The selection of data sources is done partly based on convenience, but also according to the research goals. The subject of the research program is tightly related to continuous delivery, so it is expected that the case companies are also valid subjects for the studies.

5 Results and future research agenda

Systematic literature review has been started in July 2014 and studies have been selected. Used search string was "("continuous integration" OR "continuous delivery" OR "continuous deployment") AND software". From five databases, 526 search results were attained. After removing duplicates and totally irrelevant articles, 251 remained. Next, abstracts were read and inclusion criteria were applied, which yielded 85 articles. After analyzing full texts and executing backward snowballing, 27 articles remained. The data is extracted by doing qualitative coding, which is in progress.

Proposed future studies and schedule are summarized in Table 1. First, background understanding is gained with the ongoing SLR (1.). From previous research, we can synthesize different problems (RQ1), causes for the problems (RQ2) and solutions for the problems (RQ3). Second, an initial case study will be conducted (2.) to refine the results of the SLR and the methodology of the dissertation. The case study will investigate a single context more deeply (RQ2) and it will be extended to longitudinal (4.), because the continuous delivery practices are under constant improvement (RQ3) in the selected case. Another study is made of the improvement process (3.) in multiple cases to formulate a general model for continuous delivery improvement (RQ3). Finally, the accumulated understanding is validated and generalized with a final multiple-case study (5.).

Table 1. Studies and Schedule of the Dissertation

Subject	Method	Schedule
1. What do we know about continuous delivery?	SLR	Fall 2014
2. Continuous delivery in a large software project	CS	Spring 2015
3. Model for continuous delivery improvement	Multiple-CS	Fall 2015
4. Large-scale continuous delivery transformation	Longitudinal CS	Spring 2016
5. Multiple-case study of continuous delivery	Multiple-CS	Spring 2017
Summary for the dissertation	–	Spring 2018

References

1. Beck, K.: Extreme programming explained: embrace change. Addison-Wesley Professional (2000)
2. Claps, G., Svensson, R.B., Aurum, A.: On the journey to continuous deployment: technical and social challenges along the way. Information and Software Technology (2014)
3. Cruzes, D., Dybå, T.: Recommended steps for thematic synthesis in software engineering. In: 2011 International Symposium on Empirical Software Engineering and Measurement. pp. 275–284 (Sep 2011)
4. Dösinger, S., Mordinyi, R., Biffl, S.: Communicating continuous integration servers for increasing effectiveness of automated testing. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. pp. 374–377. New York, NY, USA (2012)
5. Downs, J., Plimmer, B., Hosking, J.: Ambient awareness of build status in collocated software teams. In: 34th International Conference on Software Engineering. pp. 507–517 (Jun 2012)
6. Eck, A., Uebernickel, F., Brenner, W.: Fit for continuous integration: How organizations assimilate an agile practice. Savannah, Georgia, USA (2014)
7. Feitelson, D., Frachtenberg, E., Beck, K.: Development and deployment at facebook. IEEE Internet Computing (2013)
8. Fitz, T.: Continuous deployment (Feb 2009), `http://timothyfitz.com/2009/02/08/continuous-deployment/`
9. Fitz, T.: Continuous deployment at IMVU: Doing the impossible fifty times a day (Feb 2009), `http://timothyfitz.com/2009/02/10/continuous-deployment-at-imvu-doing-the-impossible-fifty-times-a-day/`
10. Forrester Research: Continuous delivery: A maturity assessment model (Mar 2013), `http://info.thoughtworks.com/Continuous-Delivery-Maturity-Model.html`
11. Fowler, M.: Continuous integration (May 2006), `http://martinfowler.com/articles/continuousIntegration.html`
12. Fowler, M.: ContinuousDelivery (May 2013), `http://martinfowler.com/bliki/ContinuousDelivery.html`
13. Guimarães, M.L., Rito Silva, A.: Making software integration really continuous. In: Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering. pp. 332–346. Berlin, Heidelberg (2012)
14. Humble, J.: Continuous delivery vs continuous deployment (Aug 2010), `http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/`
15. Kitchenham, B.: Guidelines for performing systematic literature reviews in software engineering. Tech. rep., Keele University Technical Report (2007)
16. Neely, S., Stolt, S.: Continuous delivery? easy! just change everything (well, maybe it is not that easy). In: Proceedings of the 2013 Agile Conference. pp. 121–128. Washington, DC, USA (2013)
17. Olsson, H., Bosch, J., Alahyari, H.: Towards r&d as innovation experiment systems: A framework for moving beyond agile software development. In: IASTED Multiconferences - Proceedings of the IASTED International Conference on Software Engineering, SE 2013. pp. 798–805 (2013)
18. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14(2), 131–164 (2009)

19. Ståhl, D., Bosch, J.: Automated software integration flows in industry: A multiple-case study. In: Companion Proceedings of the 36th International Conference on Software Engineering. pp. 54–63. New York, NY, USA (2014)
20. Van Der Storm, T.: Backtracking incremental continuous integration. In: 12th European Conference on Software Maintenance and Reengineering, 2008. pp. 233–242 (Apr 2008)
21. West, D., Grant, T.: Agile development: Mainstream adoption has changed agility. Forrester Research (2010)
22. Yin, R.K.: Case study research: Design and methods, vol. 5. Sage publications, second edn. (1994)

Measurement in Software Startups[1]

Sohaib Shahid Bajwa
Faculty of Computer Science,
Free University of Bozen – Bolzano,
Bozen (Bolzano), Italy
bajwa@inf.unibz.it

Abstract. This PhD research plan aims to present the proposed research on measurement in software startup contexts. It describes the motivation behind conducting this study and also proposes the research questions. This plan also explains the research methods that will be employed. The preliminary results, after conducting interviews with several early stage software startups, show that early stage software startups do not collect measures during software development. However, they collect measures related to the usage of software using automated tools at their earlier stages. Our next step is to explore measurement in established software startups.

1 Research Area

The discipline of my research work is Software Engineering (SE). This research work has following sub areas: Software Startups, Software Measurement (Product metrics, Management metrics etc.) and Software Business.

2 Research Questions

This PhD study is at the earliest stage. We have proposed the following research questions:

RQ. How do software startups measure (product, customer development, progress)?

RQ 1. What are the measures that software startups collect?
RQ 2. Why do software startups collect these measures?
RQ 3. How do software startups collect and use these measures?
RQ 4. How often do software startups collect these measures?

The aim of this study is to explore the measurement in software startups. This study would help us to identify and understand the information needs of software startups. The study would also propose different measures that software startups need

[1] Supervisors: Prof. Pekka Abrahamsson (pekka.abrahamsson@unibz.it) and Dr. Xiaofeng Wang (xiaofeng.wang@unibz.it)

to collect in order to make better decisions. Empirical studies will be conducted to analyze the usefulness of these measures for software startups.

The expected outcome of this PhD would be a conceptual framework that will help software startups to collect, analyze and utilize different measures according to their information needs.

3 Rationale and Significance of the Study

Software startups are established worldwide in a very large number every day [1]. It is relatively easy to start a software startup nowadays due to technology advancement, smartphone, cloud infrastructure etc. Software companies e.g. Facebook, Instagram, LinkedIn, Spotify etc. are some examples of successful software startups in modern time. These software companies were initially a startup but later become a well-established business. However it does not mean that all software startups are successful. According to [2], 98% of software startups fail.

It is important to understand what exactly a startup constitutes. Ries [3] defines that a startup is a human institution which is designed to create a novel product or service under extreme uncertainty. A startup has the following characteristics:

Young and Immature: A software startup has a little or no operational history.

Lack of resources: A software startup has limited resources both in terms of funds and technical skills.

Multiple Influences: Customers, investors, and/or competitors might influence the development process.

Dynamic Technologies and Markets: Technology is changing rapidly especially in IT industry e.g. new network technologies, computing technologies etc.

Software startups try to develop products which are new and useful at the same time. They do not know whether they would be successful or not. They lack clear requirements. They have high time pressure, limited resources and tight deadlines. It is difficult to find right skillful people for software startups. All above-mentioned factors make startup a challenging but exciting field.

