SOFTWARE ENGINEERING IN START-UP COMPANIES

ERIKS KLOTINS

Department of Software Engineering doctoral dissertation series No 2019:12

SOFTWARE ENGINEERING IN START-UP COMPANIES

ERIKS KLOTINS

Doctoral Dissertation in Software Engineering

Department of Software Engineering
Blekinge Institute of Technology
SWEDEN

Department of Software Engineering
Publisher: Blekinge Institute of Technology,
SE-371 79 Karlskrona, Sweden
Printed by Exakta Group, Sweden, 2019
ISBN: 978-91-7295-384-0
ISSN: 1653-2090

To my family

I do not think there is any other quality so essential to success of any kind as the quality of perseverance. It overcomes almost everything, even nature.

John D. Rockefeller

ABSTRACT

Start-up companies have emerged as suppliers of innovation and software-intensive products. Small teams, lack of legacy products, experimental nature, and absence of any organizational processes enable start-ups to develop and market new products and services quickly. However, most start-ups fail before delivering any value.

Start-up failures can be attributed to market factors, shortcomings in business models, lack of motivation, or self-destruction, among other reasons. However, inadequacies in product engineering precede any market or business-related challenges and could be a significant contributing factor to start-up failures. At the same time, state-of-the-art software engineering (SE) practices are often neglected by start-ups as inadequate.

At the beginning of this work, SE in start-ups had attracted very little attention from researchers. Thus, there was no coherent view of SE state-of-practice in start-ups and no starting point for a focused investigation.

In this thesis, we explore how start-ups practice SE, what specific SE challenges should be addressed, and what new SE practices are needed to support the engineering of innovative software-intensive products and services.

A substantial part of this work is exploratory and aimed to explore SE state-of-practice in start-ups. Our initial findings suggest that start-ups overlook the best SE practices. Teams of a few people working on experimental and straightforward software see no upside of following the best practices. However, late start-ups face substantial challenges as their teams grow and products become more complex. The key difficulties concern installing adequate SE practices supporting collaboration, coordination of work, and management of accumulated technical debt. To support the evolution of engineering practices in start-ups, we propose the start-up progression model outlining engineering goals, common challenges, and useful practices with regards to the start-up life-cycle phases.

Further findings suggest inadequate support for market-driven requirements engineering (MDRE). Specifically, on how to aggregate needs and wishes of a large and loosely defined set of stakeholders who may not be able to articulate their needs and expectations. To address this challenge, we propose a method for the identification and prioritization of data sources and stakeholders in MDRE.

Analyzing SE context in start-ups and other organizations developing innovative and market-driven products, we have found many similarities. While start-ups have challenges, they do not appear to be unique. Thus,

most start-up challenges can be addressed by transferring the best practices from other engineering contexts.

We conclude that there is a little need for start-up specific engineering practices. Best software engineering practices are relevant to address challenges in start-ups. The key engineering challenge in start-ups is the management of the evolution of SE practices to match the growing complexity of the product and the organization. Our work also highlights the need for better MDRE practices to support new market-driven product development in both start-ups and other types of organizations.

ACKNOWLEDGMENTS

The effort leading to this thesis have been quite a journey. On my way I have met many people who have inspired me and provided unanticipated support. There are too many names to list them all. I dedicate this section to everyone who I have met on on my way.

First and foremost, I express my gratitude to my advisors Dr. Michael Unterkalmsteiner and Prof. Dr. Tony Gorschek for their diligence, support, and impeccable example. Thank you for accepting nothing less than excellence from me.

I wish to thank all my colleagues who are some of the nicest, smartest and most hard working people I know. You have provided an excellent environment for collaboration, learning and personal growth.

Members of Software Start-up Research Network and especially Prof. Dr. Pekka Abrahamsson and Dr. Xiaofeng Wang have provided a platform of collaboration that have encouraged and greatly influenced my work.

I also wish to recognize all the people who dedicated their time and knowledge by participating in my studies. I am thankful, for your valuable input and feedback to my work.

None of this work would be possible without the support and encouragement from my wife Madara. She motivated me to undertake this project and despite personal sacrifices unconditionally supported me through it. Her contribution to this thesis is not less than mine.

My journey towards this thesis would not have even started without the wisdom from my parents. From my early age they had thought me to the importance of hard work. They have encouraged me to stay curious, explore, and seek to understand.

Karlskrona, November 1, 2019

PREFACE

PAPERS IN THIS THESIS

This compilation thesis includes the following seven papers[1].

Chapter 2: Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. "Software engineering in start-up companies: An analysis of 88 experience reports." Empirical Software Engineering 24.1 (2019): 68-102.

Chapter 3: Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. "Software Engineering Anti-Patterns in Start-Ups." IEEE Software 36.2 (2019): 118-126

Chapter 4: Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladnicki, Nirnaya Tripathi, and Leandro Bento Pompermaier, "A progression model of software engineering goals, challenges, and practices in start-ups", IEEE Transactions on Software Engineering, 2019.

Chapter 5: Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladnicki, Nirnaya Tripathi, and Leandro Bento Pompermaier, "Exploration of Technical Debt in Start-Ups," 75–84. International Conference of Software Engineering, ACM, 2018.

Chapter 6: Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladnicki, Nirnaya Tripathi, and Leandro Bento Pompermaier, "Use of Agile Practices in Start-ups,", Under Submission, 2019.

Chapter 7: Eriks Klotins, "Software start-ups through an empirical lens : are start-ups snowflakes?" in International Workshop on Software-intensive Business: Start-ups, Ecosystems and Platforms, 2018.

Chapter 8: Eriks Klotins, Veselka Boeva, Krzysztof Wnuk, Michael Unterkalmsteiner and Tony Gorschek "A Collaborative Method for Identification and Prioritization of Data Sources in MDRE", Under Submission, 2019.

CONTRIBUTION STATEMENT

Eriks Klotins is the main author of all chapters in this thesis. As the lead author he took the main responsibility for design, data collection and analysis, and publishing the results in peer-reviewed venues. Furthermore, he

1 Papers marked with "Under Submission" have been submitted but were not reviewed yet at the time of this writing.

is the sole author of Chapter 1, the introduction, and Chapter 7. His and the co-authors contribution to the remaining chapters is described next:

CHAPTER 2: Tony Gorschek and Michael Unterkalmsteiner contributed to the idea of using experience reports to establish a foundation for further and more focused research. Eriks Klotins performed data collection, coding, and analysis. Both co-authors provided frequent feedback on intermediate and the final versions of the paper.

CHAPTER 3: Michael Unterkalmsteiner contributed to the idea of formulating the anti-patterns from the results of the earlier experience reports study, described in Chapter 3. Eriks Klotins performed the data analysis and formulated the patterns. Both co-authors provided frequent feedback on intermediate and the final versions of the paper.

CHAPTER 4: Tony Gorschek contributed with the idea of using case survey for scalable and in-depth data collection from start-ups. Eriks Klotins designed the survey instruments, Tony Gorschek and Michael Unterkalmsteiner provided reviews and additional input. Eriks Klotins, Michael Unterkalmsteiner, Tony Gorschek, Nirnaya Tripathi, Leandro Bento Pompermaier, and Rafael Prikladnicki contributed to the data collection by reaching out to their networks and soliciting start-up cases. Eriks Klotins and Panagiota Chatzipetrou performed data analysis. Eriks Klotins created the start-up progression model and formulated the results. Both Michael Unterkalmsteiner and Tony Gorschek provided feedback on intermediate and the final versions of the paper.

CHAPTER 5: This chapter is based on the same dataset as Chapter 4. Eriks Klotins and Panagiota Chatzipetrou performed data analysis. Both Michael Unterkalmsteiner and Tony Gorschek provided feedback on intermediate and the final versions of the paper.

CHAPTER 6: This chapter is based on the same dataset as Chapter 4. Eriks Klotins and Panagiota Chatzipetrou performed data analysis. Both Michael Unterkalmsteiner and Tony Gorschek provided feedback on intermediate and the final versions of the paper.

CHAPTER 6: Ideas presented in this chapter originates from discussions with Michael Unterkalmsteiner and Tony Gorschek.

CHAPTER 7: The idea was conceived jointly by Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. Krzysztof Wnuk wrote parts of related work and the introduction. Veselka Boeva provided the formal definition of the method. All co-authors reviewed and edited the final draft.

RELATED PAPERS NOT INCLUDED IN THIS THESIS

- Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. "Software engineering knowledge areas in startup companies: a mapping study", International Conference of Software Business. Springer, Cham, 2015.
- Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H. Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, Henry Edison, Carlos Fernandez-Sanchez, Juan Garbajosa, Tony Gorschek, Eriks Klotins, Laura Hokkanen, Fabio Kon, Ilaria Lunesu, Michele Marchesi, Lorraine Morgan, Markku Oivo, Christoph Selig, Pertti Seppanen, Roger Sweetman, Pasi Tyrvainen, Christina Ungerer, Agustin Yague "Software startups–a research agenda." e-Informatica Software Engineering Journal 10.1, 2016.
- Eriks Klotins, "Using the case survey method to explore engineering practices in software start-ups." 2017 IEEE/ACM 1st International Workshop on Software Engineering for Startups (SoftStart). IEEE, 2017.
- Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. "Software-intensive product engineering in start-ups: a taxonomy." IEEE Software 35.4, p. 44-52, 2018.
- Nirnaya Tripathi, Eriks Klotins, Rafael Prikladnicki, Markku Oivo, Leandro Bento Pompermaier, Arun Sojan Kudakacheril, Michael Unterkalmsteiner, Kari Liukkunen, Tony Gorschek. "An anatomy of requirements engineering in software startups using multi-vocal literature and case survey." Journal of Systems and Software, 146, p. 130-151, 2018.
- Sami Hyrynsalmi, Eriks Klotins, Michael Unterkalmsteiner, Tony Gorschek, Nirnaya Tripathi, Leandro Bento Pompermaier, and Rafael Prikladnicki. "What Is a Minimum Viable (Video) Game?", 217–231. Springer, Cham, 2018

OTHER PAPERS NOT INCLUDED IN THIS THESIS

- Michael Unterkalmsteiner, Tony Gorschek, Robert Feldt, and Eriks Klotins, "Assessing Requirements Engineering and Software Test Alignment – Five Case Studies", Journal of Systems and Software, pp 62-67, 2015.
- Eriks Klotins, Prashant Goswami, "Criteria for PhD student performance assessment in Software Engineering and Computer Science programs in Sweden", Lararlardom, 2016.

FUNDING

We would like to acknowledge that work described in Chapter 8 was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

Increasingly more products and services are interweaved with software. Producing, operating, and maintaining software has become a significant task in nearly every industry. Hence, innovations in software components can provide substantial benefits and competitive advantages [Andreessen, 2011, Cascio and Montealegre, 2016]. Markets for software are fast-paced and continuously expanding with ever-evolving customer needs [Moen et al., 2004]. Such context creates opportunities for new companies (start-ups) to capitalize on emerging needs by rapidly creating and taking over large segments of the market [Carmel, 1993].

Entrepreneurs have leveraged on new technologies, expanding markets, and venture capital since the dawn of the modern economy [Stiles, 2009, Chernow, 2007]. However, the rise of software start-ups could be dated with 1993, when the first web browser *Mosaic* was released and made access to the web convenient for masses [Vetter et al., 1994]. Later that decade, the company behind *Mosaic* went public and reached $2.9 billion in value, thus creating a high interest in investing in Internet companies. A combination of easily available capital at the time, hopes for high returns, and excessive speculation led to the infamous dot-com bubble in 2000 [Howcroft, 2001]. Even though bursting of the bubble wiped out many, a few start-ups survived. Companies such as Google, Amazon, PayPal, and eBay emerged from the burst and became some of the largest companies in the world. Other well-known software companies such as Facebook, Netflix, Uber, and Twitter emerged shortly afterward and have produced both customer value and substantial returns of investment.

Start-ups have shown their capability to rapidly capitalize on market opportunities, in comparison to slower-moving large organizations. At the same time, most start-ups cease to exist within the first few years of activity and before delivering any value. On the surface, start-up failures can be attributed to a bad product idea, business, or market challenges. However, we found that the capability to build software efficiently with minimal resources and limited knowledge about emerging market needs is the foremost challenge (Chapter 3). The challenge of exploring an unmet market need and developing a feasible solution within resources and time

constraints precedes any business or market difficulties. At the same time, start-ups often overlook the best engineering practices, follow an ad-hoc approach to software engineering, and only a few start-ups specific engineering practices exist [Giardino et al., 2016, Berg et al., 2018, Klotins et al., 2015].

Several prominent authors, for example, Ries [2011], Blank [2013b], and Sutton [2000], portray start-ups as special and game-changing. Analysis of contemporary media reveals excessive hubris and overinflated expectations around start-ups.

Work in start-ups is presented as an attractive alternative to boring, tedious work in established companies. Start-up culture is often characterized by university dropout celebrity CEOs, inhabitation of ostentatious co-working spaces, no strings attached venture capital, and overnight success [Hyrkäs et al., 2016]. Such perceptions contribute to the idea that established best engineering practices are irrelevant in start-ups.

In this thesis, we aim to understand software engineering in start-ups. At the beginning of this work, software engineering in start-ups had attracted little attention from researchers. The small number of papers addressing the engineering aspects of start-ups did not form a coherent body of knowledge [Klotins et al., 2015]. Thus, this work is mostly exploratory and focuses on mapping the research area.

One of the objectives of mapping the research area is to understand the overlap between start-ups and established organizations. The overlapping engineering areas could benefit from transferring the best practices from established companies. Detailing start-up specific engineering areas would create opportunities for developing new and start-up specific engineering practices.

Studying start-ups is difficult due to their short lifespan and elusive nature. A start-up may cease to exist before appearing on any records and leave little behind after closing down. At the beginning of this work, there was a lack of understanding about engineering context in start-ups. Thus, making focused and relevant inquiries difficult [Klotins et al., 2018b]. Furthermore, start-up engineers see a little upside in volunteering their time for researchers as their companies may not survive long enough to benefit from the results.

This thesis makes three contributions to software engineering in start-ups. We illustrate the contributions and the flow of our research in Fig. 1.1. In the figure, each box denotes a chapter in this thesis. Vertical ordering of the boxes shows how studies described in each chapter were laid out in time. Arrows between the boxes denote the flow of ideas and connections between the chapters. The three columns denote each contribution.

The first contribution (*Exploration*) aims to understand how start-ups practice software engineering. Initially, the exploration was open and aimed to scope the research area (Chapter 2), later it was narrowed to specific

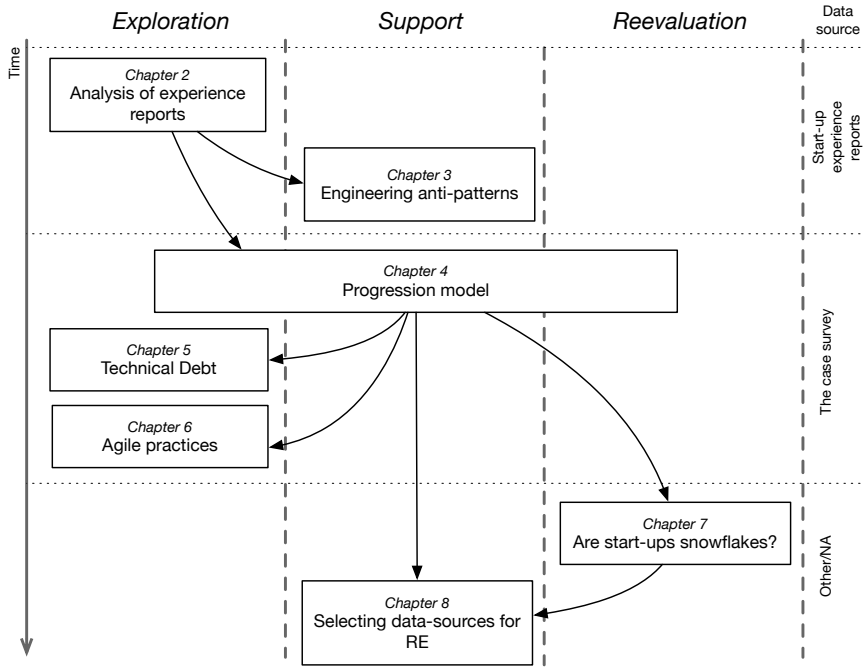


Figure 1.1: Overview of the contribution. Boxes denote how chapters of this thesis map to the contributions, and were laid out in time (y-axis). Arrows denote the flow and development of ideas between the chapters.

topics, such as the evolution of engineering practices (Chapter 4), technical debt (Chapter 5), and use of agile practices (Chapter 6).

The second contribution (*Support*) aims to support software engineering in start-ups by identifying engineering anti-patterns (Chapter 3), the start-up progression model of goals, challenges, and practices (Chapter 4), and proposing a method for selecting data sources for market-driven requirements engineering (Chapter 8).

The third contribution (*Reevaluation*) examines the initial premise that start-ups are unique and need a different approach to software engineering (Chapter 7). We show that there is insufficient empirical evidence to claim that start-ups are different from established software companies and require a new approach to software engineering. Our further results show that start-ups face very similar challenges to established organizations working on new market-driven product development. We pinpoint the lack of relevant market-driven requirements engineering practices as the fundamental difficulty in new software product development in all types of companies.

The rest of this chapter is structured as follows: In Section 2, we present relevant background and related work. We present our research approach, the gap, research questions, methods, and validity threats in Section 2. In Section 3, we summarize results from the studies included in this thesis. In Section 4, we discuss our results and answer the research questions. Section 5 concludes this chapter.

1 BACKGROUND AND RELATED WORK

1.1 *Software startups*

Different sources offer different definitions of a start-up. One of earliest characterizations of software start-ups is provided by Carmel [1994a]:

> *"Product development [in start-ups] is driven by the following five goals: (1) minimize time-to-completion, (2) increase innovation/features, (3) maximize quality, (4) minimize product cost, and (5) minimize development cost. It is impossible to pursue all five goals at once. Developers must make trade-off decisions - implicitly or explicitly. Any such decision will, by definition, affect time-to-completion."*

Sutton et al. [2000] elaborates four characteristics of software start-ups: Youth and immaturity, limited resources, multiple influences and dynamic technologies and markets. They argue that start-ups share many similarities with small companies, however it is the combination and severity of challenges that make start-ups exceptional.

Ries [2011] offers a broader definition of a start-up without a notion of any specific technology or domain:

> *"A startup is a human institution designed to create a new product or service under conditions of extreme uncertainty."*

These definitions provide the basis for our own formulation of a software start-up:

> *"With a software start-up company we understand a recently created institution with a focus of launching an innovative software-intensive product or service to market."*

Recently created institution entails that there is no operating history, and all basic processes need to be established. Such an institution relies on external funding, initially on savings on its founders, later on, a risk capital, or other forms of investments. Thus, resources are scarce, teams are small, and use of any heavy engineering practices is limited [Giardino et al., 2014a].

Development of an innovative product or service entails a substantial level of uncertainty about customer needs, markets, product features, technologies, and the role of other influences. Thus, one of the first focus areas of a start-up is the exploration of a problem domain and customer needs. A software-intensive product is a product that contains an essential and non-trivial software component [ISO/IEC 42010, 2011].

We differentiate between new companies providing bespoke software development services, i.e., consultancies, serving the needs of a specific customer, and companies with a focus on one product on service aimed for the mass market. Mass-market products and services can have variations to fit different market segments or select customers. Nevertheless, mass-market products are market-driven, i.e., aimed to suit the needs of many customers.

We also differentiate between toy and industry start-ups. A toy start-up could be created as part of the entrepreneurship or engineering course to encourage students to practice solving real problems in a somewhat realistic environment [Järvi et al., 2015, Wang et al., 2016, Fagerholm et al., 2018]. However, such start-ups are ultimately created for training and educational purposes in a supported environment. Thus, lessons learned from toy start-ups are unlikely to be relevant for industry practitioners [Ivarsson and Gorschek, 2010].

Launching a product or service to market is a complicated endeavor. It requires a combination of carefully orchestrated activities. Market research is needed to estimate the market need and potential of the product. A business model is required for the sustainable evolution of the start-up. Sales and marketing activities are needed to attract customers to the product. Last but not least, the product itself needs to be invented and produced. Inadequacies in any of these activities could hinder start-ups' chances of success. While recognizing the complexity of start-up activities, in this thesis, we focus exclusively on the engineering aspects.

1.2 *Interpreatation of success*

An inherent objective of start-ups is to perform well and succeed in developing and marketing a software-intensive product. The software product and therefore, applied software engineering practices are significant factors to start-up performance and success. To select and gauge the suitability of engineering practices, it is useful to consider the meaning of engineering "performance" and "success" in start-ups.

Success is defined as a favorable termination of an endeavor [Oxford, 1989]. In start-ups, a favorable end is usually associated with a buyout from another company or an initial public offering (IPO) in a stock market. A start-up may choose to pursue such outcomes if the risks from continuing

operations are higher than expected gains from the exit [Wennberg and DeTienne, 2014].

Success as an end goal is not particularly useful to gauge intermediate results. Also, linking specific engineering practices to a buyout or an IPO is difficult. However, we argue that the chances of success are higher if a start-up uses sustainable engineering practices and maximizes its resources. The longer a start-up can survive and exibit good performance, the higher are chances of succeeding.

Performance is the ability to produce results to a predefined target [Laitinen, 2002]. It can be measured by observing changes in turnover, market share, product quality, and internal efficiency [Reijonen and Komppula, 2007, Kitchenham and Lawrence, 1996, Egorova et al., 2009].

Utilized engineering practices determine the product quality and to a large extent, internal efficiency. Inadequacies in product engineering can affect the turnover and market share, for example, by limiting the speed of onboarding new users, customizing the product for new markets, and adding new features [Ralph and Kelly, 2014].

With good engineering performance, we understand the following [Egorova et al., 2009]:

1. Customers are involved in the engineering process
2. Good understanding of customer's problems
3. The team follows relevant best engineering practices
4. Business and engineering objectives are aligned
5. Tasks are completed on time and within budget

Engineering aims to produce a high performing product, characterized with [Egorova et al., 2009]:

1. Good quality in relation to customer expectations
2. Customer satisfaction with features and quality
3. Reliability and suitability for the intended purpose

We acknowledge that start-up performance and success depend on a variety of factors beyond software engineering, e.g., the idea, expertise of the core team, organizational structure, economic context, selection of marketing and sales strategies, and business models, among other factors [Chorev and Anderson, 2006]. Nevertheless, product engineering is one of the core activities of start-ups and has a profound influence on the overall start-up outcome.

1.3 *Market-driven and crowd requirements engineering in start-ups*

Start-ups attempt to benefit from economies of scale by developing one product that suits many customers. The number of potential customers

is often too high to single out each individual, thus customers are considered as one market or crowd [Regnell and Brinkkemper, 2005b, Groen et al., 2017]. The term crowd is used to describe a loose set of existing and potential customers, and other stakeholders who are interacting with the product and each other, often beyond reach of the product organization [Groen et al., 2017]. Performing requirements engineering in such context is challenging.

Traditional requirements engineering methods suggest to identify product stakeholders and elicit their requirements by means of interviews, surveys, observation and other methods [Kotonya and Sommerville, 1998]. However, such approach is not viable if stakeholders are unknown, too many, or not interested to collaborate. As a consequence, companies use a mixed approach of inventing requirements, involving key stakeholders in product decisions, looking at product analytics, and analyzing competitor offerings [Dahlstedt et al., 2003]. Feature ideas are formulated as hypotheses and validated in the market with continuous experimentation [Fagerholm et al., 2017, Blank, 2013a, Ries, 2011]. However, we lack evidence of such methods to be rigorously adopted by start-ups.

Related work from start-ups suggest that requirements engineering practices evolve together with the organization. Requirements engineering practices start from rudimentary and become more specific over as the company grows [Gralha et al., 2018].

The key requirements engineering difficulties in both start-ups and other types of organizations are [Dahlstedt et al., 2003, Tripathi et al., 2018, Alves et al., 2006]:

- Absence of customers for requirements elicitation before the product is launched. After the launch, customers may not be motivated enough to provide in-depth feedback, too many in numbers to be contacted individually, or simply not able to formulate their needs.
- Companies face multiple influences to requirements engineering leading to information overload. Requirements engineering is influenced by customer feedback, bug reports, requests for customization, market trends, internal ideas, irrelevant opinions, and so on. Distinguishing good ideas from irrelevant input is challenging.
- Requirements validation takes place after the product is released to market. Implementation of unvalidated features could be a substantial investment with uncertain outcome. Validation is particularly challenging when developing innovative features.
- Handling the trade-off between generalized and customer specific features. Generalized features may relevant to a wider market. However, implementing customer specific features could help to win the customer. Focusing too much on custom features can lead to feature creep and difficult product maintenance. At the same time, imple-

menting overgeneralized product reduces its value to individual customers.

2 RESEARCH APPROACH

2.1 *Research gaps*

The lack of scientific research addressing software engineering in start-ups was pointed out by Paternoster et al. [2014] and subsequent literature studies, for example, Klotins et al. [2015] and Berg et al. [2018]. The existing literature on software engineering in start-ups suggests that:

- Start-ups have different characteristics and objectives than established organizations [Sutton, 2000, Blank, 2013b, Coleman and O'Connor, 2008].
- Engineering of new software products in an uncertain environment is one of the key challenges in start-ups [Giardino et al., 2015b].
- Start-ups use an ad-hoc approach to software engineering and accumulate technical debt on the way [Giardino et al., 2016].
- Scaled-down practices from established organizations may fail to address start-up specific challenges [Yau and Murphy, 2013].
- Even though start-ups are aware of the Lean Start-up methodology [Ries, 2011], its application is cumbersome, and the results are unsatisfactory [May, 2012].
- Several engineering methods and models are proposed for start-ups, for example, Giardino et al. [2014b], Zettel et al. [2001], and Bosch et al. [2013], however they lack empirical validation and any evidence of industrial application.
- Most of the empirical work on start-ups is superficial, e.g., analyzes a few cases or shallow metrics. Such finding highlights the difficulty of getting in-depth access to live start-ups.

The lack of a coherent view on how software engineering is practiced in start-ups hinders the evaluation of existing engineering methods in start-up context and development of new start-up specific methods [Klotins et al., 2018b]. Such state-of-the-art highlights the following gaps:

- The *engineering context* in start-ups is poorly understood. It is critical to understand the context where an engineering practice is going to be applied to evaluate its suitability. The context comprises of type of product being developed, practices, tools, processes, and people around it. A wider context includes the start-up itself and the market it is operating [Petersen and Wohlin, 2009].
- *Scope of software engineering* in start-ups is not clear. It is critical to understand what is the scope of software engineering in start-ups

to focus our research efforts. SWEBOK [IEEE Computer Society, 2014] presents several engineering knowledge areas. However, not all knowledge areas could be applicable in start-ups, and SWEBOK may overlook start-up specific engineering areas. Furthermore, it is vital to understand how different knowledge areas are connected and influence each other in a start-up context.

- Lack of *support for specific challenges.* Insufficient understanding of engineering context and scope hinders both transfers of existing practices and development of new start-up specific engineering practices.

A fundamental challenge in a new research area is to address unknown unknowns. That is, things we do not know that we don't know [Logan, 2009]. To study start-ups, we need to design our inquiries to allow the unknown concepts to emerge. The unknown unknowns add complexity to the research as we cannot directly transfer existing knowledge, context models, and domain ontologies from other research contexts to start-ups. The lack of relevant tools to study start-ups is an important component of the research gap.

2.2 *Research questions*

Research questions formulate specific problems to be solved through research. Answering the questions solves the research problems and, potentially, leads to new more focused inquiries. We state specific research questions in each chapter. However, here we provide and overview and link different research questions to specific contributions.

We structure the research questions by their association to the contributions, to *explore*, *support*, and *reevaluate*, software engineering in start-ups.

EXPLORE: RQ1: How is software engineering practiced in start-ups?

RQ1.1: How are software engineering knowledge areas applied in start-ups? (Chapter 2)
RQ1.2: How do start-ups estimate aspects of technical debt? (Chapter 5)
RQ1.3: How do start-ups use agile software engineering practices? (Chapter 6)

Rationale: With this research question we aim to understand engineering context and set a foundation for more specific investigation and development of start-up specific engineering methods.

SUPPORT: RQ2: How should start-ups practice software engineering?

RQ2.1: What software engineering anti-patterns can be ascertained in start-ups? (Chapter 3)

RQ2.3: What software engineering patterns can be ascertained in start-ups? (Chapter 4)

RQ2.6: How to select and prioritize data sources for requirements engineering? (Chapter 8)

Rationale: With this research question we aim to provide guidance to start-ups in their engineering endeavors based on the experience from other start-ups?

REEVALUATE: RQ3: What are the differences between start-ups and established organizations?

RQ3.1: How much empirical evidence exists in relation to supporting that start-ups are unique in terms of engineering practices? (Chapter 7)

RQ3.2: How does start-ups compare to established organizations in relation to the relevance, and utilization of engineering practices? (Chapter 7)

Rationale: With this research question we examine how unique are the start-ups to gauge the potential of transferring the best engineering practices between start-ups and established organizations.

2.3 *Research methods*

2.3.1 *General overview of research methods*

Several research methods could be applicable to study software engineering and to answer our research questions. In this section, we provide an overview of relevant methods. In Table 1.1, we compare the methods.

A FORMAL EXPERIMENT aims to test a theory in a controlled environment. Researchers formulate hypotheses to express relationships between independent and dependent variables. The experiment context is carefully controlled to minimize any confounding factors [Wohlin et al., 2012].

The analysis of experiment results often based on statistical inference, e.g., by determining the statistical significance between the outcomes of two treatments. In practice, experiments involving human subjects require a sufficient number of participants performing a specific task in a controlled environment [Sjøberg et al., 2005].

Providing a controlled yet realistic environment and a sufficient number of participants is a significant difficulty in designing software engineering experiments.

AN INDUSTRIAL CASE STUDY aims to perform an in-depth study of a phenomenon in its natural context. Case studies frequently involve multiple methods of data collection, for example, interviews with practitioners,

artifact analysis, and surveys. The data is analyzed with both qualitative and quantitative methods to explain the studied phenomenon [Runeson and Höst, 2008].

In contrast to formal experiments, case studies are more flexible and can be used for studies when boundaries between the phenomena and its context are not clear [Yin, 2003]. In practice, conducting a case study in an industrial setting requires a substantial commitment from a company and a clear focus of the inquiry.

ACTION RESEARCH aims to introduce treatment in a real-life scenario and observe the effects. The researcher is actively involved in the studied scenario, e.g., as a member of an engineering team, administers the treatment, e.g., a new tool or way of working, and observes its effects [Robson, 2002].

Applying the action research method requires the context and the treatment to be well understood beforehand, and a certain level of confidence that the treatment does not introduce any harmful effects.

SIMULATION aims to execute a model of a real-life phenomenon. The model allows to tune the input parameters, play out "what-if" scenarios, and analyze the outcomes [Shull et al., 2007].

Simulations are useful when mechanisms comprising the studied phenomenon are too complex for a human to comprehend in a time-efficient manner. However, a good understanding of the involved mechanisms is a prerequisite for designing a simulation [Romeu, 1985].

A SURVEY aims to collect standardized responses from a large sample, commonly but not necessary, through questionnaires or data collection forms. Surveys provide an overview of a population rather than depth of individual cases [Robson, 2002].

POST-MORTEM ANALYSIS aims to study past events. However, it could also retrospectively analyze an ongoing process, e.g., a project. The post mortem analysis may be conducted by analyzing project documentation, interviewing relevant people, and looking at the produced artifacts. Thus, this type of research can be viewed as both a survey and a case study [Wohlin et al., 2003].

DELPHI METHOD aims to gather expert viewpoints on a phenomenon through interviews and surveys. Researchers combine the different viewpoints and iterate between the experts until a degree of consensus is reached. This method is useful to study specific and highly complex situations where the general population, or a representative sample of it, could not be knowledgeable enough to provide useful inputs [Okoli and Pawlowski, 2004].

Table 1.1: Comparison of research methods, adapted from Runeson and Höst [2008], and Wohlin et al. [2012]

Method	Primary objective	Primary data	Design type	Cost
Experiment	Explain	Quantitative	Fixed	High
Case, ethnographic studies	Explore	Qualitative	Flexible	Medium
Action research	Improve	Qualitative	Flexible	Medium
Simulation	Explore	Quantitative	Fixed	Low
Delphi method	Build consensus	Qualitative	Flexible	Low

ETHNOGRAPHIC STUDIES can be seen as a specific type of case studies to analyze large amounts of data over a long duration with a focus on cultural practices [Easterbrook et al., 2008].

MIXED METHODS APPROACH aims to combine different methods to complement each other to compensate for individual weaknesses. There are three broad strategies of how research methods can be combined [Creswell and Creswell, 2017]:

- In *convergent parallel mixed methods*, the researched collects both qualitative and quantitative data simultaneously and combines both types of data in the analysis.
- In *explanatory sequential mixed methods*, the researcher collects and analyzes quantitative data which is complemented with qualitative data explaining the quantitative results.
- in *exploratory sequential mixed methods*, the researcher collects and analyzes qualitative data to identify relevant variables and propose hypotheses which are then tested with quantitative data and analysis.

2.3.2 *Motivation for choice of research methods*

Selection of research methods depends on both the questions to be answered and the research context. We base our selection on our research questions, see Section 2.2, and the following considerations:

- Earlier literature studies show lack of a coherent view of engineering aspects in start-ups. At the first stage of our research, we are not able

to formulate focused research questions; that is, we do not know what is worth studying.

- Collecting empirical data from start-ups is difficult due to their fragile nature, reliance on tacit knowledge, and the lack of focused questions from our side. Thus, any data collection would require practitioners to volunteer their time for unstructured interviews or composing their answers on broad subjects in writing. The amount of effort required from a single start-up needs to be minimized to get a buy-in to participate in our research. However, that also limits the amount of data we can collect.
- To strengthen the generalizability of our results, we need to study a broad and diverse sample of start-ups. Studying a number of start-ups in the same incubator, accelerator, or similar community would introduce validity threats to our research. Start-ups in the same community could be prone to the same specific influences, rely on the same set of advisors, and influence each other. Visiting a sufficient number of unrelated companies for data collection would require substantial resources and time.

In this thesis, we primarily use two research methods. Chapters 2 and 2 analyzes start-up post-mortem reports. In Chapters 4, 5, and 6 we use a case survey research method. In the rest of this section, we motivate the use of each method and compare them with other, alternative methods.

The post-mortem reports were readily available and provided insights on practitioners lessons learned from a broad sample of start-ups. The practitioners had voluntarily written these reports to share their experiences with the community. Thus, there was no extra effort required to obtain the data.

Based on the findings from the experience reports, we map the research area and design a more focused investigation [Klotins et al., 2018b]. To address the trade-off between depth and breadth of a study, we use a case survey method. A case survey uses a standardized data collection instrument which aims to collect both qualitative and quantitative data from a case [Petersen et al., 2017a]. To scale up the data collection, we invited other researchers to participate in our study and help with promoting our study within their networks. Such setup enabled us to collect relatively rich data, compared to traditional surveys, and to scale up the data collection beyond what would be feasible with individual case studies.

Research methods requiring a deep prior understanding of the phenomenon such as experiments, simulations, and action research, are not suited for our studies. We largely focus on exploration, i.e., what software engineering methods, why, and how, are applied in start-ups. Furthermore, the study context is complex. Software engineering in start-ups is influenced by and affect a myriad of poorly understood environmental, orga-

nizational, and human aspects, thus isolating individual variables while preserving realism is not feasible. Introducing a new engineering method in a poorly understood and high stakes environment could create ethical issues.

Delphi research method is not considered as suitable due to the lack of obvious reliable experts in the field apart from start-up practitioners. However, knowledge of individual start-up practitioners is based on their own limited experience from one or a few start-ups. Relying on expert advice alone would introduce external validity threats.

Case studies and surveys are the most suited to answer our research questions. To compensate for individual shortcomings of these methods, we use qualitative post-mortem analysis to formulate relevant topics for more focused investigation. We combine the case study method with a survey to develop a scalable method for studying large number start-ups with minimal interruption to their work.

2.4 *Data and analysis methods*

A selection of research methods in combination with data and analysis methods determine what kind of conclusions a researcher can make [Runeson et al., 2012]. In this section, we discuss the data sources and applied data analysis methods.

2.4.1 *Data*

Research data are values and measures characterizing the studied phenomenon. Through the research process, data is analyzed to answer the research questions and build an understanding of the phenomenon. Research data can be classified by their originality:

- *Primary data* is information collected through original or first-hand research. Primary data is collected directly from the source by means of measurements, surveys, and interviews.
- *Secondary data* is information collected and analyzed by someone else in the past. For example, results from the analysis of primary data.
- *Tertiary data* is information based on a collection of primary and secondary sources. For example, reviews of studies analyzing primary data.

Data can be also categorized by utilized data collection technique [Lethbridge et al., 2005]:

- *First degree methods* use direct access to participants' population through, for example, meetings, interviews, questionnaires, observation, and workshops.

- *Second-degree methods* utilize access to participants' work environment, however does not require for participants and researchers to interact, for example, through non-intrusive participant monitoring systems.
- *Third-degree methods* study participants work by looking at the results and byproducts of their work, for example, by analyzing documents, artifacts, log files and database entries.

Research data can be broadly categorized into qualitative and quantitative data [Seaman, 1999]:

- *Quantitative data* quantifies some aspect of the studied phenomenon and are likely to be expressed as numbers on an ordinal, interval, or ratio scale.
- *Qualitative data* explains some aspect of the studied phenomenon are likely to be expressed in text, e.g., interview transcripts, notes, or similar.

In this thesis, we use a variety of data sources containing both qualitative and quantitative data and utilize various data collection techniques. We summarize and categorize the used data sources in Table 1.2.

The experience reports analyzed in Chapters 2 and 3 constitute a primary data source. The reports are written by start-up practitioners who were directly involved in the described situations and expressed their original insights without any intermediary analysis. However, using already written reports is a third-degree data collection method as the reports were created independently and not for the purpose of our study [Lethbridge et al., 2005]. The reports contain mainly qualitative data describing a sequence of events, applied practices, methods, challenges, lessons learned, and outcomes from start-up cases. We have provided the complete dataset of 88 reports containing 194 303 words as supplemental material to Chapter 2.

The case survey is used in Chapters 4, 5, and 6 and contains a mix of both qualitative and quantitative data. The case survey design follows convergent parallel mixed methods approach by collecting quantitative measures, e.g., team size, followed by a qualitative explanation, e.g., how the team composition changed over time. This data was collected directly from start-up practitioners and constitutes the first-degree collection of primary data. For confidentiality reasons, the complete dataset of 84 start-up cases and 23 940 data points are available upon request to the author.

In Chapter 7 analyze studies identified by an earlier literature review [Paternoster et al., 2014]. The studies represent a secondary and qualitative data source.

In Chapter 8, we analyze both qualitative and quantitative data obtained during two workshops in the industry. We use quantitative data from

practitioners as inputs to a proposed method and collect qualitative reflections on the output. This can be seen as an explanatory sequential mixed-method approach.

2.4.2 *Data analysis methods*

Data analysis methods can be broadly divided into qualitative and quantitative approaches matching the kind of data being analyzed. Quantitative data analysis methods use mathematical and statistical techniques to describe the dataset, develop predictive models, and to test hypothesis [Runeson and Höst, 2008].

Qualitative methods attempt to identify relevant patterns from unstructured data, such as textual descriptions, notes, and transcripts. This type of analysis typically involves coding of the data and further iterative analysis of the codes. Qualitative analysis can be conducted on different levels of formalism [Runeson and Höst, 2008]. Robson [2002] describes the following approaches to formalism:

Immersion approach is the least structured and reliant on intuition and interpretive skills of the researchers.

Editing approach uses a few predefined codes and defines additional codes during the data analysis.

Template approach relies on a list of predefined codes.

Quasi-statistical approach is the most formal and uses descriptive statistics to analyze textual data, e.g., by calculating frequencies of words or specific phrases in the data.

Grounded theory is a special case of qualitative data analysis, and is aimed to develop theory through coding and systematic inductive reasoning [Corbin and Strauss, 1990]. However, we do not utilize grounded theory to its full extent. The canons of grounded theory dictate to build a theory in parallel with data collection and continue data collection until theoretical saturation is reached. Working with limited datasets, we cannot fulfill this requirement.

In this thesis, we use both qualitative and quantitative methods, often in combination with each other. In Table 1.3, we summarize the applied data analysis methods.

2.5 *Ethical considerations*

This thesis analyzes empirical data concerning real-world situations. Our selection of research methods, results, and dissemination strategies may lead to unwanted effects harming individuals, organizations, or society as a whole. In this section, we discuss our ethical considerations about informed consent, scientific value, effects on humans and organizations, and confidentiality [Singer and Vinson, 2002, Runeson and Höst, 2008].

Table 1.2: An overview of data sources used in this thesis

Chapters	Data sources	Data kind	Level of researcher involvement	Degree of originality
2-3	Experience reports	Qualitative	Third degree, the reports were created independently from our study	Primary
4-6	Case survey	Mixed	First degree, the data was collected directly from participants for the purpose of the study	Primary
7	Literature	Qualitative	Third degree, the data was collected independently from our study	Secondary
8	Industry workshops	Mixed	First degree, the data was collected directly from participants for the purpose of the study	Primary

Table 1.3: An overview of data analysis methods applied in this thesis

Chapters	Approach	Level of formalism	Applied methods
2	Qualitative	Template approach	Coding of experience reports using a taxonomy, development of themes and their relationships
3	Qualitative	Immersion approach	Coding of experience reports using a taxonomy, root-cause analysis
4	Both qualitative and quantitative	Editing approach	Statistical analysis of quantitative data combined with coding of textual data
5-6	Quantitative	-	Statistical analysis of quantitative data
7	Qualitative	Template approach	Examination of earlier studies with aim to extract empirical evidence pertaining start-up characteristics
8	Qualitative	Immersion approach	Analysis of practitioner reflections during workshops

INFORMED CONSENT refers to the full disclosure of necessary information for participants to decide whether to partake in a study and have the freedom to withdraw at any moment. This type of ethical concern is relevant for Chapters 4-6, and 8.

In Chapters 4-6 we use case survey to collect data from start-up practitioners. The subjects were invited to participate in the study through personal contacts and social media. The participation was voluntary and presented as an opportunity for practitioners to share their experiences, i.e., what engineering practices worked and what did not work in their start-ups. The subjects provided their responses using an on-line questionnaire. Thus, they could fill it at their convenience and withdraw at any moment.

In Chapter 8, we describe the use of two industry workshops. The two workshops followed a similar design. First, we reached out to the companies with proposals for a workshop and outlined our research aims and expected outcomes. The proposal contained a brief agenda of the workshop, including a description of what data we aim to collect. The subjects participated in the workshops at their free will.

EFFECT ON HUMANS AND ORGANIZATIONS refers to potential adverse effects on individuals and organizations that may arise from participating in research. This type of ethical concern is relevant for Chapters 4-6, and 8.

This thesis is mainly exploratory to understand how start-ups practice software engineering. We do not attempt to prescribe any solutions to the studied companies. That said, there could be some exchange of untested ideas between researchers and the subjects. We minimize the adverse effects from our interactions by emphasizing that our work is angled towards exploring software engineering start-ups, in contrast to prescribing any specific solutions.

Interactions with practitioners take their time away from performing work tasks. We design our research instruments to minimize any interruptions to their work. First, we create our inquires to be focused and take as little time from practitioners as possible. Secondly, we design our research instruments to be flexible in terms of timing. For instance, practitioners can respond to the case survey at their convenience, and companies can decide when to schedule the workshops to minimize any interruptions of their work.

CONFIDENTIALITY concerns how raw data is handled by researchers and to what extent individual participants of a study can be identified from the results.

In Chapters 4, 5, and 6 we collect and analyze data from real companies. To maintain traceability between data and the actual start-ups, we collect company names and email addresses of the participants. We use this in-

formation to remove duplicate cases and to contact participants for the dissemination of the results. However, providing traceability information is not mandatory. Through the analysis, we anonymize the data by removing names, locations, and descriptions of specific situations that could potentially link our results to specific cases.

In Chapter 8, we describe results from two industrial workshops. During the workshops, we discussed practices, roles, processes, challenges, and other aspects relevant to the selection of data sources for requirements engineering in their organizations. This information may be sensitive. Before publication, we cleaned the results from any potentially sensitive information and asked the participants to review the final draft. We further requested for participants' permission to reveal the company names.

SCIENTIFIC VALUE comprises of the importance and validity of the results. The importance concerns risks from researching anticipated benefits. The validity concerns to what extent the presented results faithfully represent reality. This concern is relevant to all chapters of this thesis.

The importance of our work can be characterized by the amount of capital start-up companies receive and their high failure rate. Start-ups globally receive several hundred billion USD in venture capital (Chapter 2). With an optimistic 25% survival rate, this constitutes a substantial amount of wasted capital in failed start-ups. If our work can improve start-up odds of success even by a marginal amount that would result in a significant impact on the industry and capital return.

We ensure the validity of our research by describing aspects related to scientific rigor Ivarsson and Gorschek [2010]. We describe the study context, research design, and threats to validity. The context descriptions are aimed to help readers to evaluate to what extent their setting is similar to the study. With the research design description, we intended to explain what sampling method we used, what data collected, how the data was analyzed, and how we obtained our results. Threats to validity aim to reflect on the limitations of a study. The peer-review process before dissemination further strengthens the validity of our results.

Where possible, we provide raw data and intermediate results as supplemental material to our publications. Supplemental material is aimed to increase transparency of our research and help the readers to verify our results independently.

EXCEPTIONS concern situations where the above concerns do not apply, e.g., when analyzing anonymous data or already public data when the expectation of privacy does not exist.

In Chapters 2 and 3, we analyze publicly available experience reports that were voluntarily created, uploaded, and published by the authors independently from our inquiry. Most reports contain information revealing

the identities of the authors and the names of their companies. However, in this case, there is no expectation of confidentiality and privacy.

2.6 *Validity*

We discuss threats to validity separately in each chapter. However, in this section, we discuss overarching concerns relevant to the whole thesis. We discuss four types of validity concerns, construct validity, internal validity, external validity, and reliability [Yin, 2003, Runeson et al., 2012].

CONSTRUCT VALIDITY concerns to what degree operational measures represent the studied phenomenon. This type of concern is particularly relevant in Chapters 2 and 4. To address this threat, we use multiple strategies.

We limit the use of any particular set of concepts and categories on the subjects. We use a mix of methods to maintain the focus of our research and to identify new focus areas. For example, we apply multiple-choice questions and provisional coding [Saldaña, 2015], to gain insights on particular engineering topics. We complement the structured approach with open-ended questions, e.g., "What would you do differently next time?", and *in vivo* coding to gain additional insights.

Another threat to construct validity stems from our research context. Each studied start-up is viewed through the eyes of one individual, i.e., author of an experience report or questionnaire respondent. These individuals were closely involved in the studied cases, and their responses could be biased. For example, the subjects may fail to reconstruct past events correctly or attempt to rationalize their shortcomings with external circumstances [Pronin et al., 2002]. However, due to a relatively large and diverse sample of studied companies, this threat is minimal.

We further strengthen construct validity by using findings from one study as an input for the next one, see Fig 1.1. Therefore, chapters in this thesis complement each other and compensate for individual deficiencies.

INTERNAL VALIDITY concerns the trustworthiness of cause-and-effect relationships in the studied phenomenon Runeson et al. [2012].

Internal validity is not a significant concern to this thesis. We abstain from prescribing any particular practice as a treatment to start-up challenges, and we do not link the use of any practices to start-up success. Instead, we focus on illustrating and contextualizing the challenges and potential solutions. Thus, we help readers to understand their specific challenges and to select appropriate solutions.

EXTERNAL VALIDITY determines to what extent the findings are relevant outside the studied sample. Ivarsson and Gorschek [2010] propose four components of industrial relevance, subjects, context, scale, and research methods. We utilize research methods investigating real-life situations, exploratory case survey, and experience reports.

Subjects participating in our research are start-up founders, engineers, and other practitioners with first-hand experience of product engineering in start-ups. Thus, their insights are on par with the general population of start-up professionals.

The context and scale of our research are specific and real start-up cases. With collect and report contextual information characterizing the start-ups, thus readers can understand and compare their specific context with our report. In our writing, we differentiate between data and results from the cases and our conjectures.

Another concern to external validity is to what extent the studied sample is representative of the general start-up population. A representative sample would allow generalizing results over the whole population. To precisely characterize the representative sample, an understanding of the whole population is needed [Omair et al., 2014]. However, the total number of start-ups and the relevant characteristics of start-ups are unknown. Thus, it is not possible to quantify the representativeness of our sample.

Throughout this thesis, we use convenience sampling and cannot guarantee that all kinds of start-ups are equally represented. However, we specifically target as broad and as diverse sample as possible. Chapters 2 and 3 primarily analyzes lessons learned from closed start-ups. Chapters 4, 5, and 6 primarily explores operational start-ups.

Our total sample contains start-ups at different levels of maturity, with both small and large teams, with and without external support from accelerators and incubators, from different geographical regions, established by founders with both minimal and extensive experience in start-ups and industry in general. To our best understanding, we have attained a good enough sample to answer our research questions with confidence [Marshall, 1996].

RELIABILITY concerns to what extent the results and analysis are independent of specific researchers.

Studies in this thesis are primarily designed, conducted by the author. Thus, the interpretation of results may suffer from the author's biases. To minimize this threat, all instruments used in this thesis, such as survey questionnaires, data collection forms, and taxonomies, are jointly reviewed among all authors of each study. Furthermore, the authors of each study jointly reviewed results and their interpretation and complemented the interpretation with alternative explanations where relevant.

We further address the reliability by ensuring transparency and traceability throughout data collection, analysis, and formulation of results. We attempt to link our specific conclusions to exact data supporting these statements. Therefore, a reader can examine our analysis process and independently verify our conclusions.

3 RESULTS

In this section, we link results from different chapters and set the basis for answering our research questions. A high-level overview of the chapters is illustrated in Fig. 1.1.

IN CHAPTER 2, we analyze start-up experience reports to investigate how start-ups practice software engineering. We aim to identify specific knowledge areas for a more focused inquiry.

We found that requirements engineering is a crucial activity in start-ups, and is applied to refine the initial product idea. However, the applied practices are often rudimentary and not well aligned with other aspects of a start-up, for example, business objectives.

Inadequacies in requirements engineering often lead to product quality issues, both in terms of identifying the right set of functionalities and determining the right level of quality. Issues in requirements engineering contribute to wasted resources on building irrelevant features, and missed market opportunities from launching a sub-standard product.

A common issue in requirements engineering is feature creep. Start-ups often report blindly adding features to their products, hoping to win customers. However, such a strategy is counterproductive as it waters down the initial product idea, adds complexity to further product development and depletes resources.

We found that other engineering knowledge areas have a mostly supportive role in understanding customer wishes and needs. A common practice is to release a minimum viable product and then iterate it with customer feedback. Early product releases are designed, scoped, and timed to aid requirements discovery and validation. The utilized technologies and practices largely depend on teams' skills and experience. Therefore, the team composition, attitudes towards following good engineering practices, and the ability to jointly advance towards set objectives is critical to iterate the product rapidly.

IN CHAPTER 3, we continue analyzing the experience reports perform root-cause analysis of common engineering pitfalls. We investigate the real causes of problematic situations in start-ups and formulate three anti-patterns.

Symptoms of the first anti-pattern are substandard product functionality and quality and long time-to-market. The root causes of such symptoms stem from inadequacies in requirements engineering, e.g., vague requirements, frequent changes in priorities, and lack of focus. The consequences of this pattern are resource overruns, degrading team morale, missed opportunities, and eventual closure of a start-up.

We identify that poor software design decisions exacerbate the adverse effects of inadequate requirements engineering. The use of suboptimal and immature technologies and not taking advantage of third-party components prolong time-to-market. Excessive time-to-market delays requirements validation with customer input and increase the opportunity cost of developing potentially irrelevant features.

The second anti-pattern concerns poor product results in the market. A symptom of this anti-pattern is the difficulty of attracting customers, complicated on-boarding process, and expensive product maintenance. The root causes of this anti-pattern are lack of early customer involvement in product engineering, excessive technical debt, focus on wrong quality attributes, and wrong level of per-customer customization.

The third anti-pattern concerns slower than expected growth, churning customers, and slow entry in new markets. We identify that the root causes for such symptoms could be the lack of support for continuous requirements engineering, insufficient organizational support for existing features and customers, and excessive technical debt.

We conclude that start-ups should focus on better software engineering practices, not more of them. Thus, timely addressing the root causes of common issues and improving the chances of product success.

IN CHAPTER 4, we use the results from two earlier chapters to formulate a focused inquiry into engineering goals, challenges, and practices. We perform an analysis of 84 start-up cases characterized by 23'940 data points. This chapter delivers two contributions, the start-up life-cycle model and the start-up progression model.

Based on related work, we formulate a start-up life-cycle model. The key idea for the model is that start-ups evolve through four distinct life-cycle phases. To advance to the next stage, start-ups must achieve specific goals, and start-ups succeed and fail at each stage for different reasons. We use this model to explain the evolution of engineering practices in start-ups.

The start-up progression model comprises four life-cycle phases and six engineering knowledge areas. We map utilized engineering practices, goals, and challenges in each knowledge area to a specific start-up life-cycle phase. Thus, creating a model mapping the evolution of software engineering in start-ups.

We conclude that the main engineering challenge in start-ups is to manage the evolution of engineering practices. A practice that performed well

in a small team and single product may not be sufficient as the team and product grow. Thus, start-up engineers must continuously assess and develop engineering practices to match the scale and complexity of the product.

IN CHAPTER 5, we analyze the perception of technical debt, its precedents, and outcomes. We use the same dataset as in Chapter 4, albeit with a focus on technical debt.

We found an association between the estimates of technical debt and start-up outcome. Closed and acquired start-ups estimate their technical debt significantly higher than active start-ups. Such results add support to our earlier findings that technical debt is a substantial challenge in start-ups contributing to their failure.

We identify team size and the level of engineering skills as critical determinants for attitudes on incurring technical debt. Larger teams tend to be more pragmatic and report more precedents for accumulating technical debt than smaller teams. Furthermore, start-ups in the growth phase perceive the most adverse effects of excessive technical debt. Such results suggest that start-ups realize the amount of incurred technical debt when they face an influx of customers, and the development team grows.

The effects of technical debt in start-ups are impaired product quality and team productivity. However, the adverse effects could also stem from software decay. Distinguishing between avoidable technical debt, i.e., trade-offs between speed and quality, inadvertent technical debt stemming from poor engineering skills, and software decay caused by growing product complexity is needed to provide practical support to minimize the adverse effects.

IN CHAPTER 6, we analyze associations between the use of agile practices, team, and product factors. We use the same dataset as in Chapters 4 and 5, however with a focus on agile practices.

Our results suggest that start-ups use agile practices to a large extent, however without following any specific agile methodology, e.g. scrum or XP. The most frequently reported practices are backlog, version control, refactoring, user stories, and unit tests. Although most of the studied start-ups characterize their development mehtodology as agile, the use of agile practices does not imply that start-ups follow agile principles as set forth by Agile Manifesto.

We identify a number of statistically significant associations between the use of agile practices, team, and product factors. Interestingly, use of agile practices is associated with both positive and negative situations. Practices, such as unit testing, continuous integration, story mapping, and iterations, are associated with positive estimates on testing, quality, less resources and time pressure, and improved source code quality. However, practices

such as backlog, definition of done, and burndown chart are associated with unfavorable estimates on resources constraints and attitudes towards following the best practices. We identify a total of 22 such associations.

To explain the associations, we set forth a number of propositions. We identify team's attitudes towards following the best practices as one of the main denominators for using agile practices. However, the association is negative, that is, teams could be using practices such as definition of done and burndown charts to introduce more control over unsatisfactory performance. We also identify that use of agile practices is associated with more favorable estimates on the overall product quality, testing, and resources constraints.

IN CHAPTER 7, we examine to what extent empirical evidence support the unique characteristics of start-ups.

By examining 15 broadly used start-up characteristics, we found that most are anecdotal with little empirical support. Furthermore, we found that some of the characteristics are general to any projects, not only start-ups. For example, balancing project scope, resources, and time is hardly a start-up specific problem.

We conclude that there is no basis to claim that start-ups are unique from the software engineering angle. There is not enough empirical evidence to claim that start-ups need a different approach to software engineering.

We identify that the rapid evolution of the start-up, team, and product requires more, not less, careful orchestration of engineering practices. Thinner margins for error require more diligent planning and risk management practices. Lastly, the use of venture capital could introduce new objectives for attaining particular financial objectives. The financial objectives may not align with the goal of delivering sustainable customer value. Thus, the management of stakeholders and their expectations in start-ups requires additional support.

IN CHAPTER 8, we propose a systematic and collaborative method for selecting stakeholders and data sources to support specific product decisions.

Most existing stakeholder selection practices are suitable for classifying already known human stakeholders. However, in market-driven and crowd requirements engineering contexts, stakeholders may be unknown, uninterested, or not able to articulate their needs. Thus, there is a great potential of using data, documents, existing artifacts, process analysis, and similar indirect sources to learn about stakeholders' actual needs. However, considering all sources is not practical and could lead to an overload of requirements ideas.

Requirements engineering in a market-driven context can be seen as a series of micro-decisions guiding further product development. What data

sources are the most relevant to support each decision changes depending on the decision at hand. Furthermore, requirements engineering is a collaborative activity and often involves multiple perspectives, e.g., engineering, marketing, finances, and business.

We propose a method comprising of systematic steps to collaboratively identify and compare the most relevant data sources for a given problem situation. The method helps to explore and illustrate different perspectives and resolve discrepancies between analysts' perspectives.

We pilot the method in two industry workshops with positive results. However, we identify the potential to improve the method by providing tool support and taxonomies categorizing evaluation criteria and candidate data sources.

4 SYNTHESIS

I this section, we formulate answers to our main research questions, see Section 2.2. We answer the sub-questions in each chapter.

4.1 *RQ1: How is software engineering practiced in start-ups?*

Our results show that start-ups do not follow any specific methodology or approach. The selection of engineering practices is ad-hoc and depends on the skills and experience of the engineering team. New practices are often introduced as remedies to already problematic situations (Chapters 2 and 4).

We found that the main driver behind using a sub-standard set of practices, apart from lack of engineering skills, is the opportunity cost. Such results contradict some of the earlier results, e.g., Paternoster et al. [2014], suggesting that lack of funds is the main reason for not following the best engineering practices (Chapter 4).

The use of sub-standard practices often leads to the accumulation of technical debt, reduced productivity, and product quality issues. However, this becomes a concern only if a start-up survives long enough and attempts to scale up its operations. Identification and removal of technical debt is a substantial engineering challenge in maturing start-ups (Chapters 5).

Exploring the customer problem and devising a feasible solution, that is, requirements engineering, is the core engineering activity in start-ups (Chapters 2- 4). However, requirements engineering is also the most challenging task due to uncertainty around markets, customer needs, technology limitations, and the lack of suitable practices (Chapter 8). We found that start-ups mostly invent the product requirements and attempt to validate their ideas with prototyping, frequent product releases, and drive further product development with customer input (Chapter 4).

The invention of requirements is a crucial step. Invented ideas stem from teams' domain knowledge and previous experience. Thus, knowledgeable teams have a better chance of inventing relevant and good-enough requirements. However, our studies show numerous cases where teams attempt to invent requirements in a poorly understood domain leading to a challenging validation process and a failure of the product idea altogether (Chapter 2, 3, and 4).

The utilization of other software engineering knowledge areas varies. We have observed a tendency that start-ups who follow agile software design principles, leverage open-source components, and pick a mature technology stack can deliver their products to early customers faster and with higher quality. However, start-ups who attempt to devise large upfront designs, use unreliable technologies, and invest their own resources in developing commodity features generally perform worse (Chapters 2 and 4).

There is little control over the engineering process, especially in early start-ups. Companies report using high-level plans to guide their progress. Superficial planning and control lead to poorly anticipated resources and schedule overruns. As soon the product is launched, start-ups start monitoring external metrics such as a number of customers, revenue, retention, and churn. Late start-ups also monitor internal performance metrics (Chapters 4 and 6).

In our studied sample, there several start-ups who have explicitly stated sub-standard engineering as the primary cause for failure. The shortcomings concern the inability to establish a feedback loop with customers and a poor choice of technologies leading to slow progress, product quality issues, and expensive maintenance (Chapters 2 and 4).

We have not yet identified a case where utilization of the best engineering practices would have caused any adverse effects. However, the selection of best engineering practices for a given situation remains a challenge.

4.2 *RQ2: How should start-ups practice software engineering?*

Analysis of the differences between closed and active companies reveals that team composition is crucial in early start-ups. A team should possess relevant domain knowledge, technical capabilities to build the product, and should commit time to work on the start-up (Chapter 4).

Domain knowledge is crucial to identify a relevant product idea and to understand the market, e.g., to identify potential hires, partners, competitors, investors, customers, and other relevant stakeholders. Start-ups can acquire domain knowledge on the go, for example, through market research and rigorous requirements engineering. However, we argue that having such knowledge beforehand substantially increases start-ups' chances of rapidly launching a relevant product.

Technical capability refers to access to technologies, skills, tools, partnerships, and other resources to build the product. The access can be secured either by creating an in-house engineering team or outsourcing the work to another company.

Start-ups who have created in-house teams report faster results with higher quality than companies who have outsourced the engineering effort. The decision on whether it is worth outsourcing the product engineering depends on the availability of a qualified workforce, the scope of the product, among other aspects.

Our results show that teamwork issues may disrupt a start-up. We identify that other commitments, physical distance, motivation issues, the need to continue working a day job to finance the start-up, significantly impair the teamwork in a start-up. Unsolved teamwork issues could lead to the eventual closure of a start-up.

A key practice in start-ups is to establish contact with potential customers early and involve them in product engineering. The early contact helps to gauge the relevance of the product idea and to refine it. Later such customers become early adopters of the product. A key challenge in working with specific customers is to balance customer-specific features with market-oriented features. Customers could also be uninterested to invest their time and not able to articulate their needs towards the product (Chapters 2, 3, 4, and 8).

In addition to consulting potential customers, start-ups analyze similar products, competitor offerings, market trends, and leverage their own experience to identify new requirements ideas (Chapters 4, and 8).

Start-ups should manage their risks by selecting well known and mature frameworks, components, libraries, and programming languages. Mature development frameworks are supported by broad communities providing training, advice, tooling, components, experts, and the best practices. Start-ups can significantly benefit from such support in contrast to investing in inventing and maintaining new technologies in-house (Chapter 4).

As the start-up evolves, the engineering process should evolve as well. Practices and methods that worked well with few engineers and a small product could become obsolete when the product becomes more complicated. Generally, there is a need for more practices supporting communication, coordination, and automation in the organization. Ideally, the evolution of engineering practices is proactive, not reactive (Chapter 4).

As a start-up evolves, the business requirements towards the product evolve as well. The initial aim could be to quickly validate the product idea, later the aim shifts towards providing reliable and efficient service and reaching specific financial targets. Such business objectives must be recognized and translated to engineering objectives (Chapter 4).

4.3 *RQ3: What are the differences between start-ups and established organizations?*

Looking at the results of how start-ups practice software engineering, we noticed the lack of novel and unique findings. We have not identified any start-up specific engineering practices. Most of the engineering challenges in start-ups are also observed in established companies (Chapter 4). Thus, start-up challenges could be solved by transferring relevant practices from other contexts. Such results inspired us to reexamine the differences between start-ups and established organizations (Chapter 7).

By examining relevant literature, we found that the most widely recognized start-up characteristics are anecdotal. We found that challenges that are identified as characteristic to start-ups are also present in other types of organizations. Such findings point towards an opportunity for bi-directional transfer of the best engineering practices between start-ups and established organizations.

We searched for suitable solutions to requirements engineering challenges in start-ups and discovered a general lack of good practices for market-driven requirements engineering (Chapter 8). Both start-ups and established companies struggle to manage a continuous flow of requirements from various sources. Identification of what sources and requirements are relevant is an unsolved challenge.

Start-ups often utilize venture capital to fund their operations. Venture capitalists can be seen as another critical group of stakeholders with needs to achieve ambitious financial targets quickly. Management of the trade-off between reaching such targets, delivering customer value, and attaining long term objectives of the start-up could add layer of complexity.

The most crucial engineering challenge in start-ups is to manage growth. As we illustrate in Chapter 4, engineering practices evolve from simple to advanced as the team, organization, and the product grows in size and complexity. In comparison, established organizations are more stable and have more resources to support such evolution. There have been proposals for agile maturity models, e.g., Patel and Ramachandran [2009] and Salo and Abrahamsson [2007], however, the suitability of such models for use in start-ups remains unknown.

Due to their small size, dependency on venture capital, and focus on a single product, start-ups have a thinner margin for errors compared to established organizations. The errors concern both product decisions, i.e., what features to offer, and process decisions, i.e., determining the most efficient way of delivering the features. Larger organizations could compensate for the errors and inefficient practices with extra resources (Chapter 7).

Thinner margins for error points towards the need to get the product engineering right the first time. Therefore, the best engineering practices are more, not less, relevant in start-ups.

5 Conclusions and Further Work

In this thesis, we aimed to establish a coherent view of how start-ups practice software engineering. A coherent view is needed to: a) enable a more focused, in-depth studies on specific challenges, b) provide an engineering road-map for start-ups supporting their engineering efforts. c) support the transfer of the best engineering practices between start-ups and other organizations.

To reach our aim, we have conducted two considerable data collection efforts resulting in the analysis of 172 start-up cases, (Chapters 2 and 4). Through the analysis, we have mapped the research area and identified relevant start-up life-cycle stages, engineering knowledge areas, practices, challenges, and objectives. As one of our main contributions, we formulate the start-up progression model that works both as a map of the research area and a practical guide for start-ups (Chapter 4). The model pinpoints requirements engineering and the evolution of engineering practices as primary focus areas in start-ups.

In regards to improving software engineering in start-ups, we identify a need for scaled-down, start-up specific lean and agile methodologies supporting both market-driven context and continuous evolution of the engineering process.

Our results show many similarities between start-ups and other types of organizations in terms of new market-driven product engineering (Chapters 4 and 7). Such results highlight the opportunity for the transfer of the best practices between start-ups and established organizations.

Our analysis of specific challenges in start-ups highlights the lack of relevant requirements engineering practices to support new product development in crowd and market-driven contexts (Chapters 2, 4, 7, and 8). Both start-ups and established organizations face the same requirements engineering challenges when working with a large number of loosely defined stakeholders.

As a starting point for future work, we propose a method for selecting data-sources to support requirements elicitation (Chapter 8). The method is our first step towards a vision of data-driven requirements engineering, considering both individuals and inanimate data sources. With the increasing amount of data available, we envision that data could be used to augment current, manual effort-intensive, requirements engineering practices. The use of statistical methods and machine learning could add a quantitative angle to understanding the needs of large groups of stakeholders.

2

SOFTWARE ENGINEERING IN START-UP COMPANIES –ANALYSIS OF 88 EXPERIENCE REPORTS

Start-up companies have become an important supplier of innovation and software-intensive products. The flexibility and reactiveness of start-ups enables fast development and launch of innovative products. However, a majority of software start-up companies fail before achieving any success. Among other factors, poor software engineering could be a significant contributor to the challenges experienced by start-ups. However, the state-of-practice of software engineering in start-ups, as well as the utilisation of state-of-the-art is largely an unexplored area.

In this study we investigate how software engineering is applied in start-up context with a focus to identify key knowledge areas and opportunities for further research.

We perform a multi-vocal exploratory study of 88 start-up experience reports. We develop a custom taxonomy to categorize the reported software engineering practices and their interrelation with business aspects, and apply qualitative data analysis to explore influences and dependencies between the knowledge areas.

We identify the most frequently reported software engineering (requirements engineering, software design and quality) and business aspect (vision and strategy development) knowledge areas, and illustrate their relationships. We also present a summary of how relevant software engineering knowledge areas are implemented in start-ups and identify potentially useful practices for adoption in start-ups.

The results enable a more focused research on engineering practices in start-ups. We conclude that most engineering challenges in start-ups stem from inadequacies in requirements engineering. Many promising practices to address specific engineering challenges exists, however more research on adaptation of established practices, and validation of new start-up specific practices is needed.

Software start-ups are important suppliers of innovation and innovative software products [Baskerville et al., 2003], providing products and services that are a significant part to the economy [Startup Compass Inc., 2015]. This potential is strengthened further as the use of cutting-edge technologies enable start-ups to develop, launch and evolve software products fast and with very few resources [Baskerville et al., 2003].

A challenge is that most start-up companies collapse before any significant achievements are realized [Tovstiga and Grossmann, 2012]. This is explained by market conditions, lack of commitment, financial issues or, simply put, a bad product idea. However, product engineering activities takes substantial resources from start-ups, especially in the early stages [Crowne, 2002, Giardino et al., 2015a]. Inadequacies in used engineering practices could lead to under or over-engineering the product, wasted resources, and missed market opportunities.

One of the main qualities of start-ups is their ability to quickly take advantage of new business, market and technology opportunities [Giardino et al., 2014a, Bajwa et al., 2017a]. Decisions, such as what features to build, how and when, belongs to the realm of engineering and have a huge impact on how the start-up responds to new opportunities. For example, certain decisions may hamper flexibility of the product, thus reducing the speed of adapting the product for entering new markets. Taking one suboptimal decision may have only a small effect on start-up's prospects, however the compound effect of the decisions determines whether the start-up is able to remain on the edge of innovation or is struggling to keep its product running. This is a source of risk and opportunity with potential effects to all aspects of the company.

Yau and Murphy [2013] argue that practices adapted from established companies attempt to solve problems that are not present in start-ups, while ignoring start-up specific challenges, such as time-to-market as the primary goal, and accumulating technical debt [Giardino et al., 2016]. Even though similar challenges can be present in established organizations too and addressed by state-of-the-art practices, it is the combination of multiple challenges that makes engineering in start-ups difficult. There is a gap in understanding how these start-up specific challenges influence the engineering process and what engineering practices are suitable for such context [Klotins et al., 2015, Paternoster et al., 2014].

This lack of understanding results in that there are very few, if any, start-up context relevant software engineering processes/methods/models/frameworks (called practices from now on). At the same time, a substantial amount of money is invested in start-up companies. In the first three quarters of 2015 alone, start-up companies received investments of 429 billion USD in the US and Europe [PitchBook Data, 2015, PitchBook Data, Inc.,

2015]. With an optimistic start-up failure rate of 75% this constitutes 322 billion USD of capital potentially wasted on building unsuccessful products [Startup Compass Inc., 2015, PitchBook Data, 2015, PitchBook Data, Inc., 2015]. To what extent inadequacies in software engineering practices are responsible or linked to success rate is very hard to judge. However, even if the effect of improved engineering practices only would result in a few percent change in success rate, it would yield significant impact and capital return. Thus, the focus of our study is to explore specifically software engineering in start-ups and pinpoint specific areas for further research that are likely to benefit start-up practitioners.

Researchers have recognized the importance of software engineering in start-ups. Bosch et al. [2013] and Deakins and Dillon [2005] propose adaptations of iterative and incremental development methods to address engineering challenges in start-up companies. However, this work is preliminary and has not been validated yet in practice [Klotins et al., 2015, Paternoster et al., 2014]. Giardino et al. [2016] report on an interview study aiming to understand how start-ups select their product development strategy and how start-ups consider product quality attributes. Giardino et al. [2015a] also investigated the key challenges in software start-ups and report that technology uncertainty is the key challenge in software start-ups. However, none of these studies provide a comprehensive answer of what engineering practices are relevant in start-ups.

In this paper we use empirical data from 88 start-up experinece reports to provide the first insight into what engineering knowledge areas are relevant in software start-ups. Our aim is to explore what engineering practices the start-ups report as relevant, how these practices are applied and what results they yield. To analyze the reports we use qualitative data analysis methods [Seaman, 1999, Garousi et al., 2016]

To cater for the fact that the experience reports cover also business and marketing aspects, which are tightly intertwined with software engineering aspects, we use a software and business practices taxonomy to support the analysis of the reports. Even though we acknowledge importance of good business, market and other practices, the focus of this paper is strictly on software engineering practices.

The main contribution of this paper is the identification and description of the state-of-practice in software start-up companies, pinpointing to several relevant software engineering areas that need further research. Moreover, we present related work to each relevant software engineering knowledge area, illustrating potentially useful engineering practices for start-ups.

The remainder of this paper is structured as follows: Section 2 presents background and related work and Section 3 introduces the research methodology. The results are presented in Section 4, analysed and discussed in Section 5. Section 6 concludes the paper.

2.1 *Software start-ups*

As early as 1994, Carmel [1994a] recognizes small software companies being exceptionally successful at innovation and delivery of new products. Termed software start-ups, these companies share many features with small and medium enterprises such as market pressure, youth and immaturity, and limited resources [Sutton et al., 2000]. However, start-up companies are different due their goals and challenges. Contrary to established companies who aim to shape their products to address a known market need, start-up companies attempt to identify an unmet market need and to invent a product satisfying this need [Startup Compass Inc., 2015].

The engineering context in start-ups is characterized by uncertainty, lack of resources, rapid evolution and an immature team among other factors [Sutton et al., 2000]. However, the start-up context also provides flexibility to adapt new engineering practices and reactiveness to keep up with emerging technologies and markets [Giardino et al., 2014a].

Product related issues are reported as the key challenge in start-up companies [Giardino et al., 2015a]. However, recent literature mapping studies identify a lack of research in the area from a software engineering perspective [Klotins et al., 2015, Giardino et al., 2014a]. Moreover, most publications to date are experience reports lacking in-depth analysis and rigorous research methods on empirical data [Klotins et al., 2015].

Despite attempts to explore the start-up phenomenon, only a few studies specifically focus on understanding how software engineering is done in start-up companies. Yau and Murphy [2013] and Sutton et al. [2000] recognized that engineering practices aimed at established companies are not suitable for start-ups. As a result, various models for software development in start-up context were proposed [Deakins and Dillon, 2005, Bosch et al., 2013, Zettel et al., 2001], however there is very little evidence of application and validation of these models [Klotins et al., 2015].

An agenda to specify important research topics regarding software engineering in start-ups has been created by the Software Start-up Research Community [Unterkalmsteiner et al., 2016]. Among other topics, engineering practices in start-ups is identified as an important research area.

Rafiq et al. [2017] and Melegati et al. [2016a] have studied requirements engineering practices in start-ups and provide an insight into how product ideas evolve and what practices are used to connect founders' vision with customer needs.

Crowne [2002] proposes a start-up life-cycle model and identifies goals and the key challenges at each of the life-cycle phases. According to the model, in the first phase a company develops an early version of the product. The goal of the second phase is to improve quality of the product until

it can be commissioned with little effort. In the third phase, the company grows and conquers the market. In the fourth phase the company matures into an established organization. However, there is no detailing on the use or adaption of engineering practices, or their evolution over time and phases.

Giardino et al. [2014b] present a behavioral model aimed at explaining start-up failures. They argue that a start-up must first explore the problem domain and then validate a proposed solution, however mismatching validation activities with exploration activities could lead to a failure. This is directly related to engineering practices such as overall requirements engineering or scoping.

Olsson and Bosch [2015] identify challenges of using customer feedback in large software-intensive product engineering. A case study reveals that to compensate for inadequate utilization of user feedback, companies invent requirements and steer product direction by "gut feeling". Invented requirements lead to a large amount of unused and incorrectly implemented features contributing to product failure.

Software engineering in start-ups shares many similarities with companies using agile development practices, such as iterative development, empowered small team, and ongoing planning [Ramesh et al., 2007, Chow and Cao, 2008]. However, customer involvement, which is one of the key agile principles, is difficult to establish as start-up companies lack a distinct set of customers. Hence, start-ups operate in a market-driven environment [Dahlstedt et al., 2003]. In a market-driven environment, requirements are often invented and validated through frequent product releases [Dahlstedt et al., 2003, Alves et al., 2006].

There has been a substantial work to identify general operating practices for start-ups, such as The Lean Start-up [Ries, 2011] and Customer Development Model Blank [2013a]. However, there exist very few peer-reviewed studies in software engineering fora, with May [2012] being the exception, reporting on the application of said practices. Therefore, it remains to be explored to what extent these or any other practices are adopted by start-ups.

2.2 *Scope of software engineering in start-ups*

Sutton et al. [2000] argues that start-up companies are sensitive to many influencing factors, such as customers, partners, changes in technologies and markets. The developed software is often a component of another product and which is an essential part of the start-up company itself [Baskerville et al., 2003]. Therefore, software engineering in start-ups must be studied jointly with its dependencies and influences to other areas of a start-up [Broy, 2006].

The interrelation of software engineering, product development and other areas in an organization is recognized by ISO/IEC 42010:2011 which provides a definition of software-intensive system as: "any system where software contributes essential influences to the design, construction, deployment, and evolution of the system as a whole to encompass individual applications, systems in the traditional sense, subsystems, systems of systems, product lines, product families, whole enterprises, and other aggregations of interest" [ISO/IEC 42010, 2011].

To fully understand the importance of software engineering in start-ups we widen our field of view and capture other concepts that influence or depend on software engineering. To define the scope for this study we use a taxonomy, further elaborated in Section 2.3.

2.3 *Software engineering and business practice taxonomy*

A taxonomy is useful for categorization and mapping of knowledge, facilitating identification of gaps and boundaries of a phenomenon [Šmite et al., 2014]. Seaman [1999] suggests to use of preformed codes, e.g. a taxonomy, to facilitate coding in qualitative studies. Hence, to support the identification of software engineering and other relevant practices we developed a taxonomy listing knowledge areas and practices for start-ups - the SoftWare and Business Process, hereinafter the SWBP, taxonomy. The taxonomy consists of software engineering knowledge areas as defined by SWEBOK, and several business aspects oriented knowledge areas inspired by Osterwalder et al. [2005] and [Zachman, 2003].

Even though SWEBOK is not specifically designed for software start-up companies and lacks emerging areas of software engineering, such as value-based software engineering [Boehm, 2003, Azar et al., 2007] and market-driven requirements engineering [Dahlstedt et al., 2003, Karlsson et al., 2007], it promotes a consistent view on software engineering, and is well recognized within software engineering community [Sicilia et al., 2005, Budgen, David Turner, Mark Brereton, Pearl Kitchenham, 2008].

Other frameworks, such as CMMI [Software Engineering Institute, 2006] and SPICE [Dorling, 1993], are oriented towards software process identification, assessment and improvement. However, as indicated by Giardino et al. [2014a] and Sutton et al. [2000], start-up organizations are very immature in a process sense, thus identification of engineering processes could be difficult.

Several business practice frameworks, such as ABPMP BMP CBOK [ABPM, 2009], papers by Osterwalder et al. [2005] and Zachman [2003], aim to map and categorize business practices, however none of them is specific to start-up companies [Blank, 2013b]. Due to youth and immaturity of start-up companies, traditional business processes are difficult to identify.

Therefore, we created a simplified business practice taxonomy based on ABPMP BMP CBOK. The exact categories of the taxonomy are provided in the supplementary material[1].

We use the unified SWBP taxonomy to identify and categorize all relevant practices to explore software-intensive product development in start-ups.

3 RESEARCH METHODOLOGY

3.1 *Research questions*

Our research goal is to investigate how software engineering is practised in start-up companies and how software engineering contributes to other areas of a start-up, such as business and marketing. To break down the research goal further, we formulate the following research questions:

RQ1: What software engineering knowledge areas do software start-up companies consider most relevant?

We explore what knowledge areas and specific categories in software engineering are of interest for start-up companies. This enables to focus further research on the areas that have the potential to actually support software start-up companies.

RQ2: How are the identified knowledge areas applied in start-ups?

We explore how the identified knowledge areas (RQ1) are implemented in start-up companies. This increases the understanding of how, or whether at all, software engineering knowledge and practices are applied.

RQ3: What are the relationships between different knowledge areas?

We explore the relationships between the applied practices to understand how practices and knowledge areas affect each other. The understanding of the relationships enables the analysis of how significant each practice is in context of other practices.

RQ4: What practices are missing to support software engineering in start-up companies?

Having identified relevant knowledge areas (RQ1), how the knowledge areas are implemented in terms of practices (RQ2), and relationships between the practices (RQ3) we identify gaps in applied practices. We review related work to identify practices that could be useful for adaptation in start-ups.

3.2 *Data sources and collection*

We use a collection of start-up experience reports [CBInsights, 2015] as the data source. The reports represent a primary data source, i.e. we perform

1 http://eriksklotins.lv/files/exp-reports-study-supplemental-material.pdf

original analysis of data, and we perform third degree analysis, i.e. we independently analyze artifacts that are already available [Lethbridge et al., 2005].

The experience reports describe lessons-learned from start-up companies, written by one of the participants after critical events occurred in the start-up, for example, a key person leaving a company, a product launch, a buyout or closure of the company. Even though the reports represent "multi-vocal literature", i.e. are unstructured and vary in length, focus and style [Ogawa, R. T.; Malen, 1991, Tom et al., 2013], they provide a rich insight on how start-ups perceive and approach software-intensive product engineering [Garousi et al., 2016]. The reports cover a much broader scope than just product engineering thus providing insights how product engineering is influenced by and what influence engineering has on other factors in start-ups.

We screened the collection to remove inaccesible or otherwise irrelevant reports, prior to the detailed analysis, according to the following criteria:

1. The report is inaccessible, for example, the link provided in the website to the actual report is broken.
2. The report clearly does not describe experience from a start-up company, for example, the report describes experiences from an established company.
3. The report clearly does not describe experiences from a software start-up, i.e. the start-up does not do software engineering in any way.

From the initial set of 93 reports, 5 reports were removed and 88 remained for further analysis. Secondary data sources are start-up profiles [cru] that record company track length, geographical location and members of their founding teams. Figure 2.1 illustrates how each data source contributed to answering the stated research questions.

We created a simple database tool to import, store and maintain traceability between different pieces of data. Prior to storing, we trimmed irrelevant information, i.e. ads, unrelated pictures, links, from the reports and split the reports into chunks by a paragraph for further analysis. A paragraph was selected as unit of analysis as it is large enough to convey a complete statement and small enough to maintain traceability. The splitting was first done with an automated tool and later manually revised. The final chunks were 1 - 16 sentences long.

3.3 *Analysis design and execution*

To analyze the reports we use qualitative data analysis methods, that is, different types of coding and themeing of concepts [Seaman, 1999, Saldaña,

2015]. Figure 2.1 illustrates the used data sources, the three analysis steps, the output and how it answers the research questions.

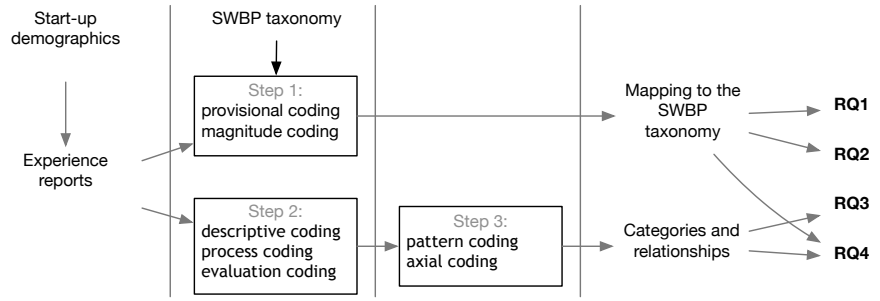


Figure 2.1: Overview of the coding process and research questions

In step 1 we apply provisional coding we explore the data and identify and categorize statements in the reports related to software engineering or business development [Saldaña, 2015]. We associate statements in the reports with preformed codes from the SWBP taxonomy (see Section 2.3), as suggested by Seaman [1999].

Our study is specifically angled towards software engineering practices and their connections to business knowledge areas, thus the SWBP taxonomy supports identification of relevant data to answer our research questions. We associate statements in the reports with a knowledge area, category or a sub-category from the taxonomy. More explicit statements are mapped to sub-categories, while more general statements are associated with a knowledge area in general. Mapping to the lowest level enables analysis both on knowledge area level and a more specific analysis of what sub-categories in a particular knowledge area are addressed. To code ambiguous statements we use simultaneous coding [Saldaña, 2015], mapping the statement to multiple categories of the SWBP taxonomy. To further describe each identified statement we added two magnitude sub-codes [Saldaña, 2015]. The first sub-code captures the impact direction of a described practice. We use the values “positive", “unknown" and “negative" to capture the report’s author own reflection on the impact of a practice. For the second sub-code we use the values “product", “business" and “both" to capture the affected object. The scope for each code is a sentence in a report. Sub-codes are added on top of each code.

In step 2 we perform another pass on each report and look at the reported experience as a whole to identify key concepts, analysis points, leading to gains or losses in software engineering or business development. Looking at the reporter’s opinion on what activities had significant impact on software engineering, we attempt to identify contextual factors and activities causing the high impact situation. We make use of *descriptive*

(to summarize), *process* (to capture ongoing action) and *evaluation* (to assess the situation) coding [Saldaña, 2015] jointly to capture analysis points in a report. Through analysis of the described situation we aim to differentiate between reported symptoms (e.g. running out of resources) and actual causes (e.g. poor resource planning due to lack of experience). Coding in this step is open, i.e. we let codes to emerge naturally without use of the taxonomy. Examples of coding in the steps 1-2 are provided in the supplementary material [2].

In step 3 we apply pattern and axial coding [Saldaña, 2015] to combine similar analysis points and to establish relationships between emerging categories. As similar activities and contextual factors recur in the data, we group them under a candidate category. We merge, split and update the candidate categories during the coding process. A category gains full category status when its category description allows understanding of characteristics, conditions, consequences and interaction of the expressed concept. To understand how different concepts influence each other we further employ axial coding [Corbin and Strauss, 1990] looking for possible causes and consequences across all analysis points forming each category. This enables deeper understanding of a concept and provides multiple explanations for its emergence and impact. Field memos in a form of white board drawings, notes and mind-maps were created to record any discoveries in the data. During the analysis we kept track of what specific statements from reports actually support the category or the relationship and continuously update a category description. We use this information to further develop or discard patterns emerging from the data.

Saldaña [2015] suggests to connect categories with underlying concepts by applying theming portions of data. Where possible, we associate the categories with practices from the the SWBP taxonomy. The association connects the categories to state-of-the-art enabling further elaboration and exploration of a category.

3.4 *Answering the research questions*

We answer RQ1 (What software engineering knowledge areas do software start-up companies consider most relevant?) by counting how frequently each knowledge area is discussed in the reports. Some reports repeatedly address the same issue resulting in multiple identical codes and impact sub-codes. Such repeated codes are useful for further qualitative analysis, however are counted only once in the quantitative analysis in order not to inflate the importance of a knowledge area. By differentiating between the reported positive or negative impact on software engineering or business development aspects, we identify potential inadequacies in the application

2 http://eriksklotins.lv/files/exp-reports-study-supplemental-material.pdf

of knowledge in that particular area, specific for the software startup context. The analysis of this research question is presented in Section 4.2.

We answer RQ2 (How are the identified knowledge areas applied in start-ups?) by summarizing reported practices relevant to each knowledge area.

We answer RQ3 (What are the relationships between different knowledge areas?) by developing a graph illustrating relevant knowledge areas and their relationships. By looking at the number of data-points we identify the most important relationships for further exploration. We discuss the relationships in context of related work.

We answer RQ4 (What other practices are missing to support software engineering in start-up companies?) by identifying the most interconnected categories and their relationships in Fig. 2.9 (RQ3). We assess the central categories in context of the related categories, examine if the answer to RQ2 suggest any particular practice pertaining the relationship, and review related work for candidate practices.

3.5 *Validity threats*

We present four categories of validity threats as proposed by Runeson et al. [2012].

3.5.1 *Construct validity*

Construct validity is concerned with to what extent the studied operational measures represent what the researcher is attempting to investigate [Runeson et al., 2012].

A possible threat is that we may fail to recognize relevant practices in the reports. To address this threat we use a taxonomy to support identification and categorization of statements from the reports.

We use the SWBP taxonomy as a framework to identify and categorize statements from the reports.

However, the SWBP taxonomy is partly based on SWEBOK and may not up to date with emerging concepts in software engineering such as value based software engineering [Boehm, 2003, Azar et al., 2007] and market-driven requirements engineering [Dahlstedt et al., 2003, Karlsson et al., 2007]. To address the threat that some important aspects are missed due to the lack of a comprehensive taxonomy we use two separate coding strategies, i.e. provisional coding (based on a taxonomy) and analysis points (independent from any taxonomy).