The role of software in economy has become increasingly crucial nowadays [5]. A larger number of startups are established nowadays and most of these startups are software startups. According to [6], startups create an average of 3 million new jobs annually only in US. The increasing popularity of software startups demand extensive research in this area. Academia has not paid proper attention to address the different problems and challenges that software startups face. Very few empirical studies are conducted related to software startups.

Steven Blank, one of the pioneers of software startup research, has developed a customer development process which is described in detail in "*The Four Steps to the Epiphany*" [7]. According to him, customer development and product development are two different but highly co-related concepts that need to be addressed separately [7]. In order to attract right market and validate idea, customer development process should be separated but aligned with the product development process.

Software startups have been applying agile methods to develop products. It is important to note that agile methods are applied when a problem is fairly understood

[8]. In a startup contexts, neither the problem, nor the solution is understood. Another approach is to apply lean principles in software startups. Ries [3], introduced the idea of Lean Startup which is based on lean manufacturing principles and Steven Blank's customer development model. The two key terms that are related to Lean Startup are [3]:

Minimum Viable Product: An incomplete product that displays different functionalities. It helps to assess customer value.

Pivot (or persevere): Pivot is a point where a software startup company changes direction. Persevere means that they would continue with the current strategy.

Many successful software startups are continually challenged by this stage of pivoting [8]. They do not end with what they have initially started. We find few case studies that have successfully implemented lean in software startups. In [9], lean methodology is applied in software startups and results are promising.

There is not much literature available on software startups. Recently, a systematic mapping study [11] shows that there is a lack of primary studies related to software development in startups [11]. In [8], the authors have proposed a software development model for early stage software startups. This model is based on the lean principles and helps software startups to make decisions e.g. when to abandon an idea or when to move forward with it. Another study [12] describes the industry-academia collaboration to create MVP. According to it [12], industry-academic collaboration reduces the company specific risks when testing customer value. In [18], the failure factors of early stage software startups are identified. Many of the software startups fail because they heavily focus on solution, rather than focusing on potential customers.

Measurements help us to control, estimate and improve process, project and product [10]. On the other hand, software measurement helps organizations to estimate and predict software characteristics to support better decisions [19]. In [19] [20], the current state of software measures is addressed. These studies have not discussed any measure/metrics related to software startups. One approach to collect measures is the Goal Question Metric (GQM) approach – which suggests that one should collet only goal related measures [21].

Software startups need to answer many questions quickly e.g. what features should be included in a MVP? When and how to pivot or persevere? At what stage we need to accelerate our progress? Are we going in the right direction? Software startups need to continuously monitor and understand their current stages. Without having a right set of measures/metrics, it becomes very difficult to make right decisions especially in software startups where they have short time and limited resources. One cannot understand current stage, measure progress, or make better decisions until starting collecting metrics. What to measure, when to measure and how to measure in software startup contexts are not addressed appropriately. Software startups should collect measures/metrics that are very critical to their business.

Measurement in established companies is discussed in the literature, but it's not yet explored in software startups. Software startups operate differently than established and mature software companies. This different nature and the dynamic behavior of software startups demand that measurement in software startups should be explored.

According to the best of our knowledge, measurement in software startup contexts is not yet explored. There are some studies related to software startups recently, but

none of these studies have discussed the measurement in software startups. In 2013, a book [13] was published that provided different key measures/metrics that were essential for the success of business in a startup. According to the book [13], measures should be collected keeping in mind the business type and the stage of startups.

The primary purpose of this study is to explore measurement in software startup contexts. This study investigates both software related measures and business related measures. We will also investigate how measurement is done in early stage software startups and in established software startups.

4 Research Methodology

Research methods include qualitative, quantitative or mix methods. We use a mixed method that includes both qualitative and quantitative research methods [14].

A thorough literature review will be carried out to synthesize prior work addressing measurement in software startups, and the consideration of different measures/metrics in making decisions in software startup contexts. The literature survey will cover material that has appeared in scholarly articles, journals, books and web references (published primarily by the ACM, the IEEE, Elsevier, Springer and Wiley). We will conduct exploratory interviews (preliminary study) to understand the information needs of software startups.

According to Wohlin et al. [16], survey, case study and experiment are the three main empirical research strategies in the SE field. A detailed web survey will be carried out to investigate the current measures/metrics that software startups collect for different purposes. This web survey will be in the form of semi-structured questionnaire. The questionnaire will contain both the close-ended and open-ended questions. The web survey would also allow us to obtain insights to the rationales behind collecting these measures. In addition, it will help us to understand when and how often software startups collect these measures.

We will also conduct multiple case studies on software startups. The main purpose of conducting the case study is to have a deeper understanding of measurement in software startups.

A software startup has a dynamic nature. Taking into account the dynamic behavior of software startups, the case study will be a cross-sectional study. The cross-sectional study takes a snapshot of a research phenomenon at a particular point in time. One can study and compare different groups (software startups) at a specific point in time.

4.1 Research Benefits

This study will enable researchers and practitioners to understand the information needs of software startups, and the challenges that software startups face. It will also help software startups to better understand their current stages, measure progress and better support their decisions.

The following table shows the time plan of the PhD work:

Table 1: Activities and Duration (Months)

Activity	Duration (Months)
Literature review	7 Months
Exploratory interviews	5 Months
Survey – design and data collection	4 Months
Survey – analysis	4 Months
Case study – data collection	4 Months
Case study – analysis	6 Months
Prepare and revise thesis	6 Months
Total	***36 Months (3 Years)***

(Note that while the activities listed above are shown as distinct and occurring in sequence, there will naturally be overlap among them).

5 Preliminary Findings

We have conducted four interviews in three early stage software startups as the pilot study. These interviews are transcribed and coded using open coding [15]. We have used an online tool, *dedoose* [17] to code and analyze data. The preliminary findings are:

- Early stage software startups do not collect measures related to product during software development.
- They collect measures regarding the usage of the application (no. of active users, no. of new users per week etc.). They use different tools (google analytics etc.) to collect measures.
- The decision on which features to include in their MVPs is based on their intuitions.
- The decision about whether to pivot or persevere is based on the feedback from the customers.
- There is no software related measure to know the current state of the software and to measure the progress.
- They do not have specific methodology to achieve their visions.

The results also show that early stage software startups face many challenges. Lack of time, lack of technical skills, initial funding, and teamwork are major challenges. Due to these challenges, it may not be feasible for them to spend time and resources to identify and collect the related metrics according to their information needs.

6 Next Step

Out next step is to carry out an industrial survey in established software startups. This industrial survey would help us to understand the current state of practice related to measurement/metrics in established software startups.