Due to lack of detail or terminology differences between the reports and the taxonomy it could be challenging to map certain statements to specific software engineering practices. We address this threat by a) applying multiple codes to the same statement to capture multiple interpretations

and b) mapping the statements to different levels of the taxonomy. As a possible consequence of multiple codes per statement, we may incorrectly estimate the number of statements addressing a particular category of the taxonomy. However, multiple codes also enable identification and analysis of closely related and overlapping practices.

Another threat to construct validity is a possible bias stemming from the nature of subjective experience reports. In their essence, the reports are self-evaluations by the authors. They may have overlooked their own shortcomings, e.g. a lack leadership or technical skills, and rationalized their experience with external circumstances [Pronin et al., 2002]. However, due to relatively large sample and diverse population we likely cover different personality types, therefore minimizing this threat [Feldt et al., 2010]. An alternative solution could be to use grounded theory [Corbin and Strauss, 1990] and study a smaller sample in more detail, thus strengthening the internal validity at the expense of generalizability.

Even though we applied multiple remedies to address the threats that stem from the nature of the data we have collected, we are conservative in the conclusions that we draw from the data and analysis.

3.5.2 *Reliability*

This aspect is concerned with the extent to which the results and analysis are independent from the specific researchers [Runeson et al., 2012].

We address reliability by ensuring transparency and traceability throughout our data collection and analysis, providing a detailed description of the applied research method. All analyzed experience reports are provided as supplemental material[3].

We have kept traceability information throughout our results and analysis linking specific conclusions with supporting statements in the experience reports.

3.5.3 *Internal validity*

Internal validity is threatened when a researcher is investigating one factor affecting another factor and there exists a third, unknown factor, that confounds the studied relationship without the researchers knowledge [Runeson et al., 2012].

A possible threat here is the single researcher bias in the coding process. To address this threat we applied researcher triangulation. Twice in the coding process, at the beginning and later in the process, selected reports were analyzed independently by three researchers and the results discussed to identify and eliminate any individual biases. In addition to the

3 http://eriksklotins.lv/files/exp-reports-study-supplemental-material.pdf

triangulation, intermediate results were discussed among the researchers and compared to state-of-the-art.

3.5.4 *External validity*

External validity concerns the ability to generalize the result of research efforts to industrial practice and to what extent the results are of interest outside the investigated case [Runeson et al., 2012].

A potential threat to external validity is sampling. For our study we have used convenience sampling and studied a relatively large number of cases from different geographical regions, market segments, team composition variations, product types, and consider both successful and failed cases.

The majority of start-up cases in our sample (63 of 88, 72%) are closed companies. As elaborated in Section 4.1 and Fig. 2.4, this proportion is representative of the whole start-up population which is nevertheless biased towards failed start-ups. Therefore we avoid to present any prescriptive advice that derives from the studied companies.

4 RESULTS

4.1 *Overview of the data set*

We have analyzed experience reports of 88 start-ups. The sample of companies is diverse in developed products, geographical location and founders experience. We attempt to estimate to what extent the studied sample is representative to the whole population.

Fig. 2.2 shows an overview of when companies in our sample were founded and when the reports were published. A total of 56 out of 88 companies (63%) have provided information on their founding and closure time. The companies were founded between 2001 and 2013 (Median = 2010). For 77 companies we were able to identify when they have published their experience report. The reports were published between 2006 and 2015 (Median = 2013).

Fig. 2.3 summarizes the operational track length of the companies, which varies between less than a year to 8 years. The majority of companies for which we know the track length, 38 out of 56 companies (68%), have operated between one and three years.

A total of 65 out of 88 companies (73%) have provided location information. Within this group, 48 out of 65 companies (74%), were located in US, 13 companies were located in Europe, and the rest were located in India, Australia and Canada.

In Fig. 2.4 we summarize status of the studied companies. In contrast to what is stated on the website [CBInsights, 2015], some of the companies have re-emerged, continue working, or were acquired by other com-

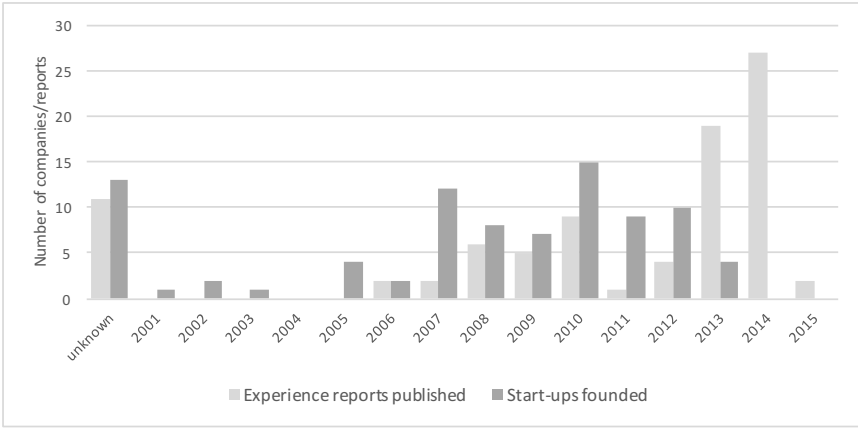


Figure 2.2: Overview of the start-up founding and report publishing time

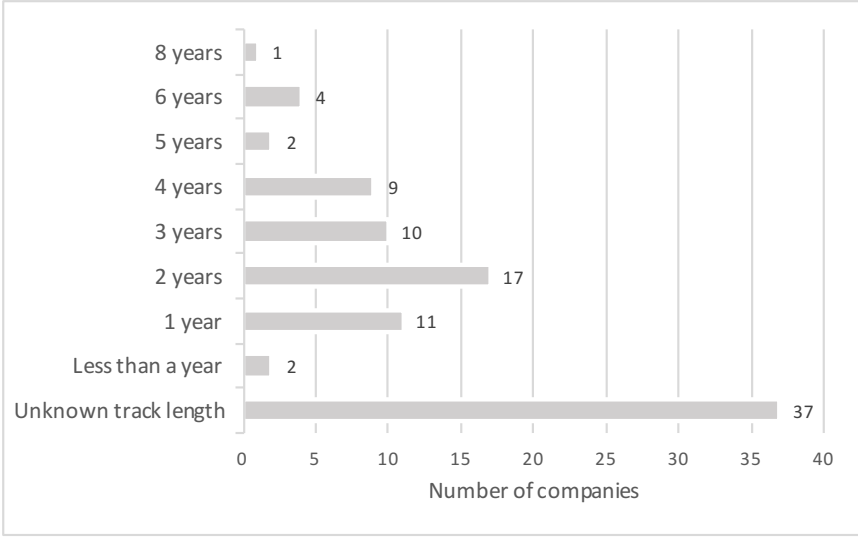


Figure 2.3: Overview of the operational track length

panies, thus can be considered as relative successes. Analyzing the reports and publicly available information, as of June 2016, we have identified different outcomes of the evaluated companies. We distinguish between the following:

- Operational: The company is still in operation. This category also includes companies that have re-emerged with similar products. Also,

companies that have pivoted, e.g. redesigned their product. Examples of such companies are:

  - GroupSpaces, available at http://groupspaces.com/
  - Pumodo, available on iTunes as 'Champion - Football Livescore, League and Cup Action'
  - PatientCommunicator, available at http://patientcommunicator.com/

- Acquired: The company, the team or its intellectual property was acquired by another company. Examples of such companies are:
  - Decide, acquired by eBay in 2013
  - PackRat, acquired by Facebook in 2011
  - ReadMill, acquired by Dropbox in 2014
- Inactive: The company has ceased any active sales or commercial product development activities, however the product is still available to users. This category also includes inactive companies that have made their products available as open-source.
- Closed: The company has ceased any operations, the product is not available to users.

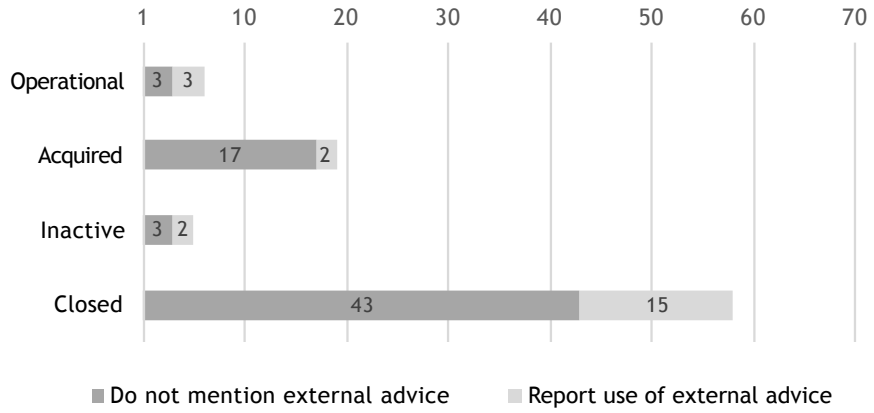


Figure 2.4: Outcomes of the companies and impact of external advice

In Fig. 2.4 we distinguish between companies that have reported participation in incubator programs or otherwise received an external expert advice, e.g. consulted with investors or mentors. There is no clear tendency of external advice being a determinant to company survival, acquisition or close-down.

In the general start-up population, the failure rate is about 75% [Startup Compass Inc., 2015]. Companies that are closed down or inactive we con-

sider as failed start-ups. Companies that are operational or have been acquired we consider successes. As Fig. 2.4 shows, in our sample 6 companies are still operational and 19 have been acquired, resulting in a total of 25 (28%) companies that can be seen as successful, while 58 (66%) of the companies have been closed, thus can be considered as failed. Although we do not know financial details of the operational and acquired companies, the percentage of failed companies in the general population (75%), and closed and inactive companies in our sample (72%) is similar.

4.2 *Knowledge Area Overview*

As a result of provisional coding (see Step 1 in Fig. 2.1), we identified and mapped 876 statements from the experience reports to the SWBP taxonomy. Saldaña [2015] suggests to look into how many reports mention a particular code instead of a total count of codes in the dataset. Therefore, we removed identical codes per report from further frequency analysis. After this filtering our dataset contains 755 codes.

We use the number of codes to illustrate what knowledge areas and their subcategories are common in the reports and what rarely occurred [Saldaña, 2015]. There could be several explanations why a knowledge area is discussed in an experience report. One is that activities associated with a particular knowledge area were conducted and had some interesting effect. This explanation suggests that a knowledge area is relevant for start-ups either because it is useful (and yields positive results), or requires adaptation for use in start-ups (if the knowledge area was applied with good intentions but provided unanticipated results). Another explanation is that a company did not apply a potentially useful knowledge area, however reflected on their mistake in a report. This suggests that a knowledge area could be useful in hindsight.

Fig. 2.5 shows a summary of this analysis. The horizontal axis shows the number of reports mentioning a particular knowledge area; direction (positive or negative) indicates impact. Differently shaded bars show whether the impact is discussed as having impact on business development, software engineering or both. The total length of a bar indicates the total number of codes referring to particular knowledge area. Note, that a knowledge area could be discussed from various aspects in a single report, thus resulting in multiple different codes per same knowledge area. The number of statements discussing a particular knowledge area but not specifying any impact are shown in circles between the bars. In total, 209 (28%) statements address software engineering aspects, and 296 (39%) address business aspects of start-ups.

As Fig. 2.5 shows, software requirements engineering, software design and professional practice are the top three most discussed knowledge ar-

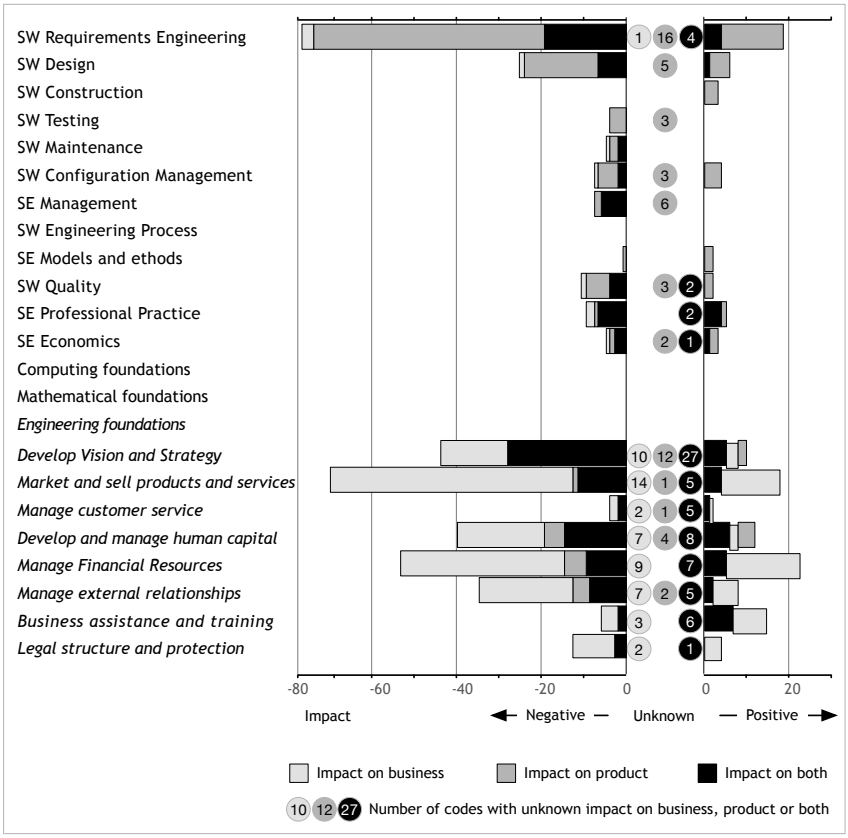


Figure 2.5: Overview of the number of statements associated with knowledge areas

eas in relation to inadequacies in product engineering. Software engineering process, computing foundations, mathematical foundations and the engineering foundations knowledge areas are not discussed at all in the reports.

Figures 2.6-2.8 illustrate what specific subcategories of requirements engineering, software design and engineering professional practice are discussed in the reports. From the requirements engineering knowledge area, requirements validation, analysis and elicitation are the most discussed sub-categories. From the software design knowledge area, the most discussed sub-category is user interface design. From the software professional practice the most discussed sub-category is group dynamics and psychology.

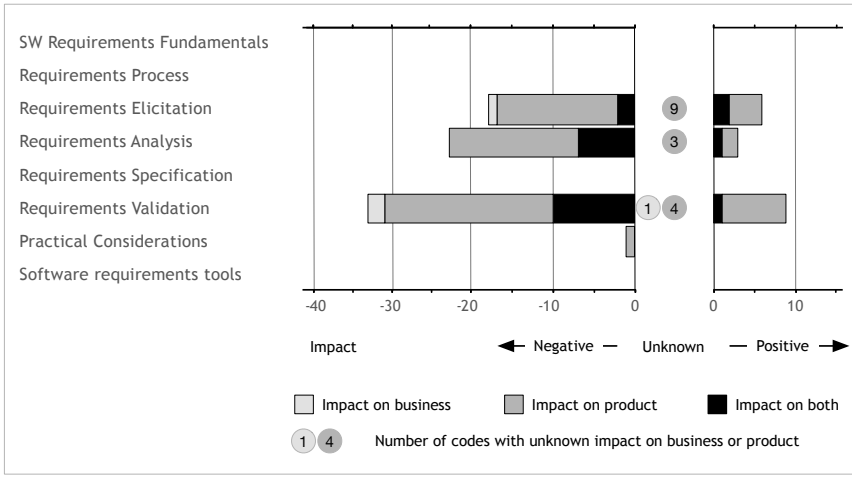


Figure 2.6: Breakdown of the software requirements engineering knowledge area

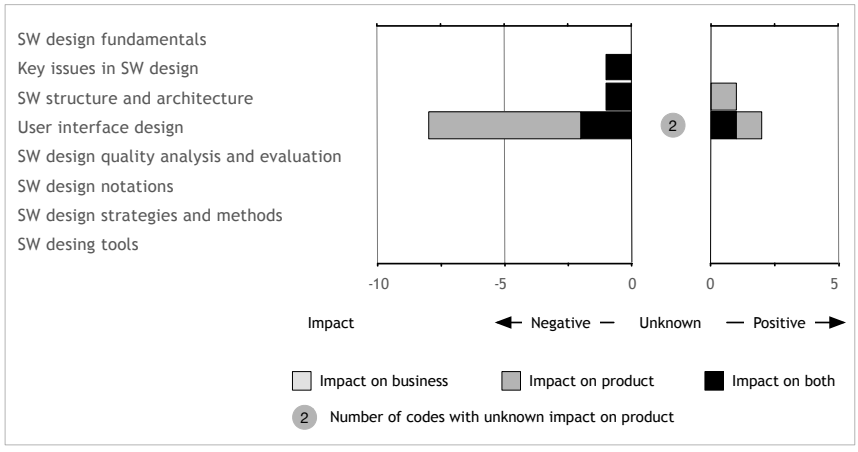


Figure 2.7: Breakdown of the software design knowledge area

5 ANALYSIS AND DISCUSSION

In this section we synthesize our analysis into the software engineering knowledge areas reported by startups to answer the remaining three research questions. First, we report how a knowledge area is applied, what specific practices are mentioned and what specific challenges are discussed in the reports in relation to the knowledge area (RQ2). Second, we explore what relationships between knowledge areas are reported to understand

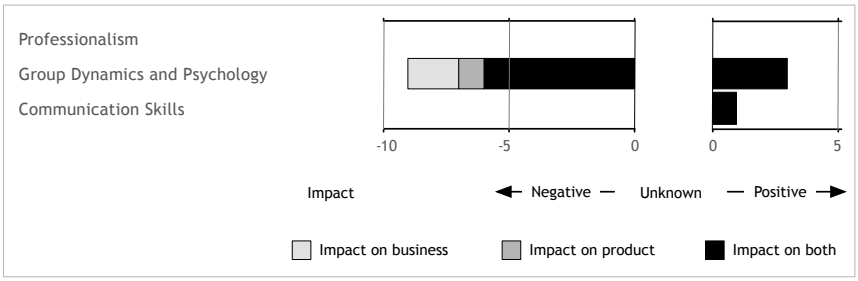


Figure 2.8: Breakdown of the software professional practice knowledge area

how software engineering knowledge areas influence each other (RQ3). Third, we look into related work from similar engineering contexts, compare challenges between start-up and other engineering contexts, and identify potentially useful practices to solve engineering challenges in start-ups (RQ4).

Fig. 2.9 shows key knowledge areas and their relationships of software-intensive product engineering in start-up companies (identified by applying pattern and axial coding as explained in Section 3.3). In Fig. 2.9, boxes represent knowledge areas, arrows denote a relationship between knowledge areas. A relationship indicates that a parent category provides input, i.e. information, to a child category. In further subsections we discuss the knowledge areas and their relationships in detail.

To support traceability between our analysis and data in the reports, we use references to the original data. The references are represented by identifiers in curly braces after a statement, formatted in the following way: C<{*company*#>-<*chunk*#>}. The identifiers refer to the supplementary material available online[4].

5.1 *Develop vision and strategy knowledge area*

Our analysis shows the process of identifying a product value proposition as a bridge between marketing and engineering aspects of a product. Value proposition provides an essential input for starting software requirements engineering activities. The value proposition is a structured description of a product idea. It outlines what is the target audience for the product is, what benefits the product aims to deliver, and what the competitive features of the product are [Carlson and Wilmot, 2006].

The reports discuss identification of the value proposition as an iterative process where the initial formulation is brainstormed, and then improved

4 http://eriksklotins.lv/files/exp-reports-study-supplemental-material.pdf

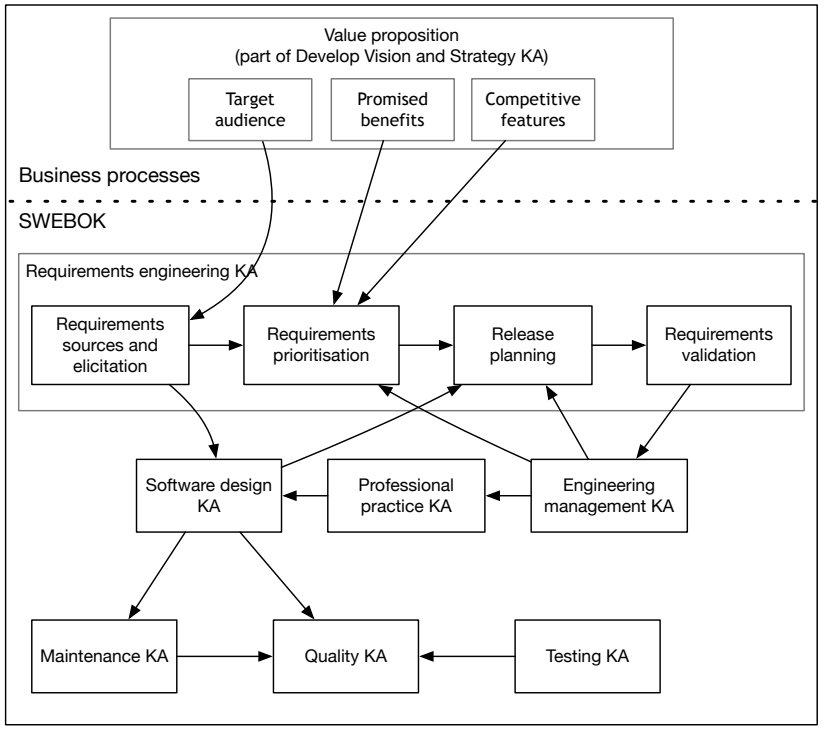


Figure 2.9: Software engineering categories and their relationships

by means of market research, customer interviews, prototype demonstrations and similar activities {C2-50, C11-10, C24-3, C29-63, C37-30, C48-5, C50-9, C52-26, C63-45, C64-33, C67-45}. The value proposition bridges the gap between market research (with a goal to explore market potential of the product) and requirements engineering (with a goal to identify a feasible solution) [Hague et al., 2004, Dahlstedt et al., 2003].

As shown in Fig. 2.9 a structured formulation of the product idea helps to identify specific goal level requirements which then are broken down into more specific functional and quality requirements by requirements engineering activities. Target audiences help to identify stakeholders for requirements elicitation activities. Inadequacies in the value proposition may hinder requirements engineering activities. For example, an unclear overall product goal makes it difficult to specify criteria for requirements prioritization, release scoping and for identifying stakeholders.

Software requirements engineering is a set of activities to capture the needs and constraints placed on a software product, and to identify a feasible solution that contribute to solving a real-life problem. Therefore, requirements engineering can take both problem and solution oriented view [IEEE Computer Society, 2014].

As shown in Fig. 2.5, requirements engineering is the most discussed software engineering knowledge area in the reports. Further analysis of statements from the reports, illustrated in Fig. 2.9, suggests that requirements engineering is the central software engineering activity in start-ups.

Start-up companies operate in a market-driven environment, thus initial requirements are invented by a start-up team [Dahlstedt et al., 2003, Ambler, 2002]. In similar contexts outside start-ups, requirements are validated by internal feasibility reviews, interviews, surveys, crowd-funding success and other techniques that are applicable in the pre-development stage [Fabijan et al., 2012, Ambler, 2002]. Requirements negotiation takes place to prioritize what features to implement next [Tingling and Saeed, 2007].

The experience reports suggest that start-ups use a similar approach to requirements engineering. Software is developed in short iterations aimed to implement and validate a slice of requirements. Results from the validation are used as input for subsequent iterations. As Company #1 reflects on quickly building a prototype, testing it and only then undertaking more extensive mobile application development:

> *"We had a mobile website prototype in front of users within a week and iterated based on that before building out the native [mobile application] version."*

Requirements engineering drives the software development process by helping to acquire domain knowledge, explore problem domain, and to identify potential solutions [Hofmann and Lehner, 2001a, IEEE Computer Society, 2014]. As put by Company #66:

> *"One of the key lessons I learned is that great startups have a blindingly obvious, ideally really large and painful problem that the company is trying to solve. Solving this problem should drive almost every decision in the startup."*

Exploring the problem domain and user needs is one of the key practices in early stage start-ups [Crowne, 2002, Churchill and Lewis, 1983]. Our findings are consistent with Hofmann et al. [Hofmann and Lehner, 2001a] who argue that inadequacies in requirements engineering are the single largest cause of software project failure.

In the following subsections we discuss sub-categories of the requirements engineering knowledge area.

5.2.1 *Requirements sources and elicitation*

This category represents practices to collect requirements and to identify sources from where engineers can collect requirements [IEEE Computer Society, 2014, Dahlstedt et al., 2003].

The reports suggest that start-ups operate in a market-driven context and that the initial requirements are derived from the product value proposition. Interviews, surveys, observations and demonstration of prototypes are reported as methods to adjust goals, discovering new requirements, and to validate existing requirements {C1-45, C14-20, C29-2, C48-5, C50-9, C59-24, C75-4, C79-102, C86-10}

The data from the experience reports suggests also that start-up teams use local businesses, people from their social network and even their teams as requirements sources. Similar products, industry standards and regulations, and partnership agreements are discussed as important requirements sources in addition to customer feedback. Examination of similar products is reported as useful to identify base functionality of a product and to spot opportunities for innovation {C06-12, C1-45, C14-8, C14-20, C33-73, C39-10, C50-8, C61-43, C69-6, C75-4, C79-102, C86-10}.

The reports suggest that the utilization of customer feedback depends on access to requirements sources and interviewer's skill to discover actual customer needs. The access could be limited by, for example, physical distance and inadequacies in identifying potential customers. Mistaking curious people for potential customers can lead to false requirements hindering the product's market potential. Some companies report testing customer interest by asking for an upfront payment {C14-34, C21-7, C02-25}. As Company #14 states:

> *"I think we did not understand that the real purpose of selling was validation (or invalidation) and had the 'always be closing' mindset at a too early stage of the company. Later, I have been joking that during the validation process, if customers don't buy, you should open a champagne bottle and celebrate that you found one way that didn't work and are now a lot closer to success."*

The start-up companies reflect on the importance of early customer feedback and the dangers of not using customer input in the requirements engineering process. Even though gathering of customer input is discussed as difficult due to a physical distance and vague understanding of the target market, customer input is reported as an essential part of requirements engineering. Companies that have neglected early customer feedback report poor product reception in the market and wasted resources on developing

unwanted features, often leading to the company's collapse {C03-10, C22-4, C35-23, C50-8, C52-20, C34-2, C59-24, C75-4, C75-17, C75-19, C76-11, C78-15, C78-19, C86-10, C88-4}.

A commonly reported difficulty is to create an engaged community of early customers of the product. This community facilitates requirements elicitation, validation and other activities were direct customer feedback is essential. The reports suggest that initially a person may show genuine interest in the product, however, if the product does not solve an actual problem for the customer, the interest fades away quickly {C06-6, C35-20, C52-22, C59-7, C65-16, C67-38, C69-6, C82-16, C83-6}.

The reports suggest that misuse of customer feedback stems from difficulties to identify and access requirements sources, i.e. customers, and poor elicitation methods, for example asking the wrong questions {C4-22, C4-72, C59-7, C59-24, C61-43, C79-102, C83-6}. As Company #2 reflects:

> *"People compliment you on the idea because they believe it will be so useful for people other than themselves. i.e., they get into advisor mode."*

DISCUSSION: As shown in Fig. 2.9, using value proposition to identify concrete requirements sources and software requirements is one of the first steps in product engineering activities. Inadequacies in value proposition and requirements engineering activities could hinder any further engineering activities. Unclear quality and functional requirements lead to over or under-engineering of the product.

Identification and access to useful requirements sources is essential for requirements elicitation [Mitroff, 1983]. In a market-driven context, a company must solve the practical problem on how to select a manageable number of users, e.g. early customers, to perform requirements elicitation activities.

Pacheco and Garcia [2012] suggest to classify all likely users and to study all of the user classes to identify their role in the product. Pruitt and Grundin [2003] suggest that the use of superficial characters representing users of the product, i.e. 'personas', helps to identify different user groups and to facilitate discussion around the requirements. The personas could be created with help of a small group of customers or domain experts and further detailed with interviews, surveys and ethnographies to create more detailed descriptions of the users and their needs [Miller and Williams, 2006, Pruitt and Grundin, 2003]. This lightweight practice could be useful for start-ups when actual customers are not readily available.

Fabijan et al. [2012] suggest different customer feedback collection techniques useful at different development stages. Since access to actual users for face-to-face interviews is usually limited, start-ups could use indirect requirement sources such as listing the product idea on a crowd-funding

website [Fabijan et al., 2012, Pruitt and Grundin, 2003], validating the product idea and discover new requirements with less effort.

Due to practical restrictions, only a limited number of potential users can be involved in elicitation and any requirements are generalized over a larger population. However, such approach poses risks of biases, such as sampling (e.g. consulting only expert users as requirements sources), and data collection method (e.g. utilizing only quantitative surveys). Wilson [2006] argues that triangulation and use of multiple methods, measures and approaches must be explicitly interweaves in requirements elicitation process. He argues, that the best results can be achieved by mixing qualitative and quantitative methods, and using multiple complimentary data sources.

Karlsson et al. [2007] report that technology focused companies often neglect user feedback in favor of inventing requirements internally. This is partly due to difficulties obtaining feedback on a new product that is unknown for a market, and partly due to focus on technology rather than actual customer needs [Karlsson et al., 2007]. The reports suggest that start-ups often use interviews to elicit requirements from users, however users are not always able to articulate their needs. Davis et al. [2006b] identify four typical situations in requirements elicitation and argue that each requires specific elicitation techniques. For example, if a user and the analyst share knowledge about a specific requirement, simple questioning to verify the requirement could be sufficient. However, if a requirement is unknown to both sides then mutual exploration of the problem and requirements discovery are a more suitable approach to elicitation [Davis et al., 2006a]. This resonates very well with findings by Kujala [2008] arguing that it is beneficial to empower and involve a group of key customers in daily development activities.

5.2.2 *Requirements Prioritisation*

Requirements prioritisation is a requirements analysis activity to categorize requirements by how essential they are for meeting overall goals of the product. The requirement priorities need to be balanced against resources, time and other constraints [IEEE Computer Society, 2014].

Requirements prioritisation is discussed most commonly in relation to identifying features for the smallest viable feature set, i.e. a minimum viable product (MVP) [Junk, 2000]. The MVP is reported as useful to showcase the main advantages of the product to users and to spot inadequacies in product features or design early {C75-17, C14-11}.

The reports indicate that customers, own ideas, competitors and similar sources provide a constant flow of ideas for new features and improvements. However, due to resource limitations, only a few can be implemented. Start-ups report on selecting features that deliver the most value

to their customers. However, this process is reported as difficult without mentioning any specific practices {12-110, 14-27, 15-25, 33-6, 43-6, 48-9, 50-9, 52-6}.

Requirements prioritization is reported as challenging, specifically the selection of prioritization criteria. To maintain a product focus and to stay within resource, time and quality constraints, the company must prioritize what features are the most relevant to deliver a promised value proposition {C50-11, C50-11, C57-12, C71-24}.

Some companies reflect that their challenges with requirements prioritization originate from a vague value proposition, i.e. unclear product goals and benefits. The reports suggest that consequences of poor requirements prioritization are over-scoped product releases and wasted resources on implementing unwanted features {C76-5, 50-18, 69-14, 79-48}.

DISCUSSION: As illustrated in Fig. 2.9, requirements prioritization goals are defined by the product value proposition.

Quantifying value is a complex task and often involves making a compromise between interests of different stakeholders. When maximizing value is used as a prioritization goal, different perspectives of value need to be considered. Khurum et al. [2012] propose a breakdown of software value aspects therefore enabling discussion about different perspectives on value.

Lehtola et al. [2005] identifies a need for alignment between business and engineering activities in a market-driven setting. The authors discuss use of roadmapping as a technique to align product and market perspectives. A road-map helps to connect immediate engineering goals with higher level objectives and to facilitate the discussion between different stakeholder perspectives, i.e. customers, business and engineering.

5.2.3 *Release Planning*

Release planning is closely related to requirements prioritisation and concerns the identification of sets of requirements that can be delivered to customers and provide competitiveness in the market [Carlshamre and Regnell, 2000]. In a market-driven setting there is a constant pressure to deliver features faster [Giardino et al., 2014a]. However practical challenges, such as requirements interdependencies, need to be resolved.

When planning product releases, start-ups follow two general approaches: frequently releasing small increments and delivery of a fully-fledged product. The frequent delivery approach starts by creating a very simple functionality, even mock-ups, and continues until the product matures. A continuous delivery process allows to conduct continuous requirement validation and to immediately adjust the product direction {C33-33, C48-4, C53-46, C54-9, C54-11, C87-60, C46-13}. Fully-fledged releases take more

time to build, thus continuous validation of the product direction is challenging. Moreover, as validation takes place after the release, substantial effort is put on risk to be wasted {C14-11, C14-42, C52-20, C78-19, C78-20, C87-60, C46-13}. Attempts to launch a fully-fledged version are most commonly discussed in relation to neglect of user input and focus on technology rather than an actual customer need {C14-11, C14-42, C52-20, C78-19, C78-20, C87-60, C46-13}.

Due to market pressure or internal uncertainty of what customers expect from the product, companies desire to satisfy customers with a more complete and polished product. However, implementation of more features or higher quality requires more resources and postpones the opportunity to demonstrate the product to users, thus hindering requirements validation activities. Companies that have leveraged on early user feedback and have launched a less complete product, report fewer difficulties in marketing the product {C35-23, C50-11, C52-20, C57-12, C71-24, C75-17, C75-18, C76-5, C78-19, C82-12, C86-11}.

We found that companies often overscope their releases aiming to deliver a more "ground-breaking" product in hopes for more positive user feedback {C14-11, C52-20, C46-13, C78-19, C87-60}. As Company #59 states:

> *"We should have concentrated on the core idea and launched a Minimum Viable Product (MVP) to test the concept, as we initially had planned even though we never had heard of the concept of an MVP. We kept building more features, since we always felt that 'the service needs X because Flickr has it too' or 'he/she said he needs that feature'."*

Overscoping could be a consequence of poor requirements prioritization.

DISCUSSION: As illustrated in Fig. 2.9, release planning is closely related to requirements prioritisation and requirements validation. Prioritization provides means for identifying requirements to be included in a product release. Requirements in the release are demonstrated to customers and, thereafter, validated by customer feedback.

Bjarnason et al. [2010a] recognize that scoping of product releases is challenging. They report that an unclear vision of overall goals, constant inflow of requirements, and miscommunication are some of the reasons for over-scoping the releases [Bjarnason et al., 2010a]. As shown by the experience reports and supported by Bjarnason et al. [2010a], consequences of over-scoped releases are unmet customer expectations, wasted effort and delays.

Dahlstedt et al. [2003] and Alves et al. [2006] report that in market-driven requirements engineering most requirements validation takes place after the product is released to the users. Therefore, frequent releases enables

early identification of potential flaws in the value proposition or the requirements.

Incremental delivery of the product and frequent adjustment of plans are described as key practices of Scrum [Rising and Janoff, 2000]. Rising and Janoff [2000] reports that organizing development in sprints and prioritizing features for upcoming release helps to deal with uncertainty and changing requirements. Moreover, the Scrum method implies that after each iteration an assessment of progress, user feedback and re-prioritization of tasks takes place. Such rigorous approach to development and planning helps to break down the product to manageable chunks and progress is made even if requirements change. Predictable timing and scope of product releases encourages users to adopt the product [Rising and Janoff, 2000].

5.2.4 *Requirements Validation*

Requirements validation covers practices to ensure that engineers have understood the requirements and the proposed solution actually solves the original problem [IEEE Computer Society, 2014].

The reports suggest that start-up companies aim to focus their activities around continuous requirements validation. The most commonly discussed technique is to implement requirements in a early version of a product, i.e. a prototype, demonstrate it to the users and to collect feedback, commonly called a feedback loop {C01-47, C14-8, C29-3, C35-20, C52-22, C54-9, C55-16, C61-7, C71-23, C75-18, C86-14, C87-56, C34-4, C46-13}.

User feedback is used both to validate the requirements and to identify new user requirements for the product. In addition, interviews with users are reported as useful to review and discuss the requirements before prototyping {C02-11, C06-6, C14-8, C29-33, C63-13, C34-2}.

The companies report on using various metrics to gather quantitative data how customers use the product. The collected metrics are used to validate requirements and to steer further product development {C1-66, C48-5, C50-18, C53-72, C57-12, C64-17, C75-10}.

However, many companies have failed to establish the feedback loop either due to the lack of an internal engineering process to manage user feedback or the difficult access to users {C49-21, C50-18, C34-2, C86-10}. As Company #14, building a software tool for ordering photo prints on-line, reflects:

> *"Iterations took longer than planned for us, because small print labs were often quite busy and did not have time to immediately have a look at the new version and give feedback. [..] When they finally had time to try out the new version, if they felt that it still needed improvement or they came up with a new feature that would be needed, the launch was likely to be postponed by at least a month."*

DISCUSSION: As shown in Fig. 2.9, requirements validation in start-ups is closely related to release planning and provides an input to planning activities. Release planning determines what features are released and, therefore, undergo validation. Outcomes from the validation are used to adjust further product direction.

The most recurring issues in requirements validation are the lack of a structured process to utilize user feedback and the inability to select relevant metrics. However, the experience reports offer little details on specific practices addressing these issues. Our findings are consistent with Olsson and Bosch [2015] who identify similar issues in established companies developing software-intensive products.

Hanssen and Fægri [2006] report on involving expert users in a deploy-test-evaluate loop. The expert users are central in testing and evaluating each release. However, the authors also emphasize the required overhead to maintain the user-developer relationship, to keep the users engaged and to make strategic decisions on the product direction.

Further research is required to understand how to identify users to be involved in development process and to what extent methods by Hanssen and Fægri [2006] could be applied in start-ups.

5.3 *Software design knowledge area*

Software design is a set of activities and a result of defining software architecture, components, interfaces and other characteristics of the system, supporting its construction. Software design can take place before the construction process as in plan-driven contexts, or interweave with the construction process as in an agile setting [IEEE Computer Society, 2014, Yang et al., 2016].

As shown in Fig. 2.7, most (22 out of 34) statements associated with the Software Design knowledge area lack details for mapping to subcategories. The remainder of the statements specifically discuss the User Interface Design subcategory.

The reports offer very little information on the actual construction of the product, coding or integration of components. Instead, the reports discuss design decisions behind selecting one or another construction technology, components or design goals.

Statements from the reports suggest that start-ups aim to release their products or services fast, thus spending little time on upfront software design. Start-ups opt for incremental designs and faster product releases {47, 3101, 3956}. Scalability and flexibility of the product are identified as primary goals of software design {264, 1245, 1836, 1922, 2982, 3249, 3956, 1305, 733,1166}.

Start-ups report on attempts to leverage on cutting-edge technologies with the aim to gain a competitive advantage such as faster time-to-market or additional features. However, new technologies are often reported as immature causing product quality issues. As Company #3 states

> *"Sure, it was seven years ago, pre-iPhone and pre-Android, so it was ahead of its time, we had to use Adobe Flash on a browser which sucked in so many ways I can't even start to explain how bad it was. Technology would be so much better and more important all mobile today."*

Selection of technology also concerns third-party solutions that can be integrated and configured to constitute the product. Third-party components are used as a method to deliver functionality with little development effort. Leveraging on existing functionality of third-party components is reported as a key practice in software design. Some companies that have not leveraged on third-party components admit lack of skill and experience in software design {C04-23, C69-12}.

Several reports mention good product user experience as an important quality and their efforts to improve it. However, no specific practices regarding user experience engineering are mentioned {6-11, 14-15, 35-17, 66-35, 63-54}. User interface design is recognized as having an impact on customer behavior and attitude towards the product. As Company #66 reflects on user interface and user experience design:

> *"The team never properly sat down and brainstormed the UX. Quick decisions were made to get the MVP out the door and these had serious impacts on how the product was received by customers."*

Constantinides et al. [2010] and May [2012] also recognize the importance of user interface and its impact on product adaptation.

Other goals of user interface design are the development of a product's visual appeal, to establish a brand identity, to gain attention from media {C02-87, C33-36, C67-13}, search engine optimization {C17-16}, and promoting viral effects in social networks {C66-35}. An iterative approach of frequently updating the user interface and measuring changes in the user behavior is reported as a viable practice to build user interface of a product {C01-47, C02-80, c12-126, C14-33, C33-33, C52-19, C82-13, C46-13, C54-29, C63-06, C02-81}.

The reports suggest that start-up companies use brainstorming {C66-42}, mock-ups and wire-frames {C14-15} to design user interfaces of a product. Frequent iterations {C33-36}, experiments {C66-35} and usability tests {C14-15} are applied to continuously improve the user interface {C02-87, C35-36, C61-17, C66-35}.

However, under a tight schedule, the process could be abandoned and user interface designs are done in a hurry with little consideration {C14-15}

causing quality issues later. Attempts to salvage a product that is unsuccessful for other reasons by tweaking the user interface leads to wasted resources with little or no gain {C02-87, C14-15, C63-54}.

DISCUSSION: The reports suggest that when the understanding of requirements is vague, it is useful to put together a quick prototype demonstrating a feature in question. The prototype is used to gather user feedback before any extensive development takes place {C33-14, C57-26, C71-18, C78-25, C79-50, C79-115}. However, sticking more features into a makeshift product degrades the architecture and technical debt accumulates over time. Maintaining a manageable level of technical debt and creating an architecture supporting changing requirements and enabling quick prototyping is a major challenge {C04-58, C14-12, C22-04, C33-42, C50-11, C58-15, C64-31}.

Software design activities in start-ups are closely related to requirements engineering. Non-functional requirements determine the required level of quality as an input for software design activities. However, vague or invented non-functional requirements could lead to under or over-engineering of a product {C04-22,C04-71, C23-3, C75-23, C55-14}.

Quality requirements constraining internal aspects of the product, such as time-to-market or maintenance costs, are repoted as often overlooked. Poor or neglected quality requirements may create pitfalls in the long run: inadequately high maintenance costs when product is launched or overly long release cycles {C04-22,C04-71, C23-3, C75-23, C55-14, C32-3, C78-25, C35-22, C79-115}.

Creating software design that requires minimal lead time and can accommodate changing requirements while maintaining high product quality is a challenge. The reports suggest that it takes skilled engineers to build such designs and reflections on how inadequate engineering skills had contributed to poor design leading to poor product quality {C79-50, C75-23, C53-68, C72-18, C73-40, C01-02, C02-80, C35-52, C43-17, C49-26, C52-19, C67-33}.

A study by Woods [2015] suggests that goals of software architects often clash with goals of agile teams, however there are simple principles that allow both to benefit from each other. For example, breaking the software into smaller components and delivering incrementally helps to avoid large upfront designs. Communicating architecture principles to the developers help the team members to understand why architectural structures exist and what are most important characteristics of the architecture. Yang et al. [2016] identifies forty three software architecture approaches that can be used in an agile context. The identified approaches range from naive (considering architecture only for current iteration) to use or architectural design patterns and cost-benefit analysis.

State-of-the-art in agile software architecture offers a variety of practices and guidelines that could be relevant for start-ups. However, it needs to be explored which exact practices are most efficient to address start-up specific challenges.

5.4 *Software engineering professional practice knowledge area*

The software engineering professional practice knowledge area comprises skills, knowledge and attitudes that an engineer must posses to practice software engineering [IEEE Computer Society, 2014]. The reports discuss various aspects of professional practice, such as decision making, motivation, trust, importance of good software engineering skills, and ability to learn.

Difficulties in communication are discussed as having a significant impact on decision making, motivation, trust and general climate in the team. As Company #79 describes the communication between co-founders:

> *"Overall, the most important [challenge] is that Nathan and I had difficulty communicating in a way which would allow us save the company, and that this really drained out motivation."*

The reports contain descriptions of team structures ranging from hierarchical to flat. In a start-up team the highest authority are founders. However, some founders empower and involve other team members in making important decisions while others exercise autocracy in all aspects of their companies {C01-46, C12-54, C62-17, C74-40, C78-17}.

Autocracy is discussed in the reports as a cause and consequence of lack of trust between team members, miscommunication of company goals and lack of transparency in decision making. However, involving the whole team in every decision hinders performance and team motivation. Separating areas of responsibility and empowering team members to make decisions are discussed in the reports as viable practices for decision making {C05-06, C12-83, C16-08, C17-19, C46-05, C66-42, C69-04, C71-13, C74-40, C77-22, C78-17, C84-12}. Company #17 points out that in a dynamic start-up environment it is difficult to make decisions based on previous experience. Instead decisions should be based on data and experimentation:

> *"No one has any idea what is going to work and what's not. Don't listen to the people who think they know. Sure, this one didn't pan out, but each failure helps us navigate the thousands of decisions we will need to make for the next one. That knowledge helps us build better things that will last longer."*

Mutual trust between team members is reported as an important factor for good teamwork and decision making. Founding teams with joint

previous experience reflect on team issues more positively and reflect on mutual trust as a contributing factor to good teamwork. The inability to communicate mutual expectations, intentions and motivate own decisions hinders trust {C49-26, C62-19, C72-18, C73-32, C73-34}.

Capabilities to learn new emerging practices, adapt to an uncertain environment and collaborate are reported as essential in a start-up environment. Some reports discuss how unanticipated personality traits have contributed to team break-up and company collapse, suggesting that good team composition is essential in start-ups {C02-47, C14-2, C21-17, C29-4, C73-32, C46-5}.

A working environment encouraging communication and collaboration, such as a dedicated office space and joint activities, boosts performance and increases motivation {C56-16, C61-65, C71-14, C73-9, C73-34, C77-22}. Working remotely is reported to have negative effects in the long term {C21-14, C75-15, C77-3}, however, when done with consideration, working remotely can have positive effects, i.e. to avoid disturbances in the office {C75-12} or being closer to the target market {C01-45, C21-14}.

The reports discuss how initial optimism for fast success vanishes and development tasks shift from inventing the product to less exciting activities such as handling customer service {C02-50, C02-74, C04-20, C04-22, C77-3}. When a critical motivator is not present anymore, a team member may leave the company or start following his own agenda {C01-28, C12-142, C21-16}. Motivation to work in a start-up is often discussed in the reports in relation to shared goals and vision. A lack of shared understanding about changing goals is reported as a consequence of poor value propositions {C01-58, C12-142, C22-07, C53-86, C71-14, C74-40, C76-06. C77-16, C79-18}.

Due to time constraints, start-up companies choose people that fit the team by "character rather than skill" {C63-6, C73-34, C76-6}, implying that one's commitment and teamwork skills are more important than technical skills.

The reports discuss emotional issues of working in a start-up. Taking responsibility of many tasks at once creates anxiety, leading to burnout and loss of motivation. The reports suggest that the founders loss of motivation to continue operating a start-up, leads also to the end of the company. Anxiety and burnout are also reported as outcomes and poor work and personal life balance {C01-18, C27-35, C73-8}.

Software engineering is an inherently human and team based intellectual activity. Team factors have emerged as critical in many different development environments and have effect to nearly any other activity [Fagerholm and Münch, 2012, Khan and Spang, 2011, Chow and Cao, 2008, Sudhakar, 2012, Giardino et al., 2016]. However, as shown in Fig. 2.9, the reports suggest that software design is the most affected software engineering knowledge area. Difficulties in communication, inadequacies in skills

and decision making are exposed through sub-optimal software design and poor product quality. This finding is consistent with Giardino et al. [2016] reporting that team's disregard of structures and engineering processes lead to deterioration of product architecture.

Fagerholm et al. [2015] reports a study on different factors affecting developer performance in lean and agile environments. In this study performance is used to benchmark efficiency and effectiveness of a team. They explore different factors facilitating performance, creating performance awareness, disrupting performance and others. This study shows that there are few key factors contributing to good developer experience. For instance, control of own work, decision power, and good environmental atmosphere contributes positively to overall team performance. Several factors, such as open office, collaboration and competition, and subordinance can both enhance and worsen a team performance.

De Melo et al. [2013] explores agile team productivity and lists several team related processes contributing to productivity, staff turnover and commitment. For example, good conflict management, sharing of expertise, and team coordination are essential to high developer commitment, low staff turnover and high productivity.

As exemplified with these two studies, state-of-the-art identifies the key ingredients for high performing teams in agile and lean environments. The same factors could be applicable in start-up teams. However, start-ups face certain specific limitations in team formation. Firstly, start-ups are founder centric and team environment highly depends on dynamics between the founders [Criaco et al., 2014]. Secondly, lack of resources limit access to highly skilled individuals, especially in the early stages. This could lead to sub-optimal initial team composition, and more effort is required to develop the team as a whole to reach the desired performance level [Fagerholm et al., 2015, Giardino et al., 2016].

5.5 *Software Quality knowledge area*

Software quality is a multi-faceted concept in software engineering. Nearly all other knowledge areas aim to somehow contribute to software quality [IEEE Computer Society, 2014]. Kitchenham and Lawrence [1996] identifies five perspectives on quality: transcendental, user, manufacturing, product and value-based view.

Fig. 2.5 illustrates that little details are provided in the reports regarding software quality. Superficial statements indicate that start-up companies see software quality as product's characteristics to meet users needs, thus focusing on the user perspective of quality [Kitchenham and Lawrence, 1996]. The reports discuss usability, especially performance, user experi-

ence and reliability as their focus areas {C50-15, C50-22, C61-17, C67-23, C67-26, C67-32, C78-15, C86-13, C86-14, C87-22, C37-4, C46-10}.

The reports reflect on issues stemming from product quality: poor product reception due to an insufficient level of quality, or emphasis on the wrong quality aspects (for example, scalability over time-to-market) {C14-15, C63-54, C66-35, C54-35, C69-12 }.

Poor product adoption or loss or reputation are reported as consequences of poor product quality. However, very little details are provided on quality requirements and on any quality assurance procedures.

DISCUSSION: While start-ups discuss very little how to achieve software quality, the consequences of inadequate quality, such as ruining the product's image or overly expensive product maintenance, are discussed. We observed that software quality is more discussed in closed companies. This tendency suggests that the importance of software quality may be realized only in hindsight. However, we could not confirm any statistically significant connection between company outcome and statements pertaining software quality.

As shown in Fig. 2.9, quality originates from product design. The analysis of the reported software design practices in Section 5.3 reveals that companies often put excessive resources on improving certain quality attributes with little market need. The analysis of the reported requirements engineering practices in Section 5.2 shows that there is little discussion on quality requirements. Moreover, no practices to assure quality requirements were identified (see Section 5.7).

These findings indicate a gap between requirements engineering and software design. We argue that vague and wrongly prioritized quality requirements contribute to inadequate product design affecting the product's potential to deliver the promised value.

Regnell et al. [2008] argue that in a market-driven context, product quality aspects have different thresholds. If a quality indicator is below a certain threshold, a product is useless. Within certain thresholds the product is useful but does not differentiate itself from competition. Above a certain threshold the product becomes competitive and at some point the quality becomes excessive and costly [Regnell et al., 2008]. The concept of quality thresholds enables a company to identify important quality indicators and to perform requirements elicitation to determine the threshold values. Understanding of the threshold values and multiple perspectives of value enables the company to set and specify quality goals. Azar et al. [2007] propose a lightweight method to balance multiple influences to quality requirements and to determine optimal product quality goals. As elaborated in Section 5.3, companies often fail to determine the required level of quality and waste resources on excessive quality features. More research

is required to understand to what extent value-oriented requirements engineering practices could be applied in start-ups.

5.6 *Software Engineering Management knowledge area*

The engineering management knowledge area concerns organizational aspects of software engineering, such as initiating, planning, monitoring, controlling and reporting of software engineering activities [IEEE Computer Society, 2014]. While the dynamic environment in start-ups makes any detailed plans outdated quickly, the engineering process still must be controlled, follow resource and time constraints, and produce a result that is aligned with overall goals of the company.

With respect to the software engineering management knowledge area, the reports discuss effort estimation, monitoring and product discontinuation practices.

The reports suggest that the companies aim to achieve certain business goals, either to qualify for further external funding or to establish sufficient cash-flow to support development efforts. Pursuing these goals require investments in product development. Thus, estimation of the required resources is an important step to assess feasibility of the goals. The reports discuss how overly optimistic estimates contribute to the collapse of a company due to the lack of resources to finish the product {C29-35, C66-35, C77-5, C79-11, C79-120} or missed market opportunities {C29-35, C79-11}. However, the reports do not discuss any specific effort estimation method.

The start-ups report that any initial plans are based on assumptions {C01-77, C02-74, C06-8,C52-30,1967-1978,C24-12} and are adjusted as data from requirements validation comes in { C39-9, C06-6, C82-7, C50-8, C57-18, C34-5, C65-21, C65-14, C35-22, C14-47, C66-20, C37-12, C61-10, C61-21, C61-17, C29-40, C33-27, C04-92, C53-110, C54-33, C02-75, C02-91}. Feedback from customers helps to determine the next immediate step, e.g. to improve certain features or collect feedback from a different stakeholder group. Even though the companies frequently refer to adjusting their plans based on success of a product release, we found very little details about this process {C01-43, C12-142, C22-04, C02-32, C29-07, C33-44, C52-08, C52-20, C53-50, C46-13, C60-40, C82-16, C86-10, C51-12}.

The reports suggest that start-ups attempt to estimate their progress by looking at various metrics that should be carefully selected and tied to business goals. Measuring the wrong thing or measuring too many things may lead to data overload and difficulties to interpret many conflicting measurements {C01-66, C02-93, C05-5, C14-13, C29-25, C33-46, C35-26, C48-5, C50-18, C57-12, C64-17, C74-36}. As Company #50 reflects on selecting KPI:

> *"As a business leader you need to figure out the metric that matters most for your company and understand that the more you measure, the less prioritized you'll be. Don't fall into the trap of trying to measure everything."*

The reports discuss different approaches to product discontinuation. Companies with products that had attracted a significant number of users report a timely notification to the users about the product discontinuation, and instructions on how to back-up their data and migrate to different solutions {C18-4,C47-6}. Some companies leave the product accessible but cease any further development or maintenance efforts {C05-7, C10-15,C32-4,C36-5}. Few companies report open-sourcing the product {C80-13, C65-41}.

The reports suggest that decisions stemming from engineering management influence release planning, e.g. by determining the release strategy, and requirements prioritization by setting prioritization goals. Engineering management also influences the professional practice knowledge area by setting expectations and constraints on the engineering team.

DISCUSSION: Shahin and Mahbod [2007] proposes criteria for defining organizational goals and a structured method to select key performance indicators for assessing progress towards said goals. A combination of SMART (Specific, Measurable, Attainable, Realistic and Timely) criteria to define goals and analytical hierarchy process (AHP) to select metrics to assess progress towards the goals [Shahin and Mahbod, 2007] is a feasible alternative to the "gut feeling" approach described by Olsson and Bosch [2015] and Terho et al. [2015].

Terho et al. [2015] argue that fundamental changes to the start-ups' business plans, i.e. pivoting, are largely based on a gut feeling and are caused by an urgent need, e.g. need for more revenue. However, the collection on operational data (key performance indicators) helps to make more motivated decisions with specific goals [Terho et al., 2015]. Therefore, use of a structured method to select important metrics, for example Shahin and Mahbod [2007], could improve decision making in start-ups.

Garengo et al. [2005] report that the lack of resources, little attention to formalization and a reactive approach are factors that hinder implementation of performance indicators in small and medium enterprises. Cocca and Alberti [2009] argue that key performance indicators are essential to make informed decisions and propose best practices in implementation of performance indicators. The study lists qualities of a good performance indicator and exemplifies maturity grids as a tool to in decision making. Shahin and Mahbod [2007] propose a lightweight technique based on the analytical hierarchy process (AHP) to select and prioritize key performance indicators. These practices for selecting and implementing key

performance indicators could be considered for adaptation in start-ups [Shahin and Mahbod, 2007, Cocca and Alberti, 2009].

Giardino et al. [2014a] emphasize the uncertainty in the start-up environment and argue that development teams in start-ups are formed by low-experienced engineers. The lack of joint and individual experience makes the application of expert judgement based effort estimation methods difficult [Molokken and Jorgensen, 2003a]. Usman et al. [2015] report that the most widespread effort estimation technique in agile teams is planning poker, and the most popular size metric is story points. Moreover, the most common planning levels are current iteration and release. Whether the same methods are equally widespread in start-ups as well requires further research. However, group estimates, i.e. planning poker, is reported as being more accurate than individual estimates and could be very well applied in start-ups [Haugen, 2006].

Existing literature presents very little discussion on software product discontinuation. Jansen et al. [2011] presents a structured plan on how to discontinue a software product. The proposed plan includes adequate pre-planning, transferring customers and partners to another solution and finally reallocation of the product team. Given the lack of resources it is unlikely that start-ups put more than absolute minimal effort on product discontinuation. However, the list of steps to discontinue a product proposed by Jansen et al. [2011] could serve as a roadmap for product discontinuation in start-ups.

5.7 *Software Testing knowledge area*

Software testing is the dynamic verification that a product works as expected on a set of selected test cases. Some of the main tasks of testing are to determine what to test, specify input data and expected software behaviors, and to organize the process of software testing [IEEE Computer Society, 2014].

The reports contain only general statements addressing the software testing knowledge area. The statements suggest that start-up companies perform testing activities only when obvious issues emerge. For example, when performance had degraded below an acceptable level {C24-16, C67-26, C86-13}. Feedback from users is used to spot discrepancies in the product instead of performing rigorous internal testing {C14-23, C29-36, C67-26}.

Some companies report product failures in operation with substantial loss of resources and reputation. As company #54 states:

> *"Finally, the server went down, scuttling the entire operation. Hill started handing out margaritas by the fistful to keep everyone happy.*

[..] The app picked up a number of 1-star reviews following the debacle"

. The reports did not specify whether the failures were due to lack of specific requirements or failure to meet such requirements.

As illustrated in Fig. 2.9 software testing has a direct impact on software quality. The product must have an acceptable level of quality on all relevant aspects, or the product is simply useless to customers [Regnell et al., 2008].

DISCUSSION: Giardino et al. [2016] argue that product quality has a low priority in software start-ups. Instead of rigorous internal testing, start-up companies utilize user feedback to determine if a level of quality is acceptable. A possible explanation is that due to frequently changing or unclear requirements there is no other reliable input for testing [Graham, 2002]. However, as elaborated in Section 5.5, inadequacies in product quality can severely damage a product's reputation. Therefore, a company must carefully assess the risks stemming from the reliance on user side testing.

Another possible explanation is that a large part of testing is done by developers during the development process. This explains why the product appears to be in shape when released (code defects are removed), however failures in operation indicate a lack of design and stress testing [Runeson et al., 2006]. Our findings suggest that start-ups are overlooking a potentially important knowledge area.

5.8 *Software Maintenance knowledge area*

A result of software development is a delivery of a software product to is users. However, post delivery defects may emerge, operating environment change or users could propose new requirements. The software maintenance phase begins when software is released to customers and ensures that the software continue to operate as intended. Software maintenance activities fall into perfective maintenance (to improve some quality the software, e.g. performance), corrective maintenance (to remove defects), adaptive maintenance (to adapt the software to a changed environment) and preventive maintenance (to prevent problems before they occur) [IEEE Computer Society, 2014].

Start-up companies report on software maintenance costs {C23-3, C32-3} and resource allocation for maintenance activities. The reports discuss the struggle of performing timely corrective maintenance due to understaffed teams. Long response time to product faults is reported as having a negative impact on product adoption {C77-3}. Adaptive maintenance to keep up with the product and any third-party component changes is reported as a concern {C14-12, C38-9}.

DISCUSSION: The reports contain very little details on how start-ups manage software maintenance. However, inadequately high costs of keeping the product running is reported in relation to poor software design. As discussed in Section 5.3, goals of software design shift from faster time-to-market to reducing maintenance effort. This shift takes place when the product feature set stabilizes and more and more users start-using the product [Crowne, 2002].

If this shift is not executed properly, a large number of users can overwhelm the product, exposing any inadequacies in product design and quality. Tackling these inadequacies may require substantial resources and time, contributing to the collapse of the company.

Batista Webster et al. [2005] propose a taxonomy for evaluating risks pertaining to software maintenance. The taxonomy could be used in start-ups to identify and address potential maintenance risks during product development. Tom et al. [2013] argue that taking risks in engineering, i.e. creating technical debt, is a trade-off between shorter time-to-market and internal product quality. However, how exactly technical debt is handled in start-ups and to what extent this taxonomy is exhaustive and relevant in start-up context requires more research [Batista Webster et al., 2005, Tom et al., 2013].

6 CONCLUSIONS AND FUTURE WORK

This study is the largest (by a number of studied cases) and broadest (by addressed software engineering knowledge areas) investigation into engineering aspects of start-ups to-date. With this study we paint a rich picture on how start-ups reflect on utilizing software engineering, what engineering practices start-ups use, and why. This study is aimed to characterize software engineering in start-ups, thus providing the necessary groundwork for conducting further and more detailed investigation into software-intensive product engineering in start-up context. To achieve our goal we perform third level analysis of start-up experience reports from 25 relatively successful and 63 closed start-ups.

Our results show that start-ups apply market-driven requirements engineering practices to discover and validate ideas for innovative products. However, the applied requirements engineering practices are often rudimentary and lack alignment with other knowledge areas. As a consequence, inadequacies in requirements engineering hinder other engineering activities and might lead to unwanted technical debt, poor product quality, and wasted resources on building irrelevant features. Further work is needed to identify good requirements engineering practices in start-ups.

We have found very little discussion regarding software testing. However, the reports discuss disastrous events when a product had failed in

hands of customers. We conclude that software testing practices could be overlooked by start-ups. Further research is needed to understand state-of-practice in software testing in start-up context.

Other software engineering knowledge areas have a supportive role in continuous requirements identification and validation. For example, software design knowledge must support fast evolution of product prototypes, used to gather customer requirements, to a robust solution for easy maintenance.

The results of this study are intended to be useful to researchers in supporting further research in the area. The results can also be useful to start-up engineers willing to learn from experience of others. We have analyzed our findings in context of related work, thus hinting practitioners towards potentially useful practices. Future work includes examining key knowledge areas in more detail, and exploring to what extent the use of certain practices contributes to achieving start-up goals.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Krzysztof Wnuk for insightful discussions, comments and hints to related work.

3

SOFTWARE ENGINEERING ANTI-PATTERNS IN START-UPS

Software start-up failures are often explained with a poor business model, market issues, insufficient funding, or simply a bad product idea. However, inadequacies in software engineering are relatively unexplored and could be a significant contributing factor to the high start-up failure rate.

In this paper we present the analysis of 88 start-up experience reports, revealing three anti-patterns associated with start-up progression phases. The anti-patterns address challenges of releasing the first version of the product, attracting customers, and expanding the product into new markets.

The anti-patterns show that challenges and failure scenarios that appear to be business or market related are, at least partially, rooted in engineering inadequacies.

1 INTRODUCTION

Software start-ups are important suppliers of innovation and software-intensive products and services. There has been a steady growth in capital invested with 2017 setting a record of 16 billion EUR invested in European start-ups [sta, 2017]. However, only a few percent of start-ups manage to deliver any value. Thus a significant capital is wasted on building unsuccessful products [Blank, 2013b]. On the surface, the low success rates can be explained by market challenges, the difficulty in attracting customer interest, and resource shortages among others. However, digging deeper, we found indications that the capability to build software efficiently with minimal resources and limited knowledge about emerging target markets is the foremost challenge in software start-ups – and this is to a large extent closely related to engineering practices [Giardino et al., 2015b]. If even only a small part of start-up failure can be attributed to failures in the actual engineering practices and principles applied it translates into a significant loss of investments. Thus this paper focuses on the engineering applied in start-ups.

Very little is known about software engineering in start-ups and to what extent the lack of customer interest and resource overruns are related to deficiencies in engineering [Giardino et al., 2014a]. Earlier studies suggest that scoping and building a minimum viable product is a substantial challenge and precedes any market or business related challenges [Giardino et al., 2014a, Crowne, 2002]. Thus, failure in engineering could hinder any subsequent attempts to market the product and to build a sustainable business around it.

To explore how software engineering is applied in start-ups and how inadequacies herein could be linked to start-up failures we examine 88 start-up experience reports. We apply qualitative analysis methods to identify recurring failure scenarios and their root causes. We present our results in the form of three anti-patterns: (1) not getting the first product release out, (2) not attracting customers to the product, and (3) challenges of scaling the product for new markets.

The differentiation between symptoms and root causes, as presented in this paper, of potentially dangerous scenarios enable practitioners to assess their situation better and apply corrective practices to the root causes. Moreover, the analysis pinpoints potentially interesting research directions to further understand software engineering practices and principles used in start-ups.

2 RESEARCH METHODOLOGY

To maintain transparency of our results we present four steps of our research methodology.

2.1 *Threats to validity*

We identify two main sources of validity threats, the first stemming from the used data source, and the second, from our interpretation of the reports.

The analysed reports were not created for this study, thus may lack important details about the software engineering process, including precise details about the engineering context, such as team size, roles, and skills. Moreover, the reports are essentially subjective self-evaluations of practitioners. The practitioners could rationalize their shortcomings with external circumstances. However, these threats are alleviated due to a relatively large sample, and diverse population.

Another possible threat stems from single researcher bias in the coding process. To address this threat, we applied researcher triangulation. At multiple points in the coding process, selected reports where independently analysed by all three authors, the results reviewed and discussed.

Table 3.1: Overview of research methodology

#	Step	Description
1	Data collection	We used a repository of start-up experience reports [CBInsights, 2015]. Reports in the repository are written by start-up practitioners and describe lessons-learned and reflections after critical events such as product launch, buyout, or closure of the company. Although these reports were not compiled for this study, they are a primary data source providing an original insight of how do start-ups operate. The reports are not limited to software engineering; they also cover business, marketing, teamwork and personal issues relevant to start-ups.
2	Screening	The original dataset consisted of 93 reports. We screened contents of the reports and removed 5 reports from companies not developing software, and established companies. Most of the reports are between 1000 and 2000 words long.
3	Coding	We analysed the reports by following qualitative data analysis methods and used descriptive (to summarize), process (to capture ongoing action) and evaluation (to assess the situation) coding jointly to capture analysis points in the experience reports. Through analysis of the described situation, we aimed to differentiate between reported symptoms (e.g., running out of resources) and actual causes (e.g., poor resource planning due to lack of experience). The resulting codes briefly summarize key situations in a company along with contributing factors to the situation.
4	Development of anti-patterns	We grouped similar codes and chained them to create cause-effect diagrams. We use the practitioners' reflections and related work to suggest countermeasures to each anti-pattern.

2.2 *Studied sample*

The studied reports reflect on events between 2001 – 2015 and describe experiences from a diverse set of companies in developed products, geographical location, founders' backgrounds, and different development scenarios. In Figure 1 we illustrate our sample of studied start-ups.

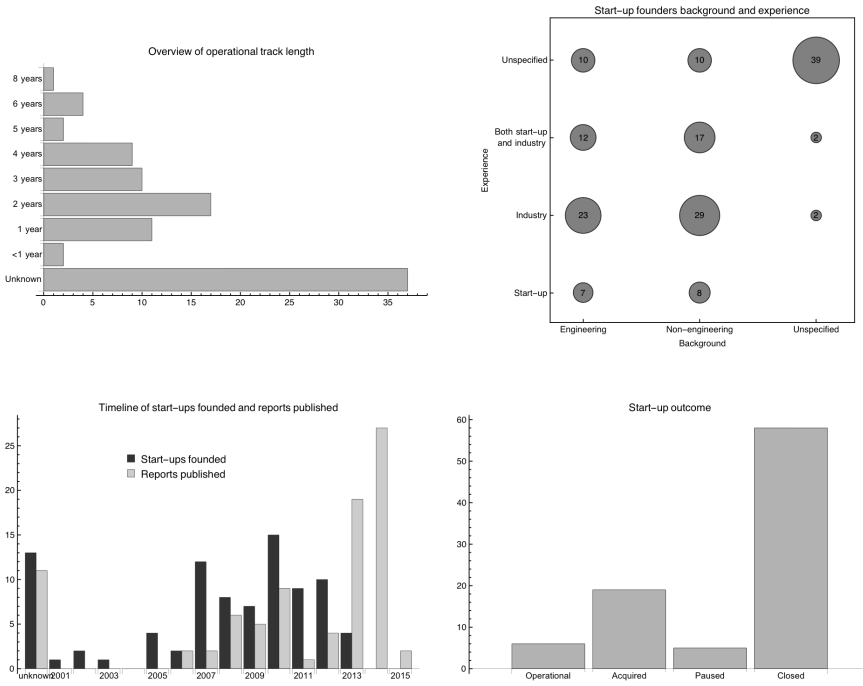


Figure 3.1: Demographics of the sample

To compile the demographical information, we rely on information in the reports and publicly available information about the companies.

3 START-UP ENGINEERING ANTI-PATTERNS

Our analysis shows that start-ups experience different challenges depending on how far into product development they are. From the reports, we identified three phases in start-up progression characterized by specific goals, and challenges. Phase I is concerned with building and releasing the first release of a product. Phase II is concerned with attracting first customers to the product beyond early adopters and beta testers. Phase III is concerned with scaling the product further into new markets.

As illustrated in Figure 2, the reports suggest three distinct progression phases, each with a specific aim and symptoms for the anti-patterns. The plotted line represents the expected growth of a company. The forks denote alternative scenarios, that is, anti-patterns that could hinder the progression of the start-up. Along with the causes at the bottom of the figure, we denote how many reports were the basis for identifying each cause.

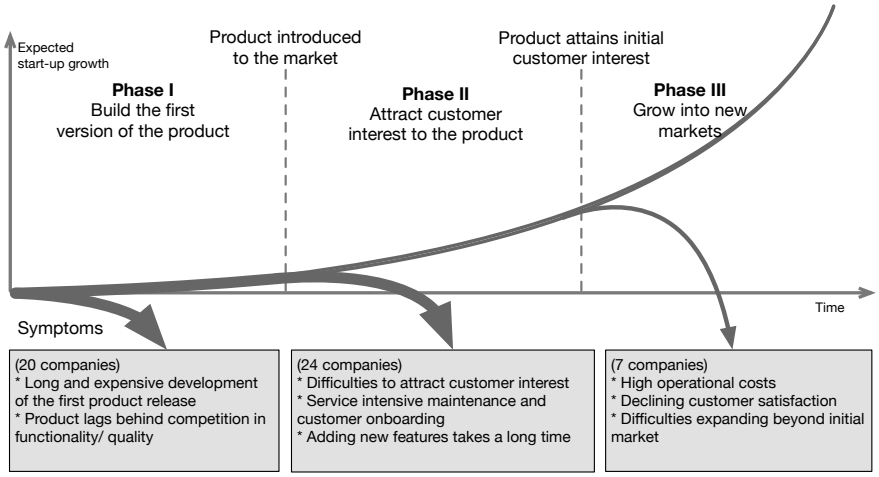


Figure 3.2: Start-up milestones and symptoms for anti-patterns

3.1 *Anti-pattern I: (not) releasing a market worthy product*

The reports suggest that one of the initial engineering challenges is to build and release the first version of a product: a minimum viable product, a bare-bones version of the product, good enough to be used for its main purpose and to test if there is enough customer interest to justify further investments in product development.

With the advantage of hindsight, the start-ups emphasize the importance of small and fast to develop first releases of their products. However, some start-ups reflect that it took them an overly long time to build the first version, stretched resources, and drained motivation, which cost them a market opportunity. Symptoms and outcomes of anti-pattern I are illustrated in Figure 3.

Reflections in the reports suggest two main causes for this anti-pattern. Cause 1, design decisions (see Figure 3), is concerned with selecting technologies and components that constitute the product. The choice lies between developing features in-house, utilizing open-source components, or buying certain functionality from a third party supplier [Petersen et al., 2017a]. In their reports, start-ups reflect that building commodity features

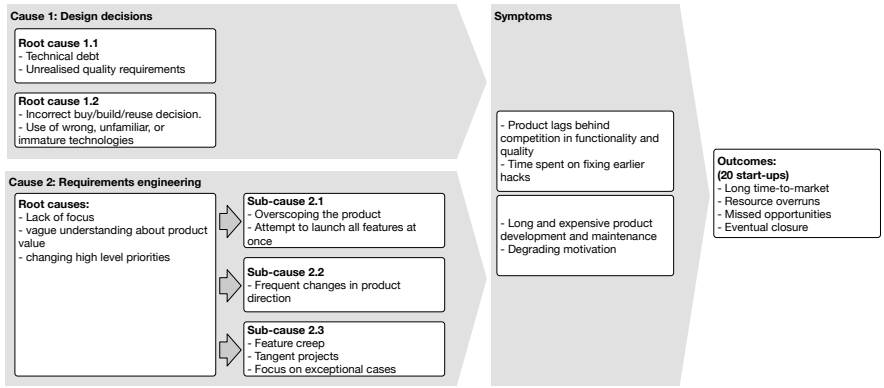


Figure 3.3: Breakdown of anti-pattern I

that can be obtained by other means was a waste and significantly prolonged product development. Furthermore, the quality of in-house developed features was often significantly lower than of alternatives regarding functionality, performance, and maintainability.

A potential root cause for inadequate product quality is unrealized quality requirements. Quality requirements drive architecture decisions, including the selection of components and technologies which determine the level of external quality. Incorrectly assessing the required level of quality could lead to rework delaying the product release to market.

Cause 2 in Figure 3 is concerned with selecting features for the first product release. We identify three sub-causes all stemming from the same root cause – inadequacies in requirements engineering.

Overscoping is a consequence of poor requirements engineering and a potential cause for long and expensive development of the product, see sub-cause 2.1. Attempting to launch more than minimum set of features, attempting to implement all possible product use-cases beyond what is necessary for a product to be usable, generates potential waste and inflates the product scope beyond feasible. As in the case of Saaspire:

> *"The custom platform created a massive overhead on our development work. We ended up over-engineering our systems so they could support both today's and tomorrow's products."*

The second sub-cause is general of product direction. With a vague understanding about customer needs, the solution is based on invented requirements and unclear priorities. As there is no real association between product features and needs of specific customers, requirements change whenever a new idea about an interesting feature comes to mind. Thus, the product direction changes frequently prolonging the software develop-

ment process indefinitely. Furthermore, systematically abandoning already built features creates waste, damages teams' morale and drains motivation.

The third sub-cause is related to focusing efforts on key features. Some start-ups report that they always felt that the product lacks a feature thus kept adding more features. Furthermore, some companies report starting side projects along the primary product thus increasing the scope of the work substantially. As described by Disruptive Media:

> *"We kept building more features since we always felt that the service needs X because Flickr has it too or he/she said he needs that feature."*

3.1.1 *Analysis of the causes and remedies*

Releasing the first version of their product is a test whether a start-up team can coordinate their work, and has enough skills to produce a meaningful output. Looking at how many start-ups encountered this anti-pattern (20, see Figure 2), getting the first product out could be a substantial challenge for many.

The practitioners argue that product time-to-market can be substantially reduced by combining existing open source or third-party components. Such components typically offer a set of related functionalities out-of-the-box, thus counterbalancing the extreme focus on the key features only [Munir et al., 2016]. However, a decision what to build, buy or what components to use depends on desired product qualities. Therefore, a clear understanding of what quality aspects are important and what is the expected quality level is a prerequisite [Regnell et al., 2008].

A strategy to minimize overscoping and to shorten time-to-market is to invest in understanding the customer needs and to strip any non-critical features from the product [Melegati et al., 2016b]. In hindsight, the start-ups suggest scoping the first release to solve one simple problem for a customer. Earlier research suggests focussing on how a customer will be using the product rather than generating lists of feature ideas [Cloyd, 2001].

Aiming for fewer features will save development time and resources. Minimizing effort of developing the first release will also minimize the effort of rework in case some of the features turn out to be unsuccessful. That said, minimizing effort by scoping the product should not be confused with reducing effort by lowering the engineering quality. The goal should be to build fewer features at good quality.

3.2 *Anti-pattern II: (not) attracting customers*

The reports from start-up that have launched their products suggest that the next challenge is to bring the product to market. While coordinated

marketing and sales activities are important to attract customers, the companies reflect that characteristics of the product, stemming from earlier engineering decisions, hindered their marketing activities.

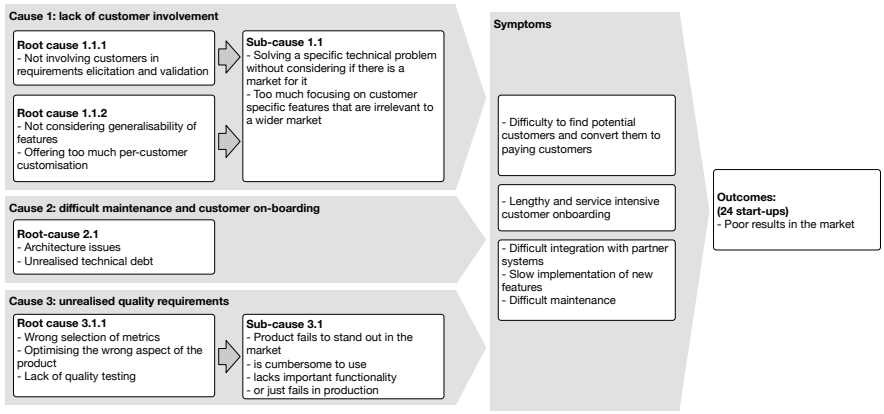


Figure 3.4: Breakdown of anti-pattern II

The analysis of the reports shows 3 potential causes, see Figure 4, that could hinder efforts to market the product. The most common symptom is the difficulty to find potential customers and convert them into paying customers. Reflections in the reports suggest that lack of customer involvement at the right level in developing the product (Cause I) largely contributes to marketing issues later. We identified two related scenarios. The first is that customers are not involved in requirements elicitation and validation, and the product the product turns out to be irrelevant to market, as described by one practitioner:

> *"We rarely had meaningful conversations with our target end-users. We huddled together to decide on ideas that sounded nice, built prototypes, put on our salesman hats, and didn't understand why we weren't closing deals."*

The second is that some customers are involved, and the product is tailored to their specific needs and lacks generalizability to a wider customer base, see root-cause 1.1.2.

The companies reflect that after launching a minimum viable product, they rush to add more utility features to make the product more usable for customers. For example, by adding integrations with other systems, data import and export functionalities, support for other software and hardware platforms. The ability to add new features quickly demonstrates to potential customers that the product is evolving fast and any functional or quality shortcomings can be removed swiftly. However, the speed of

adding new features could be reduced by product architecture issues and unrealized technical debt.

Another potential cause for poor results in the market is that the product does not stand out among the competition, is cumbersome to use, lacks important functionality, or is unreliable. As in a case of Flowtab,

> *"A big-bang product launch involving many partners and customers become a catastrophe when the app failed."*

The root cause for these scenarios lies in unrealized quality requirements, lack of quality testing, or using the wrong metrics to optimize the wrong aspect of the product. Multiple start-ups reflect that their attempts to improve results in the market by tweaking user interfaces turned out ineffective because their product functionality lacked relevance. As described by Dinnr:

> *"The hypothesis of making the product visually more appealing and people will come back more often because you address their emotions didn't work out. You can't design your way out of a fundamental flaw."*

Start-ups that have nailed features and quality of their minimum viable product report that slow customer onboarding is a significant hindrance to attracting customers at a desired rate. As experienced by Treehouse Logic:

> *"If a client confirms they want to work with us, they must hire an agency to handle the integration work. The reality is that the client expects full service and involving another party was too much for most of them."*

3.2.1 *Analysis of causes and remedies*

Speed is one of the most praised start-up advantages over larger companies. However, a start-up can build itself into a corner by accumulating technical debt. Technical debt slows down development of new features, introduces hard to address quality issues, and degrades teams' morale [Tom et al., 2013]. Unwanted technical debt seeps in due to poor engineering decisions, sloppy individual attitudes, and insufficient coordination of engineering work. Unwanted technical debt can be prevented and removed by planned refactoring of the product and considering the impact on technical debt in all product related decisions.

An earlier study of customer complaints about mobile apps shows that reports functional errors, sluggishness, and unexpected app crashing are the most common among customer complaints [Khalid et al., 2014]. Such results suggest, that engineers should pay extra attention of removing any such issues from the software before releasing it to the public.

Optimally, there is a pool of interested customers even before the product is launched. As suggested by several reports, such customers are a valuable source of requirements and assure that there is a market need for the product. The reports also suggest that this is not always the case, and the product launch is often the first time when a start-up attempts to reach out to customers. Therefore, to minimize the risk of market failure, start-ups should reach out to potential customers and involve them in developing the product.

At this phase, the response from markets could trigger a pivot in products' direction regarding features, used technologies, and targeted market segments. As shown by earlier research, flexibility to abandon ideas that do not work out and be prepared to explore new opportunities is crucial [Bajwa et al., 2017b]. Thus, a lightweight product engineering process, such as Scrum, can support quick testing of feature ideas with little upfront work [Rising and Janoff, 2000].

3.3 *Anti-pattern III: A good problem to have*

Start-ups that had attracted a substantial number of customers reflect on challenges concerning growth beyond their initial markets. At this stage, start-ups report declining customer satisfaction, difficulties expanding into new markets, and high operational costs hindering sustainability of the company. The outcomes of this anti-pattern are difficulties establishing a sustainable business model and growth issues.

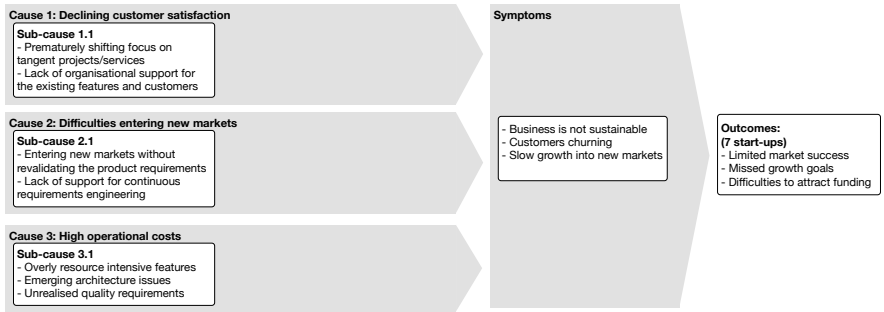


Figure 3.5: Breakdown of anti-pattern III

The reports discuss declining customer satisfaction, cause I, in association of start-ups undertaking tangent projects, and focusing efforts to enter new markets. Such activities could take attention and resources off existing customers, thus hindering their satisfaction. The root cause for such a scenario is the lack of an organizational framework supporting growing number of customers, continue to develop the product and to expand.

According to the reports, another common challenge is to expand into new markets. Part of the difficulty is to conduct sales and marketing activities over a geographical distance. However, another part of the challenge, cause II, is to identify what are the differences in customer requirements between markets and to tweak the product for the new market. As experienced first-hand by Dinnr:

> *"I thought that because businesses [like ours] have sprung up everywhere around Scandinavia, it will be a breeze to start something similar in London. [..] The lesson is: No, it doesn't have to work in country #2 only because it works in country #1."*

Cause III, hindering start-ups at this stage is cope with high product operational and maintenance costs. The increasing costs are associated with extra workforce and computing power required to serve an increasing number of users. When costs grow larger than the actual value provided by the product, the business model of the company cannot be sustainable. As in case of Serendip, a start-up that used machine learning and big-data analysis to create customized playlists:

> *"The high costs of processing millions of posts every day, and serving relevant and engaging playlists to our users are really bigger than we can handle."*

3.3.1 *Analysis of causes and remedies*

Experiencing challenges associated with growth into new markets is a good place to be for a start-up. However, as described in several reports, these challenges can bring the company down if not addressed timely.

High operational costs could be associated with inadequate requirements analysis and product design earlier. Specifically, not considering how much it would cost to run a feature and how much of customer value the feature creates. For instance, Everipix provided a freemium service to organize photos. However, costs of hosting pictures on Amazon Web Services turned to be higher than customers were ready to pay for the service. Thus, the company could not turn to profit and eventually went bankrupt. A potential remedy could have been to focus on features organizing the photos and to minimize the need of hosting the pictures.

When expanding into new market segments start-ups should make sure that the product is relevant in these new segments and to make necessary adjustments. This can be done by repeating requirements validation and aiming to discover new, the market segment specific requirements. For example, user interfaces and user documentation may need translations.

The analysis of the start-up experience reports shows an association between shortcomings in software engineering practices and principles and start-up failures. The association highlights the importance of using good software engineering practices in start-ups. However, improving engineering practices does not necessarily mean spending more time and resources on engineering work. Timely and efficiently addressing root causes of potential problems down the road could save resources.

In essence, the focus should be on better engineering, not more of it. Thus saving time, resources and can greatly improving chances of successful product launch.

We present the anti-patterns along three distinct phases of start-up progression. Thus, putting the anti-patterns into temporal perspective. In combination, this creates a map that can support start-ups in improving their goals and prevent common pitfalls relevant to their specific progression stage.

For researchers, the anti-patterns along with the start-up progression phases present a framework that can be further developed by identifying new milestones, improved anti-patterns, and best engineering practices.

4

A PROGRESSION MODEL OF SOFTWARE ENGINEERING GOALS, CHALLENGES, AND PRACTICES IN START-UPS

Software start-ups are emerging as suppliers of innovation and software-intensive products. However, traditional software engineering practices are not evaluated in the context, nor adopted to goals and challenges of start-ups. As a result, there is insufficient support for software engineering in the start-up context.

We aim to collect data related to engineering goals, challenges, and practices in start-up companies to ascertain trends and patterns characterizing engineering work in start-ups. Such data allows researchers to understand better how goals and challenges are related to practices. This understanding can then inform future studies aimed at designing solutions addressing those goals and challenges. Besides, these trends and patterns can be useful for practitioners to make more informed decisions in their engineering practice.

We use a case survey method to gather first-hand, in-depth experiences from a large sample of software start-ups. We use open coding and cross-case analysis to describe and identify patterns, and corroborate the findings with statistical analysis.

We analyze 84 start-up cases and identify 16 goals, 9 challenges, and 16 engineering practices that are common among start-ups. We have mapped these goals, challenges, and practices to start-up life-cycle stages (inception, stabilization, growth, and maturity). Thus, creating the progression model guiding software engineering efforts in start-ups.

We conclude that start-ups to a large extent face the same challenges and use the same practices as established companies. However, the primary software engineering challenge in start-ups is to evolve multiple process areas at once, with a little margin for serious errors.

Software start-ups are small companies created to build and market a software-intensive product [Sutton et al., 2000]. Start-ups are characterized by rapid evolution, small teams, uncertainty about customer needs and market conditions, and a high failure rate [Paternoster et al., 2014, Berg et al., 2018]. However, leveraging cutting-edge technologies, risk, and speed, start-ups can launch software products fast [Giardino et al., 2014a, Baskerville et al., 2003].

Aims, objectives, and challenges of product engineering change quickly as a start-up evolves [Crowne, 2002]. State-of-the-art engineering methods offer little support for understanding the evolving context and selecting the right practices [Giardino et al., 2016, Paternoster et al., 2014]. A miscalculation in choosing engineering practices could lead to over or under-engineering of the product, and contribute to wasted resources and missed market opportunities [Giardino et al., 2014a].

According to industry reports, a record of 19.2 billion EUR of venture capital was invested in European and 80 billion USD in US start-ups in 2017 alone [Wijngaarde et al., 2018]. Building the first version of a product is a substantial engineering challenge and precedes any market or business related difficulties [Giardino et al., 2015a, Crowne, 2002]. Thus, shortcomings in applied engineering practices could waste the investment, and hinder any subsequent attempts to market the product and to build a sustainable business around it. Even if a fraction of start-up failures could be attributed to engineering failures, that would still present an opportunity for better, start-up specific, engineering practices, and a relevant area of research.

An increasing number of studies attempt to explore engineering practices in a start-up context, for example, requirements engineering [Melegati et al., 2016b, Gralha et al., 2018, Tripathi et al., 2018], technical debt [Klotins et al., 2018a], and user experience [Hokkanen and Väänänen-Vainio-Mattila, 2015a]. Several studies, such as Unterkalmsteiner et al. [2014] and Crowne [2002], attempt to explore and present conceptual models of product engineering in start-ups. However, none of these studies provide a full and detailed answer to what engineering practices start-ups use concerning different engineering process areas and start-up evolution stages. The need to better understand product engineering in start-ups, and to provide relevant support for practitioners, has been highlighted by Unterkalmsteiner et al. [2016] and Klotins et al. [2018b].

With this study, we aim to understand how start-ups use different engineering process areas, and how utilized practices evolve over start-up life-cycle. Through our analysis, we present a progression model of what engineering aspects, that is, goals, practices, and challenges are relevant in start-ups in their evolution stage. The model is aimed at supporting

practitioners in their decision making process and at pinpointing specific engineering challenges for further investigation.