References

1. Smagalla, D.: The truth about software startups, MIT Sloan Manage. Rev. (USA), vol. 45, no. 2, p. 7, (Winter 2004)
2. Mullins, J., Komisar, R.: Getting to Plan B: Breaking Through to a Better Business Model. Harvard Business Review Press (2009)
3. Ries, E.: The Lean Startup: How Constant Innovation Creates Radically Successful Businesses. Penguin Group, London (2011)
4. Sutton, S. M.: The role of process in software start-up. IEEE Software, Vol. 17, Issue. 4, pp. 33–39 (2000)
5. Andreessen, M.: Why software is eating the world. [Online]. Available: http://goo.gl/6CEVN (Accessed: Sep. 08, 2014)
6. Kane, T.: The Importance of Startups in Job Creation and Job Destruction, Kauman Foundation, Tech. Rep., July (2010).
7. Blank, S.: The Four Steps to the Epiphany: Successful Strategies for Products that Win, 3rd ed., Cafepress.com. (2005)
8. Bosch, J., Olsson, H. H., Björk, J., Ljungblad, J.: The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups, Lecture Notes in Business Information Processing (LNBIP), Vol 167, pp 1- 15, (2013)
9. Taipale, M.: Huitale – A Story of a Finnish Lean Startup, Lecture Notes in Business Information Processing (LNBIP), Vol. 65, pp. 111- 114, (2010)
10. Fenton, N., Pfleeger, S.L.: Software Metrics: A Rigorous & Practical Approach 2nd edn. PWS Publishing Company (1997)
11. Paternoster, N., Giardino, C., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software development in startup companies: A systematic mapping study. Information & Software Technology, Vol. 56, Issue 10, pp. 1200-1218, (2014)
12. Münch, J., Fagerholm, F., Johnson, P., Pirttilahti, J., Torkkel, J., Jäarvinen, J.: Creating Minimum Viable Products in Industry-Academia Collaborations. Lean Enterprise Software and Systems, 137-151. (2013)
13. Croll, A., Yoskovitz, B.: Lean Analytics: Use Data to Build a Better Startup Faster. O'Reilly Media Inc. (2013)
14. Creswell, W.: Research Design - Qualitative, Quantitative and Mixed Method Approaches, Sage Publications. (2002)
15. Corbin, J., Strauss, A.: Grounded theory research: Procedures, canons, and evaluative criteria. Qualitative Sociology, Vol. 13, No. 1, pp. 3–21, (1990).
16. Wohlin, C., Runeson, P., Host, M., Ohlsson, M. C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers, ISBN: 0-7923-8666-3, (2000)
17. Dedoose tool [Online]. Available: www.dedoose.com (Accessed: Sep. 08, 2014)
18. Giardino, C., Wang, X., Abrahamsson, P.: Why Early-Stage Software Startups Fail: A Behavioral Framework. ICSOB, pp 27-41. (2014)
19. Gómez, O., Oktaba, H., Piattini, M., García, F.: A Systematic Review Measurement in Software Engineering: State-of-the-Art in Measures. Communications in Computer and Information Science Vol. 10, pp 165-176, (2008)
20. Kitchenham, B.: What's up with software metrics? - A preliminary mapping study. J. Syst. Softw. Vol. 83, no. 1, pp 37-51 January (2010)
21. Basili, V. R., Caldiera, G., Rombach, H. D. The goal question metric approach. 1-10. (1996)

Research Plan: Visualizations for Software Analytics

Author: Anna-Liisa Mattila
Supervisor: Tommi Mikkonen

Department of Pervasive Computing,
Tampere University of Technology,
Korkeakoulunkatu 1, FI-33720 Tampere, Finland
anna-liisa.mattila@tut.fi

Abstract. In this research proposal visualizing software engineering data in order to get better overall picture to the software project is described. Using visualizations in the initial step to form metrics can reduce amount of data that needs to be collected and analyzed later thus lowering the costs of software analytics. Also finding better metrics and understanding the big picture of the project is an expected impact. Proverb *"'What you measure is what you get"'* is many times true so measuring the right things for each projects is extremely important.

Keywords: Visualization, Software Analytics, Software Process Improvement, Research Plan, Research Proposal

1 Introduction

In software engineering there are lots of questions where companies and practitioners would like to get answers. Questions vary from enhancing software process to how customers actually use the product [2]. In order to answer these questions data from software process as well as from end users is collected, analyzed, and used as a basis for decision making and process improvement.

Data collected from the software process can be anything from version control logs to bug reports and mailing list conversations [7]. Data amounts collected are usually large and data types diverse. Analysis of the data is not a simple task. Data visualization can play important role in getting insights from these data sets as visualizations can present large amount of information in relatively small space [12]. From good visualization abnormalities, patterns and noise can be detected easily with human visual perception [5]. Interesting findings can be then analyzed further using other analysis methods. In this paper research about visualizing software engineering data in order to get better over all pictures for the projects is presented.

The paper is structured as follows. In Section 2 overview to the research area is given based on current literature. Research objectives are discussed in Section 3 and methods for conducting the research are described in Section 4. In Section 5 results and future work are addressed. In Section 6 the final conclusions are drawn.

2 Background

Software analytics have become an interesting and important topic in software engineering community as gathering data from software process is getting more common. Gathering data, like amount of code lines, and analyzing it to discover code complexity and building quality metrics is not exactly a new trend [3]. However the direction is for more extensive data collection to get deeper understanding of the process and to have actual data behind the decision making process [4].

There are many aspects to consider when initializing software analytics to a project. Data collection is a thing that needs to be adressed. How it is done so that the return of investment is good enough [9]? Collecting data from the software process by hand is tedious and error prone but on the other hand building automated data collection and analysis system costs. Many tools used in software development already collects data from the process: bug tracking systems, version control systems, etc. It can be worthwhile to actually use the data from these tools as a basis for the analytics. However, different stake holders have different questions to ask from the data. Even if we are able to answer the questions based on the already collected data, many different views to the data might be needed [1]. Good visualizations to the data plays important role in understanding the collected data and in determining which data is actually worth of collecting and analyzing.

3 Research Objectives

Objectives of the research are exploring the use of visualizations in software data analysis to gain better insight to the software project's state. Expected results are generalizable guidelines for visualizing variety of software engineering data in order to show improvement points in the software process and to determining what data is interesting for further analysis and usable as a metric.

The main research questions of the proposed research are:

Q1 How visualizations should be applied in order to show the interesting findings of software data?

Q2 How visualizations could be used in finding the right metrics?

Our aim is to work in collaboration with Finnish software companies which can provide software engineering data from their projects. Software engineering data is always somehow dependent on its context. Companies use different tools and processes which affects to the data that can be collected. There is lot research to do in order to do generalized guidelines or models for selecting metrics and visualizing the data in a useful way. Our plan is to apply different kind of visualization methods to variety of software engineering data.

Defining the metrics is important part of taking software analytics in to the software process. Proverb *"'What you measure is what you get"'* is often true

thus understanding the big picture of the project is important before determining metrics for the project. Visualizations can give good overall picture to the software project to which the metrics are designed. The suggested approach for determining metrics is iterative bottom up approach – the tools used in the software projects already collects data from the projects. The data, already existing, is visualized to form an overall picture of the project. Reasonable data sources and metrics for measuring the project is formed based on the knowledge gained from the visualizations. The metrics are formed iteratively because when knowledge from the visualization is gathered, the questions we would like ask from the data might change or become irrelevant and another questions might arise from the data.

4 Research Methods

The main research methods used in the proposed research are action research and case study. For ground survey a literature review following guidelines of systematic literature review introduced in [6] is conducted. The research is divided into three phases: 1) Pre study to get insights and hypothesis from the stake holder companies. 2) Developing visualizations and determining metrics in action research cycle. 3) Post study to conclude the findings of the research.

Action Research is doing research rather in the environment where the results are actually used than in the laboratory [8]. Action research is a cycle process that contains five steps: *diagnosing*, *action planning*, *action taking*, *evaluating* and *specifying learning*. The problem is defined in the diagnosing phase which is the starting point of the cycle. Action research is an iterative process and output of a cycle acts as an input for next iterations diagnosing phase. These steps are defined by Susman and Evered in [11].

Our aim is to do the proposed research with Finnish companies in their real software development process context. The action research is selected as one of the research methods to provide the most valid results which benefits participants the most.

Case Studies are observation studies that can be used for many purposes from generating hypothesis to explanation seeking as well as improving an aspect of the studied phenomena [10]. Whereas action research's aim is to take action and influence to the process studied in case study the process is just observed without taking action. We aim to do case studies with all stake holder companies as pre studies for gathering hypothesis for the research. The results from the case studies are used as an input for the first action research iteration. Also post studies to conclude the results from research done with companies is done as case study.

5 Current State and Future Work

The research has started at the beginning of the year 2014 and it is in its early state where no results have yet been published. Systematic literature review of

visualization methods and tools is ongoing work. Three industrial cases have been established with Finnish companies. First results from the cases are planned to be published at the beginning of 2015. Research questions introduced are expected to be answered within the year 2018.

6 Conclusions

In this paper research about using visualizing software engineering data is introduced. The objectives of the research are to study how visualizations should be applied in order to show the overall picture of the software project and how visualizations could be used in forming metrics for the project.

Use of visualizations in software analytics to determine what is actually interesting can make taking software analytics as part of the software process more feasible to the companies. Visualizations can guide forming the right metrics and thus decrease the amount of data which needs to be collected and analyzed, but also give better overall picture of the project.