We use an adapted case survey method [Petersen et al., 2017a] to collect and analyze primary data on engineering practices in 84 start-up cases. The cases vary by geographical location, development stage, outcome, and time of operation among other factors, thus presenting a relatively large and diverse sample [Berg et al., 2018]. To explore start-up goals, challenges, and used practices, we propose the start-up life-cycle model and analyze cases within the same development stage, and with the same outcome. We apply qualitative methods to identify patterns in the data and arrive at explanatory results. The explanatory results are verified and complemented with statistical analysis providing a firm basis for our conclusions.

Our study provides several novel contributions. Firstly, we present a start-up life-cycle model, aimed at illustrating the dynamically evolving nature of start-ups. Secondly, we use the life-cycle model to describe what practices, goals, and challenges are relevant to start-ups at different life-cycle stages. Thirdly, we present the start-up progression model aimed at guiding practitioners and at illustrating relevant areas for further exploration.

The remainder of the paper is structured as follows: in Section 2 we define software start-ups and summarize existing work in the area, in Section 3 we describe our research methodology, in Section 4 we report and analyze our results, in Section 5 we interpret and discuss our findings, Section 6 concludes the paper.

2 Background and Related Work

2.1 *Software start-ups*

As early as 1994 Carmel [1994a] reported on small companies building and marketing innovative software products. These small companies, or start-ups, prioritize time-to-market over product quality, teams are small and self-motivated, and engineering practices informal. Later studies, for example, Giardino et al. [2014a], and Sutton et al. [2000] report the very similar characterization of start-ups.

Start-up companies are known for their high failure rate. About 75 - 99% of start-up products fail to achieve any meaningful results in market [sta, 2017, Blank, 2011]. The high failure rate could be explained by market challenges, team issues, difficulties in securing funding and so on. However, the capability to build software efficiently with limited understanding about stakeholder needs and with limited resources is the foremost challenge in software start-ups and precedes any market or business related challenges [Klotins et al., 2017].

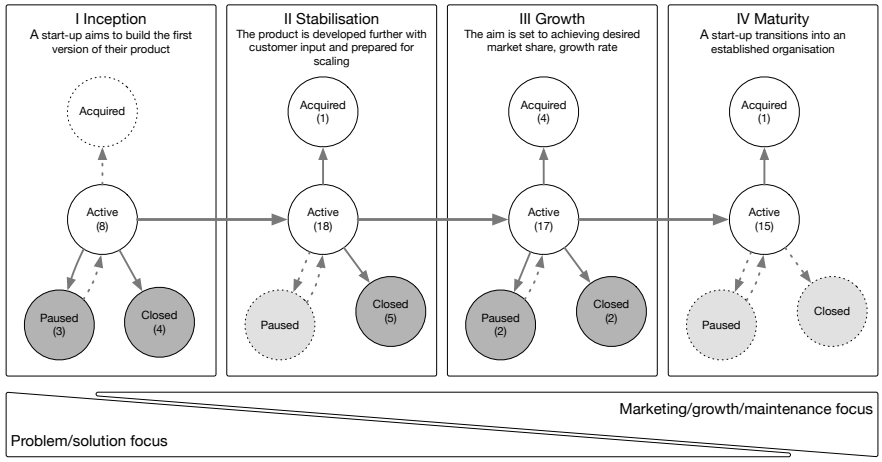


Figure 4.1: The start-up life-cycle model based on Crowne [2002] and Churchill and Lewis [1983]. The white bubbles indicate "good" and desired states. The shaded bubbles show the undesirable states. Arrows denote possible transitions between the states. States and transitions that are possible, were however not observed in this study, are denoted by dashed lines. The numbers in brackets indicate how many cases representing each state were observed.

2.2 *What do we know about software start-ups?*

A broader interest to study software start-ups from software engineering perspective was launched by publication of a systematic review on existing literature in the area [Paternoster et al., 2014]. Selected results from this review were also published in IEEE Software [Giardino et al., 2014a]. In the review, authors point out the potential of software start-ups as vehicles for innovation, and lack of relevant research in the area. The review lists a number of contextual challenges to software product engineering in start-ups compared to established companies.

Two subsequent literature reviews were published by Klotins et al. [2015] and Berg et al. [2018] aiming to map state-of-the-art in start-ups with SWEBOK [IEEE Computer Society, 2014] knowledge areas. They conclude that there are many gaps and opportunities for developing start-up specific engineering practices. However, they attempt to analyze state-of-the-art in start-ups with SWEBOK that isn't start-up specific and lacks several relevant knowledge areas, for example, market-driven requirements engineering and value driven software engineering.

Giardino et al. [2016] proposes the Greenfield start-up model identifying the main engineering categories and their relationships. The model identifies severe lack of resources, teamwork, rapid development, little focus

to quality, evolutionary approach, and technical debt as the key categories. The relationships reveal that development speed is achieved by capable teams, little focus on quality assurance and internal product quality. However, such approach leads to accumulation of technical debt and hindered performance in the long term.

2.3 *Software engineering practices in start-ups*

Earlier studies point out that start-ups often do not follow best engineering practices and opt for an ad-hoc approach to engineering. Such an approach to engineering is partly due to the immaturity of start-ups, rapidly changing environment, and lack of engineering expertise. However, there is also limited support from academia pinpointing engineering practices that could be suited for such context [Klotins et al., 2018b, Yau and Murphy, 2013, Unterkalmsteiner et al., 2016].

Part of the difficulty to practice and study software engineering in start-ups is the lack of knowledge transfer between start-ups. All knowledge about a domain, ways-of-working, and practices is carried by individuals, thus is lost when a start-up closes down, and needs to be reinvented with every new start-up. Thus, a large part of start-up research is to establish a body of knowledge with the best engineering practices for start-ups [Barton, 1995, Klotins et al., 2018b].

Several studies attempt to explore product engineering practices in start-ups. For example, Gralha et al. [2018] and Melegati et al. [2016b] explore requirements engineering practices in start-ups. These studies suggest that requirements engineering is one of the key engineering process areas in start-ups, and help to explore the market opportunity and to devise a feasible solution.

Hokkanen et al. [2016] present a framework guiding user experience design work in a start-up context. They argue that for a product to be successful in a market it has to fulfill minimal functional and user experience requirements. The framework identifies and prioritizes various user experience elements guiding user experience focus.

In our earlier work (Klotins et al. [2018a]) we explore technical debt in start-ups. We found that excessive technical debt could be a cause for missed market opportunities and contribute to start-up failures. Furthermore, they identify several strategies that could help to expose unwanted technical debt early.

The earlier work suggests that engineering practices are rapidly evolving to accommodate the changing engineering context [Hokkanen et al., 2016, Gralha et al., 2018, Giardino et al., 2016, Klotins et al., 2017]. Even though, individual areas, for example, requirements engineering and user experience design, has been studied, there is a lack of a coherent view of

how different engineering areas fit and evolve together. Such complete understanding is needed to analyze why specific practices are applied, and to spot potential misalignment between engineering process areas.

2.4 *Start-up life-cycle models*

To illustrate the changing objectives of a start-up, there have been attempts to define start-up life-cycle models. Blank [2013b] identifies two stages, search for a viable market opportunity, and building a viable business around the opportunity. Each stage consists of several objectives outlining the need to explore and validate a potential customer need first, then propose and validate a potential solution, and finally scale up the operations. This model, however, is generic and offers little guidance for software engineers.

Inspired by Churchill and Lewis [1983] and Crowne [2002] we combine start-up product evolution stages with relevant organization states (Fig. 4.1). In this paper, we use the four stages of a start-up as a basis for explaining start-up evolution and define them as:

1. Inception - a stage between ideation of a product until the start-up releases the first product release to the first customer. The primary goal of this stage is to scope and build the minimum viable product by balancing needs of a customer, available resources, and time [Junk, 2000].
2. Stabilization - a stage between first product release until readiness for scaling. In this stage, a start-up aims to ensure that the product can be decommissioned without adding extra effort to the development team. That is, the product should be easy to maintain, scale, and surrounding infrastructure, for example, operations and customer support, are in place.
3. Growth - at this stage the focus is set on attaining the desired market share and growth rate. Although the efforts shift towards marketing and sales, the engineering team should cope with a flow of new customer requirements, and product variations for different markets.
4. Maturity - in this stage a start-up transitions into an established organization aiming to preserve established market share and to optimize its operations. The engineering team should install routines for operating and maintaining the product.

These four stages represent the shift in start-up objectives. Early stages focus on finding a relevant problem, devising a feasible solution. Later, the focus shifts to marketing and improving the efficiency of start-up's operations [Blank, 2013a, Crowne, 2002].

In case a start-up decides to radically change some fundamental aspect of its product, that is, to pivot [Bajwa et al., 2016], the product likely moves to an earlier phase in the life-cycle model. For example, discarding existing features and developing new ones implies abandoning any marketing or stabilization efforts related to the abandoned features. Developing entirely new features throws the start-up back to inception stage for scoping, validation and piloting.

At any of the four product stages a start-up strives to: a) remain active and continue operations (that is, not to fail), b) advance to the next stage or c) be acquired by another company for profit to shareholders. Alternatively, at any of these stages, a start-up may be closed down or paused. We define organization states as the following:

1. Active - the team actively works on the product.
2. Paused - the team has stopped working, however there is an intention to resume at some point in future. Reasons for pausing a start-up could be, for example, a temporary shift in founders priorities, temporary lack of funding for product development or marketing among other scenarios.
3. Acquired - another company acquires the start-up for a profit to shareholders. The team is disbanded or merges with the other company.
4. Closed - the team is disbanded or works on something else.

The combined model of start-up and product life-cycle states is shown in Fig. 4.1. We use this model as an input to our study and to capture the state of each studied case.

3 Research Methodology

We use a case survey method to collect and analyze primary data from start-up companies. The method is aimed to balance studying a few cases in depth with traditional (multi) case studies and quantitatively studying many cases [Petersen et al., 2017b]. Case studies offer greater level of detail and internal validity by closely examining multiple data sources. Surveys attempt to collect data from a large number of cases, thus achieving a higher degree of generalizability [Runeson et al., 2012].

The main advantages of the case survey method are its ability to collect richer information than conventional survey, and extendibility to study more cases. That said, the data collected by a case survey are limited by the scope of the questionnaire instrument [Larsson, 1993, Petersen et al., 2017b]. Validity threats of our study are discussed in Section 3.3.

The original case survey method description suggests using coding and statistical methods to analyze the data [Larsson, 1993]. However, we ex-

Table 4.1: Steps of the research method

Step	Case survey method (Larsson [1993])	Theory building (Eisenhardt [1989])	The applied method
1	-	Define research questions and any a priori constructs	We start the study with a broad aim to collect primary data on how start-ups practice software engineering. We define our research questions in Section 3.1, and present the start-up life-cycle model in Fig. 4.1.
2	Select cases of interest	Specify a population	We aim to study start-up companies building software-intensive products and reach as diverse sample of start-ups as possible.
3	Design the data extraction form for elicitation	Craft instruments and protocols for data collection	We design a questionnaire for data collection. It is aimed at start-up practitioners with practical experience with software engineering in start-up setting, see Section 3.2.1.
4	-	Parallelize data collection and analysis	We parallelize the final steps of questionnaire design with the data collection to identify any issues with the questionnaire before scaling up the data collection.

Table 4.1: Steps of the research method (continued)

Step	Case survey method (Larsson [1993])	Theory building (Eisenhardt [1989])	The applied method
5	Conduct the coding	Conduct within case analysis, search for cross-case patterns using divergent techniques	First, we use open coding to extract relevant information from textual data. Secondly, we use the start-up life-cycle model, see Fig. 4.1, to categorize the cases and conduct cross-case analysis, see Section 3.2.3.
6	Use statistical approaches to analyze the coding	Iterative shaping of hypotheses, search evidence for ”why" behind relationships.	We condense findings from the previous step into more broader patterns, and perform a parallel quantitative analysis complementing qualitative results, see Section 3.2.4.
7	-	Comparison with conflicting and similar literature	We compare patterns from the previous step with literature. We seek for additional support and explanation for our findings, see Section 3.2.4.
8	-	Aim for theoretical saturation when possible	-

tend the method by adding more steps from the theory building process proposed by Eisenhardt [1989]. While both methods are compatible, Eisenhardt provides more details on inducing a theory from data, that is, in this study, the start-up progression model. We present a mapping between the two methods and the combined method in Table 4.1.

3.1 *Research questions*

To guide our study, we define the following research questions (RQ):

RQ1: What patterns pertaining software engineering can be ascertained in start-up companies?
Rationale: Very little is known of what software engineering practices start-ups use and what is the motivation for using or avoiding specific practices. Earlier studies report the use of light-weight, ad-hoc practices with emphasis on requirements engineering [Klotins et al., 2015, Paternoster et al., 2014, Melegati et al., 2016b]. However, most earlier reports use secondary data, explore only a few cases, and are based on limited understanding of engineering context in start-ups.

With this research question, we identify commonalities in engineering goals, challenges, and used software engineering practices in start-ups with respect to their life-cycle stage, see Fig. 4.1. An understanding of what goals and challenges shape the use of engineering practices are essential to:

a) Judge suitability of commonly used practices.
b) Devise new engineering practices to navigate specific challenges and to achieve particular goals.
c) Provide a blueprint of software-intensive product engineering in a start-up context [Klotins et al., 2018b].

We formulate three sub-questions to explore goals, challenges, and practices specifically.

RQ1.1: What goals, relevant to software engineering, can be ascertained in start-up companies?
Rationale: We aim to explore what goals are driving software engineering in start-ups concerning their life-cycle stage, see Fig. 4.1. A fine-grained understanding of the goals, i.e. drivers for engineering activities, helps to understand the context of why certain engineering practices are used, or avoided [Kirk and MacDonell, 2014, Klotins et al., 2018b].

RQ1.2: What challenges relevant to software engineering can be ascertained in start-up companies?

Rationale: Engineering challenges is another context factor, alongside goals, shaping engineering practices in start-ups. We aim to explore what specific challenges, associated with start-up life-cycle stages, can be ascertained in start-ups.

RQ1.3: What software engineering practices do start-ups use?
Rationale: We aim to explore what engineering practices start-ups apply as a response to life-cycle stage-specific goals and challenges.

3.2 *Data collection and analysis*

3.2.1 *Questionnaire design*

We base the data extraction on a questionnaire eliciting practitioner experiences about their specific start-up case. The scope of the survey is inspired by our earlier work the Start-up Context Map [Klotins et al., 2018b] and covers team, requirements engineering, value, quality assurance, architecture and design, and project management aspects of start-ups.

During the questionnaire design, we conducted multiple internal and external reviews to ensure that all questions are relevant, easy to understand and that we receive meaningful answers.

The internal reviews were conducted among the authors to determine the scope and flow of questions. For the external reviews we invited 10 members of Software Start-up Research Network[1] to provide their input. Firstly, we asked them to fill in the questionnaire and answer all the questions as if they were engineers in a start-up. Then, we organized a joint on-line workshop to discuss participants reflections and potential improvements to the questionnaire. Their responses were removed from the final dataset.

Finally, we piloted the questionnaire with four practitioners from different start-ups. During the pilots, respondents filled in the questionnaire while discussing questions, their answers and any potential issues with the first author of this paper.

As a result of these reviews, we improved the question formulations and removed some irrelevant questions. The finalized questionnaire contains, 10 sections, 85 high level questions and 285 sub-questions, 73 of the sub-questions are free-text, others are on an ordinal or nominal scale. Note that one question in the questionnaire may result in multiple sub-questions, for example, a high level question may result in two sub-questions one capturing a Likert-scale answer, another free text motivation for the answer. The sections cover many software engineering topics, see Table 4.2. Note that through the analysis process some topics were merged, and some

1 Software Start-up Research Network: http://softwarestartups.org

lifted out. The last column of the table, process area, shows a mapping between sections of the questionnaire and the resulting process areas.

From all the questions, 54 sub-questions focus on capturing respondents agreement with statements addressing their engineering practices and engineering context. To capture respondents level of agreement with a statement we use a Likert scale: not at all (1), a little (2), somewhat (3), very much (4). The values indicate the degree of agreement with a statement. Statements are formulated consistently in a way that lower values indicate less agreement, however higher values indicate more agreement. We specifically avoid neutral (neither agree or disagree) answer option to force respondents to state their opinion. However, we provide the "I do not know" option to allow respondents to explicitly skip the question.

All the questions and answer options are available as supplemental material[2].

3.2.2 *Distribution of the survey and data collection*

To distribute the survey we reached out to The Software Start-up Research Network[3] and asked other researchers to use their networks and connections. All authors of this paper actively promoted the survey through their on-line accounts and personal contacts. The first author promoted the study in several start-up themed events. The data collection took place between December 1, 2016 and May 15, 2017.

To support the data collection we created an on-line tool. The landing page of the tool contained a short description of the study aims, and a promotional video encouraging start-ups to share their experiences. The questionnaire was public and freely available to everyone. To screen the responses and gauge their suitability for our study, the questionnaire contains a multitude of demographical questions about the start-up and the respondent. For example, what is respondents relationship with the start-up, their role in the company, and how long time ago they were in contact with the start-up?

A total of 369 practitioners started to answer the questionnaire. However, many of the responses were incomplete. We removed responses with less than 50% of questions answered. We further removed several duplicates and responses from non-software start-ups. The response rate of the questionnaire was 23%, 84 out of 369.

3.2.3 *Coding scheme and cross-case analysis*

We perform the cross-case analysis using both qualitative and quantitative methods.

2 Full questionnaire:
http://eriksklotins.lv/files/GCP_questionnaire.pdf

3 Software Start-up Research Network: http://softwarestartups.org

Table 4.2: Topics covered by the questionnaire

#	Topics	Questions	Number of sub-questions	Process area
1	Start-up demographics	1 - 8	12	-
2	Product demographics	9 - 17	18	-
3	Respondent demographics	18 - 23	10	I
4	Team demographics	24 - 30	12	I
5	Requirements engineering	31 - 47	57	II, III
6	Software architecture	48 - 55	19	V
7	User interface design	56 - 59	9	V
8	Development practices	60 - 69	80	I - VI
9	Quality assurance	70 - 77	48	IV
10	Project management	78 - 85	20	VI
	Total		285	

The responses are already structured by questions, and for multiple-choice questions, responses are already categorized. Thus, we need to code only responses from open-ended questions. Such questions help us to gain a finer understanding on how each topic is implemented in start-ups. Since there is no established body of knowledge of software engineering in start-ups, we use open, in-vivo, coding to gain insights how respondents themselves perceive and use software engineering. Thus, we applied open coding to identify described stakeholders, practices, artifacts used or produced, steps taken, and motivations for certain decisions [Corbin and

Strauss, 1990]. Each open-ended question addresses a different topic, thus we developed an individual coding scheme for each question.

The questionnaire is formulated to capture both used practices, and the practitioners' experiences with using the practices. Respondent reflections were captured in separate questions formulated along the lines of: "In hindsight, what would you have done differently?". In our results, we report and describe applied practices and respondents' experiences with the practices separately. However, the progression model in Fig. 4.4 contains only practices that were reported as used.

We use the start-up life-cycle model, see Fig. 4.1, to group cases by start-up life-cycle state. We analyze start-ups within each group and look for recurring engineering practices, challenges, goals, and contextual factors in their responses. We document these findings with memos. A memo describes a finding, start-up cases that were basis for the finding, category of the finding, that is, whether it pertains goals, practices, context factors or challenges, and to what state of the start-up life-cycle model the finding pertains to. We developed a total of 1856 codes and 366 memos. An example of open coding and memos is available as supplemental material on-line[4].

The initial coding is performed by the first author. The resulting memos are discussed and refined by the first and the second author jointly. In this process, memos with limited support from the data are removed, or merged with other similar memos. Interesting analysis points were marked for additional analysis. This is an iterative process leading to formulation of our findings.

As the final steps of our analysis, we use descriptive statistics and contingency tables to identify new and to confirm already identified patterns [Haberman, 1973]. We illustrate our results with frequency analysis and histograms, and test statistical significance of our findings.

We use the Chi-Square test of statistical association to test if the associations between the examined variables are not due to chance. To prevent Type I errors, we used exact tests, specifically, the Monte-Carlo test of statistical significance based on 10 000 sampled tables and assuming ($p < 0.05$) [Hope, 1968].

To examine the strength of associations we use Cramer's V test. We interpret the test results as suggested by Cohan [1988]. That is, we consider thresholds of $0.1, 0.3$, and 0.5 for weak, moderate and strong associations.

To explore the specifics of an association, such as which cases are responsible for this association, we perform post-hoc testing using adjusted residuals. We consider an adjusted residual significant if the absolute value is above 1.96 ($Adj.residual > 1.96$), as suggested by Agresti [1996].

4 Example of the open coding:
http://eriksklotins.lv/files/GCP_open_coding.pdf

The adjusted residuals drive our analysis on relationships between start-up demographics and reported engineering practices. However, due to the exploratory nature of our study, we do not state any hypotheses upfront and drive our analysis with the research questions. We document statistically significant findings with field memos for further analysis.

3.2.4 *Development of patterns*

To develop our results further, we revise and group together similar memos from the previous step to formulate broader findings. This process is aimed at building up evidence for supporting a particular finding. As suggested by Eisenhardt [1989], we apply the following practices:

- We analyze outlying, extreme, or otherwise surprising findings to shape our findings.
- We collect multiple variables on each topic and seek to triangulate our findings with multiple variables, and cases.
- We search for cases that present contradictory evidence to our propositions and shape our propositions to cover the negative evidence.
- To decide if our findings constitute a salient pattern we use a combination of criteria. Firstly, we look if a finding appears in more than one case. Secondly, we look if multiple variables support the finding, in particular, whether respondents have pointed it out in their reflections. Thirdly, if the finding is supported by statistical analysis.

As a result of this step, we identify 55 patterns pertaining to software engineering in start-ups. Similar to the memos before, a pattern describes a specific practice, challenge, context factor, or goal, along with information what cases were the basis for formulating this pattern, and information in what context this pattern was observed. The patterns are further grouped into 6 engineering areas and categorized into 33 patterns illustrating goals, practices, and challenges.

Goals are patterns describing a desirable outcome toward which an effort is directed. Such desirable results are identified either explicitly by question formulation, e.g. "What is the primary quality goal?", or by the practitioners' own reflections on why a certain practice was used. In few occasions, a goal overlaps with a practice, e.g. to use product usage metrics to gauge start-up performance, see G16 and P14 in Fig. 4.4. Such overlap indicates an association between use of the practice and attainment of the goal.

Practices are engineering activities helping start-ups to advance through the life-cycle stages. A practice could be means-end, such as establishing a team, could help to solve a problem, e.g. by documenting feature ideas, or help to gather knowledge on the current needs, e.g. by determining "good-enough" level of quality [Sheppard et al., 2007].

Challenges are difficulties in practicing software engineering and progressing through the life-cycle stages. We identify challenges from respondents reflections on how practices were applied and what they would do differently the next time. Some challenges overlap with practices, e.g. to establish a feedback loop with customers, see for example, C4 and P5 in Fig. 4.4. Such overlap indicates an association between the practice and the challenge, i.e. that start-ups attempt to use a specific practice, however find it challenging.

As a result of this step, we identify patterns pertaining to software engineering in start-ups. Similar to memos before, a pattern describes a specific practice, challenge, context factor, or goal, along with information what cases were the basis for formulating this pattern, and information in what context this pattern was observed. The patterns are further grouped into engineering process areas, resulting in the start-up progression model.

3.3 *Threats to validity*

Larsson [1993] identifies a number of validity concerns for case survey research.

Descriptive validity, factual accuracy could be compromised if responses from start-ups lack important information leading to an incorrect interpretation of the cases. To address this threat we iterated the questionnaire instrument with both researchers and start-ups to assure that all important aspects of software engineering in start-ups are covered and our questions are understandable to practitioners. Furthermore, we provided an explicit option to capture "I do not know" answers and, at the end of each section, asked "What would you do differently next time?" question to capture practitioner reflections on the most important lessons learned.

Respondent bias stems from participants' inability or unwillingness to provide accurate responses.

We collect respondents experiences and estimates of events that may have occurred in the past. Thus, the quality of the responses depend greatly on respondents memory and ability to reconstruct past events. Respondent responses suggest that majority of them, 67 out of 84, 80%, are currently involved with their start-ups or have been involved in the last 12 months. Thus, the majority of responses concern relatively fresh experiences.

Some of the respondents, 36 of 84, 43%, have specified that their main area of expertise in their start-up is other than software engineering, see Fig. 4.2. There could be a concern whether they can provide reliable answers to questions about software engineering. Earlier work suggests that start-up teams are closely-knit, team members perform multiple roles, and the effort is focused on launching one product [Giardino et al., 2016, Pater-

noster et al., 2014]. Thus, even if a respondents primary area of expertise is not software engineering, they are closely involved in product work.

We further explore potential differences in answers due to time since the last contact with the start-up and respondents area of expertise with statistical tests. As part of the last step of cross-case analysis, see Section 3.2.3, we test if respondents association with the start-up, age, time since the last contact, area of expertise, amount of experience in the area of expertise, and amount of experience working with start-ups, has any effect on their responses. Significant results from this analysis are presented in respective process areas.

There is a possibility that respondents are unwilling to provide honest responses, or twist the responses to what they believe researchers hope to hear. In the particular case of startups, and especially when respondents are answering about a failed case, it may be hard for them to admit were they failed and if they were not following what is perceived as the best practices in software development.

Participation in the study was voluntary, and we advertised the questionnaire as means to help other start-ups by sharing what worked and what did not in their start-ups. Thus, we minimize biases stemming from participants being forced to participate and to provide dishonest answers. The questionnaire contains a mix of multiple-choice, Likert scale, and free text providing multiple means of capturing the experiences, and mitigating the chance of extreme responses. We offer "What would you do differently next time?" questions to capture respondents own lessons learned.

Interpretive validity, objectivity of the researcher is concerned with potential bias stemming from researchers misinterpreting the data. To address this threat, the first three authors of this paper frequently met and discussed any intermediate findings as part of the analysis process. As a result, findings with weak support from the data were identified and removed, see Steps 4-6 in Table 4.1.

Generalizability of our results is determined by the number and diversity of the studied cases. We explore a diverse set of 84 start-up cases varying by geographical location, domain, product life-cycle stage, the extent of team expertise, and start-up outcome. That said, operational companies are overrepresented in our sample, see Fig. 4.3, and could bias our conclusions towards active, that is, to some extent successful start-ups. To compensate for this potential bias, we analyze operational and closed companies separately, compare the results, and apply statistical methods to determine significance of any conclusions.

Our sample mostly contains start-ups from Europe and South America, and start-ups from North America and Asia are underrepresented. For example, start-ups in the U.S. may have access to a broader and more homogeneous market than their European counterparts who need to adapt

their products to the diversity of Europe's markets. Such differences could have an effect on software engineering goals, challenges, and practices.

Repeatability of case surveys increases the objectivity of the findings. To strengthen repeatability of this study, we provide the data extraction form, full demographical information of the studied cases, and raw data[5] as supplemental material.

4 RESULTS AND ANALYSIS

After removing incomplete and irrelevant responses, we analyze 84 responses from start-ups building software-intensive products.

Some of the first questions in the questionnaire collects demographical information about the start-up, such as current state, state of the product, when the team started working on the product. The responses show that our sample consists of start-ups established between 2004 and 2017, with the majority (60 of 84, 71%) of start-ups actively working on their products at the time of the survey; 24 are closed, paused or acquired by other companies, see Fig. 4.3.

Responses on the product state show that our sample contains start-ups in all life-cycle stages, inception - 15 (18%), stabilization - 24 (29%), growth - 26 (30%), and maturity - 15 (18%), however 4 companies have not specified their product state. Most start-ups have their main office in South America (39 of 84, 46%) and Europe (33 of 84, 39%), the rest are located in Asia and North America. Such underrepresentation of North America and Asia could be explained by origin of the authors. The authors represent Europe and South America and were actively promoting the study in their networks.

With the questionnaire we collect respondents demographical information, such as age, experience, area of expertise, relationship with the start-up, and how recent they have been involved in the start-up. Responses show that most of the respondents (62 of 84, 74%) are founders, others are employed by start-ups (16 of 84, 19%), or are otherwise associated. The respondents are about evenly distributed regarding whether their area of expertise is software engineering or not, an how much prior experience they have in their area of expertise (left side of Fig. 4.2). However, most have little experience with start-ups before the current case (right side of Fig. 4.2).

Other details characterizing our sample and engineering context in start-ups are presented along with their respective process areas, discussed in the remainder of this section. A list of studied cases, respondents, and their demographical information is available as supplemental material[6].

5 Upon request to the first author, Eriks Klotins, eriks.klotins@bth.se
6 All cases and their demographical information:

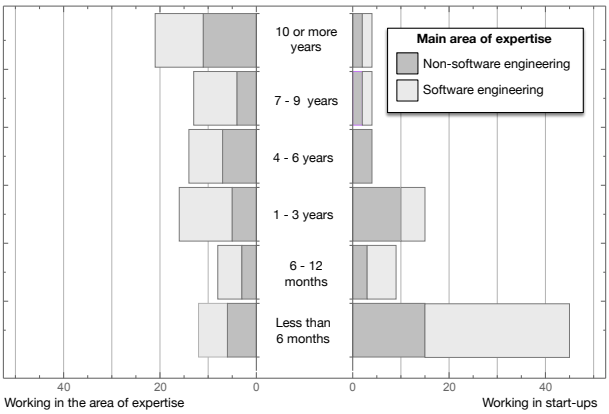


Figure 4.2: Distribution of respondent background and prior experience

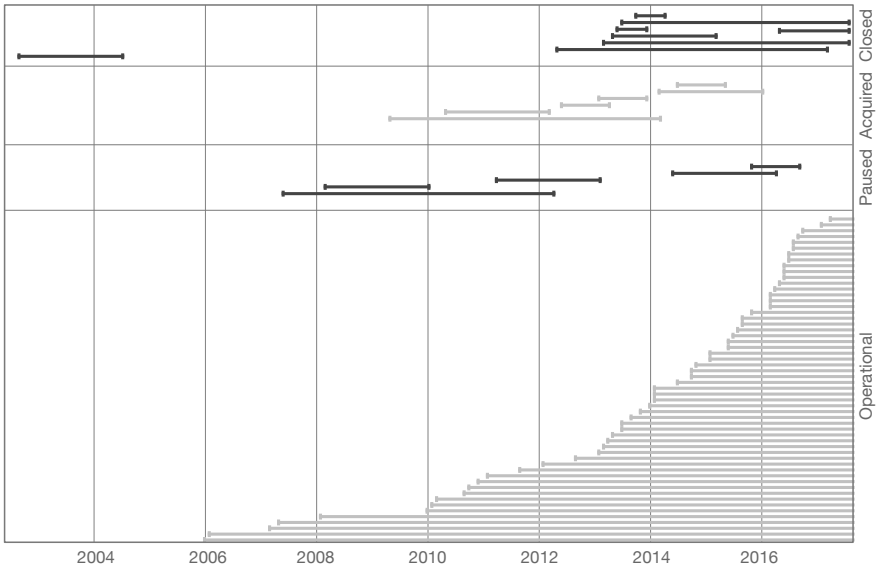


Figure 4.3: Gantt chart illustrating operation time and outcome for studied companies. 9 respondents had not answered when they started working on the product, thus these cases are not shown in the figure

We structure our results into 6 process areas: team, requirements engineering, value, quality assurance, architecture and design, and project management, see Table 4.2. Within each process area we consider goals, challenges, and practices, i.e., engineering aspects, see the legend in Fig. 4.4.

http://eriksklotins.lv/files/GCP_demographics.pdf

We analyze the process areas in relation to the four start-up evolution stages, inception, stabilization, growth, and maturity. In Fig. 4.4 we provide an overview of the results and present the start-up progression model. To maintain traceability between our description and the figure we enumerate our findings in the following way: goals (*Gx*), challenges (*Cx*) and practices (*Px*). On the top-right corner of each bar we denote how many cases were basis for formulating the finding.

The purpose of the progression model is to provide an overview of the critical stages of product development, main concerns, and relevant practices. For researchers, the model summarizes the key engineering concerns for further investigation and provides a structure for adding new results. For practitioners, the model helps to identify the current product stage, the key objectives to be attained and lists the key engineering practices for attaining said objectives. We discuss each process area, goal, challenge and practice in the following sections.

To present respondent estimates on Likert scale questions, we illustrate the distribution of answers with in-line histograms, for example, ( , "*To what extent it is true that most product/service features are invented rather than discovered?*").

The four bars denote the distribution of respondent agreement with a statement on a Likert scale ("not at all", "a little", "somewhat", and "very much'). The above example shows that most respondents "somewhat" agree with a statement (shown after the histogram), and the distribution of responses is skewed towards agreeing with the statement.

4.1 *Team process area*

The team process area concerns start-up teams with respect to formation of a team, individual skills, attitudes, capabilities, and coordination between individuals.

4.1.1 *Goals*

Establishing a team posing sufficient skills and expertise is one of the first goals of a start-up (G1). Earlier studies suggest that the team is the catalyst for product development in start-ups [Giardino et al., 2016]. Team characteristics such as cohesion, coordination, leadership, and learning are recognized as essential for software project success [Dingsøyr et al., 2016].

Across our sample, the median team size is 4 - 8 people and 1 - 3 of them primarily work on software engineering. We observe a tendency of growing median team size over the start-up life-cycle, from 1 - 3 in the inception stage, to 4 - 8 in the stabilization and growth stages, and to more than 20 in the maturity stage.

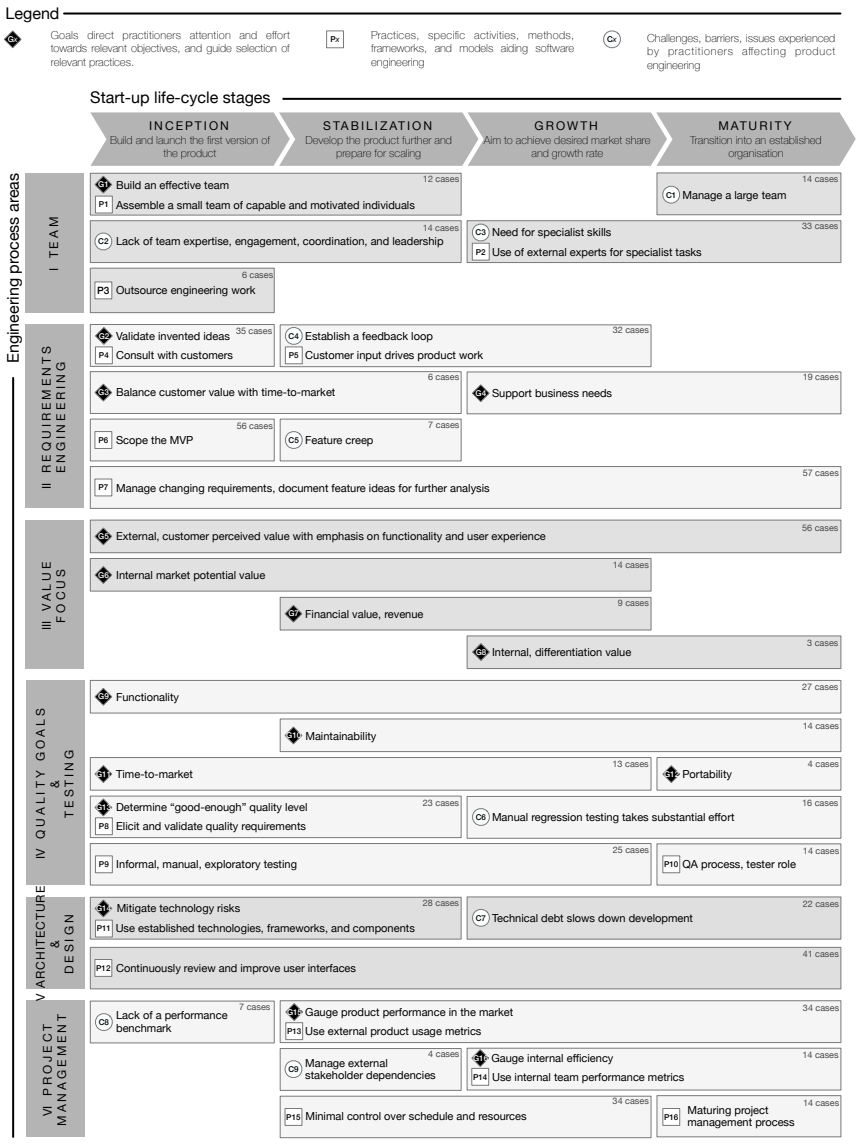


Figure 4.4: The start-up progression model outlining software engineering goals, challenges, and practices in start-ups

The responses suggest that to build a team, start-ups need to address communication issues, shortages of the domain and engineering expertise, commitment issues, and to create accountability. Statistical analysis on accountability in the teams shows an association ($Cramer's V = 0.334, p < 0.05$) between closed companies and a lack of accountability. Thus, form-

ing a unit that poses relevant and complementary expertise, and that can work together efficiently with little overhead is an essential early goal in start-ups, see G1 in Fig. 4.4.

By comparing responses from start-ups at different life-cycle stages, we found that establishing a team is a concern in the early stages of a start-up. We observe that start-ups at the inception and stabilization stages reflect on issues associated with creating a team, see G1 and C2 in Fig. 4.4. However, start-ups in the maturity stage reflect on challenges related to managing a large team (C1).

4.1.2 *Challenges*

Looking into respondent reflections on their team formation, we found that the majority, 61 out of 84, 72%, reports some team-related challenges. The challenges concern team formation, management, expertise, leadership, and coordination, see C1 - C3.

Start-ups at all life-cycle stages report shortages in engineering skills and domain expertise. The most severe shortages are reported by start-ups at the inception stage where only 3 out of 15, 20%, respondents, estimate their engineering, and 1 out of 15, 6%, rate their domain knowledge as sufficient. However, we observe a tendency of estimates improving over life-cycle stages. A potential explanation could be survivor bias and start-ups who managed to acquire the necessary competencies were able to advance [Shermer, 2014]. We also observe a tendency that less skilled teams spend more time in the inception stage and often fail to release the product altogether. As one start-up reflected on their lessons learned from their team formation:

> *"We should have looked for more developers with experience in delivering products fast, and do not rely on corporate "experts" that deliver anything but completed software."*

Statistical analysis shows a strong association between domain knowledge and engineering skills in the team ($Cramer's\ V = 0.513, p < 0.05$). Teams with less domain knowledge also lack engineering knowledge, and teams with adequate domain knowledge are likely to have sufficient engineering skills. The level of expertise is also associated with the state of the start-up ($Cramer'sV = 0.316, p < 0.05$). Operational start-ups estimate their skills and knowledge significantly higher than paused or closed companies. The level of team skills and experience is reported as one of the key success factors in software projects [Chow and Cao, 2008], and new business creation[Politis, 2008]. Thus, there could be a causal relationship between the level of team skills and start-up outcomes.

From the 41 start-ups in growth or maturity stages, 80%, or 33 companies, reflect on the need for specialist skills and the challenge to acquire

them (C_3). For example, finding employees with domain-specific experience, or engineers with specific technical skills. Often start-ups reflect, that it would have been beneficial to acquire such expert skills earlier. However, the responses do not reveal the cause for the difficulty.

The shortage of experts with domain-specific experience could be explained by short supply of experts in a narrow and potentially new area. Such explanation highlights the importance of education and training in start-ups. However, an alternative explanation could be that experts choose not to work with start-ups due to, for example, an uncertain future of the company.

By analyzing the responses on the team's attitude towards good engineering practices and their engineering skills we found that less skilled teams are less predisposed towards following good engineering practices such as avoiding code smells ($Cramer's\ V = 0.342, p < 0.05$) or thorough testing of their product ($Cramer'sV = 0.366, p < 0.05$). Further analysis reveals that the attitude towards following good engineering practices is associated with start-up outcomes. Operational start-ups recognize benefits from following good engineering practices, such as maintaining good software architecture ($Cramer's\ V = 0.400, p < 0.05$) and avoiding code smells ($Cramer's\ V = 0.320, p < 0.05$). Paused start-ups report seeing fewer benefits from good engineering practices. This finding suggests an association between start-up outcomes and attitudes towards utilizing good engineering practices.

Interestingly, respondents with more individual experience estimate their team attitudes towards good engineering practices and quality of engineering work significantly worse than less experienced respondents ($Cramer's\ V = 0.384, p < 0.05$). This finding could be interpreted as follows: a) inexperienced engineers overestimate the quality of their and their teams' work, or b) experienced engineers underestimate their work. Kruger and Dunning [1999] have studied cognitive biases in estimating own abilities. Their findings suggest that low-ability engineers cannot objectively evaluate their actual competence or incompetence, and are likely to overestimate their abilities.

We have also found that early start-ups do not implement any objective metrics to assess their team performance, see section 4.6. Therefore, implementation, evaluation, and improvement of engineering practices depend on competence and gut-feeling of early start-up engineers.

Individual skills, competencies and teamwork capabilities have been recognized as essential success factors in software engineering projects [Wohlin et al., 2015, IEEE Computer Society, 2014, Chow and Cao, 2008]. Earlier studies suggest that start-ups rely on implicit knowledge, and have very little, if any, organizational capital [Giardino et al., 2016, Seppänen et al., 2017]. Thus, establishing a small and efficient team is essential to compensate for the lack of organization. Our results show that early team issues

could be a reason for stalling product engineering and collapse of a company even before the product is launched to market.

Another challenge reported by 67%, 56 of 84 companies, is to engage the team and coordinate the product work. The difficulty stems from team members having other priorities outside the start-up in combination with a bloated, poorly organized, and distributed team. The reported issues in such teams are poor communication, lack of clear responsibilities and unbalanced skill set with further effects on productivity, degrading motivation, and poor execution of multiple parallel activities, e.g., product engineering work and marketing, among other challenges.

From all team related concerns we distill 2 main challenges. The first challenge pertains team building and comprises of a lack of team expertise, engagement, and coordination at the inception and stabilization stages, see C2. The second challenge, see C1, is to manage a large and potentially distributed team at the maturity phase. Respondents mention difficulties to coordinate and maintain efficient teamwork across multiple teams and time-zone. Such findings suggests that start-up principals need to recognize and adapt for different teamwork challenges as the company moves forward.

Part of the challenge to initiate teamwork is the absence of leadership to establish an engineering team and to drive the product engineering work. As stated by one start-up:

> *"All of the founders had other occupations. 3 professors (2 of them located in the USA), and one business owner located in Turkey. There was a lack of communication. The developers were hired part-time. There was no-one for whom this start-up was their primary occupation. Maybe hiring a manager would have helped."*

Looking more into leadership we found 4 cases explicitly pointing out the need for technical leadership. A technical leader, or CTO, is needed to set up an engineering team and to lead product engineering work. As one respondent, employed by a start-up, stated:

> *"I would fire the current CTO and hire a new CTO, who is more technically sound. In the startup, the most prominent thing is that the CTO should be technically sound. Otherwise, it is hard to drive the product forward and to motivate the development team."*

A potential pitfall is to select the technical product leaders based their executive powers and not professional competences. This could be the case when, for example, a less competent founder refuses to give away the CTO role to a more suited employee [Crowne, 2002].

An association between good teamwork practices and software project cost and quality is studied in the context of established companies [Krishnan, 1998]. Higher team capabilities regarding technical skills and domain

knowledge, are strongly associated with a lower number of discovered defects and lower software maintenance costs. Moreover, commitment to a shared goal and internal team communication mechanisms are essential for project success. Our results indicate that such findings are relevant in the start-up context as well.

We looked into how the respondents' relationship with the start-up affects their responses. We found that founders of start-ups are significantly more optimistic about the quality of planning ($Cramer's\ V = 0.381, p < 0.05$), and quality of product engineering ($Cramer's\ V = 0.385, p < 0.05$), compared to hired engineers and external contractors. Such results suggests a potential fault-line in the teams between founders and employees in terms of how they perceive the engineering context.

As shown by Chow and Cao [2008], joint decision making and knowledge sharing, critical to project success, depend on efficient communication. However, fault-lines splitting a team into two or more sub-groups, result in impaired communication and further adverse effects from stemming from communication issues. Causes and effects of team fault-lines are observed and studied in the context of globally distributed teams, for example,Gopal et al. [2011] and Staats et al. [2012].