References

1. Baysal, O., Holmes, R., Godfrey, M.W.: Developer Dashboards: The Need for Qualitative Analytics. Software, IEEE 30(4), 46–52 (2013)
2. Begel, A., Zimmermann, T.: Analyze This! 145 Questions for Data Scientists in Software Engineering. In: Proceedings of the 36th International Conference on Software Engineering. pp. 12–23. ICSE 2014, ACM, New York, NY, USA (2014)
3. van Genuchten, M., Hatton, L.: Metrics with Impact. IEEE Software 30(4), 99–101 (2013)
4. Hassan, A.E., Hindle, A., Runeson, P., Shepperd, M., Devanbu, P., Kim, S.: Roundtable: What's Next in Software Analytics. Software, IEEE 30(4), 53–56 (2013)
5. Keim, D.A.: Information Visualization and Visual Data Mining. Visualization and Computer Graphics, IEEE Transactions on 8(1), 1–8 (2002)
6. Kitchenham, B.: Procedures for Performing Systematic Reviews. Keele, UK, Keele University 33, 2004 (2004)
7. Menzies, T., Zimmermann, T.: Software Analytics: So What? Software, IEEE 30(4), 31–37 (2013)
8. Reason, P., Bradbury, H.: Handbook of Action Research: Participative Inquiry and Practice. Sage (2001)
9. Robbes, R., Vidal, R., Bastarrica, M.: Are Software Analytics Efforts Worthwhile for Small Companies? The Case of Amisoft. IEEE Software 30(5), 46–53 (2013)
10. Runeson, P., Höst, M.: Guidelines for Conducting and Reporting Case Study Research in Software Engineering. Empirical software engineering 14(2), 131–164 (2009)
11. Susman, G.I., Evered, R.D.: An Assessment of the Scientific Merits of Action Research. Administrative science quarterly pp. 582–603 (1978)
12. Tufte, E.R., Graves-Morris, P.: The Visual Display of Quantitative Information, vol. 2. Graphics press Cheshire, CT (1983)

Evaluating and managing technical debt in software development lifecycle

Jesse Yli-Huumo, Andrey Maglyas (supervisor), Kari Smolander (supervisor)

Lappeenranta University of Technology, Finland
{jesse.yli-huumo,andrey.maglyas,kari.smolander}@lut.fi

Abstract. Increasing competition within software industry is forcing companies to develop their products faster to market in order to acquire customers. Balancing the idea of releasing poor-quality software early or high-quality after deadline is challenging for companies. Taking shortcuts and workarounds in development can give companies the needed speed to release their product in time. However, if these shortcuts are never paid back, they might show up as omitted quality and extra costs in the future. This research is studying the challenges between development and deadlines that can also be called as 'technical debt'. We are interested on the causes of the technical debt to the software development lifecycle and the effects occurring from it. Moreover, the focus in on evaluation and management strategies regarding controlling and reducing technical debt. Our goal is to create a theoretical model about the evaluation and management of technical debt in software development lifecycle. We use grounded theory method for creating a theoretical model through several case studies and field interviews with professionals from both technical and business background. As a result of the research, we will have a theoretical model of technical debt evaluation and management that can be applied to practice for improving companies internal and external processes that will help to create high-quality products on time and on budget.

1 Background

The competition within the software industry has been increasing and new services, solutions and innovations are being brought to the consumers constantly. To obtain the market share companies must think about time-to-market strategy to gain advantage over competitors. Competition might drive companies to situation where they need to omit quality and take shortcuts in different phases of software development life cycle in order to meet these time-to-market demands and to acquire customers. Taking shortcuts and speeding development might give a company the needed advantage, but they also incur 'debt'. If this debt is never paid it accumulates and might cause some serious effects to the product and the company. This phenomenon in software development is called "technical debt".

The term technical debt was first introduced in 1992 by Ward Cunningham [1]. He explained a situation where long-term code quality is traded for a short-term gain. Deficiencies in software can be compared with financial debt [2]. Implementing shortcuts to the system architecture incur 'debt' that must be paid back eventually. If this debt is

not properly managed, it might accumulate as 'interest,' affecting the overall quality of the developed software systems. Although technical debt has negative consequences in a long-term, it can be used as a competitive advantage in a short-term. Time-to-market and constant customer feedback through releasing software faster than competitors allow companies to gain a bigger market share.

Often technical debt is equated with shortcuts and workarounds in the source code of the software. However, taking shortcuts and workarounds can happen in different stages [3] of software development life cycle [4]. Lack of documentation [5] or lack of requirements specification [6] can increase technical debt in the requirements phase. Architectural flaws in the design phase can be seen as design debt [7]. In the testing environment shortcuts in running and writing test cases can also incur testing debt [5].

Technical debt is not always a shortcut or workaround that is done with intentional decision. Instead, according to McConnell technical debt can be divided into an intentional and unintentional debt [8]. Intentional technical debt occurs when organization makes a conscious decision to incur debt that might come from the pressure to release the product. Unintentional technical debt is non-strategic and usually a result of a poor job or bad decisions done unknowingly that might reveal themselves after years.

This research is a part of the Need for Speed -program. The aim of the Need for Speed is to create the foundation for the success of the Finnish software intensive business in the new digital economy. The research program focuses on business models, tools and processes that help companies speed up their business to proactively take advantage of the real-time economy. Our research is focusing on bringing an empirical data about the challenges between managing the development and deadlines.

2 Research objectives

The current literature identifies several different technical debt causes and effects. Technical debt is not always caused by technical reasons, but also for business reasons [9]. In a short-term technical debt has often positive effects such as the time-to-market advantage [9]. If this debt is not managed, it tends to turn to economic consequences and quality issues in a long-term [10][11]. There are also some cases where the short-term benefit overweighs the cost of long-term consequences [12]. However, the current literature is lacking a clear mapping between the causes and effects of technical debt and its sub-categories.

Evaluating and managing is an important part of reducing and paying back the technical debt. There are some strategies and practices suggested in the current literature for dealing with technical debt. In the portfolio management [13] technical debts are collected to a 'technical debt list' that is being used to reduce the technical debt based on their cost and value. There are also some guidelines [14][15] developed for improving refactoring, coding and teamwork that might reduce the amount of technical debt. However, the current literature includes very few evaluation and management strategies for technical debt and lacks empirical evidence.

Based on these observations, the aim of this research is to investigate a topic that has not been studied a lot in previous research: Evaluation and management of technical debt in software development lifecycle. The topic is important because technical debt

is an essential part of current software development and companies must be able to manage their technical debt in order to keep the product healthy and profitable in competition. Our focus in this research is more on the management side of the technical debt causes and effects, rather than on the qualities of technical debt in source code and how to measure them.

Technical debt in software development lifecycle in almost inevitable and this is the reason why companies must be able to evaluate and manage it. Research in this area has not been much carried out and this study should improve the research gap. In this regards, the main research question will be **"How to evaluate and manage technical debt in the software development lifecycle?"** The objective of this research is to create a theoretical model which describes the role of technical debt evaluation and management in the software development lifecycle. It also helps to understand the causes and effects of technical debt to software development. The main research question is divided into sub-questions that will focus on the certain aspects of the research problem.

1. What are the causes and effects of technical debt in software development life cycle?
2. What management and technical perspectives can be used in the evaluation and/or management of technical debt?
3. What are the current management strategies and practices for managing and reducing technical debt in software life cycle?

Combining the information gathered during the research will help us to create a theoretical model of the technical debt evaluation and management in software development lifecycle. This model will describe the evaluation and management of technical debt with practices that help companies to manage and reduce it. In addition, we will discuss the role of technical debt in software development lifecycle with causes and effects of it. Companies must be able to change their practices and processes more suitable for dealing with the technical debt. As a result of the research, we will have a theoretical model of technical debt evaluation and management that can be applied to practice for improving companies' internal and external processes that will help to create high-quality products on time and budget.

3 Research methodology

To answer the main research question, sub-questions and to develop a model for the evaluation and management of technical debt, we have decided to use qualitative research and grounded theory method that was developed by Glaser and Strauss in 1967 [16]. The grounded theory is built on two main concepts: constant comparison and theoretical sampling. With constant comparison in grounded theory we are able to have simultaneous involvement in data collection and constructing analytic codes and categories from the data. Theoretical sampling creates an iterative theory construction process, where the next data sample is chosen based on the analysis of previous sample.

Answering the research questions requires also qualitative research which will include case studies and field interviews. Due to the nature of the project, the selected

companies for the research are primary dictated by a list of partners in the research project. Since our goal is to increase our knowledge of the relationship in technical debt causes, effects, evaluation and management rather than the measurement and qualities of technical debt in source code, we decided to use case study methodology with semi-structured interviews for data collection. This research will include multiple case studies with face-to-face interviews from professionals with technical and business background related to each case.

4 Results

So far we have conducted total of 30 interviews from four different companies with professionals from both technical and business side. The cases include both middle and large-size companies from different industry areas. We have also two other case studies planned for the future research.

We have made one publication of the gathered results that has been accepted to PROFES 2014 conference. In this publication we studied a one middle-size software development company and their two different product lines. With this study we were able to identify several different causes for technical debt. These can be further divided into technical debt that is a caused with intentional decisions and technical debt incurring unintentionally. We were also able to identify several short- and long-term effects of technical debt to a software project. As expected, the effects of technical debt seemed to be positive in a short-term, but turned negative in a long-term such as extra costs and quality issues. Although we did not find any specific approach for managing technical debt, we were able to identify some practices for reducing technical debt such as refactoring and coding standards/reviews. In addition, the company that took part in this first publication stated after the conducted research that they are going to start use similar technical debt 'list' suggested in a study conducted by Guo and Seaman [13]. In the future we are going to conduct a research about the effects of using this 'list' to manage and reduce technical debt.

In the future we are focusing on the analysis of current case studies. Even though the current results have been promising and revealed several interesting aspects of the research topic, we think that the amount of observed data is not enough at the moment for creating a theoretical model for evaluation and management of technical debt. So far we have been able to observe a lot of results regarding the causes and effects of technical debt. However, the amount of data regarding the evaluation and management of technical debt has been fairly low. Therefore, we are planning to conduct more case studies and field interviews in the future to gather more data regarding the evaluation and management aspect of technical debt.

References

1. W. Cunningham, "The WyCash Portfolio Management System," Experience Report, 1992.
2. "TechnicalDebtQuadrant." [Online]. Available: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html. [Accessed: 07-Jul-2014].

3. B. L. Yu, K. L. Wooi, Y. T. Wai, and F. T. Soo, “Software Development Life Cycle AGILE vs Traditional Approaches,” in *IPCSIT vol. 37 (2012)*, Singapore, 2012.
4. E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” *J. Syst. Softw.*, vol. 86, no. 6, pp. 1498–1516, Jun. 2013.
5. N. Zazworka, R. O. Spínola, A. Vetro’, F. Shull, and C. Seaman, “A Case Study on Effectively Identifying Technical Debt,” in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, New York, NY, USA, 2013, pp. 42–47.
6. B. Ojameruaye and R. Bahsoon, “Systematic Elaboration of Compliance Requirements Using Compliance Debt and Portfolio Theory,” in *Requirements Engineering: Foundation for Software Quality*, C. Salinesi and I. van de Weerd, Eds. Springer International Publishing, 2014, pp. 152–167.
7. N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, “Managing Technical Debt in Software-reliant Systems,” in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, New York, NY, USA, 2010, pp. 47–52.
8. S. McConnell, “Technical Debt-10x Software Development | Construx,” 01-Nov-2007. [Online]. Available: http://www.construx.com/10x_Software_Development/Technical_Debt/. [Accessed: 25-Mar-2014].
9. E. Lim, N. Taksande, and C. Seaman, “A Balancing Act: What Software Practitioners Have to Say about Technical Debt,” *IEEE Softw.*, vol. 29, no. 6, pp. 22–27, Nov. 2012.
10. N. Zazworka, M. A. Shaw, F. Shull, and C. Seaman, “Investigating the Impact of Design Debt on Software Quality,” in *Proceedings of the 2Nd Workshop on Managing Technical Debt*, New York, NY, USA, 2011, pp. 17–23.
11. Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. Q. B. da Silva, A. L. M. Santos, and C. Siebra, “Tracking technical debt #x2014; An exploratory case study,” in *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 2011, pp. 528–531.
12. C. S. A. Siebra, G. S. Tonin, F. Q. B. Silva, R. G. Oliveira, A. L. O. C. Junior, R. C. G. Miranda, and A. L. M. Santos, “Managing Technical Debt in Practice: An Industrial Report,” in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, New York, NY, USA, 2012, pp. 247–250.
13. Y. Guo and C. Seaman, “A Portfolio Approach to Technical Debt Management,” in *Proceedings of the 2Nd Workshop on Managing Technical Debt*, New York, NY, USA, 2011, pp. 31–34.
14. V. Krishna and A. Basu, “Minimizing Technical Debt: Developer’s viewpoint,” in *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA 2012)*, 2012, pp. 1–5.
15. V. Krishna and A. Basu, “Software Engineering Practices for Minimizing Technical Debt,” presented at the SERP’13 The 2013 International Conference on Software Engineering Research and Practice.
16. B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. 2009.

The Dynamics of Test-Driven Development

Davide Fucci
supervised by Burak Turhan

University of Oulu, Finland
davide.fucci@oulu.fi
burak.turhan@oulu.fi

Abstract. Test-driven development (TDD) has been the subject of several software engineering experiments. However the controversial results about its effects still need to be contextualized. This doctoral research will show how TDD could be better assessed by studying to what extent developers follow its cycle. Moreover, I aim at exploring the low level aspects of the TDD process. This knowledge is foreseen to be beneficial for software industries willing to *adopt* or *adapt* TDD.

Keywords: Test-driven development, Process conformance, software quality, developers' productivity

1 Introduction and Research Gap

Test-driven Development (TDD) has been advertised as a development practice that aids the production of high quality code in short time iterations, therefore making it appealing for both academia and industry. If the promises of TDD were to be kept in practice, it would have a big impact on the software industry; on the other hand, academics have been interested in the study of TDD to confirm whether the promised effects are real and repeatable. TDD is an incremental development practice in which tests are written before the implementation code, reversing the conventional development process. Therefore, the tests drive the implementation of production code. During the TDD process, developers (1) write a test that describes a small piece of functionality they wish their production code will have; (2) run the test to ensure that it fails; (3) write the *minimal* amount of code that necessary for the test to pass; (4) run the test(s) in order to ensure that the new implementation is correct, as well as not breaking existing functionalities; (5) refactor the code to improve the design (e.g.; removing duplications) and (6) re-run the tests to check that all the functionalities still work [2]. TDD is associated with an increased external quality [1, 2] of the code and it is believed to achieve this effect mainly by encouraging developers to support the code they write with a comprehensive suite of unit tests and keeping the code itself easy to test through refactoring. Furthermore, using TDD with an automated testing framework builds a regression test suite against which future changes to the code can be verified. Proponents claim that TDD promotes a modular, simple and emergent design thanks to the short iterations and continual refactoring in contrast with the classic, test-last approach, traditionally

focused on big up-front design [1, 2]. There is currently a gap in research about TDD. In general, its effects, have not been assessed through sound evidence in which the extent to which the developers follow the TDD cycle, i.e.; *process conformance* has been taken into account. In order to achieve such goal the process itself needs to be studied at a fine-grained level. Alongside, the results from existing research about TDD have not been transferred to the industry; therefore another gap exists regarding how the research results can help the industry stakeholders adopting and making decision about TDD in practice.