Earlier studies suggest that start-ups have small, flat and empowered teams. Empowerment of individuals is supposed to reduce the need for bureaucracy and improve flexibility [Paternoster et al., 2014]. However, our results suggest that even though teams are small, there still exists a communication gap between founders and employees. An explanation could be that principal decisions from founders could be ill-communicated, thus perceived by employees as unjustified, and degrading motivation and trust in a team [Boies et al., 2015].

4.1.3 *Practices*

We looked at practitioner responses to identify how team formation challenges are addressed in their start-ups. We recognize two general scenarios how start-up teams are formed (P1). One scenario is creating a new team from people without previous joint experience. The second scenario is that a team originates from a former organization and already has some teamwork experience. Not every start-up could have the opportunity to reuse an existing team, thus we consider these both strategies as variations of the same practice to establish a team.

The responses suggest that newly created teams start with a few founders and new people are added when there is a need for additional skills or human resources. Such organizations have yet to establish teamwork practices, acquire domain knowledge, and vet their engineering capabilities. Thus, new teams are prone to teamwork issues, shortages of skills and expertise.

Teams originating from earlier projects are slightly larger and have more diverse competencies (compared to new teams), shared history concerning established ways of working, roles, responsibilities, and had already ironed out initial team formation issues. While we do not know the exact relationships between team members in all cases, 9 respondents mentioned that their teams have been working on earlier projects suggesting that they have shared experience before the current project. An example of this scenario would be a small consultancy company, i.e., offering customized services, that identifies an opportunity to develop a product for mass-market. They keep the consultancy business going to support the start-up endeavor, eventually aiming to become a product organization.

Making use of an existing team helps to alleviate initial team formation challenges, see C2 in Fig. 4.4, and minimize the risk of the team breaking apart. Moreover, an existing team likely has experience in relevant product engineering technologies, markets, and the product domain. Therefore, such teams have an advantage over recently formed teams with no shared background and experience. Similar results, pointing out that start-up founders' earlier experience shapes their skills and attitudes helping to cope with uncertainty are reported by Politis [2008]. However our results show that skills and expertise of the team as a whole plays an important role.

Start-ups report different tactics to address the lack of engineering competences. As an alternative to establishing an own engineering team, 3 start-ups mention outsourcing product development work to another company (P3). The motivation for outsourcing is to quickly build the first version of the product without the effort of creating an own team. However, outsourcing the engineering work comes with challenges to negotiate requirements and communicate efficiently. As one respondent stated:

> *"We started the development offshore with an external company, now development is in-house. With fewer people, we have the same cost, but we at least tripled the productivity."*

Start-ups at all life-cycle stages mention the use of external consultants to help with specialist tasks (P2) in addition to their engineering team. For example, 6 start-ups mention that they have used external user interface specialists to help with product design. Some mention using external developers for mobile application development, security, and optimization related tasks.

4.1.4 *Lessons learned*

Our results support earlier findings that team is the catalyst for product engineering [Giardino et al., 2016]. Team issues could hinder start-up potential to advance through the life-cycle stages.

For start-ups, our findings present several implications:

1. Team formation is an essential early activity. To attain a highly performing team, a team building program must be implemented, focusing on establishing respect for everyone in the organization, identify and communicate individual performance standards, develop ways of efficient communication, identify clear individual and group goals, reward teamwork, and team-building efforts, and encourage loyalty to the team [Tippett and Peters, 1995].
   Suggested reading: Bubshait and Farooq [1999] presents critical concepts influencing team performance and lists building blocks for establishing highly performing teams.
2. Engineering and domain expertise are important for efficient teamwork. Engineering expertise is essential to build the product fast. However, domain understanding helps to identify and interpret software requirements [Hadar et al., 2014]. Start-up teams are often formed by inexperienced people or work in new domains, thus knowledge sharing and mutual learning is essential.
   Suggested reading: Eppler and Sukowski [2000] presents processes, tools, and factors for enabling team knowledge management. Cockburn and Highsmith [2001] explore the role of individual competences in agile team performance.

4.2 *Requirements engineering process area*

The requirements engineering process area concerns the elicitation, analysis, validation, documentation and scoping of software requirements. The identification and validation of a relevant product idea, that is requirements identification, validation, and scoping, are some of the most important activities in start-ups [Blank, 2013b].

4.2.1 *Goals*

One of the first steps in any software project is to identify needs and constraints placed on a software product [IEEE Computer Society, 2014]. Respondents from start-ups at the inception stage agree that product features are to a large extent invented, and are based on founders experience, and understanding about the domain ( , *"To what extent it is true that most product/service features are invented rather than discovered?"*).

As a consequence of invented requirements, one of the first objectives in a start-up is to break down invented ideas into software requirements and to validate these requirements, optimally with inputs from target customers (G2). As one practitioner stated:

> *"The real purpose of requirements elicitation is to invalidate requirement ideas as quickly as possible without unnecessary effort on meta documentation or implementation."*

Responses show that as soon as the product is launched to market, start-ups put more emphasis on using their customers as requirement sources. Requirements invention becomes less common. Responses, to what extent product requirements are invented, shifts towards disagreement when start-ups mature. ( , "*To what extent it is true that most product/service features are invented rather than discovered?*"). Thus, the goal to quickly invalidate own ideas is most relevant at the inception stage and before start-ups have established a feedback loop and could use customer input for feature ideas, see C4 and P5 in Fig. 4.4.

Requirements validation is mainly about what functionality and quality to offer. However, it is essential to differentiate between the validation of the idea itself (requirements) and how it is provided (design) [Van Lamsweerde, 2003]. The differentiation is significant. A great idea can be received poorly if the design of it, regarding how the solution is offered, is bad.

Results from established companies working on market-driven products are similar. Mature companies, alike start-ups invent or discover requirements indirectly through analysis and observations. Established companies face similar challenges to validate requirements before a product is launched. This is compensated by internal requirements analysis and frequent releases [Dahlstedt et al., 2003, Alves et al., 2006].

To achieve the goal of the inception stage and to release the first version of the product, see Fig. 4.1, start-ups must determine the scope of the minimum viable product (MVP). The MVP is a trade-off between features, quality, time, and cost [Junk, 2000] used to gauge market interest in the product and to establish an early customer base justifying further investments in the product.

When asked how the MVP was scoped and what quality attributes were considered necessary, the majority of respondents, 56 out of 84, 67%, pointed out that the priority is to maximize customer value through functionality, usability, and user experience, while keeping engineering effort minimal, see P6. Practice of scoping the MVP is related to the goal of balancing customer value with minimal development effort (G3). As one respondent reflects the MVP should be scoped by the current needs, and not by wishful thinking:

> *"Any requirement that is essential for validating the growth or value hypotheses is priority. Anything that is to support the business when the user base is larger than 100 users is not a priority. Scoping is an exercise of un-prioritizing features from being added to the MVP."*

Responses on release scoping goals from start-ups in stabilization, growth and maturity stages suggest a shift in scoping goals compared to the inception stage. The release scoping goals of inception and stabilization stages are to maximize customer value and to validate the product idea, however later goals shift to supporting business goals, such as monetization and growth (G4). We present related results on value focus and product quality goals in Sections 4.3 and 4.4.

4.2.2 *Challenges*

Comparing responses from active and closed start-ups at stabilization and growth stages we observe that internal sources, such as brainstorming and invention of requirements, are the most popular requirement sources and used by 94% of active, and 71% of closed start-ups.

Other requirement sources are similar products, used by 83% active and by 43% closed start-ups, and market trends, used by 57% active and by 29% closed start-ups. However, only 43% of closed start-ups have used input from potential and existing customers compared to 91% of active start-ups. This difference is statistically significant (Cramer's $V = 0.463, p < 0.05$).

Looking at the free text responses for an explanation, we found that all closed start-ups, and 10 out of 35, 29% of the active start-ups had difficulties establishing contact with their potential customers and involving them in the product work (C4, P5). The common shortcomings are involving customers too late in product work, for example, only after release to market, and sampling of potential customers for requirements elicitation and validation. As one respondent reflected:

> *"We informally asked our friends and family about existing solutions, then we brainstormed with the information collected about how to design the product and what features it should have. We never used a formal method to elicit requirements, we just informally decided the features our product should have."*

The importance of stakeholder involvement in new product development as a success factor has been pointed out by nearly every study on requirements engineering, for example, Chow and Cao [2008], Karlsson et al. [2007], Hoyer et al. [2010], and Blank [2013a]. However, as our results show, finding the right sample of customers and convincing them to invest time in collaboration is challenging.

Establishing relationships with potential customers is a goal of sales activities as well. A recent study on new product development from the sales perspective highlights the overlap between requirements engineering and sales processes, pointing out that getting to know customers is essential for both technical and commercial outcomes of the project. Involving both

engineering and sales roles in establishing customer contacts helps to ensure continuity between requirements engineering and commercial relationships [La Rocca et al., 2016].

Some, 7 of 24, 30%, start-ups at the stabilization stage reflect that they had believed that adding more features to the product would improve their chances of success. That constitutes feature creep, see C5, and stems from difficulties to elicit useful feedback from customers, and generalizing customer specific requirements. Start-ups report that feature creep drained financial resources and added extra complexity to the product. As one company reflected on their lessons learned:

> *"We should have done usability tests with real customers with a prototype of the product and iterate faster based on user feedback and actual usage, not just guessed what customers might want."*

Feature creep is reported as a general challenge in developing new software products. It is caused by adding new requirements late in the product development without appropriate analysis. The new requirements could stem from external stakeholders, thus appear very appealing to include in a release, or could be discovered and self-approved by the team during the development process. Either way, a company should analyze the impact and assure that higher business goals are not compromised [Elliott, 2007].

4.2.3 *Practices*

Responses suggest that start-ups use a mix of requirements sources. Across all cases, internal sources such as requirements invention and brainstorming are the most reported by 76 out of 84, 90%, start-ups, followed by potential and existing customers (66 cases, 79%), and analysis of similar products (59 cases, 70%). Market trends and business goals are less utilized, standards, laws, and regulations are used by start-ups in regulated domains, such as medicine.

Respondents suggest that they have used their previous experience in the domain, both professional and from using similar products, to identify "obvious" requirements and requirements sources. However, customers are the most valuable source of requirements, as stated by one respondent:

> *"The most important source is existing and potential customers where we have continuous dialog in place. This is somewhat self evident. Understanding of competition, business models and regulations is secondary to that."*

Elicitation, validation: Start-ups report using observation (64%) and interviews (61%) to elicit requirements from customers. On-site customer and surveys are less used, 32% and 46% respectively. Prototypes and mock-ups

are used by 61% of respondents to support brainstorming and to elicit feedback in customer interviews.

Responses suggest that elicitation is triggered by internal ideas that are further elaborated and iterated with input from customers. As one company reflected:

> *"Elicitation is a cyclical process of getting info from customers/products, analyzing and then brainstorming to feed into prototypes and mock-ups. Early in the project, this was at a very high level. Later when we concentrated development on specific features, the steps were repeated in a more detailed manner."*

Requirements elicitation overlaps with requirements validation (P4). Or, as one respondent put it: "the elicitation techniques are used to IN-validate requirements/ideas/features". The most frequently reported validation technique is internal reviews used by 55%, 46 start-ups, followed by prototype demonstrations to customers reported by 49%, 41 start-ups. A/B tests to measure customer reaction on new features are used by 25%, 21 start-ups. Use of A/B testing increases over the start-up life-cycle from 6% at inception stage to 34% at the maturity stage, potentially because A/B tests and other data-driven methods require a significant number of customers interacting with the product and such numbers may not be available at early stages [Olsson and Bosch, 2014].

By looking into associations between difficulties in requirements elicitation and demographical information we found that younger respondents, 25 - 34 years old, report more difficulties in collecting and prioritizing requirements than older, 35 - 44 years old, respondents ($Cramer's\ V = 0.599, p < 0.05$). An explanation for this finding could be that older age is associated with a broader network of personal contacts and more extensive domain experience supporting identification of relevant ideas for product features.

Similar results are reported by P. Azoulay [2018] suggesting that with older age comes more experience, business acumen, broader social network, and greater access to financial resources, thus older founders are more likely to succeed commercially.

Start-ups in all life-cycle stages report that requirements are changing and they have an informal process to manage changes (P7). The most common source of changes is input from customers. Responses suggest that start-ups aim to work in short iterations and frequently re-prioritize their backlogs. Thus, requirements changes do not have any significant adverse effects. As one respondent described their change management process:

> *"Yes, requirements change! We happily abandon the obsolete requirements and remove any code from the product if there is any. New requirements are re-prioritized with a goal to validate our growth and value hypotheses."*

From our sample, only 7%, 6 start-ups, have explicitly stated that they do not document their requirements in any way. The most used specification format is informal notes and drawings, reported by 56%, 47 respondents, others report more formal techniques, such as the use of templates and formal specifications, for documenting requirements. Most commonly, start-ups document requirements on a feature level (43%, 35 cases), and a function/action level (21%, 18 cases). Requirements are written down as ideas which are later elaborated (P7). As one respondent reflected on their practice:

> *"We used an on-line tool (Trello) to write all the features we wanted to implement and, at the same time, to organize their development. Trello cards served as the requirements to be implemented. There were no steps in documenting requirements per-see, we just talked informally about what to implement and then wrote it down in order to not forget it."*

With further statistical analysis, we found that the understandability of requirements is associated with the state of a company ($Cramer's\ V = 0.319, p < 0.05$). Significantly more closed companies, compared to active ones, report that even though requirements were written down, they were difficult to understand and use in practice. Teams with insufficient domain knowledge are also more likely not to document their requirements. However, teams with adequate domain knowledge are more likely to use templates for documenting their requirements ($Cramer's\ V = 0.345, p < 0.05$). These findings suggest that more rigorous requirements documentation could be a way to acquire, document and distribute critical domain knowledge in the team. Alternatively, teams with better skills and domain knowledge see the upside of documenting requirements. Either way, we observe an association between understandability of requirements, improved domain knowledge, and progression of a start-up.

Requirements documentation enables to create a plan, outlining what features to implement when, i.e., develop a product road-map. The responses suggest that 46 start-ups, 55%, have such a road-map. From start-ups at inception, stabilization and growth stages, about half, 40 - 50% of the star-ups, report having a road-map. However, product road-maps are used by nearly all, 87%, of mature start-ups. Between closed and active start-ups, 67% of operational, 40 cases, have road-maps. However, only 2 or 18% of closed start-ups had road-maps. Respondents reflections suggest that road-maps are used as planning documents internally and often synced with primary stakeholders. Parts of a road-map that concerns near future are more detailed, however long-term plans are described at a higher, milestone level.

As suggested by the start-up life-cycle model, the first significant milestone for start-ups is to release the minimum viable product, see Fig 4.1.

Responses suggest many strategies for scoping the MVP (P6), such as using their domain knowledge, input from potential customers and partners, soft launch by releasing the product to one customer at the time, time-boxing, and "gut-feeling". With the MVP, start-ups aim to deliver the essential features in a shortest time possible. As one practitioner described the MVP scoping process:

> *"We carefully removed any feature or function not essential to testing growth and value hypotheses. Of which two, the value hypotheses is prioritized."*

Value is specified as the primary prioritization goal by 85%, 71 of 84, of respondents. Implementation time is the secondary goal reported by 38%, 32 cases. Requirements prioritization is done by consulting customers and other stakeholders. We observe that at the inception and stabilization stages, start-ups consider customer needs, while at the growth and maturity stages, business requirements, such as a need for revenue and growth, are discussed as well. As one practitioner described their prioritization process:

> *"To prioritize we use this question: how much money or new customers we will have if we implement the new feature?"*

In their reflections, nearly all respondents from both active and closed start-ups, suggest that they should have spent more time with customers to understand and analyze their needs better. As one respondent describes their lessons learned:

> *"In the start we let innovations lead our goals too much. Then we got our first customers and did not listen to them much. We did not track in a enough detailed way what the customer is exactly doing with our product. We let just one dedicated person to be in contact with certain customers and did not share details in the team. Even a small company needs to setup a program to make interaction with customers transparent and actionable."*

Comparing our results on requirements engineering in start-ups with results from established market-driven companies we observe many similarities. For example, in both contexts requirements are primarily invented, used practices are light-weight and informal, and focused around quick releases to elicit customer feedback [Dahlstedt et al., 2003, Alves et al., 2006]. However, we observe a difference in prioritization practices. Established companies rely more on effort estimates in prioritizing requirements, while start-ups use value as the primary prioritization target [Dos Santos et al., 2016].

The discrepancy in prioritization goals could be explained by a lack of unified and quantifiable view on value. Thus, established companies

opt for scalable and straightforward prioritization criteria [Khurum et al., 2012]. Moreover, established companies are likely to operate within a set budget and schedule constraints further motivating the need to adhere to effort estimates. However, start-ups are more customer-centric, flexible, and work on a smaller number of features, thus can use value as prioritization target [Giardino et al., 2014a].

4.2.4 *Lessons learned*

Our results support earlier findings that requirements engineering is one of the key engineering activities in start-ups [Melegati et al., 2016b]. Moreover, our results show an association between requirements engineering practices and state of a start-up. We present the following implications for practitioners:

1. Starting to collect input from potential customers as early as possible is the key to identifying the most relevant requirements. By involving customers in new product engineering, companies can achieve a higher degree of efficiency, ensure product fit with customer needs, and achieve higher customer engagement and satisfaction [Hoyer et al., 2010]. Moreover, early customer relationships are a basis for sales activities when the product is launched.
   Suggested reading: Cui and Wu [2016] compiles earlier work from marketing and innovation literature and presents practical guidelines on how to involve customers and use their knowledge in developing innovative products.
   Hoyer et al. [2010] present a framework for value co-creation in product development comprising of motivators, outcomes, and potential impediments.
2. Feature creep can be managed by requirements analysis focusing on how many customers will find a feature useful. Only features that concern the majority of the customers should be included in the roadmap.
   Suggested reading: Elliott [2007] proposes to use the Pareto principle in deciding whether a feature is part of the core product or is customer specific. He argues that about 20% of features are used by 80% of customers, and the key to avoiding scope creep is to pinpoint the 20%. Also, the paper lists several strategies for handling feature creep.
3. Writing down requirement ideas, their source and rationale can help to acquire, maintain and distribute domain knowledge in the team. For example, even a basic requirements specification helps to establish a common vocabulary and avoid delays and time-consuming interactions caused by confusion and misunderstandings between stakeholders [Buchman and Ekadharmawan, 2009].

Suggested reading: Hadar et al. [2014] explore the role of domain knowledge in requirements elicitation. They analyze both positive and negative effects of prior domain knowledge in elicitation interviews. Domain knowledge helps to ask more focused questions, provides a common language, and saves time in learning the basics.

4. Release scoping, especially scoping of the minimum viable product, should be done carefully and optimally with clear goals and input from all stakeholders. Our results from start-ups are similar to results from established companies suggesting that facing uncertainty companies are likely to overscope their product releases [Bjarnason et al., 2010b]. Suggested reading: Bjarnason et al. [2010b] presents a root-cause analysis and effects of release overscoping in software projects.

4.3 *Value focus*

The software value concept characterizes the broader aims of a company and aligns all activities in an organization towards defined value goals [Boehm, 2003]. Respondent responses often mention value as a criterion for prioritizing requirements and scoping product releases. However, a value is a vague term and could mean different things to different stakeholders. To explore how start-ups interpret the phrase we map their definitions of value to software value aspect taxonomy [Khurum et al., 2012]. We summarize these different views as engineering goals.

4.3.1 *Goals*

Across the sample, the dominant view on value is the customer perspective, reported by 48% or 40 start-ups. Respondents define customer value as perceived benefits, regarding functionality, user experience, and hedonistic value, derived from the product, and potential for the company to capitalize this value, see G5-G8. As one respondent phrased it:

> *"With value we understand the benefit for a customer to use our product. Higher value strengthens our position in the market compared to competitors and helps to increase the price of the product."*

The second most reported interpretation of value stems from the internal business perspective, i.e., how the company estimates product value from their own, internal perspective (23% or 20 start-ups). Respondents define this perspective as both market potential, e.g., how many customers could benefit from a particular feature (G6), and differentiation value, e.g., how a feature will help to stand out in the market (G8). As one respondent put it:

"Value is something that can be used to market/distinguish the product from competition."

Financial value concerning revenue is the third most reported interpretation of value, by 11% or 9 respondents (G7). However financial value is often defined in combination with other value perspectives, such as customer or internal value. As one respondent described their multi-faceted view on value:

"Value is what could give more revenue to the company and make the product easier to use for users and motivate them to purchase, so that we could have more transactions each day."

We compare value definitions between start-ups at different life-cycle stages and observe several tendencies. Customer value pertaining perceived benefits of the product of is the dominant value perspective in all life-cycle stages, reported by 23 - 46% of respondents. Internal value capturing product's market potential value, for example, ability to serve more customers efficiently and to access broader markets, is reported by 20% of respondents at inception and growth stages, 12% at the growth stage, and not reported at all by start-ups at the maturity stage. Financial value of the product is not reported at all by companies at inception stage, however it grows over start-up life-cycle and peaks at the maturity stage where 20% of start-ups have reported it. Internal, differentiation value are more reported by start-ups at inception and maturity stages, and appear less relevant at stabilization and growth stages.

A similar analysis conducted in established product companies shows somewhat different results [Alahyari et al., 2017]. While the ranking of value is similar, customer value being the most important, followed by internal value and financial aspects, we observe differences in the definition of these values. For example, established companies interpret customer value as delivery time and perceived quality. However, none of the start-ups in our sample have mentioned time-to-market or product quality in their value definitions. The internal value in established companies is understood as internal product quality, technical debt, supporting tools and processes. Start-ups, in turn, focus on market potential and differentiation value.

Differences in the studied samples can explain these discrepancies in value focus. Established companies could see more value in consistently delivering quality features to existing customers and keeping technical debt low. However, start-ups aim to identify an unmet customer need in a high-potential market [Sutton et al., 2000, Blank, 2011].

4.3.2 *Lessons learned*

Our results show shifting views on value definition among start-ups in different life-cycle stages. However, there is a shortage of similar results (except for Alahyari et al. [2017]) for comparison and deeper analysis. Nevertheless, we present the following implications for practitioners and gaps for further investigation:

- Understanding of value focus can help practitioners to the scope and align their engineering and business activities to maximize specific types of value. Alignment of activities helps to ensure that technology actually supports specific business goals and deliver the intended value to stakeholders [Carlson and Wilmot, 2006].
- Our results show that at least two value perspectives are relevant at any life-cycle stage. The value focus shifts from customer value and internal market value at inception stage to financial and differentiation value at the maturity stage, see Fig. 4.4. Understanding of the multi-faceted nature of value can help to facilitate communication between stakeholders [Khurum et al., 2012].
  Suggested reading: Khurum et al. [2012] present a taxonomy of software value aspects establishing a common vocabulary and understanding on different value aspects.
  Carlson and Wilmot [2006] present a practical guide on how to identify, use and develop an understanding of the value and use value to align organizational efforts in building innovative products.

4.4 *Quality goals and testing process area*

This process area concerns product quality goals and practices to attain these goals. Product quality is a mix of functionalities, non-functional attributes, and broader constraints determining commercial success of a product (e.g., cost of product development vs. returns from marketing the product). We aim to explore what aspects of software quality are considered significant by practitioners and what practices are used to attain such aims.

4.4.1 *Goals*

Respondent answers suggest that product functionality is the most common quality goal (G9) across all life-cycle stages, reported by 32%, 27 respondents. Time-to-market is the second most common objective (G11), indicated by 15%, 13 start-ups. Even though time-to-market is not a product quality goal intrinsically, it has a profound influence on the product decisions [Giardino et al., 2016, Carmel, 1994a]. The frequency of other quality goals varies across life-cycle stages.

Maintainability is frequently reported quality goal in stabilization, growth and maturity stages (G10). Portability is reported only by start-ups at the maturity stage (G12). Reliability is mentioned by few cases in stabilization and maturity stages. Shifting quality goals indicate suggest changing priorities and adds support for start-up life-cycle model, see Section 2.4.

Looking further into how the required level of product quality was determined we found that start-ups reflect on a "good-enough" level of their quality goals (G13). This goal is related to scoping of the MVP (P6) and focus on non-functional features of the product. Wrongly estimating the required quality level (G13, P8) could lead to poor market reception due to less than acceptable quality, or waste by providing excessive quality [Regnell et al., 2008].

4.4.2 *Practices*

Start-ups report using different strategies, based on in-house expertise, user feedback, and iterative development (P8) to set their quality targets.

One strategy is not to set any specific quality targets and iteratively identify, and improve relevant quality aspects. As stated by one company:

> *"We take a continuous iterative refinement approach rather than setting a fixed goal, so we periodically assess which areas are in need of improvement, evaluate and prioritize the options."*

As an alternative, one start-up at the inception stage reported aiming for the simplest solution that can be improved, if needed:

> *"We look for what is the simplest, regarding size and complexity, solution to this problem. Will that solution be able to support a couple of hundred users? How many different ways can the solution be improved to support more users?"*

Regnell et al. [2008] describe a quality requirements roadmapping model for determining the required quality level. The method proposes to identify relevant quality metrics and defines useless, useful, competitive, and excessive quality ranges for each. Then, it uses these ranges to compare the own product with competition and to spot opportunities for improvement. Using such a model in a start-up could help in determining essential qualities, a minimum quality level for a product to be useful, and opportunities to differentiate among similar products. This is aligned with the focus of internal value start-ups typically have (see Section 4.3.2).

Respondent answers suggest that informal manual testing is the most common practice for making sure that the product has an acceptable level of quality at all life-cycle stages, reported by 33%, 28 start-ups (P9). However, the responses suggest that informal testing is gradually replaced with an organized QA process (P10) at the maturity stage. Alternative methods,

often used in parallel, are exploratory testing, or scenario-based testing, reported by 21% and 19% start-ups respectively.

Respondents estimates suggest that test case documentation varies from informal to systematic without any clear tendency ( , "*To what extent it is true that test cases are not systematically documented?*"). Estimates on the test case coverage are biased towards agreeing with less than complete coverage ( , "*To what extent it is true that test cases do not fully cover the product functionality?*")

We observe little differences in testing practices between start-ups in different life-cycle stages. However, start-ups at growth and maturity stages reflect that more test automation, more skills regarding software testing, and more systematic testing would have been helpful. An explanation for such results could be that consequences of the informal testing surface only when a product gains a user base. Two start-ups at the maturity stage reflect that a dedicated tester role, responsible for performing testing tasks, is needed (P10). As stated by one of the respondents:

> *"I think we need more structured testing, a manager of testing that coordinates the efforts, all being responsible [for testing the product] is NONE being responsible."*

Answers to questions on the use of automated testing show that a third, 26 out of 84, 31%, of start-ups are attempting to implement automated testing, and only 17 start-ups, 20%, have explicitly stated that no test automation is used.

Responses suggest that regression testing is an increasing concern over start-up life-cycle (C6). Start-ups at inception and maturity stages report that they spend substantial effort on manually testing the entire product ( , "*To what extent it is true that manual testing of the entire product/service is required to make sure that a release is defect free?*"). At the same time, respondents report that few defects slip through testing and are reported by customers ( , "*To what extent it is true that customers often report defects that could have been captured earlier?*"). Two respondents from start-ups at the maturity stage have stated explicitly that they are working to improve and automate their product testing.

Start-ups could benefit from more rigorous testing practices. Efficient software testing enables faster product releases, thus allowing the teams to reduce time-to-market and to iterate new features faster. More rapid release cycles contribute to faster requirements validation (G2, P5) and are known to improve customer satisfaction [Chen, 2015].

Good testing practices and use of automated tests can support the onboarding of new developers. Automated tests provide a safety net for inexperienced developers to discover any problems with their code quickly on their own. Automated test definitions serve as means of documentation

to learn how different components of a product work [Pham et al., 2017]. Therefore, having good testing practice and test automation have benefits beyond a defect-free software.

4.4.3 *Lessons learned*

Our results show that start-ups primarily focus on delivering relevant functionality. Other quality aspects change as a start-up advances through the life-cycle. Software defects aren't a concern. However, the internal quality and testing practices are important to support the sustainable evolution of the product. We identify the following implications for practitioners:

- Our results suggest that external quality isn't a concern in start-ups, potentially due to relatively small products and early adopters being more tolerant towards defects. That said, quality becomes a concern in growth and maturity stages when commercial success of a product depends on quality service.
- We observe that maintainability is an increasing concern over a start-up's life-cycle. Maintainability helps a start-up to remain fast in launching new features and sets a foundation to enable portability of a product.
  Suggested reading: Regnell et al. [2008] propose a lightweight method for quality requirements roadmapping. The method helps to establish a frame of reference for assessing current product quality and spotting opportunities for improvement.
- Respondent answers suggest a lack of test automation and superficial testing practices. However, our earlier study on technical debt in start-ups could not find any significant association between testing debt and product quality or team performance issues [Klotins et al., 2018a]. Nevertheless, good testing practices help to have faster product releases and speed up the on-boarding of new developers.
  Suggested reading: Collins et al. [2012] present an experience report on test automation practices in an agile development context. They present several lessons learned from implementing test automation and what types of test automation bring the most benefit.

4.5 *Architecture and design process area*

The architecture and design process area concerns the internal product structure, selection and use of components, construction technologies, interfaces and other aspects supporting the construction of the product.

Earlier studies suggest that start-ups leverage on open-source components, third-party services and cutting-edge technologies to construct their

products [Giardino et al., 2016]. We aim to explore how start-ups choose the components and how they design their product architectures.

The visual appearance of graphical user interfaces influences the usability of a product and affects stakeholder perception about the product. Look and feel of product user interfaces is known to have an impact on project success [Ralph and Kelly, 2014]. We explore how start-ups design user interfaces for their products.

4.5.1 *Goals*

We inquired respondents on the use of cutting-edge technologies and found that start-ups aim to minimize risks from using immature technologies by using well-known, stable technologies in the product development (G14). However, 39%, 33 of 84, of the respondents report using or experimenting with some cutting-edge components on the side. As one start-up reflected:

> *"We don't use unstable components or new 3rd party services without evaluating them thoroughly and running a small pilot project."*

For at least one start-up in our sample, poor choice of a technology platform hindered quality, delayed product releases, and was the leading cause for discontinuing the product. Therefore, selecting a technology stack is an important goal early in the start-up life-cycle.

Earlier studies have pointed out that start-ups leverage on cutting-edge technologies to develop innovative products and gain competitive advantages [Unterkalmsteiner et al., 2014, Sutton et al., 2000]. However, we could not find support for such results in our dataset.

Looking into how organizations make technology decisions in other contexts, we found that performance regarding total time and resources efficiency, maintainability and reliability are the top criteria for software component selection in established companies. Similar results are reported by Petersen et al. [2017b], and Badampudi et al. [2016]. Start-up companies could be considering similar criteria and opting for more stable components to save development time and resources. Furthermore, start-ups could be leveraging on established, well-known technologies to create new and innovative products [Maranville, 1992].

4.5.2 *Challenges*

Respondent estimates suggest that technical debt is prevalent in start-ups, especially late in the life-cycle, see C7. Nearly all start-ups report some signs of technical debt, such as shortcomings in knowledge distribution, code smells, suboptimal architecture decisions, and lack of automated testing. Testing debt, associated with lack of automated regression testing

and a need to manually re-test the entire product before releases, is the most common type of technical debt. Documentation, architecture debt, and code smells are also a concern, albeit to a lesser extent.

In our earlier study with the same dataset, we found documentation, architecture debt, and code smells to be associated with impaired team productivity and product quality. However, we could not find any significant association between testing debt, productivity or quality [Klotins et al., 2018a].

Statistical results showed a significant association between start-up team size, team skill level and level of technical debt. Larger, less skilled teams are more likely to suffer from consequences of technical debt ($Cramer's\ V = 0.386, p < 0.05$).

Comparing reports on the technical debt between start-ups at different life-cycle stages we found a pattern of architecture debt growing over time and peaking at the growth stage (C7). By looking into respondent reflections, we found that start-ups at growth and maturity stages face challenges stemming from earlier architecture decisions. As stated by one respondent:

> *"Initially, the product was started by an inexperienced developer and many design decisions were incorrect. Fixing them early would have been easy, however with the product and user base growing, changing core things became increasingly difficult. Correcting simple architecture mistakes that could have been trivial to fix early, can take weeks now."*

Our results match earlier findings suggesting that technical debt in start-ups is caused by a need to develop a product fast and an inadequate team skill level [Giardino et al., 2016]. However, our respondent answers indicate that external pressures, such as competition do not cause the need for speed. Instead, the need for fast product development is primarily caused by internal considerations to validate the product idea as quickly as possible with minimal waste.

4.5.3 *Practices*

Start-ups report leveraging on best practices and established frameworks in creating software architectures (P11), thus avoiding the effort of inventing complex solutions themselves. Nearly all start-ups report using open source or free to use tools and components. Only few report use of commercial off-the-shelf components. Such a strategy reduces the effort of software engineering, potential lock-in effects with specific vendors, and need to reinvent already existing functionality. As stated by one respondent:

> *"We used a standard MVC architecture for the back-end and the web front-end, and the standard architecture for iOs/Android. The*

back-end and web front-end was implemented with Ruby on Rails and we followed its architecture conventions to guide the design."

Few start-ups in the maturity stage, 4 out of 15, 27%, report using in-house developed, innovative technologies. For example, highly customized deployment configurations, audio and video processing tools, and a custom graph database, to support their product development. Note that such innovations in technologies are done late in the start-up life-cycle to support further evolution of an already established product.

Graphical user interface design is interrelated with user experience design. User interfaces could be the primary touch point between users and the product, thus largely contributing to users perception of the product. In our study, we explore on how are the product user interface designs created without inferring a connection to a broader practice of user experience design.

Respondent responses on user interface (UI) design practices suggest that start-ups use mock-ups, utilize design frameworks and borrow ideas from other products to develop their product UI (P12). Three start-ups report outsourcing the UI design work. Companies reflect that user interface design is an ongoing process of continuous improvement. Thus, user interface design is not a significant concern at any particular stage. At the inception and stabilization stages, UI is tested by internal reviews and by observing users interacting with the UI. At later stages of the start-up life-cycle when a large number of customers are interacting with the product, start-ups use analytical tools and A/B tests to monitor how customers interact with the UI and to identify opportunities for improvement.

The main reported goals of the UI are usability, 75%, 63 cases, and usefulness, 35%, 29 cases. When reflecting on their user interface development practices, respondents suggest that investing more time in customer tests instead of inventing the UI internally, and having clear user interface guidelines, would have been helpful.

Our results on user interface design are aligned with studies investigating user experience practices in a start-up context. Start-ups use a set of practices to iterate prototypes, collect feedback and continuously improve their user interfaces. [Hokkanen and Väänänen-Vainio-Mattila, 2015b]. Quality criteria for good user interfaces and user experience are a clear message, visual attractiveness, intuitiveness, and credibility [Hokkanen et al., 2016].

4.5.4 *Lessons learned*

Our analysis contradicts earlier results that start-ups extensively use cutting-edge technologies [Giardino et al., 2014a]. Rather, start-ups opt for a stable technology platform with standard architecture to alleviate technology risks. We formulate the following implications for practitioners:

- Start-ups mitigate technology risks by selecting stable technologies and following best practices associated with the technologies. Stable technologies and best practices improve team performance and provide a rigid platform for innovative features.
  Further reading: Petersen et al. [2017b] explore how companies make decisions to select between open-source, in-house, or COTS components. Although the study is performed among established companies, it presents different strategies and factors influencing the decisions, thus providing an overview of the process.
  Baskerville et al. [2003] present a study exploring reactive engineering practices in start-up companies and argue that software engineering in start-ups mainly focuses on integration and interoperability of components developed elsewhere.
- Our findings suggest that user interface design is a significant activity. However, it must be done by continuously monitoring and tweaking the product.
  Suggested reading: Hokkanen et al. [2016] presents a framework for minimum viable user experience pinpointing essential qualities and practices of developing good user interfaces.

4.6 *Project management*

The project management process area concerns planning and control of engineering activities in a start-up. Planning and control are known as important to optimize resource usage and attainment of specific goals [Snyder, 2014]. We aim to explore how start-ups plan and control their activities.

4.6.1 *Goals*

The start-up life-cycle model, see Fig. 4.1, outlines the main milestones, releasing a minimum viable product, stabilizing the product for growth, attaining market share, and transitioning into an established organization. We inquired start-ups on how they control and measure their progress towards success.

The responses show that start-ups primarily use external metrics, such as revenue, number of customers, and customer satisfaction, to measure their success (G15, P13). However, start-ups at growth and maturity stages also consider using internal metrics, such as team performance, adherence to deadlines, and budget plans as performance measures (G16, P14).

We observe differences between start-ups in different life-cycle stages. Start-ups at the inception stage, before they have launched their product, do not report the use of any metrics to measure their progress. However, they aim to use external metrics, e.g., number product users, as soon as the product is launched. At the stabilization stage, just after the prod-

uct is launched, the primary success metrics are external and aimed to monitor general product adoption rates. As start-ups progress through the life-cycle, metrics become more specific, attached to high-level business milestones, and consider both internal and external aspects jointly. Internal team performance metrics are monitored and used to gauge start-up performance, in addition to external, market adoption metrics.

4.6.2 *Challenges*

Our results show that start-ups aim to use internal and external metrics to gauge their progression, see G15-16 in Fig. 4.4. However, external metrics are not available in the inception stage, before the product is launched.

Start-ups at the inception stage do not report using any specific metrics to control their progress. Potentially, due to unclear, changing plans, and immature project management practices. Thus, start-ups at the inception stage lack a benchmark to gauge their progression (C8). As stated by one practitioner:

> *"We work towards the goal to improve our platform, priorities changed, and but there was no control over schedule. Sometimes tasks took a lot longer than planned."*

In their reflections, practitioners mention a need for tighter control over product engineering work, deadlines and budget. As one practitioner reflected on lessons learned:

> *"First and foremost, I would have defined short term and long term goals of the start-up. That would have helped us to align all resources and activities."*

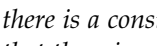
Results from the statistical analysis show that with the advantage of hindsight, practitioners estimate the performance of their teams more critically by a significant margin. Such findings suggest that it is challenging to assess performance from within a company objectively, and objective early performance indicators could be useful (C8). Objective performance indicators could be helpful to establish a performing team (C2).

Setting goals and metrics to measure the progress towards goals is a common practice in project management [Snyder, 2014, Shahin and Mahbod, 2007]. However, as shown by practitioner responses, start-ups at the inception stage do not measure their progression. Lack of measurement could lead to pitfalls such as the late realization of resource overruns, and scope creep.

Looking at the literature, we found several models aimed to guide early start-ups [Ries, 2011, Bosch et al., 2013, Blank, 2013a]. These models propose first to define and validate the customer need, then validate a solution

to this need, followed by validating product feasibility through small and large-scale prototypes. A somewhat similar approach focused on continuous experimentation is proposed by Fagerholm et al. [2017]. However, to what extent such practices are used to guide work in early start-ups remains to be explored.

Respondents estimate that their goals are rather clear and change rarely ( , "*To what extent it is true that goals in the start-up are rather unclear and change often?*"). As the source for uncertainty respondents report specific requirements from customers, demands from partners, and commercial results (C9). Such factors are out of control for start-ups. Thus, any plans should have a room for uncertainty and adjustments.

When inquired to what extent respondents experience time and financial resources shortages, most responses fall between "A little" and "Somewhat", indicating that availability financial resources and time is a concern, however not to an extreme extent, ( , "*To what extent it is true that there is a constant time pressure?*") and ( , "*To what extent it is true that there is a constant resources pressure?*")

Statistical analysis shows a linear correlation between time and resources shortages indicating that companies struggling with finances also struggle with time. However, the free-text answers does not contain any mention of specific difficulties stemming from time or financial resources shortages. The responses are consistent across the sample, and we could not identify any particular cohort where time and resource shortages would be more (or less, thereof) of a challenge.

Such results contradict earlier studies stating that extreme lack of resources is one of the key characteristics of start-ups [Giardino et al., 2014a, Sutton et al., 2000]. The trade-off between project scope and budget, and project management are reported as a concern in nearly every study exploring software project success factors, for example, Chow and Cao [2008], Junk [2000], Reel [1999], and Nasir and Sahibuddin [2011]. Thus, time and budget constraints, and associated trade-offs are not specific to start-ups exclusively. Moreover, a majority of software projects require more resources to complete than initially estimated [Molokken and Jorgensen, 2003b]. Therefore, resource shortages alone are not a differentiating factor between start-ups and established companies. That said, start-ups could differ from established organizations with thinner margins for resource overruns, less rigorous control over resource utilization, and could have more difficulty to access additional resources [Bocken, 2015].

A study of 1000 Finnish start-ups between 2010 – 2013 reveals that external funding has no association with start-up outcome. That is, start-ups with more resources at hand do not have more chances of survival and success than start-ups with more limited resources. Moreover, start-ups without external funding are likely to generate more revenue in the long term [Suominen et al., 2017]. Such findings highlight the importance of

scoping, planning and control to optimize utilization of any amount of resources.

We would like to challenge an earlier belief, see for example, Paternoster et al. [2014], Sutton et al. [2000], and Giardino et al. [2016], that time and resource shortages are characteristic to start-up companies and have a substantial effect on product engineering. Our analysis shows that resource shortages alone cannot be associated with engineering failures in start-ups. Rather, a lack of clear goals and absence of performance indicators lead to the depletion of resources with little contribution to product development.

4.6.3 *Practices*

The responses suggest that start-ups at the inception stage use elementary, if any, practices to schedule their work and keep control over time and budget, see C8. Some start-ups are constrained with a fixed budget or a hard deadline, however control over utilization of time and budget is based on "gut-feeling". As one respondent from a start-up at the inception stage reflects on their planning practices:

> *"We have something on the budget but it's very informal and it's solely based on projected product revenue, and user retention and acquisition. In hindsight, we should have done better in terms of scheduling and scoping the work."*

Following from our earlier discussion on control over resource utilization, lack of control over time and financial resources is a potential pitfall for early start-ups.

Start-ups at the stabilization stage report improving project management practices, such as setting boundaries for certain expense positions, assigning a budget to meet specific goals, and attempting to estimate their cash-flow (P15). Start-ups at maturity stage report using processes, supported by tools, for resource planning and control over resource utilization (P16). As one respondent reflects on their planning practices in a mature start-up:

> *"We initially created a budget plan to understand if our business is viable at all and to apply for a loan. Nowadays we use budget estimates as a tool for planning."*

4.6.4 *Lessons learned*

Our analysis shows that clear goals, control over resources and schedule are essential to track progress over start-up life-cycle. However, there is a challenge to objectively assess own performance before a product is launched to market. Lack of planning and control could contribute to budget and schedule overruns leading to wasted opportunities.

Lack of resources does not have a substantial effect on engineering practices and start-ups' prospects of advancing through the life-cycle. However, lack of planning and control over resource utilization, especially on early stages of a start-up is a pitfall. Start-ups need to set clear goals and metrics to assess their performance from the very beginning objectively.

Suggested reading: Dybå et al. [2014] present an overview of agile project management practices and provide practical tips on how to organize project management in uncertain and changing environments.

5 DISCUSSION

5.1 *Reflections on the research questions*

Our primary research question explores what engineering patterns, that is, common goals, practices, challenges, contextual factors, can be ascertained in start-up companies.

To identify the patterns, we use the start-up life-cycle model. The model supported clustering of cases by product life-cycle stage and outcome. Thus, observing how engineering focus changes over start-up life-cycle and enabling a discussion of what practices contribute to desirable state transitions. Furthermore, with the model and our results, we aim to provide a blueprint for studying start-ups as dynamic and multi-faceted entities. Such blueprint could enable compatibility of results from different studies, and contribute towards a coherent view of software engineering in start-ups.

In most process areas we observe evolving practices, from rudimentary at the early stages, to more mature in the latter stages. Such evolution regarding practice maturity is also noted by earlier studies, e.g., Giardino et al. [2016]. However, we present empirical results of specific practices, goals, challenges at each life-cycle stage, and discuss the results in the context of related work from similar contexts. This way we provide actionable support for practicing software engineering in start-ups.

Our results show that domain knowledge, technical expertise, and teamwork are the key components in the early stages of a start-up. Shortages of any of these components can be compensated with specific practices. For example, scarcity of domain knowledge in the team can be compensated with more rigorous requirements engineering practices helping to identify, acquire, and distribute information in the organization. Similarly, lack of technical expertise in the core team can be compensated by outsourcing engineering work to another team.