2 Prior Work

The results regarding TDD's effects are so far contradicting. Turhan et al. [16] conducted a systematic review of studies that performed a quantitative comparison of TDD with an alternative development method, such as after-the-fact testing or ad-hoc in-process testing. Their analysis suggests that the primary studies present moderate evidence in favour of TDD for improving external code quality. However, this evidence disappears when studies classified as being less rigorous were excluded. TDD's effects on developer productivity were inconsistent across different study types and levels of rigour. The primary studies also present moderate evidence that TDD improves the quality of test code and defect removal efficiency. Controversially, considering only the high rigorous and relevant studies, Munir et al. [13], found that TDD increases external quality but at the same time deteriorates productivity. Another systematic literature review by Rafique and Misic [14] included the results from the 27 primary studies showing that TDD has a small positive effect on quality but no significant effect on productivity. A survey from Begel et al. [4] involving 488 software developers, testers and project managers at Microsoft Corp., showed that even though 51% have been using TDD and another 10% was willing to adopt it in their development workflow, TDD remains one of the least used methodologies among Agile development methodologies along with pair programming. TDD was perceived as a factor mainly contributing to the higher code quality, while its overall benefits were ranked 10th among the perceived benefits brought by Agile methodologies. Many empirical studies about the effects of TDD, including the systematic reviews [5, 14, 16], reported that process conformance is often underestimated or not properly controlled by the experimenters, therefore posing threats to the validity of the studies. In the case of TDD, process conformance is the level to which the cycle is followed. In particular, the study by Causevic et al. [5] claims that inadequate process conformance might be hindering the adoption of TDD by industry. Although there are heuristics and tools [3] capable of assessing the conformance to the TDD cycle, only few studies make use of them (or other ad-hoc built solutions) to deal with the threat of low process conformance [12]. Another factor limiting the understanding, and therefore the adoption, of TDD lies in having an understanding of what are the low-level aspects of the actual process employed by the developers as opposed to the ideal one presented in the literature [1, 2]. To the best of my knowledge, the state of the art about

TDD studies does not include a in-depth analysis of the dynamics underlying the red-green-refactor cycle in practice, for both industry and academia.

3 Research Problem and Contribution

My research aims to achieve a deeper understanding of the dynamics of Test Driven Development (TDD) and to provide necessary actions in order to offer industry a more informed adoption of the practice. In the first place, I am interested in measuring how the level of conformance to the ideal practice impacts developers' productivity and the quality of the system under development. Accordingly, it is of interest to observe what actually happens when developers claim to use TDD, how the actual process deviates from the ideal one, and what are the most significant factors observed during the process.
RQ1 - How the level of conformance to TDD impacts its effects?
In turn, RQ1 is divided into:
RQ1.1 - How the level of conformance to TDD impacts external quality?
RQ1.2 - How the level of conformance to TDD impacts productivity?
The second level of analysis deals with how the TDD process takes place in practice. I have identified four components representing the process:
Granularity: the duration of each development cycle.
Uniformity: the regularity, in terms of duration, of each development cycle.
Sequencing: the dominant testing pattern in each development cycle.
Refactoring: the amount of refactoring in each development cycle.
The objective is study how each component contributes to the actual TDD process, and which have an impact on productivity and software quality.
RQ2 - What is the impact of the TDD components on its outcomes?
RQ2 is divided into sub-questions:
RQ2.1 - What is the impact of the TDD components on external quality?
RQ2.2 - What is the impact of the TDD components on productivity?
The expected contributions of my doctoral work are (1) the creation of new evidence about how, and eventually which parts of the TDD process should be employed by industry in order to bring quality and productivity improvements; and consequently (2) create guidelines regarding the adoption of the process.

4 Methodology

I employ a quantitative approach by using the low-level data collected from TDD developers' IDE by a tool, to make inference about their level of conformance to the ideal TDD process. At the same time source code is used to calculate metrics for external quality (in terms of defects) and productivity (in terms of time necessary to develop the software requirements, e.g.; user stories). The dataset composed of such metrics is then analysed, seeking for relationships in order to answer RQ1. On the other hand, the same data is analysed to measure each of the four actual TDD components to answer RQ2. I am gathering and analysing data from industrial partners of my research group, who are willing to use, or are

already using, TDD. In order to validate my approach, I first collected and analysed data gathered from Master's students in Information Processing Science at University of Oulu using TDD in their courses. The findings from RQ1 and RQ2 will be processed into guidelines to provide practical support to practitioners. Specifically, my research work will be approached through experimental research methods. A set of quasi-experiments [15] (i.e. a baseline experiment and its replications) are appropriate means to investigate the research questions. The choice of quasi-experiment is a compromise [11] due to the fact that the constructs of interest cannot be artificially manipulated as in controlled experiments. In order to have a richer understanding of the TDD process in practice and to overcome the threat for causal inference posed by the use of quasi-experiment, surveys and field observations will be used along with the experiments, to gauge the subjects' skills, experience and attitude towards TDD. Such kind of assessments could be useful to get more insights, support to the experiments' results as well as drive the focus of each experiment replication.

5 Progress Status and Agenda

At the moment four studies have been carried out [7–10]. The first two were replications of the experiment described by Erdogmus et al [6], in which the effects of TDD on external quality and productivity was studies in academic context. The papers included also a correlation study in which the number of tests—developed by both TDD and test last developers—was used to predict the final software product quality as well as the productivity of the developers. The result of those two previously published studies is that no improvement in quality nor productivity was achieved by the adoption of TDD, while in the original study TDD seemed to improve, although slightly, the external quality. These initial results suffer from a threat to external validity since their findings are hardly generalisable to the industrial context, they provide good material for theory testing. In particular, the aim was to show that the selected variables represent the constructs presented in the TDD literature and that, although with a small sample, relationships and causality exists between such constructs. From the methodological perspective such experiments allow to validate the experimental design, and increase the confidence in using ESE techniques (e.g. data analysis, reporting). In the latter two studies, the collected data is analysed under the prospective of process conformance. This pre-study will support the design of the experiments needed to answer RQ1. Alongside, the initial result shed some light on the actual process employed by the developers claiming to follow the TDD cycle. Three experiments run, focusing on the study of process conformance, took place during Fall 2013 in industrial context. Part of the data collected will be used to test the hypotheses necessary to answer RQ2. Hence, the future work revolves around RQ2, which will be accompanied by an observational field study in order to form guidelines that could help practitioners willing to employ TDD in their workflow.

References

1. D. Astels. *Test Driven development: A Practical Guide.* Prentice Hall Professional Technical Reference, 2003.
2. K. Beck. *Test-driven Development: by Example.* The Addison-Wesley signature series. Addison-Wesley, 2003.
3. K. Becker, B. de Souza Costa Pedroso, M. S. Pimenta, and R. P. Jacobi. Besouro: A framework for exploring compliance rules in automatic TDD behavior assessment. *Inf. Softw. Technol.*, pages 1–15, July 2014.
4. A. Begel and N. Nagappan. Usage and perceptions of agile software development in an industrial context: An exploratory study. In *1st Symposium on Empirical Software Engineering and Measurement*, pages 255–264. IEEE, 2007.
5. A. Causevic, D. Sundmark, and S. Punnekkat. Factors limiting industrial adoption of test driven development: A systematic review. In *Software Testing, Verification and Validation (ICST), Fourth International Conference on.* IEEE, 2011.
6. H. Erdogmus, M. Morisio, and M. Torchiano. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.
7. D. Fucci and B. Turhan. A Replicated Experiment on the Effectiveness of Test-driven Development. In *Empirical Software Engineering and Measurement, 2013. ESEM 2013. Seventh International Symposium on*, pages 364–373. IEEE, 2013.
8. D. Fucci and B. Turhan. On the role of tests in test-driven development: a differentiated and partial replication. *Empirical Software Engineering*, 2013.
9. D. Fucci, B. Turhan, and M. Oivo. Conformance factor in test-driven development: initial results from an enhanced replication. In *18th International Conference on Evaluation and Assessment in Software Engineering*, page 22, 2014.
10. D. Fucci, B. Turhan, and M. Oivo. The Impact of Process Conformance on the Effects of Test-driven Development. In *Empirical Software Engineering and Measurement, 2014. ESEM 2014. 8th International Symposium on.* IEEE, 2014.
11. V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K Sjøberg. A systematic review of quasi-experiments in software engineering. *Information and Software Technology*, 51(1):71–82, 2009.
12. M. Müller and A. Höfer. The Effect of Experience on the Test-driven Development Process. *Empirical Software Engineering*, 12(6):593–615, 2007.
13. H. Munir, M. Moayyed, and K. Petersen. Considering rigor and relevance when evaluating test driven development: A systematic review. *Information and Software Technology*, 56(4):375–394, Apr. 2014.
14. Y. Rafique and V. Misic. The effects of test-driven development on external quality and productivity: A meta-analysis. 2012.
15. W. Shadish. *Experimental and quasi-experimental designs for generalized causal inference.* Houghton Mifflin, Boston, 2001.
16. B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull. *How Effective Is Test Driven Development?* O'Reilly Media, 2010.