In the later stages, technical debt, growing team and lack of processes (e.g. quality assurance, project management, managing a large, distributed

team) hinder engineering work. However, these challenges can be anticipated and addressed with appropriate practices.

By looking at the patterns, we can identify the key concerns at each lifecycle stage.

At the inception stage, the most important concern is to assemble a small team of few individuals with sufficient domain knowledge, technical expertise and adequate financial resources to build the first release of a product. As our results show, different practices can compensate for shortcomings in domain knowledge (e.g., by more actively involving stakeholders and more rigorously documenting requirements), lack of technical expertise (e.g., by using external engineering team), and limited budget (e.g., by adjusting the scope of the release). Releasing the first version of a product depends on how efficiently a team can use their understanding of the domain and engineering expertise to produce software. Excessively large teams and difficult communication drains resources, time and motivation, and could lead to the closure of the company even before the launch of a product.

At the stabilization stage, we observe two main concerns. The first is related to team building and establishing an efficient team with defined areas of responsibility, trust, and coordination. After a start-up releases the first product version to customers, the company need to balance between developing new features and providing quality service for the existing customers. Aforementioned requires the team to be more diverse, handle a broader range of responsibilities and more stringent internal structure. The second concern is related to establishing a feedback loop with customers. Our results show an association between using customer feedback and technical and commercial success. Identifying early customers and involving them in the engineering work is one of the essential tasks in a start-up.

At the growth stage, we observe that concerns related to business, monetization, and marketing become relevant and have an influence on product engineering. For example, companies' own goals such as the need for monetization and expansion, must be taken into account in addition to customer needs. Serving a growing number of potentially diverse customers require the product to be flexible, scalable, and reliable. Providing such qualities require expert engineers. Growing complexity of the product creates a need for more robust testing and deployment practices, alleviating a need for manual work to prepare every release. Furthermore, increasing complexity of the organization and the product expose technical debt which must be addressed to support further growth.

At the maturity, stage start-ups are concerned with managing a growing, and potentially, distributed team. At this point, more rigorous project management practices are needed to organize and control engineering work, and more thorough engineering processes could be introduced marking a transition into an established organization.

6 CONCLUSION

In this paper, we investigated how 84 software start-ups utilize software engineering to build innovative software-intensive products. To frame the results we proposed start-up life-cycle model and looked into team, requirements engineering, value focus, quality and testing, architecture and design, and software project management process areas. This framing highlights stage and process area specific goals, challenges, and practices. We have discussed our findings in the context of related work and formulated practical lessons learned aimed at practitioners.

We conclude that all explored process areas are relevant for start-ups and, in essence, not different from established companies. That said, potentially the key difference and difficulty to practice software engineering in start-ups is to manage the evolution of practices in all the process areas at once with only a little room for error. Even though start-ups are experimental by nature and making errors in the process is inevitable, there is a distinction between taking calculated risks, and neglecting the best engineering practices.

To further understand and support software engineering in start-ups we aim to conduct a series of structured workshops to conduct an assessment of software engineering practices in start-ups using results obtained in this study. With the results of the workshops, we aim to refine further the patterns and a methodology for using the patterns in improving engineering practices in start-ups.

5

EXPLORATION OF TECHNICAL DEBT IN START-UPS

Software start-ups are young companies aiming to build and market software-intensive products fast with little resources. Aiming to accelerate time-to-market, start-ups often opt for ad-hoc engineering practices, make shortcuts in product engineering, and accumulate technical debt.

In this paper we explore to what extent precedents, dimensions and outcomes associated with technical debt are prevalent in start-ups.

We apply a case survey method to identify aspects of technical debt and contextual information characterizing the engineering context in start-ups.

By analyzing responses from 86 start-up cases we found that start-ups accumulate most technical debt in the testing dimension, despite attempts to automate testing. Furthermore, we found that start-up team size and experience is a leading precedent for accumulating technical debt: larger teams face more challenges in keeping the debt under control.

This study highlights the necessity to monitor levels of technical debt and to preemptively introduce practices to keep the debt under control. Adding more people to an already difficult to maintain product could amplify other precedents, such as resource shortages, communication issues and negatively affect decisions pertaining to the use of good engineering practices.

1 INTRODUCTION

Start-ups are important suppliers of innovation, new software products and services. However, engineering the software in start-ups is a complex endeavor as the start-up context poses unique challenges to software engineers [Giardino et al., 2015a]. As a result of these challenges, most start-ups do not survive the first few years of operation and cease to exist before delivering any value [Blank, 2013b, Giardino et al., 2014a].

Uncertainty, changing goals, limited human resources, extreme time and resource constraints are reported as characteristic to start-ups [Paternoster et al., 2014, Giardino et al., 2015a]. To cope with such forces, start-ups make a trade-off between internal product quality and faster time-to-market, fa-

voring the latter. As a consequence, start-ups accumulate technical debt [Giardino et al., 2016].

Technical debt is a metaphor to describe not quite right engineering solutions in a product that adds friction to its further development and maintenance. The extra effort associated with this friction, i.e. the "interest", needs to be paid every time a sub-optimal solution is touched [Kruchten et al., 2012]. Over time, the cumulative interest may exceed the effort needed to remove the debt, i.e. the "principal". The compound effects of sub-optimal solutions can reduce development team efficiency and overall quality. However, there is the belief among start-ups that any amount of technical debt can be written off if a feature or the whole product is not successful in the market [Giardino et al., 2016].

The strategy to accumulate technical debt can backfire if a start-up survives long enough and fails to put its technical debt under control. An unstable and difficult to maintain product adds risk to the company, for example, by limiting the ability to quickly enter into new markets (i.e. to pivot [Terho et al., 2015]) or to launch new innovative features [Tom et al., 2013, Klotins et al., 2017]. That said, we are not advocating for the removal of all technical debt. Rather, we are interested to see an overview of how technical debt influences start-ups and to enable start-up teams to make better decisions in regards to the trade-off between quality and time-to-market.

Technical debt has been extensively studied in the context of established companies and in relation to software maintenance [Li et al., 2015, Sjoberg et al., 2013]. For example, Tom et al. [2013] present a taxonomy comprising precedents, dimensions, and outcomes of technical debt. We adopt the terminology from this taxonomy to enable traceability.

Precedents are contextual factors in the development organization that contribute to the accumulation of technical debt, e.g. a lack of resources. Dimensions describe different types of technical debt, e.g. documentation, architecture, or testing debt. Outcomes refer to consequences of having excess technical debt, such as impaired productivity or quality [Tom et al., 2013].

While technical debt is a liability, development teams should manage it and use it as a leverage to attain otherwise unattainable goals [Tom et al., 2013]. In the start-up context, the concept of technical debt is explored only superficially. Giardino et al. [2016] argue that the need for speed, cutting edge technologies and uncertainty about a product's market potential are the main precedents for cutting corners in product engineering. However, if a start-up survives past its initial phases, management of technical debt becomes more and more important [Crowne, 2002, Giardino et al., 2016].

Our earlier study on software engineering anti-patterns in start-ups [Klotins et al., 2017] indicates that poorly managed technical debt could be one contributing factor to high start-up failure rates, driven by poor product qual-

ity and difficult maintenance. Negative effects of technical debt on team productivity has also been observed [Giardino et al., 2016].

In this study, we explore how start-ups estimate technical debt, what are precedents for accumulating technical debt, and to what extent start-ups experience outcomes associated with technical debt. We use a case survey as data source and apply a combination of quantitative and qualitative methods to explore technical debt in the surveyed companies. Our objective is to provide a fine-grained understanding of technical debt and its components that could provide a basis for defining start-up context-specific practices for technical debt management.

The main contribution of this paper is an empirical investigation that identifies the key precedents for the accumulation of technical debt in software start-ups, and the primary dimensions where the accumulation of debt has been observed by practitioners.

The rest of the paper is structured as follows. In Section 2 we introduce relevant concepts to understand our study. Section 3 presents the study design while results are presented in Section 4. The results are discussed and interpreted in Section 5. Section 6 concludes the paper.

2 Background and Related Work

2.1 *Software start-ups*

Software start-ups are small companies created for the purpose of developing and bringing an innovative product or service to market, and to benefit from economy of scale. Even though start-ups share many characteristics with small and medium enterprises, start-ups are different due to the combination of challenges they face [Sutton et al., 2000, Metzger and Pohl, 2014].

Start-ups are characterized by high risk, uncertainty, lack of resources, rapid evolution, immature teams, and time pressure among other factors. However, start-ups are flexible to adopt new engineering practices, and reactive to keep up with emerging technologies and markets [Giardino et al., 2014a, Sutton et al., 2000].

Start-up companies rely on external funding to support their endeavors. In 2015 alone, start-up companies have received investments of 429 billion USD in the US and Europe alone [PitchBook Data, 2015, PitchBook Data, Inc., 2015]. With an optimistic start-up failure rate of 75% that constitutes of 322 billion USD of capital potentially wasted on building unsuccessful products.

Earlier studies show that product engineering challenges and inadequacies in applied engineering practices could be linked to start-up failures [Giardino et al., 2015a, Klotins et al., 2017]. To what extent software

engineering practices are responsible or linked to success rate is very hard to judge. However, if improved software engineering practices could increase the likelihood of success by only a few percent, it would yield a significant impact on capital return.

2.2 *Technical debt*

Technical debt is a metaphor to describe the extra effort arising from maintaining and removing suboptimal or flawed solutions from a software product. Technical debt can be attributed to the software itself (e.g. source code), and also other artifacts and processes that comprise the product, and are relevant for maintenance and evolution of the product. For example, user manuals, knowledge distribution, operational processes, and infrastructure [Tom et al., 2013].

Suboptimal solutions find their way into software products due to a variety of reasons, such as ignorance of good engineering practices, oversight, lack of skills, or pragmatism [Alves et al., 2016]. Taking engineering shortcuts and delivering flawed solutions is often used as leverage to achieve faster time-to-market. However, the debt should be re-payed by removing flawed solutions from the product [Tom et al., 2013, Li et al., 2015].

When not addressed, suboptimal solutions make maintenance and evolution of software products difficult, any changes in the product require more effort than without the debt. This extra effort takes time away from developing new features and may overwhelm a team with firefighting tasks just to keep the product running, and decreases product quality altogether [Kruchten et al., 2012].

Giardino et al. [2016] argue that technical debt in start-ups accumulates from prioritizing development speed over quality, team aspects, and lack of resources. We combine results from their work, which is specific to start-ups, with a general taxonomy of technical debt by Tom et al. [2013]. We adopt the model of precedents, dimensions, and outcomes as proposed by Tom et al. [2013] and map it with the categories of the Greenfield start-up model [Giardino et al., 2016] to identify and to focus on relevant aspects of technical debt for start-ups, see Fig. 5.1.

As precedents, we study engineering skills and attitudes, communication issues, pragmatism, process, and resources. We explore technical debt in forms of code smells, software architecture, documentation, and testing. Furthermore, we attempt to understand to what extent team productivity and product quality is a challenge in start-ups. We use this conceptual model of technical debt as a basis to scope and define the research methodology, discussed next.

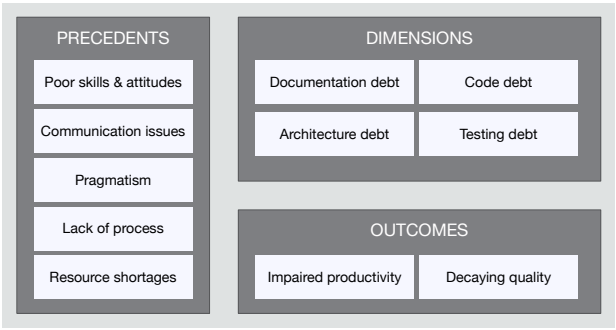


Figure 5.1: Aspects of technical debt

3 RESEARCH METHODOLOGY

3.1 *Research questions*

To achieve our goal and to drive the study we formulate the following research questions:

RQ1: How do start-ups estimate technical debt?
Rationale: Technical debt can incur in different forms, for example, as code smells, incomplete or outdated documentation, suboptimal software architecture, or shortcuts taken in testing [Li et al., 2015]. We aim to understand how start-ups estimate different types of technical debt, what types of technical debt are prevalent in start-ups and primary candidates for further investigation. In addition, what types of technical debt are least accumulated, i.e. are irrelevant or already well managed in the start-up context.

RQ2: What are precedents of technical debt in start-ups?
Rationale: Earlier studies report a number of precedents contributing to the accumulation of technical debt, such as prioritizing time-to-market over product quality and severe lack of resources [Giardino et al., 2016], developer skills and attitude, lack of process, oversight, and ignorance [Tom et al., 2013]. We aim to corroborate what precedents, identified by earlier studies in other contexts, are also present in start-ups.

RQ3: What outcomes linked to technical debt do start-ups report?
Rationale: Decreasing productivity, decaying morale, product quality issues, and increasing risks are reported as outcomes of technical debt [Tom et al., 2013, Giardino et al., 2016]. Yet, there is a belief that any amount of technical debt can be written off if a product or a specific feature does not succeed in market [Giardino et al., 2016]. We aim to corroborate what

outcomes, identified by earlier studies and linked to increased amounts of technical debt, do start-ups report.

3.2 *Data collection*

We used a case survey method to collect primary data from start-up companies [Petersen et al., 2017b, Larsson, 1993].

The case survey method is based on a questionnaire and is a compromise between a traditional case study and a regular survey [Klotins, 2017]. We have designed the questionnaire to collect practitioners experiences about specific start-up cases.

During the questionnaire design phase, we conducted multiple internal and external reviews to ensure that all questions are relevant, clear and that we receive meaningful answers. First, the questions were reviewed in multiple rounds by the first three authors of this paper to refine scope of the survey and question formulations. Then, with help of other researchers from the Software Start-up Research Network[1], we conducted a workshop to gain external input on the questionnaire. A total of 10 researchers participated and provided their input.

Finally, the questionnaire was piloted with four practitioners from different start-ups. During the pilots, respondents filled in the questionnaire while discussing questions, their answers and any issues with the first author of this paper.

As a result of these reviews, we improved the question formulations and removed some irrelevant questions. The finalized questionnaire[2] contains 85 questions in 10 sections. The questionnaire captures 285 variables from each start-up case.

From all the variables, 45 variables focus on capturing the magnitude of dimensions, precedents, and outcomes linked to technical debt[3]. The questions capture the respondents' agreement with a statement on a Likert scale: not at all (1), a little (2), somewhat (3), very much (4). The values indicate the degree of agreement with a statement. Statements are formulated consistently in a way that lower values indicate less precedents, less outcomes, and less technical debt.

In addition to questions pertaining technical debt, the questionnaire contains questions inquiring the engineering context in the start-up and applied software engineering practices.

The data collection took place between December 1, 2016, and June 15, 2017. The survey was promoted through personal contacts, by attending

1 The Software Start-up Research Network, https://softwarestartups.org/

2 http://startupcontextmap.org/exp-survey/woifenw2

3 The subset of questions used in this study is available here: http://eriksklotins.lv/uploads/TD-in-start-ups-questions.pdf

industry events, and by posts on social media websites. Moreover, we invited other researchers from the Software Start-up Research Network to collaborate on the data collection. This collaboration helped to spread the survey across many geographical locations in Europe, North and South America, and Asia.

3.3 *Data analysis*

To analyze the survey responses we used a number of techniques. We started by screening the data and filtering out duplicate cases, responses with few questions answered, or otherwise unusable responses. In the screening we attempt to be as inclusive as possible and do not remove any cases based on the provided responses.

The respondent estimates on technical debt aspects are measured on an ordinal scale, measured from 1 (not at all) to 4 (very much). Respondent and start-up demographics such as age and years of operation are measured with categorical variables on a nominal scale.

Overall, we analyze responses from 86 start-up cases, 75 data-points per each case, and 6450 data-points overall. To gain an overview of the data, the results were visualized by histograms, box-plots and contingency tables [Haberman, 1973].

We use the Chi-Squared test of association to test if the associations between the examined variables are not due to chance. To prevent Type I errors, we used exact tests, specifically, the Monte-Carlo test of statistical significance based on 10 000 sampled tables and assuming ($p = 0.05$) [Hope, 1968].

To examine the strength of associations we use Cramer's V test. We interpret the test results as suggested by Cohan [1988], see Table 5.1. To explore specifics of the association, such as which cases are responsible for this association, we perform post-hoc testing using adjusted residuals. We consider an adjusted residual significant if the absolute value is above 1.96 ($Adj.residual > 1.96$), as suggested by Agresti [1996]. The adjusted residuals drive our analysis on how different groups of start-ups estimate aspects of technical debt. However, due to the exploratory nature of our study, we do not state any hypotheses upfront and drive our analysis with the research questions.

Full results, contingency tables, histograms and calculation details are accessible on-line[4] for a full disclosure.

4 http://eriksklotins.lv/uploads/TD-in-start-ups-sm.pdf

Table 5.1: Interpretation of Cramer's V test

Cramer's V value	Interpretation
≥ 0.1	Weak association
≥ 0.3	Moderate association
≥ 0.5	Strong association

3.4 *Validity threats*

In this section we follow guidelines by Runeson et al. [2012] and discuss four types of validity threats and applied countermeasures in the context of our study.

3.4.1 *Construct validity*

Construct validity concerns whether operational measures really represent the studied subject [Runeson et al., 2012]. A potential threat is that the statements we use to capture respondent estimates are not actually capturing the studied aspects of technical debt.

To address this threat we organized a series of workshops with other researchers and potential respondents to ensure that questions are clear, to the point, and capture the studied phenomenon.

Each aspect, i.e. type of precedent, is triangulated by capturing it by at least three different questions in the questionnaire. To avoid biases stemming from respondents opinions about technical debt and to capture the actual situation we avoid mentioning technical debt in the questions. Instead, we formulate the questions indirectly to capture respondent estimates on different aspects associated with technical debt. For example, we ask whether they find it difficult to understand requirements documentation.

To accommodate for the fact that a respondent may not know answers to some of the questions, we provide an explicit "I do not know" answer option to all Likert scale questions.

3.4.2 *Internal validity*

This type of validity threat addresses causal relationships in the study design [Runeson et al., 2012]. In our study we use a model of precedents, dimensions and outcomes of technical debt. The literature, for example, Tom et al. [2013] and Li et al. [2015], suggest that there is a causality between the three. We, however, present respondent estimates on precedents,

dimensions and the outcomes separately without considering or implying any causality.

3.4.3 *External validity*

This type of validity threat concerns to what extent the results could be valid to start-ups outside the study [Runeson et al., 2012]. The study setting for participants was close to real life as possible, that is, the questionnaire was filled in without researcher intervention and in the participants own environment.

A sampling of participants is a concern to external validity. We use convenience sampling to recruit respondents and with help of other researchers, distributed the survey across a number of different start-up communities. Demographic information from respondent answers shows that our sample is skewed towards active companies, respondents with little experience in start-ups, young companies and small development teams of 1-8 engineers. In these aspects our sample fits the general characteristics of start-ups, see for example, Giardino et al. [2014a], Giardino et al. [2015a], and Klotins et al. [2016]. However, there clearly is a survivorship bias, that is, failed start-ups are underrepresented, thus our results reflect state-of-practice in active start-ups.

Another threat to external validity stems from case selection. The questionnaire was marketed to start-ups building software-intensive products, however due to the broad definition of software start-ups (see Giardino et al. [2014a]), it is difficult to differentiate between start-ups and small medium enterprises. We opted to be as inclusive as possible and to discuss relevant demographic information along with our findings.

3.4.4 *Conclusion validity*

This type of validity threat concerns the possibility of incorrect interpretations arising from flaws in, for example, instrumentation, respondent and researcher personal biases, and external influences [Runeson et al., 2012].

To make sure that respondents interpret the questions in the intended way we conducted a number of pilots, workshops and improved the questionnaire afterwards. To minimize the risk of systematic errors, the calculations and statistical analysis was performed by the first and the third author independently, and findings were discussed among the authors.

To strengthen reliability and repeatability of our study, all survey materials and calculations with immediate results are published online.

To answer our research questions we analyze 6450 data-points from 86 start-up cases. The majority of these start-ups (63 out of 86, 73%) are active and had been operating for 1-5 years (58 out of 86, 67%), see Fig. 5.2. Start-ups are geographically distributed among Europe (34 out of 86, 40%), South America (41 out of 86, 47%), Asia (7 out of 86) and North America (2 out of 86).

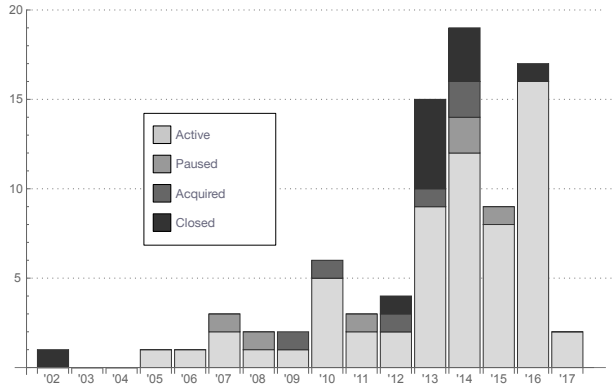


Figure 5.2: Distribution of start-ups by the founding year and their current state

Our sample is about equally distributed in terms of the product development phase. We follow a start-up life-cycle model proposed by Crowne [2002] and distinguish between inception, stabilization, growth and maturity phases. In our sample, 16 start-ups have been working on a product but haven't yet released it to market, 24 teams had released the first version and actively develop it further with customer input, 26 start-ups have a stable product and they focus on gaining customer base, and another 16 start-ups have mature products and they focus on developing variations of their products. The distribution of start-ups by their life-cycle phase and length of operation is shown in Fig. 5.3. In the figure, bubble size denotes the a number of people in the team. Most start-ups in the sample (75 out of 86, 87%) have small teams of 1-8 engineers actively working on the product.

About an equal number of start-ups had indicated that they work on more than one product at the time. Start-ups in our sample do per-customer customization to some extent: 10 companies (11%) had specified that they tailor each product instance to a specific customer, 30 companies (35%) do not do per-customer customization at all, while 43 start-ups (49%) occasionally perform product customization for an individual customer.

The questionnaire was filled in mostly by start-up founders (64 out of 86, 74%) and engineers employed by start-ups (15 out of 86, 17%). About

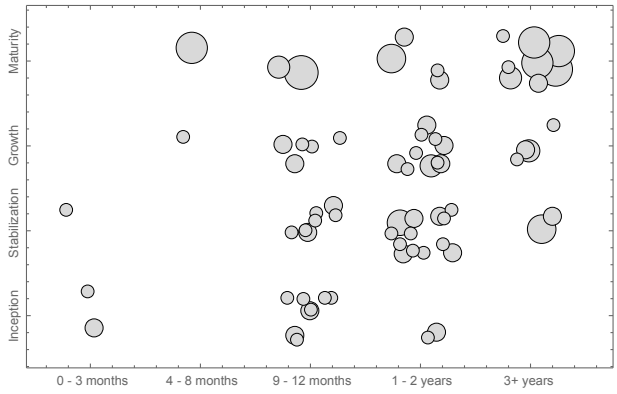


Figure 5.3: Distribution of start-ups by product phase and length of operation

a half of respondents have specified that their area of expertise is software engineering (49 out of 86, 56%). Others have specified marketing, their respective domain, and business development as their areas of expertise.

Respondents length of software engineering experience ranges from 6 months to more than 10 years. A large portion of respondents (44 out of 86, 51%) had less than 6 months of experience in working with start-ups at the time when they joined their current start-up.

4.1 *Dimensions of technical debt*

We start our exploration by looking at the extent to which the dimensions of technical debt (documentation, architecture, code, and testing) are present in the surveyed start-ups. We quantify the degree of technical debt by aggregating respondent answers on questions pertaining to each dimension, Answers were given on a Likert scale where higher values indicate more estimated technical debt in a given dimension.

Responses from the whole sample indicate that start-ups estimate some technical debt (2 on a scale from 1 to 4) in documentation, architecture, and code dimensions, while testing debt is estimated as the most prevalent (3 in a scale from 1 to 4). Fig. 5.4 shows the median (dark horizontal line), first and third quartile, and minimum and maximum estimates on all statements pertaining to a specific debt type.

To explore the estimated degree of technical debt further, we analyze the influence of respondent demographics, such as relationship with the start-up and background, and start-up demographics, such as product life-cycle phase, team skill level and longevity of the start-up, on the responses.

The analysis shows that only start-up state, that is if the start-up is active, paused, acquired, or closed, has an effect on the overall estimates of

Table 5.2: Results of Cramer's V test on associations between dimensions and start-up characteristics with $p < 0.05$

#	Characteristic	Documentation	Architecture	Code	Testing	All
1	State of the start-up	0.346	0.326	0.414	-	0.318
2	Product phase	-	0.329	-	-	-
3	Overall team size	-	-	0.427	-	-
4	Level of domain knowledge	0.334	-	-	-	-
5	Per-customer tailoring	-	-	0.423	-	-
6	Practical experience	0.337	-	-	-	-

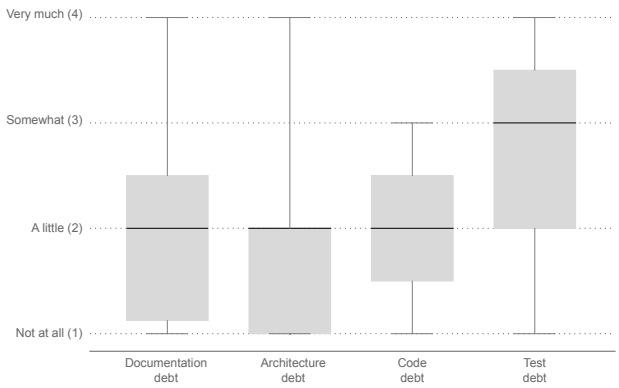


Figure 5.4: Estimates for the prevalence of different dimensions of technical debt from the case survey

technical debt, see Table 5.2. In the table we show strength (measured by Cramer's V test) of statistically significant associations ($p < 0.05$, measured by Chi-Square test) between relevant characteristics of start-ups and technical debt dimensions. In the last column we show if the characteristic has an effect on all dimensions together.

Observe that the level of engineering skills and domain knowledge pertains to the whole team. However, practical experience pertains only to the respondent. We show respondent characteristics as well to illustrate to what extent respondents background influences their responses. For instance, respondents with more practical experience estimate documentation debt more critically, see Table 5.2, and are more critical about skills shortages, see Table 5.3.

We highlight important findings in framed boxes and discuss them in Section 5.

Finding 1: Start-ups that are in the active category estimate technical debt, overall in all dimensions, lower than closed or acquired start-ups.

Product phase, team size and level of domain knowledge have effects on individual technical debt dimensions. We present these results next.

4.1.1 *Documentation debt*

Documentation debt refers to any shortcoming in documenting aspects of software development, such as architecture, requirements, and test cases [Li et al., 2015].

We look into requirements, architecture and test documentation because these are the essential artifacts guiding a software project. Requirements

capture stakeholders needs and provide a joint understanding of what features are expected from the software. Architecture documentation lists design principles, patterns and components comprising the software. Documentation of test cases supports testing activities and provides means for quality assurance [Selic, 2009].

Only 1% (7 out of 84) of start-ups in our sample have explicitly stated that they do not document requirements in any way. The most popular forms of documenting requirements are informal notes and drawings (50 out of 86, 58%), followed by organized lists (20 out of 86, 20%).

Responses from the whole sample show that start-ups have some amount of documentation debt (Median = 2.0), see Fig. 5.4. Exploring what start-up characteristics have an effect on the estimates, see Table 5.2, we found that start-ups who are active estimate documentation debt lower than acquired or closed companies. We also found that teams with sufficient domain knowledge estimate documentation debt lower than teams with many gaps in their domain knowledge.

4.1.2 *Architecture debt*

Architecture debt refers to compromises in internal qualities of the software such as maintainability, scalability, and evolvability [Li et al., 2015].

Results from the whole sample show that start-ups experience some architectural debt (Median = 2.0), see Fig. 5.4. By looking into what start-up characteristics have an effect on how respondents estimate architecture debt, we found that state of the start-up and the product phase have an effect on the estimates, see Table 5.2. Active start-ups have provided substantially lower estimates than acquired and closed companies. We found that start-ups who have just started on product engineering and haven't yet released it to market, experience almost no architectural debt. During stabilization and growth phases the estimates become more critical. However, during the maturity phase estimates become slightly more optimistic, see Fig. 5.5. Box-plots in the figure show responses from all statements pertaining to architecture debt.

4.1.3 *Code debt*

Code debt refers to a poorly written code. Signs of a poorly written code are, for example, unnecessary complex logic, code clones, and bad coding style affecting code readability. Poorly written code is difficult to understand and change [Mantyla et al., 2003, Tom et al., 2013, Palomba et al., 2014a].

Results from the whole sample show that start-ups experience some code debt (Median = 2.0), see Fig. 5.4. By looking into what start-up characteristics have an effect on how respondents estimate code debt we found

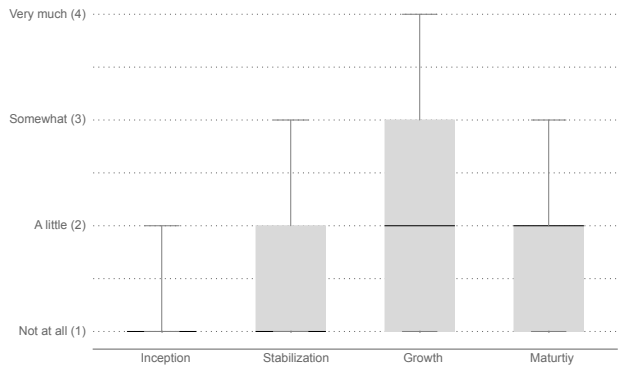


Figure 5.5: Box-plot showing how in different product phases start-ups estimate architecture debt

that state of the start-up, team size, level of per-customer tailoring has an effect on the estimates, see Table 5.2. Active start-ups estimate code debt lower than acquired start-ups. Start-ups with larger teams (9 or more people), provide higher estimates on code debt than small teams. Start-ups who do not offer per-customer customization estimate code debt lower. However, start-ups that occasionally tailor their product to needs of a specific customer estimates their code debt higher.

4.1.4 *Testing debt*

Testing debt refers to lack of test automation leading to the need of manually retesting the software before a release. The effort of manual regression testing grows exponentially with the number of features, slowing down release cycles and making defect detection a time consuming and tedious task [Tom et al., 2013].

Answers to questions inquiring use of automated testing show that about a third (26 out of 86, 30%) of start-ups are attempting to implement automated testing, and only 17 start-ups (20%) have explicitly stated that no test automation is used.

Despite attempts to automate, companies across the whole sample estimate their testing debt somewhat high ($\mathtt{Median} = 3$), see Fig. 5.4. Manual exploratory testing is reported as the primary testing practice, regardless of start-up life-cycle phase, team size and engineering experience, and length of operation.

Finding 2: Despite attempts to automate, manual testing is still the primary practice to ensure that the product is defect free.

Similar results, showing that only a small number of mobile application projects have any significant code coverage by automated tests, and listing time constraints as the top challenge for adopting automated testing, were obtained by Kochhar et al. [2015].

4.2 *Precedents for technical debt*

We asked the respondents to estimate various precedents of technical debt in their start-ups, such as attitudes towards good software engineering practices, pragmatic decisions to make shortcuts in product engineering, communication issues in the team, level of team engineering skills, time and resource shortages, and lack of established SE processes.

Box-plots with median responses from the whole sample are shown in Fig. 5.6. Higher values indicate stronger agreement with the presence of a precedent in the start-up. Poor attitude is the least common precedent for technical debt, while resource shortage is estimated as the most prevalent precedent.

Looking into what start-up characteristics influence the responses, we find that start-up team size and team's engineering skills have a significant effect on the estimates overall, see Table 5.3. Larger teams of 9 or more people estimate the precedents for technical debt higher than small teams.

In the results we show only characteristics with statistically relevant associations, thus listed characteristics differ between Tables 5.2 and 5.3.

Finding 3: Start-up team size and level of engineering skills have a significant effect on how severe the other precedents are estimated.

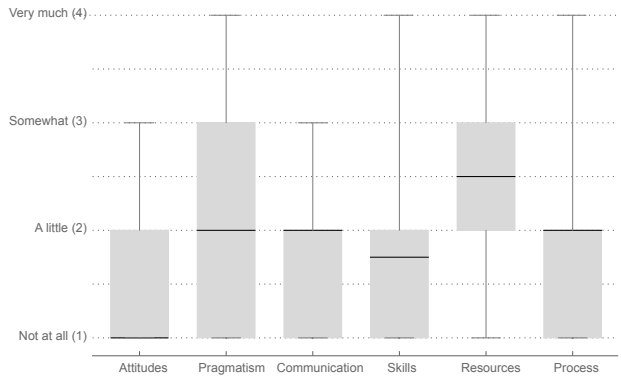


Figure 5.6: Box-plot showing how the sample estimates different precedents for technical debt

Table 5.3: Results of Cramer's V test on associations between precedents and start-up characteristics with $p < 0.05$

#	Characteristic	Attitudes	Pragmatism	Communication issues	Skills shortages	Resources shortages	Process	All
1	State of the start-up	0.339	-	-	0.285	-	-	-
2	Per-customer tailoring	-	-	-	-	0.345	-	-
3	Overall team size	-	-	-	0.360	0.344	0.375	0.386
4	Development team size	-	0.432	0.387	0.429	-	0.403	-
5	Level of engineering skills	-	-	0.331	0.465	-	-	0.377
6	Level of domain knowledge	-	-	-	-	0.324	-	-
7	Practical experience	-	-	-	0.329	-	-	-

4.2.1 *Attitude towards good engineering practices*

The responses to questions about following good engineering practices suggest that start-up engineers do realize the importance of following good architecture, coding and testing practices (Median = 1), see Fig. 5.6.

Comparing how responses on attitude differ by start-up characteristics we found that start-ups who are active, estimate their attitudes more optimistically than acquired or closed companies. That is, they agree more with benefits from following good engineering practices, such as coding conventions and throughout testing of the product.

4.2.2 *Pragmatism*

Estimates on statements about pragmatic considerations, that is, prioritization of time-to-market over good engineering practices, show that start-ups are ready to make shortcuts to speed up time-to-market (Median = 2). However, the spread of estimates suggests that different companies have very different attitudes towards deliberately introducing technical debt, see Fig. 5.6. Comparing how estimates on attitude differ by start-up characteristics we found that start-ups with larger teams of 15 or more developers estimate pragmatic precedents higher than smaller teams.

4.2.3 *Communication*

Estimates on statements about communication show that communication issues could be one of the precedents for introducing technical debt, see Fig. 5.6. We observe that communication issues become significantly more severe in larger engineering teams of 13 and more people than in smaller teams. Moreover, the results suggest that teams with better engineering skills experience fewer communication issues.

4.2.4 *Engineering skills*

Estimates to what extent start-ups face lack of engineering skills show that skills shortage could be a precedent for accumulating technical debt, see Fig. 5.6.

Comparing how estimates on skill shortages differ by start-up characteristics we found that the state of the start-up, team size, length of practical experience, and level of estimated engineering skills have the significant influence on the estimates. We found that active start-ups estimate skills shortages lower than closed down companies. A somewhat expected result is that teams with adequate engineering skills provide significantly lower estimates for challenges associated with skills shortages.

4.2.5 *Resources*

Looking at differences between estimates on time and other types of resources we found that they are tied together, see Fig. 5.6. That is, companies reporting time shortages also report resource shortages. A potential explanation for the association is that time pressure is created internally by a need to get the product out and start generating revenue, and not by an external market pressure. We also found that per-customer customization, overall team size, and level of domain knowledge has an effect on how start-ups estimate resource shortages.

Estimates of resources and time shortages show that resource issues are the highest estimated precedent for technical debt ($Median = 2.5$). We find that occasional per-customer tailoring is associated with higher estimates on resource shortages. Potentially, start-ups suffering from lack of resources opt for occasional customization to serve needs of an important customer, thus acquiring resources for further development. Start-ups with smaller teams of 1-3 people estimate resource shortages lower than larger start-ups of 9-12 people. A plausible explanation for this association could be that supporting a larger team requires more resources.

4.2.6 *Process*

Respondent estimates on the software engineering process issues show that frequent and unplanned changes occur and could cause difficulties in avoiding technical debt, see Fig. 5.6. We found that estimates on process issues become more severe as team size grows.

4.3 *Outcomes of technical debt*

To explore potential outcomes of technical debt we presented respondents with statements exploring to what extent team productivity and product quality are concerns in their start-ups. Estimates from the whole sample show that start-ups experience some quality and productivity issues ($Median = 2$) that could be associated with accumulated technical debt. We found that team size is the only characteristic that influences the estimates ($Cramer's V = 0.362$).

Looking into what types of technical debt are associated with specific outcomes linked to technical debt, we found a clear association between estimates of technical debt and estimates of the outcomes, see Table 5.4.

Code debt has the most severe impact on both productivity and quality. Architecture debt have a similar effect, albeit to a lesser extent. Documentation debt impairs productivity. However, we did not find a statistically significant association between testing debt and loss of productivity or quality.

Table 5.4: Results of Cramer's V test on associations between types of technical debt and outcomes with $p < 0.05$

#	Debt type	Impact on:		
		Quality	Productivity	Both
1	Documentation	-	0.332	0.344
2	Architecture	0.399	0.331	0.440
3	Code	0.445	0.471	0.532
4	Testing	-	-	-
	All types	0.363	0.459	0.463

Finding 4: We found that from all types of technical debt, code debt have the strongest association with productivity and quality issues.

5 DISCUSSION

5.1 *Reflections on the research questions*

Our results on how start-ups estimate technical debt show that active start-ups estimate aspects of technical debt significantly lower than closed or acquired start-ups, see Finding 1 in Section 4.1. A plausible explanation for this result could be that lower technical debt helps start-ups to have a more stable and easier to maintain product. Thus giving a start-up more room for evolving the product into something the market wants, i.e. to pivot [Bajwa et al., 2016]. However, excess technical debt hinders product evolution and could be one of contributing factors to the shutdown of a company.

An alternative explanation is that technical debt could be invisible and compensated by the team's implicit knowledge. However, when a start-up is acquired by another company and the product is transferred to another team, all the technical debt becomes visible. Difficulties to capture undocumented knowledge and the associated drop in performance of the receiving team has been recognized in the context of agile project handover [Stettina and Kroon, 2013].

Results on how different types of technical debt are estimated show that the most technical debt is estimated in the testing category, even though start-ups do attempt to automate tests, see Finding 2 in Section 4.1. A

potential explanation could be that start-ups lack certain prerequisites for full implementation of automated testing [Voas and Kassab, 1999]. Excess technical debt in other categories, for instance, difficult to test code, lack of requirements documentation, and an unclear return of investment, could be hindering the implementation of test automation, as it is also observed in traditional, more mature companies [Rice et al., 2003, Joorabchi et al., 2013, Kochhar et al., 2015]. However, we could not find any statistically significant association between testing debt and quality or productivity issues.

The comparison of results on architecture debt from start-ups in different life-cycle phases shows another interesting pattern. Start-ups who have not yet released their products to market experience very little architecture debt, the debt increases as the product is delivered to the first customer and peaks at the growth stage when start-ups focus on marketing the product and drops slightly as start-ups mature, see Fig 5.5. Marketing of the product could be a source of new challenges for the product development team. For example, the product must support different configurations for different customer segments, provide a level of service, and cope with a flow of requests for unanticipated features [Crowne, 2002, Dahlstedt et al., 2003]. Earlier shortcuts in product architecture are therefore exposed and must be addressed.

Overall team size and level of engineering skills could be the most important characteristics contributing to precedents and linked to technical debt in most dimensions, see Finding 3 in Section 4.2. Larger teams of 9 or more people experience more challenges and report higher technical debt in all categories. This finding is similar to De Melo et al. [2013] studying productivity in agile teams. Smaller teams are better aligned and more efficient in collaboration with little overhead. However, as the team size grows more processes and artifacts for coordination are needed [Staats et al., 2012]. Therefore larger teams have more artifacts that can degrade.

Team size could be an indicator of the general complexity of a start-up and the product. More people are added to the team when there are more things to be taken care of. Therefore, the technical debt could stem not only from the number of people but also from increasing complexity of the organization itself.

Our results show that increase in team size is also associated with outcomes of technical debt, a decrease in productivity and product quality, see Finding 4 in Section 4.3. This result could be explained by our earlier discussion on how larger teams require more coordination for collaboration. However, the more critical estimates by larger teams could be also associated with the increase in product complexity as new features are added. Rushing to release new features could contribute to the accumulation of technical debt until deliberate, corrective actions are taken, as observed in mobile application development [Hecht et al., 2015].

As a software product grows, it naturally becomes more difficult to maintain. For instance, if individual product components do not change and the new components are at the same quality level as existing ones, the increased number of components and their dependencies requires more effort from engineers to maintain the product and creates more room for defects [Izurieta and Bieman, 2013]. This is software decay and is not the same as avoidable technical debt stemming from the trade-off between quality and speed. Distinguishing between true technical debt and software decay is an important next step in providing practical support for software-intensive product engineering in start-ups.

5.2 *Implications for practitioners*

This study presents several implications for practitioners:

1. Start-up teams with higher level of engineering skills and respondents with more experience perceive aspects of technical debt more severely. Less skilled teams may not be aware of their practices introducing additional technical debt, and amount of technical debt in their products. Using tools and occasional external expert help could help to identify unrealized technical debt, and to improve any suboptimal practices.
2. Start-up team size correlates with more severe precedents and outcomes of technical debt. Keeping a team small and skilled could be a strategy to mitigate precedents for technical debt. To support growth of the team, more coordination practices need to be introduced, and impact on technical debt monitored. Additional coordination practices require more maintenance of coordination artifacts. Thus, there is a practical limit how large a team can grow before it needs to be divided into sub-teams.
3. There is an association between levels of technical debt and a start-up outcome. Having less technical debt could give a start-up more room for pivoting and product evolution in the long term.
4. There are certain moments when the effects of technical debt are the most severe. For example, shipping a product to a large number of customers, scaling up the team, and handing the product over to another team. The anticipation of such moments and adequate preparations could help to mitigate the negative effects of technical debt.
5. The most significant type of technical debt in start-ups is code smells. We found that poorly structured and documented code has the strongest association with issues in team productivity and product quality. However, detection of code smells can be automated with open-source tools, thus alleviating removal of this type of debt.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we report how technical debt is estimated in start-ups building software-intensive products. We explore to what extent precedents, dimensions, and outcomes, identified by earlier studies, are relevant in the start-up context. We attempt to identify what start-up characteristics have an amplifying or remedying effect on technical debt.

Our results show that, even though start-up engineers realize the importance of good engineering practices, they cut corners in product engineering, mostly due to resource pressure and a need for faster time to market. The results suggest that precedents for technical debt become more severe as start-ups evolve and severity of the precedents could be associated with the number of people working in a start-up and a product life-cycle phase.

Our results show significantly different estimates from closed, acquired and operational start-ups. The differences highlight how start-ups use technical debt as a leverage, and emphasizes the importance of careful technical debt management.

This exploratory study leads to a formulation of several hypotheses:

(a) Technical debt peaks at the growth stage when a start-up attempts to market the product.
(b) The number of people in a team amplifies precedents for technical debt.
(c) There is an association between a start-up outcome and their technical debt management strategy.

We aim to explore these hypotheses further by triangulating results from this study with qualitative data from interviews and artifact analysis.

6

USE OF AGILE PRACTICES IN START-UPS

Software start-ups have shown their ability to develop and launch innovative software products and services. Small, motivated teams and uncertain project scope makes start-ups good candidates for adopting Agile practices. We explore how start-ups use Agile practices and what effects can be associated with the use of those practices.

We use a case survey to analyze 84 start-up cases and 56 Agile practices. We apply statistical methods to test for statistically significant associations between the use of Agile practices, team, and product factors.

Our results suggest that backlog, version control, refactoring, and user stories are the most frequently reported practices. We identify 22 associations between the use of Agile practices, team, and product factors. The use of Agile practices is associated with effects on source code and overall product quality. A teams' positive or negative attitude towards best engineering practices is a significant indicator for either adoption or rejection of certain Agile practices. To explore the relationships in our findings, we set forth a number of propositions that can be investigated by future research.

We conclude that start-ups use Agile practices, however without following any specific methodology. We identify the opportunity for more fine-grained studies into the adoption and effects of individual Agile practices. Start-up practitioners could benefit from Agile practices in terms of better overall quality, tighter control over team performance and resource utilization.