Language for Choreography Modeling in Embedded Systems Domain

Candidate: Nebojša Taušan, Supervisors: Markku Oivo, Pasi Kuvaja, Jouni Markkula

Department of Information Processing Science
University of Oulu, Oulu, Finland

[nebojsa.tausan, markku.oivo, pasi.kuvaja,
jouni.markkula]@oulu.fi

Abstract. The choreography models are important artifacts for systems based on service-oriented architecture. Conventional languages for choreography modeling, however, have a limited applicability in the embedded systems domain where service-oriented architecture is increasingly present. Therefore the main problem addressed in this doctoral study is the lack of the choreography modeling language primitives needed to tackle embedded systems development challenges. This problem will be addressed through the design of customizations for an existing choreography language. The expected benefits of the language utilization include improved communication between the development teams, faster development, and error reduction.

Keywords: service-oriented architecture, choreography, design science

Research Area: Software architecture, modeling languages

1 Introduction

Service-Oriented Architecture (SOA) is a well-established approach for enterprise systems development [1], but today it is also used for the development of Embedded Systems (ES) [2]. SOA introduces the service as a main building block of the system and service interactions as a way to achieve a system's goals. Service interaction specification, therefore, represents an important development artifact which consists of two related parts—choreography and orchestration [3]. The choreography describes the sequence of service interaction steps during which the distinct management authorities (also called process participants) exchange messages. The orchestration describes service interactions from the perspective of a distinct process participant. Choreography and orchestration can be specified using general-purpose modeling languages, such as UML (uml.org), or domain-specific languages, such as those used for business process modeling in enterprise systems domain [4][5]. These languages, however, have limited applicability in the ES domain, since the nature of service interactions in ES is more complex [6][7][8]. This doctoral study will address

this problem by designing customizations for the existing choreography modeling language in a way that makes the language applicable to ES development challenges. The language customizations will provide primitives that will support (a) complex service interactions modeling, (b) role identification and management, and (c) middleware heterogeneity.

1.1 Research Problem, Research Questions and Significance

The research problem addressed in this dissertation is the lack of the choreography modeling language (CML) primitives needed to tackle embedded systems development challenges. This research problem represents a synthesis of what is published in the literature and what has been learned from the industry partners in the AMALTHEA (amalthea-project.org) project of which this dissertation is part. The identified research problem is decomposed on research questions (RQ), and for this purpose, the Wieringa's guidelines are used [9]. Following these guidelines, the problem is decomposed to three higher-level RQs and eight sub questions. These RQs are:

- RQ1: Why is choreography modeling utilized in ES development?
  a. Which practical ES development challenges can be addressed by choreography modeling?
  b. What is known in the scientific literature about the choreography modeling in ES development?
- RQ2: How to design the CML customizations for the ES development domain?
  a. How are different middleware platforms addressed with CML?
  b. How to address the ES specific interaction complexity with CML?
  c. How to facilitate the identification and management of participants' roles?
- RQ3: What effects on ES development are perceived by practitioners during the utilization of the customized CML?
  a. What are the effects of the customized CML on communication?
  b. What are the effects of the customized CML on the development speed?
  c. What are the effects of the customized CML on the number of errors?

The identified problem and its corresponding RQs are significant for several reasons. These include the relationship of the choreography with (a) better performance of a system [10][11][12], (b) network reliability and shorter network paths [13], and (c) a reduced number of bottlenecks [14]. In addition, empirical findings from the study of Lescevica et al. [15] indicate that practitioners expect improvements in orchestration and choreography capabilities.

1.2 Related Work

Languages for enterprise service interaction modeling [4][5] have a limited applicability in the ES domain [6][7][8]. Today, there are several languages that are used for service interaction in this domain. Call Control eXtensible Markup Language (CCXML) [16] and State Chart eXtensible Markup Language (SCXML) [17] are used

for telecommunication service interaction modeling. SCALE [18] and SPATEL [19] model the convergence between telecommunication services and services developed in different domains and technologies. SENSORIA reference modeling language [20] and customized Business Process Execution Language [6] are used to represent complex service interactions in the automotive domain. The TOSCA language is an emerging standard for cloud-based applications, but it is also used to model the interactions between Internet-of-Things applications [21]. The analysis of these languages, which was done in the course of this doctoral study, resulted in the conclusion that the semantics and language primitives needed for choreography modeling in ES are not sufficiently supported. This analysis result is also an argument for this dissertation work.

2 Research Methodology

This research is framed by Design Science methodology and is instantiated in accordance with the framework of Hevner et al. [22]. Therefore, this study comprises a sequence of research activities, where each activity utilizes a concrete research method(s) to explore (a) the topics relevant in practice, (b) what is known in the literature about those topics, (c) how this knowledge can be used for designing the modeling language and, (d) the benefits of language utilization. Figure 1 illustrates how the selected research activities and methods fit into the Design Science framework.

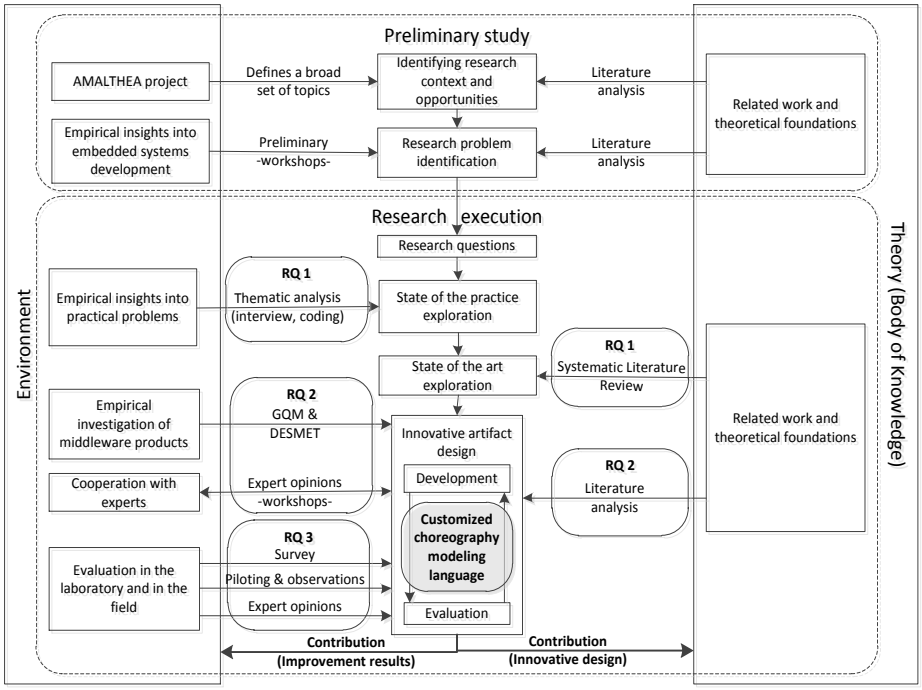

Fig. 1. Research methodology, structured based on framework from Hevner et al. [22]

The research activities and methods are divided in two groups in the center of Figure 1. The first group of activities, labeled as „preliminary study", establishes a context and identifies and analyzes the research opportunities set by the AMALTHEA project. For this purpose, the literature analysis and workshops were conducted with industry experts. The preliminary study efforts defined the research problem, which is relevant for experts and interesting for researchers.

The following group of activities is labeled as „research execution". This group is driven by the RQs derived from the identified research problem. Accordingly, to understand the practical challenges and to explore what is known in the literature about the identified problem (RQ1), the state of the practice and literature will be investigated. For these activities, research methods, including expert interviews, thematic analyses [23], and a systematic literature review [24] will be employed. Several topics will be studied to design the customizations for CML (RQ2). Methods such as the Goal Question Metric [25] and DESMET [26] will be used to investigate the influence of the middleware on CML. Methods including focus groups, company-specific document analyses, and the literature analysis will be utilized to explore and support the complex interaction modeling and role management, answer RQ2. To answer RQ3, which pertains to the evaluation of the customized CML, methods such as surveys, piloting, and observations will be used.

Finally, according to the framework of Hevner et al. [22], the design of the CML customizations will iterate over the development and evaluation activities until it is unable to contribute utility to the environment and new knowledge to the theory. The CML customizations designed in this dissertation will contribute to the software development practice by improving the communication, reducing development time, and reducing errors. The contribution to the theory will be in the form of a unique design that combines the CML with the complex service interaction support, role management, and middleware specifics.

3 Results

So far, results from the exploration of the state of the art in scientific literature, the state of the practice, and the middleware products analysis have been obtained. State of the art in scientific literature exploration provided the knowledge about the ways choreography modeling is utilized for ES development, which tools are used, what benefits can be expected and what are the research trends. State of the practice exploration resulted in the set of software architecture development challenges that can be addressed with choreography modeling [27]. Together, these challenges and the results from the state of the art in literature serve as one of the main guidepost for the design of CML customizations, and form the answer to RQ1. Exploration of the middleware products provided knowledge about the influence of different middleware implementations on CML [28]. This knowledge is used for the actual CML customization design, and partially answers RQ2.

4 Future Work

Three milestones will be sought in future work. First, the finalization of the customized CML design will provide support for (a) middleware heterogeneity, (b) role identification and management, and (c) complex service interactions modeling. Role identification and management will be supported using class-responsibility-collaboration cards technique (c2.com/doc/oopsla89/paper.html) while the complex service interactions modeling will incorporate the language for expressing and manipulation of the event-condition-action rules [29]. The second milestone is the development of the prototype that will allow user to model choreographies using the customized CML. For this purpose, the Sirius technology (http://eclipse.org/sirius/) for domain modeling will be used. The third milestone is the validation of the customized CML. Validation activities will first be conducted in the laboratory setting, after which the final CML modifications will be done. The piloting of the language with industry experts will follow. Pilot results will be collected through surveys and by recording experts' evaluations and opinions. These results will answer RQ3.

5 References

1. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: service-oriented architecture best practices. Prentice Hall Professional (2005).
2. Scholz, A., Gaponova, I., Sommer, S., Kemper, A., Knoll, A., Buckl, C., Heuer, J., Schmitt, A.: ∈ SOA-Service Oriented Architectures adapted for embedded networks. In: 7th IEEE International Conference on Industrial Informatics, 2009. pp. 599–605. IEEE (2009).
3. Peltz, C.: Web services orchestration and choreography. Computer (Long. Beach. Calif). 36, 46–52 (2003).
4. Van der Aalst, W.M.P.: Don't go with the flow: Web services composition standards exposed. IEEE Intell. Syst. 18, 72–76 (2003).
5. Barros, A., Dumas, M., Oaks, P.: A critical overview of the web services choreography description language. In: BPTrends Newsl. 3, 1–24 (2005).
6. Iwai, A., Oohashi, N., Kelly, S.: Experiences with automotive service modeling. Proceedings of the 10th Workshop on Domain-Specific Modeling. p. 1. ACM (2010).
7. Bond, G., Cheung, E., Fikouras, I., Levenshteyn, R.: Unified telecom and web services composition: problem definition and future directions. Proceedings of the 3rd International Conference on Principles, Systems and Applications of IP Telecommunications. p. 13. ACM (2009).
8. Lin, L., Lin, P.: Orchestration in Web Services and real-time communications. Commun. Mag. IEEE. 45, 44–50 (2007).
9. Wieringa, R.: Design science as nested problem solving. Proceedings of the 4th international conference on design science research in information systems and technology. p. 8. ACM (2009).
10. Pontes Guimaraes, F., Kuroda, E.H., Batista, D.M.: Performance Evaluation of Choreographies and Orchestrations with a New Simulator for Service Compositions. 17th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks. pp. 140–144. IEEE (2012).
11. Mostarda, L., Marinovic, S., Dulay, N.: Distributed orchestration of pervasive services. In: 24th IEEE International Conference on Advanced Information Networking and Applications. pp. 166–173. IEEE (2010).

12. Chafle, G.B., Chandra, S., Mann, V., Nanda, M.G.: Decentralized orchestration of composite web services. Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters. pp. 134–143. ACM (2004).
13. Cherrier, S., Ghamri-Doudane, Y.M., Lohier, S., Roussel, G.: Services collaboration in wireless sensor and actuator networks: orchestration versus choreography. IEEE Symposium on Computers and Communications. pp. 411–418. IEEE (2012).
14. Barker, A., Besana, P., Robertson, D., Weissman, J.B.: The benefits of service choreography for data-intensive computing. Proceedings of the 7th international workshop on Challenges of large applications in distributed environments - CLADE '09. p. 1. ACM Press, New York, New York, USA (2009).
15. Lescevica, M., Ginters, E., Mazza, R.: Unified Theory of Acceptance and Use of Technology (UTAUT) for Market Analysis of FP7 CHOReOS Products. Procedia Comput. Sci., vol. 26, pp. 51–68, (2013).
16. Auburn, R.J., Cafarella, M., Jackson, D., Peck, J., Sharma, P., Shanmughan, S., Stohs, C., Zhang, Y.: Voice browser call control: CCXML version 1.0,W3C (2011).
17. Barnett, J., Akolkar, R., Auburn, R.J., Bodell, M., Burnett, D.C., Carter, J., McGlashan, S., Lager, T., Helbing, M., Hosn, R.: State Chart XML (SCXML): State machine notation for control abstraction. W3C Work. Draft. (2013).
18. Niemoeller, J., Vandakas.K.,: SCALE – A language for dynamic composition of heterogeneous services, http://www.ericsson.com/res/thecompany/docs/journal_conference_papers/service_layer/101215_scale.pdf, (2010).
19. Almeida, J., Baravaglio, A., Belaunde, M., Falcarin, P., Kovacs, E.: Service Creation in the SPICE Service Platform. Proceedings of the 17th Wireless World Research Forum Meeting (WWRF17). pp. 1–7. Heidelberg: Wireless World Research Forum (2006).
20. Fiadeiro, J., Lopes, A., Bocchi, L., Abreu, J.: The Sensoria reference modelling language. Rigorous software engineering for service-oriented systems. pp. 61–114. Springer (2011).
21. Binz, T., Breiter, G., Leyman, F., Spatzier, T.: Portable Cloud Services Using TOSCA. IEEE Internet Comput. 16, (2012).
22. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. 28, 75–105 (2004).
23. Miles, M.B., Huberman, A.M.: Qualitative data analysis: An expanded sourcebook. Sage (1994).
24. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Engineering. Technical report, EBSE Technical Report EBSE-2007-01 (2007).
25. Caldiera, V.R.B.G., Rombach, H.D.: The goal question metric approach. Encycl. Softw. Eng. 2, 528–532 (1994).
26. Kitchenham, B., Linkman, S., Law, D.: DESMET: a methodology for evaluating software engineering methods and tools. Comput. Control Eng. J. 8, 120–126 (1997).
27. Taušan, N., Aaramaa, S., Lehto, J., Kuvaja, P., Markkula, J., Oivo, M.: Customized Choreography and Requirement Template Models as a Means for Addressing Software Architect's Challenges. In: 9th International Conference on Software Engineering Advances. pp.55–63. IARIA XPS Press (2013).
28. Taušan, N., Lehto, J., Kuvaja, P., Markkula, J., Oivo, M.: Comparative Influence Evaluation of Middleware Features on Choreography DSL. In: 8th International Conference on Software Engineering Advances. pp. 184–193. IARIA XPS Press (2013).
29. Sterling, L., Kuldar T.: "A Logic Programming Perspective on Rules." Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches. IGI Global (2009).