1 INTRODUCTION

Start-ups are important suppliers of innovation, new software products, and services. However, engineering the software in start-ups is a complicated endeavor as the start-up context poses challenges to software engineers [Giardino et al., 2015a]. As a result of these challenges, most start-ups do not survive the first few years of operation and cease to exist before delivering any value [Blank, 2013b, Giardino et al., 2014a].

Uncertainty, changing goals, limited human resources, extreme time and resource constraints are reported as characteristic to start-ups [Paternoster et al., 2014, Giardino et al., 2015a].

To survive in such a context, start-ups use ad-hoc engineering practices and attempt to tailor agile methods to their needs. However, scaled-down agile methods could be irrelevant and ignore start-up specific challenges [Klotins et al., 2019b, Yau and Murphy, 2013].

Giardino et al. [2016] suggest that start-ups adopt practices as a response to some problematic situations and do not consider adopting full agile methodologies, e.g., scrum or XP, at least in early stages.

Pantiuchina et al. [2017] make a similar observation and argue that start-ups focus more on speed-related practices, e.g., iterations and frequent releases, than quality-related practices, e.g., unit testing and refactoring.

In this study, we explore the use of Agile practices in start-ups. We focus on identifying the associations between certain Agile practices, product, and team factors. We aim to understand what positive, and potentially adverse effects can be associated with the use of specific practices. We use our results to formulate propositions for further exploration.

We use a case survey to collect data from 84 start-up cases [Klotins et al., 2019a]. We use statistical methods to analyze 11,088 data points and identify associations between the use of Agile practices and respondents' estimates on various team and product factors.

We identify 20 statistically significant associations pointing towards potential causes and effects of using Agile practices. We identify that the use of automated tests and continuous integration is associated with positive attitudes towards following best practice. However, the use of planning and control practices are more associated with negative attitudes towards following the best practices.

The rest of this paper is structured as follows. In Section 2, we discuss related work. Section 3 covers the research methodology, data collection, and our approach to data analysis. Section 4 presents the results. We answer our research questions and discuss implications for research and practice in Section 5. Section 6 concludes the paper.

2 Background and Related Work

2.1 *Software Start-ups*

Software start-ups are small companies created for the purpose of developing and bringing an innovative product or service to market, and to benefit from economies of scale.

Start-up companies rely on external funding to support their endeavors. In 2015 alone, start-up companies have received investments of 429 billion

USD in the US and Europe alone [PitchBook Data, 2015, PitchBook Data, Inc., 2015]. With an optimistic start-up failure rate of 75% that constitutes of 322 billion USD of capital potentially wasted on building unsuccessful products.

Earlier studies show that product engineering challenges and inadequacies in applied engineering practices could be linked to start-up failures [Giardino et al., 2015a, Klotins et al., 2017]. To what extent software engineering practices are responsible or linked to success rate is very hard to judge. However, if improved software engineering practices could increase the likelihood of success by only a few percent, it would yield a significant impact on capital return.

Some authors, e.g. Sutton et al. [2000] and Giardino et al. [2014a], point out the unique challenges in start-ups, such as high risk, uncertainty, lack of resources, rapid evolution, immature teams, and time pressure among other factors. At the same time, start-ups are flexible to adopt new engineering practices, and reactive to keep up with emerging technologies and markets [Giardino et al., 2016]. However, our earlier study [Klotins, 2018] analyzing the amount of empirical evidence supporting the uniqueness of start-ups found that most start-up characteristics are based on anecdotal evidence. Thus, there could be negligible difference between start-ups and other organizations launching new products to market in terms of software engineering.

2.2 *Agile practices*

Agile software engineering practices originate from the Agile manifesto proposing a shift from heavyweight, plan-driven engineering towards more lightweight, customer-oriented, and flexible methodologies [Beck et al., 2001]. Agile methodologies, such as Scrum, XP, and Kanban, prescribe specific sets of Agile practices [Rising and Janoff, 2000, Jyothi and Rao, 2012]. However, in practice, by-the-book methodologies are often tailored with additional practices to address specific concerns [Diebold and Dahlem, 2014, Jalali and Wohlin, 2012]. Thus, we focus our study on what practices start-ups use, without considering any specific agile methodology.

Small organizations have successfully adopted Agile practices for projects where requirements are uncertain and expected to change [Chow and Cao, 2008, Misra et al., 2012]. In theory, Agile practices could be perfect for software start-ups [Yau and Murphy, 2013]. However, successful adoption of Agile practices requires highly skilled teams and support throughout an organization [Solinski and Petersen, 2016, Chow and Cao, 2008].

Earlier work on engineering practices in start-ups suggests that start-ups initially rely on an ad-hoc approach to engineering and adopt agile

principles incrementally when a need for more systematic practices arises. The shift is often motivated by excessive technical debt hindering quality and lack of control over the engineering process [Giardino et al., 2016].

We explore associations between 56 Agile practices, product, and team factors. We use a list and descriptions of Agile practices compiled by Agile Alliance, a non-profit community promoting agile principles [Agile Alliance, 2018]. To our best knowledge, their website contains the most comprehensive list of Agile practices to date.

In this study we consider the following practices whose definitions can be found at the Agile Alliance's website [Agile Alliance, 2018]: Card, Conversation, Confirmation (3C's), Acceptance tests, Acceptance Test-Driven Development (ATDD), Automated build, Backlog, Backlog grooming, Behavior Driven Development, Burndown chart, Collective ownership, Continuous deployment, Continuous integration, Class Responsibility Collaborator (CRC) Cards cards, Daily meeting, Definition of Done, Definition of Ready, Exploratory testing, Facilitation, Frequent releases, Given-When-Then, Heartbeat retrospective, Incremental development, INVEST, Iterations, Iterative development, Kanban board, Lead time, Mock objects, Niko-Niko, Pair Programming, Personas, Planning poker, Point estimates, Project charters, Quick design session, Refactoring, Relative estimation, Role-Feature-Reason, Rules of simplicity, Scrum of Scrums, Sign up for tasks, Simple design, Story mapping, Story splitting, Sustainable Pace, Task board, Team, Team room, Test-driven development, Three Questions, Timebox, Ubiquitous language, Unit tests, Usability testing, User stories, Velocity, and Version control.

2.3 *Effects of using Agile practices*

The use of Agile practices is associated with increased product quality and fewer defects compared to plan-driven approaches [Layman et al., 2004, Ilieva et al., 2004]. We analyze associations between use of the Agile practices, product documentation, software architecture, quality of the source code, tests, and the overall product quality. In this paper, we adopt the product view on software quality, recognizing the relationship between internal product characteristics and quality in use [Kitchenham and Pfleeger, 1996].

Product documentation comprises of written requirements, architecture documentation, and test cases. Deficiencies in such artifacts are associated with hindered knowledge distribution in the team and with adverse effects on further development and maintenance of the product [Tom et al., 2013]. Note that we analyze if documentation artifacts are understandable and useful without implying any specific format.

Even though the Agile manifesto emphasizes working software over comprehensive documentation, some documentation is essential [Beck et al., 2001]. For example, user stories are one of the key agile tools to document requirements [Lucassen et al., 2015]. System metaphor is useful to communicate the logical structure of the software to all stakeholders [Khaled et al., 2004]. The use of automated testing in continuous integration and deployment pipelines require formally defined tests [Collins et al., 2012].

Software architecture denotes how different components, modules, and technologies are combined to compose the product. Symptoms such as outdated components, a need for workarounds and patches point towards deficiencies in the software architecture and the lack of attention to refactoring [Moser et al., 2007, Selic, 2009].

Source code quality is determined by the use of coding standards and refactoring practices [Palomba et al., 2014b, Mantyla et al., 2003]. Degrading architecture and poorly organized source code is associated with increased software complexity, difficult maintenance, and product quality issues down the road [Tom et al., 2013].

We analyze the quality (or lack, thereof) of automated test scripts removing the need to perform manual regression testing with every release of the product. The effort of manual regression testing grows exponentially with the number of features, slowing down release cycles and making defect detection a time consuming and tedious task [Tom et al., 2013].

We also examine associations between product quality and the use of Agile practices. With product quality, we understand non-functional aspects of the product, such as performance, scalability, maintainability, security, robustness, and the ability to capture any defects before the product is released to customers [Tom et al., 2013].

Good communication, teamwork, adequate skills, and a positive attitude towards following the best practices are recognized as essential team factors for project success [Chow and Cao, 2008]. Agile software engineering practices aim to facilitate communication, empower individuals, and improve teamwork [Dybå and Dingsøyr, 2008]. We analyze the associations between team characteristics and the use of specific Agile practices.

Attitudes determine the level of apathy or interest in adopting and following the best engineering practices. Skills characterize to what extent individual members of a start-up team possess relevant engineering skills and knowledge. Communication captures to what extent the team can communicate and coordinate the engineering work. Giardino et al. [2016] identify the team as the catalyst for product development in start-ups. Sufficient skills, positive attitudes, and efficient communication are essential for rapid product development in both agile and start-up contexts [Giardino et al., 2016, Chow and Cao, 2008].

Pragmatism characterizes to what extent a team can handle trade-offs between investing in perfected engineering solutions and time-to-market. Agile practices advocate for frequent releases and good-enough solutions [Rising and Janoff, 2000]. Such practices help to validate the product features early and gather additional feedback from customers [Klotins et al., 2019b]. On the other hand, quick product releases need to be accompanied by frequent refactoring and unit tests to manage technical debt and keep regression defects under control [Chow and Cao, 2008]. Start-ups often overlook such corrective practices [Giardino et al., 2016, Klotins et al., 2018a].

Sufficient time and resources for product engineering are essential for project success [Chow and Cao, 2008]. We analyze what Agile practices can be associated with better resource estimation and planning in start-ups. Several authors, e.g., Giardino et al. [2014a] and Sutton et al. [2000] identify resource shortages as one of the critical challenges in start-ups. However, we, in our earlier study identify the lack of adequate resources planning and control practices in early start-ups [Klotins et al., 2019a].

We look into respondents estimates on engineering process in their organizations. Process characterizes to what extent product engineering is hindered by unanticipated changes in organizational priorities, goals, and unsystematic changes in the product itself. Lack of organizational support for agile product engineering contributes to project failures [Chow and Cao, 2008]. On the other hand, Agile practices offer some room for adjusting to unclear and changing objectives [Misra et al., 2012].

Agile methods on a high level attempt to address and promise improvements in all these concerns [Dybå and Dingsøyr, 2008]. However, analyzing effects from applying whole methodologies on a large number of factors does not help to pinpoint specific practices for specific challenges. We aim to establish a fine-grained view on the use and effects of individual practices.

3 Research Methodology

3.1 *Research aim*

We aim to explore how start-ups use Agile practices and what positive and negative effects can be associated with specific practices.

3.2 *Research questions*

To guide our study, we define the following research questions (RQ):

RQ1: How are Agile practices used in start-ups?

Rationale: With this question, we identify what Agile practices and in what combinations start-ups use.

RQ2: What are the associations between specific Agile practices and product factors?

Rationale: With this question, we explore the associations between specific Agile practices, quality of documentation, architecture, source code, testing, and overall product quality.

RQ3: What are the associations between specific Agile practices and team factors?

Rationale: With this question, we explore the associations between specific Agile practices, attitudes towards following best engineering practices, pragmatism, communication, skills, resources, engineer process, and teams' productivity.

3.3 *Data collection*

We used a case survey method to collect primary data from start-up companies [Klotins et al., 2019a, Larsson, 1993].

The case survey method is based on a questionnaire and is a compromise between a traditional case study and a regular survey [Klotins, 2017]. We have designed the questionnaire to collect practitioners' experiences in specific start-up cases.

During the questionnaire design phase, we conducted multiple internal and external reviews to ensure that all questions are relevant, clear and that we receive meaningful answers. First, the questions were reviewed in multiple rounds by the first three authors of this paper to refine the scope of the survey and question formulations. Then, with the help of other researchers from the Software Start-up Research Network[1], we conducted a workshop to gain external input on the questionnaire. A total of 10 researchers participated and provided their input.

Finally, we piloted the questionnaire with four practitioners from different start-ups. During the pilots, respondents filled in the questionnaire while discussing questions, their answers, and any issues with the first author of this paper.

As a result of these reviews, we improved the question formulations and removed some irrelevant questions. The finalized questionnaire contains 85 questions in 10 sections. The questionnaire captures 285 variables from each start-up case. The full questionnaire is available as supplemental material on-line[2].

1 The Software Start-up Research Network, https://softwarestartups.org/
2 Full questionnaire: http://eriksklotins.lv/files/GCPquestionnaire.pdf

We use a list of 56 Agile practices to capture respondent's answers on what practices they use in their companies [Agile Alliance, 2018]. The answers are captured in a binary, use or not use, format. In addition to specific practices, we offer an "I do not know" and "other" options to accommodate for lack of respondents knowledge and to discover other, unlisted, practices.

We use 45 variables to capture respondents estimates on product and team factors. The questions capture the respondents' agreement with a statement characterizing a factor on a Likert scale: not at all (1), a little (2), somewhat (3), very much (4). The values indicate the degree of agreement with a statement. Statements are formulated consistently in a way that lower values indicate less, and higher values indicate more agreement with the statement.

In addition to questions about software engineering, the questionnaire contains questions inquiring about the engineering context in the start-up and applied software engineering practices.

The data collection took place between December 1, 2016, and June 15, 2017. The survey was promoted through personal contacts, by attending industry events, and with posts on social media websites. Moreover, we invited other researchers from the Software Start-up Research Network to collaborate on the data collection. This collaboration helped to spread the survey across many geographical locations in Europe, North and South America, and Asia.

3.4 *Data analysis methods*

To analyze the survey responses, we used several techniques. We started by screening the data and filtering out duplicate cases, responses with few questions answered, or otherwise unusable responses. In the screening, we attempt to be as inclusive as possible and do not remove any cases based on the provided responses.

Overall, we analyze responses from 84 start-up cases, 132 data-points per each case, and 11,088 data-points. We use the Chi-Squared test of association to test if the associations between the examined variables are not due to chance. To prevent Type I errors, we used exact tests, specifically, the Monte-Carlo test of statistical significance based on 10,000 sampled tables and assuming ($p < 0.05$) [Hope, 1968].

To examine the strength of associations, we use Cramer's V test. We interpret the test results as suggested by Cohan [1988], see Table 6.1. To explore specifics of the association, such as which cases are responsible for this association, we perform post-hoc testing using adjusted residuals. We consider an adjusted residual significant if the absolute value is above 1.96 ($Adj.residual > 1.96$), as suggested by Agresti [1996].

Table 6.1: Interpretation of Cramer's V test

Cramer's V value	Interpretation
≥ 0.1	Weak association
≥ 0.3	Moderate association
≥ 0.5	Strong association

The adjusted residuals drive our analysis on how different groups of start-ups estimate aspects of technical debt. However, due to the exploratory nature of our study, we do not state any hypotheses upfront and drive our analysis with the research questions.

3.5 *Validity threats*

In this section, we follow guidelines by Runeson et al. [2012] and discuss four types of validity threats and applied countermeasures in the context of our study.

3.5.1 *Construct validity*

Construct validity concerns whether operational measures represent the studied subject [Runeson et al., 2012]. A potential threat is that the statements we use to capture respondent estimates are not capturing the indented team and product factors.

To address this threat, we organized a series of workshops with other researchers and potential respondents to ensure that questions are clear, to the point, and to capture the studied phenomenon.

We triangulate each factor by capturing it by 3 - 4 different questions in the questionnaire. To avoid biases stemming from respondents preconceived opinions about the effects of agile practices, we separate questions about the use of the practices and questions inquiring about team and product factors.

To accommodate for the fact that a respondent may not know answers to some of the questions, we provide an explicit "I do not know" answer option to all Likert scale questions.

3.5.2 *Internal validity*

This type of validity threat addresses causal relationships in the study design [Runeson et al., 2012].

With our study we do not seek to establish causal relationships, thus this type of validity threat is not relevant.

3.5.3 *External validity*

This type of validity threat concerns to what extent the results could be valid to start-ups outside the study [Runeson et al., 2012]. The study setting for participants was close to real life as possible. That is, the questionnaire was filled in without researcher intervention and in the participant's environment.

The sampling of participants is a concern to external validity. We use convenience sampling to recruit respondents and with the help of other researchers, distributed the survey across several different start-up communities. Demographic information from respondent answers shows that our sample is skewed towards active companies, respondents with little experience in start-ups, young companies, and small development teams of 1-8 engineers. In these aspects, our sample fits the general characteristics of start-ups, see, for example, Giardino et al. [2014a, 2015a] and Klotins et al. [2019b]. However, there is a survivor bias, that is, failed start-ups are under-represented. Thus our results reflect state-of-practice in active start-ups.

Another threat to external validity stems from case selection. We marketed the questionnaire to start-ups building software-intensive products. However, due to the broad definition of software start-ups (see Giardino et al. [2014a]), it is difficult to differentiate between start-ups and small-medium enterprises. We opted to be as inclusive as possible and to discuss relevant demographic information along with our findings.

3.5.4 *Conclusion validity*

This type of validity threat concerns the possibility of incorrect interpretations arising from flaws in, for example, instrumentation, respondent and researcher personal biases, and external influences [Runeson et al., 2012].

To make sure that respondents interpret the questions in the intended way we conducted several pilots, workshops and improved the questionnaire afterwards. To minimize the risk of systematic errors, the calculations and the first and the third author performed statistical analysis independently, and findings were discussed among the authors.

To test if the order of appearance of Agile practices affects practitioner responses, we run a Spearman's rank-order correlation test [Daniel, 1990]. We examine a potential relationship between the order of appearance and the frequency chosen by respondents. The results showed that there is no statistically significant correlation ($p > 0.05$).

The majority of the surveyed start-ups (63 out of 84, 74%) are active and had been operating for 1 - 5 years (58 out of 84, 69%). Start-ups are geographically distributed among Europe (34 out of 86, 40%), South America (41 out of 84, 49%), Asia (7 out of 84), and North America (2 out of 84).

Our sample is about equally distributed in terms of the product development phase. We follow the start-up life-cycle model proposed by Crowne [2002] and distinguish between inception, stabilization, growth, and maturity phases. In our sample, 16 start-ups have been working on a product but have not yet released it to market, 24 teams had released the first version and actively develop it further with customer input, 26 start-ups have a stable product and they focus on gaining customer base, and another 16 start-ups have mature products, and they focus on developing variations of their products.

The start-ups in our sample do per-customer customization to some extent: 10 companies (11%) had specified that they tailor each product instance to a specific customer, 30 companies (35%) do not do per-customer customization at all, while 43 start-ups (49%) occasionally perform product customization for an individual customer.

The questionnaire was filled in mostly by start-up founders (64 out of 86, 76%) and engineers employed by start-ups (15 out of 86, 18%). About half of respondents have specified that their area of expertise is software engineering (49 out of 86, 58%). Others have specified marketing, their respective domain, and business development as their areas of expertise.

The respondents' length of software engineering experience ranges from 6 months to more than 10 years. A large portion of respondents (44 out of 86, 52%) had less than 6 months of experience in working with start-ups at the time when they joined their current start-up.

We provide a complete list of studied cases and their demographical information as supplemental material on-line[3].

The responses on what development type best characterizes the company, suggest that most companies, 51 out of 84, 60%, follow agile and iterative processes. A few, 2 out of 84, follow a waterfall like process, 10 companies report using an ad-hoc approach, see Fig. 6.2.

We presented respondents with a list of 56 Agile practices and asked to thick off practices that they use in their companies. Most start-ups use between 0 and 20 Agile practices. However, the majority of companies report using only a few practices, see Fig. 6.2. There is also a small cluster of companies reporting the use of more than 35 individual practices. Only 7 companies explicitly report not using any agile practices, 16 respondents have not provided their answer.

3 The studied cases: http://eriksklotins.lv/files/GCP_demographics.pdf

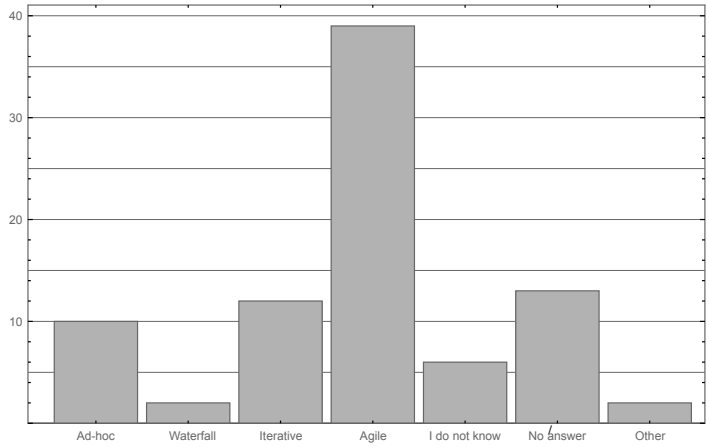


Figure 6.1: Development methodologies in the studied cases. Y-axis denote the number of companies

The most frequently used Agile practices are backlog and version control reported by 42 and 39 companies, respectively (50% and 46% out of 84 cases). The use of other practices varies, see Fig. 6.3. Respondents do not report the use of practices such as the Niko-Niko calendar (visualizing team's mood changes), project charters (a poster with a high level summary of the project), and rules of simplicity (a set of criteria to evaluate source code quality).

4.1 *Overview of the findings*

In Table 6.2, we summarize the associations between the use of certain practices and Product factors. In Table 6.3, we summarize the associations between the use of certain practices and team factors. We show only practices with statistically significant associations ($p < 0.05$). The numbers in the table show Cramers'V values denoting the strength of the associations, see Table 6.1 for interpretation of the values.

4.2 *Interpretations of associations*

An association shows that a specific practice and certain estimates on a factor are reported together. We use the Pearsons Chi-squared test ($p < 0.05$) to determine if the association is statistically significant. However, from associations alone, we cannot explain the phenomenon with confidence and guide guide practitioners in adopting Agile practices in start-ups. To explain the associations, we formulate 5 archetypes (A) of propositions characterizing potential explanations of our findings.

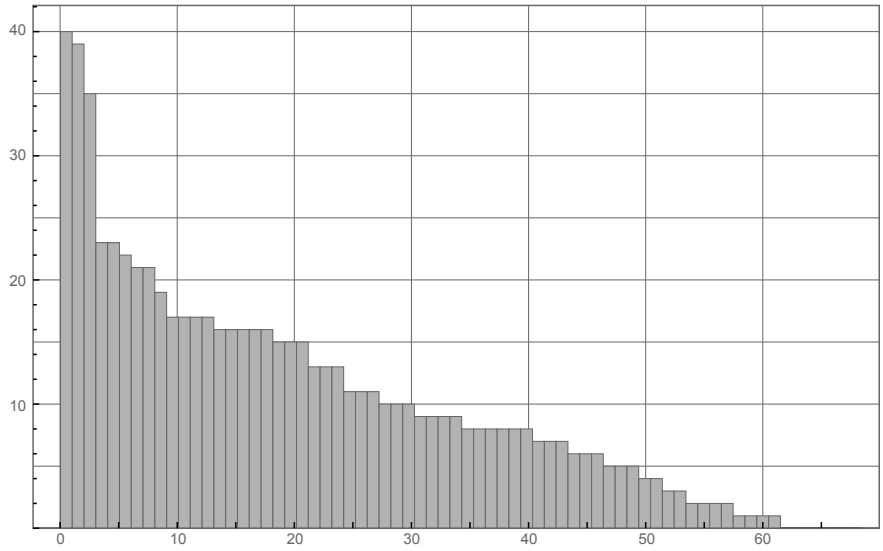


Figure 6.2: Use of Agile practices in the studied start-up companies. X-axis show studied cases, y-axis show the number of reported practices. The cases are sorted by the number of reported practices. There are a few companies reporting 30+ practices (left side of the plot), and 7 cases reporting not using any agile practices (right side of the plot).

It could be that a statistically significant association is a false positive. That is, the association between a practice and a factor is due to an error or some confounding factor.

A_0 : There is a spurious association between P and F.

An association could point towards a causal relationship between the use of a practice (P) and a factor (F). We are measuring factors through respondents evaluations, thus we cannot distinguish between actual and perceived improvements.

A_1 : Use of P improves perception of F.

Some of the associations appear to be negative, i.e. use of a practice is reported together with an unfavorable estimates. It could be that the practice has adverse effects, or the use of the practice helped to expose the a problematic factor:

A_2 : Use of P hinders F.

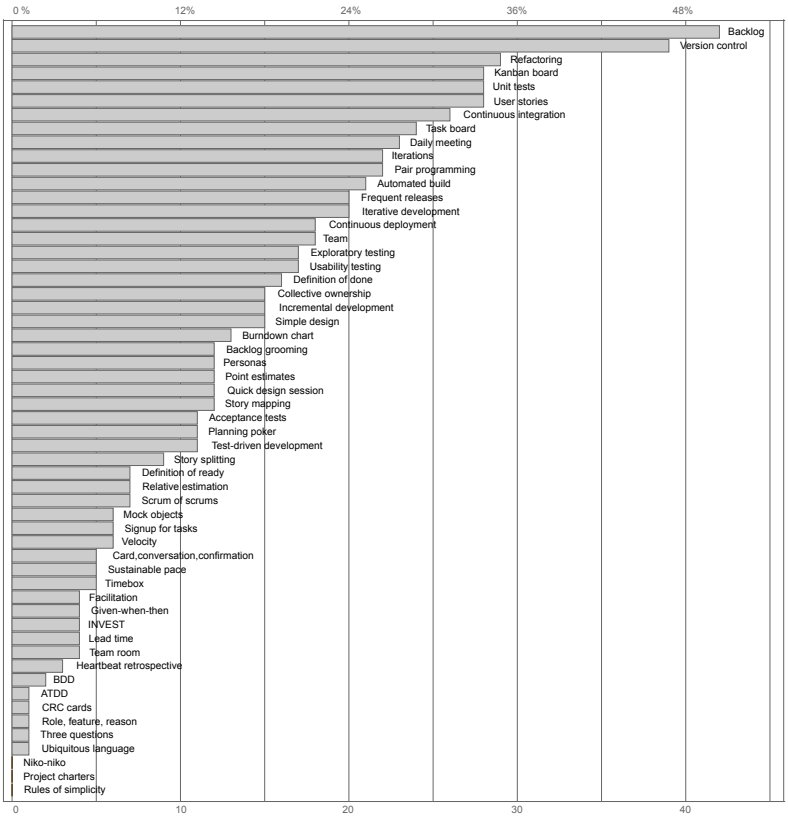


Figure 6.3: Frequency of Agile practices. X-axis denote the number of cases reporting use of a practice. On the top we show percentage, at the bottom we show the absolute number of cases.

A_3 : Use of P exposes issues with F.

It could be that a practice is introduced as a consequence to a situation. That is, we could be observing a reverse causal relationship.

A_4 : F is the cause or enabler for using P

4.3 *Specific findings*

In this section, we link together our specific findings with relevant propositions, see Fig. 6.4. In the figure we show a list of agile practices with

Table 6.2: Results of Cramer's V test on associations between product factors and use of Agile practices with $p < 0.05$. Up (↑) and down (↓) arrows denote whether the association is positive, i.e., use of the practice is associated with more positive responses, or negative, i.e., use of the practice is associated with more negative estimates from respondents.

Practice	Documentation	Architecture	Source code	Testing	Overall quality
Card, Conversation, Confirmation	-	-	-	0.422↑	-
Unit tests	-	-	-	0.391↑	-
Automated build	-	-	0.374↑	-	-
Facilitation	-	-	0.330↓	-	-
Given, When, Then	-	-	0.330↓	-	-
INVEST	-	-	0.330↓	-	-
Iterations	-	0.359↑	-	-	-
Continuous integration	-	-	-	-	0.368↑
Collective ownership	-	-	-	-	0.372↓

statistically significant associations to factors. The factors are groups into four blocks $A_1 - A_4$ representing our propositions. The arrows denote potential explanations between factors and practices.

A product backlog is an authoritative list of new features, changes, bug fixes, and other activities that a team may deliver to achieve a specific outcome [Agile Alliance, 2018].

Our results show a moderately strong ($Cramers'V = 0.401$) association between the use of a backlog and worse perceptions on the engineering process. In particular, frequent changes in requirements, unclear objectives, and unsystematic changes hindering the engineering process are reported together with the use of the backlog.

Unit testing is a practice to develop short scripts to automate examination of low-level behavior of the software [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.379$) between the use of unit tests and teams' attitudes. In particular, a positive

Table 6.3: Results of Cramer's V test on associations ($p < 0.05$) between use of Agile practices and team factors. Up (↑) and down (↓) arrows denote whether the association is positive, i.e., use of the practice is associated with more positive responses, or negative, i.e., use of the practice is associated with more negative estimates from respondents.

Practice	Attitudes	Pragmatism	Communication	Skills	Resources	Process
Backlog	-	-	-	-		0.401↓
Unit tests	0.379↑	-	-	-	-	-
Continuous integration	0.360↑	-	-	-	-	-
Automated build	-	-	-	-	-	0.346↓
Definition of Done	0.411↓	-	-	-	-	-
Simple design	-	-	-	-	0.365↓	-
Burndown chart	0.383↓	-	-	-	0.384↑	-
Story mapping	-	0.356↑	-	-	-	-
Relative estimation	0.399↓	-	-	-	0.399↑	-
Velocity	0.435↓	-	-	-	-	-
Team room	-	-	-	-	0.343↓	-

attitude towards following the best design, coding, and testing practices are reported together with using unit testing.

Our findings also show a moderately strong association ($Cramers'V = 0.391$) between the use of unit testing and less reliance on manual testing of the product.

Continuous integration aims to minimize the duration and effort of each integration episode, and maintain readiness to deliver a complete product at any moment [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.360$) between the use of continuous integration and more positive attitudes towards using sound design, coding, and testing practices.

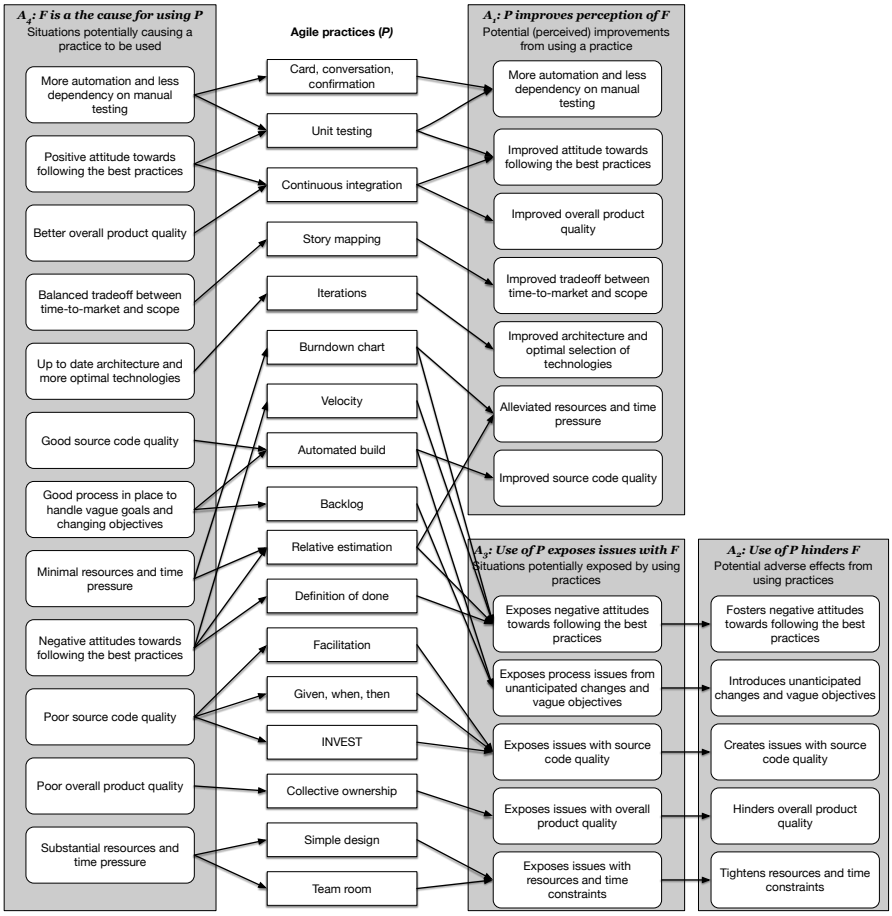


Figure 6.4: Overview of the findings and the propositions. We show agile practices and different explanations for the associations ($A_1 - A_4$).

Our findings also show a moderately strong association ($Cramers'V$ = 0.368) between the use of continuous integration and more positive estimates on product internal and external quality, and less slipped defects.

Automated build is a practice to automate the steps of compiling, linking, and packaging the software for deployment [Agile Alliance, 2018].

Our findings show a moderately strong ($Cramers'V$ = 0.346) association between the use of automated build and worse estimates on the engineering process.

Our findings also show a moderately strong ($Cramers'V$ = 0.374) association between the use of automated builds and more positive estimates on the source code quality.

Definition of done is a list of criteria which s task must meet before it is considered done [Agile Alliance, 2018].

Our findings show a moderately strong (Cramers'V = 0.411) association between the use of a definition of done and worse attitudes towards following the best engineering practices.

Simple design is a practice to favor simple, modular, and reusable software designs that are created as needed [Agile Alliance, 2018].

Our findings show a moderately strong association (Cramers'V = 0.365) between simple design practices and more pressing time and resources concerns.

Burndown chart is a graph visualizing the remaining work (x-axis) over time (y-axis) [Agile Alliance, 2018].

Our findings show a moderately strong association (Cramers'V = 0.383) between the use of the burndown chart and worse estimates on teams' attitudes towards following the best engineering practices.

Our findings also show a moderately strong association (Cramers'V = 0.384) between the use of the burndown chart and less time and resources pressure.

Story mapping is a practice to organize user stories in a two-dimensional map according to their priority and the level of sophistication. Such a map is used to identify requirements for a bare-bones but usable first release, and subsequent levels of increased functionality [Agile Alliance, 2018].

Our findings show a moderately strong association (Cramers'V = 0.356) between the use of the story mapping and a more pragmatic approach on handing the trade-off between time-to-market and following the best engineering practices.

Relative estimation comprises of estimating task effort in relation to other similar tasks, and not absolute units [Agile Alliance, 2018].

Our findings show a moderately strong association (Cramers'V = 0.399) between the use of the relative estimation and worse attitudes towards following the best testing, architecture, and coding practices.

Our results also show a moderately strong association (Cramers'V = 0.399) between the use of relative estimates and less time and resources pressure.

Velocity is a metric to calculate how long it will take to complete the project based on past performance [Agile Alliance, 2018].

Our findings show a moderately strong association (Cramers'V = 0.435) between the use of velocity and worse attitudes towards following the best engineering practices.

Team room is a dedicated, secluded, and equipped space for an agile team to collaborate on the project [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.343$) between the use of a team room and more pressing time and resource constraints.

Facilitation is a practice to have a dedicated person in the meeting, ensuring effective communication, and maintaining focus on the objectives [Agile Alliance, 2018].

Given, When, Then is a template for formulating user stories comprising of some contextual information, triggers or actions, and a set of observable consequences [Agile Alliance, 2018].

INVEST is a checklist to evaluate the quality of a user story [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.330$) between the use of any of the three practices (Facilitation, Given, When, Then, and INVEST) and worse estimates on the product source code quality.

Iterations are time-boxed intervals in an agile project in which the work is organized. A project consists of multiple iterations, tasks, and objectives for the next iteration and is revised just before it starts [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.359$) between the use of iterations and more positive estimates on the quality of product architecture. Specifically, respondents report fewer workarounds, more optimal selection of technologies, and fewer issues with outdated designs.

Collective ownership is a practice to empower any developer to modify any part of the project source code [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.372$) between collective ownership and worse estimates on the product's internal and external quality.

Card, Conversation, Confirmation is a pattern capturing the life-cycle of a user story. The life-cycle starts with tangible "card", "conversations" regarding the user story takes place throughout the project; finally, a "confirmation" is received of a successful implementation of the user story [Agile Alliance, 2018].

Our findings show a moderately strong association ($Cramers'V = 0.422$) between the use of the life-cycle pattern and less dependence on manual testing of the product.

5.1 *Answers to our research questions*

RQ1: HOW ARE AGILE PRACTICES USED IN STARTUPS Our results show that start-ups use Agile practices, even though they do not follow any specific agile methodology. Such results confirm earlier findings, e.g., Giardino et al. [2016], and Yau and Murphy [2013], stating that engineering practices and processes in start-ups gradually evolve from rudimentary and ad-hoc to more systematic.

The most frequently used practices are a backlog, version control, refactoring, user stories, unit tests, and kanban board. We could not identify any clear tendencies comparing frequencies of practices between different cohorts, e.g., team size, product stage, and team skill level.

The use of Agile practices does not imply that an organization follows agile principles as proposed by the Agile manifesto [Beck et al., 2001]. Many of the Agile practices, for example, version control, unit testing, and refactoring, among others, could be equally well applied with other types of development methodologies. That said, a majority of start-ups characterize their development methodology as agile. Exploring the maturity of agile processes in start-ups remains a direction for further exploration [Gren et al., 2015, Klotins et al., 2019a].

RQ2: WHAT ARE THE ASSOCIATIONS BETWEEN SPECIFIC AGILE PRACTICES AND PRODUCT FACTORS We identify associations between the use of Agile practices and product architecture, source code quality, test automation, and the overall level of quality. We could not identify any associations regarding the quality and understandability of product documentation.

Practices related to automation, e.g., unit tests, automated build, and continuous integration, are associated with positive estimates on product factors. Practices related to requirements quality, e.g., Given, when, then, and INVEST, show negative associations. It could be that start-ups introduce such practices as a response to the adverse effects of poor requirements. However, the causal effects of using Agile practices need to be explored further to draw any definitive conclusions.

The use of collective ownership is associated with negative estimates of overall product quality. We propose two interpretations: a) collective ownership exposes the actual state of product internal quality, b) collective ownership has adverse effects.

If two or more developers collaborate on the same part of the product, they may have a more objective view of its flaws. A single developer working on and "owning" a part of a product may be biased in estimating its quality [Ozer and Vogel, 2015].

Alternatively, inviting other developers to work on the part of a product could introduce defects. Other developers, who are not the original authors, may lack the essential contextual information to evaluate and change the component without introducing defects. Practices such as unit testing, continuous integration, and pair programming may help to prevent defects and distribute knowledge in the team. Collective ownership could be an example of a practice that must be supported by other practices to avoid adverse effects.

RQ3: WHAT ARE THE ASSOCIATIONS BETWEEN SPECIFIC AGILE PRACTICES AND TEAM FACTORS The most associations pertain to teams' attitudes towards following the best engineering practices. Both positive and negative attitudes towards the best engineering practices are precedents for using several practices. Automation practices, such as unit tests and continuous integration, are associated with positive attitudes. However, control and planning practices, such as the definition of done, burndown chart, relative estimation, and velocity, are associated with negative attitudes towards following the best engineering practices. We explain such results with the need for tighter control over the team's performance when they do not see the benefits of following the best practices.

We observe several associations between the use of Agile practices and respondents' estimates towards time and resource pressures. The use of burndown charts and relative estimates are associated with less pressure. We interpret such findings that the use of resource planning and control practices helps to plan any amount of resources better and alleviate the pressure.

We have not identified any associations about communication in the team. Other authors, e.g., Yau and Murphy [2013] and Sutton et al. [2000], have identified that in small start-up teams, communication is not an issue. Small collocated teams do not need additional support for coordination. Such finding leads us to argue that the primary reasons for introducing Agile practices in start-ups are tighter control over a team's performance and resource utilization.

5.2 *Implications to research*

With this study, we have set forth a number propositions for further investigation. Looking at the propositions, summarized in Figure 6.4, we identify several cross-cutting concerns to address with further studies in the area.

Our results suggest that start-ups adopt Agile practices one by one without following any particular agile methodology, e.g., scrum or XP. Such finding is supported earlier work, for example, Giardino et al. [2016] and Gralha et al. [2018], reporting that new practices are introduced grad-

ually and aimed at addressing specific concerns. However, existing research on adopting agile software engineering considers mostly the adoption of whole methodologies, e.g., scrum, or XP, and not individual practices [Dybå and Dingsøyr, 2008]. We identify an opportunity for more fine-grained research on how to adopt Agile practices in small organizations to address their specific concerns.

Related work identifies the need to be more flexible and to alleviate the need for rigorous upfront planning as the primary goal for adopting agile. Other objectives include the aim to improve product quality, shorten feedback loops with customers, and to improve teams' morale [Dybå and Dingsøyr, 2008]. Such objectives are superficial and do not support the adoption of specific practices or addressing specific start-up specific challenges [Giardino et al., 2014a]. We identify an opportunity to explore precedents of introducing specific Agile practices, and also longitudinal studies examining the effects of specific practices.

5.3 *Implications for practitioners*

Examining our findings, we identify several relevant patterns for practitioners.

Teams' attitudes towards following the best engineering practices appear as a strong denominator of adopting a range of Agile practices. Positive attitudes towards good practices drive the adoption of automated testing and continuous integration. Such practices have further positive effects on software quality.

Negative attitudes towards the best practices are associated with the use of practices for progress control, such as the definition of done, burndown chart, and effort estimation. Our explanation for such a finding is that teams implement such practices to have tighter control over the development process.

Our results suggest that the primary benefits of adopting Agile practices are tighter control over the team's performance and product quality. The use of progress control practices alleviates resource pressures.

6 CONCLUSIONS

In this study, we investigate associations between the use of Agile practices and perceived impact on various product and team factors. Based on our findings, we set forth a number of propositions that narrow down the space of investigation for future studies on Agile practices and start-ups.

We conclude that start-ups adopt Agile practices, however do not follow any specific methodology. The use of Agile practices is associated with improved product quality, more positive attitudes towards following the best

engineering practices, and tighter control over resource utilization. However, the exploration of the causal effects remains a direction of further work.

We have formulated several implications for researchers and practitioners. We identify an opportunity for more fine-grained studies (on practice level) into the adoption and effects of Agile practices. We conclude that Agile practices show a potential to be used in start-ups setting, however adopting individual practices without considering the supporting practices could lead to adverse effects.

7

SOFTWARE START-UPS THROUGH AN EMPIRICAL LENS: ARE START-UPS SNOWFLAKES?

Most of the existing research assume that software start-ups are "unique" and require a special approach to software engineering. The uniqueness of start-ups is often justified by the scarcity of resources, time pressure, little operating history, and focus on innovation. As a consequence, most research on software start-ups concentrate on exploring the start-up context and are overlooking the potential of transferring the best engineering practices from other contexts to start-ups.

In this paper, we examine results from an earlier mapping study reporting frequently used terms in literature used to characterize start-ups. We analyze how much empirical evidence support each characteristic, and how unique each characteristic is in the context of innovative, market-driven, software-intensive product development.

Our findings suggest that many of the terms used to describe start-ups originate from anecdotal evidence and have little empirical backing. Therefore, there is a potential to revise the original start-up characterization.

In conclusion, we identify three potential research avenues for further work: a) considering shareholder perspective in product decisions, b) providing support for software engineering in rapidly growing organizations, and c) focusing on transferring the best engineering practices from other contexts to start-ups.

1 INTRODUCTION

In recent years, software start-ups have gained attention from the research community. In 2014, a systematic mapping study by Paternoster et al. [2014] highlighted the lack of relevant research addressing software engineering in start-ups. Results of this paper are reused by most subsequent studies on software start-ups.

In 2016, Unterkalmsteiner et al. [2016] published a research agenda identifying further research directions in the area. These directions explore start-ups from software engineering perspective and only superficially

touches upon other, e.g. marketing and business, aspects of start-ups. The underlying idea is that the core of a start-up is development and maintenance of a software-intensive product. Thus, shortcomings in the product development could hinder any subsequent attempts to build a sustainable business around it [Klotins et al., 2017].

Since 2014, a substantial corpus of empirical data on software start-ups has been collected and analyzed, for example, Giardino et al. [2015a], Klotins et al. [2019b,a], and Tripathi et al. [2018]. Several models are proposed to explain software engineering in start-ups, for example, Giardino et al. [2016], Giardino et al. [2014b], and Klotins et al. [2018b].

Most of the recent research on software start-ups focus on exploring engineering context and used practices. The exploration is motivated by the premise that start-ups are "special" and "unique", thus require a special approach to software engineering, for example, Sutton et al. [2000], Blank [2013b], Gralha et al. [2018], and Duc et al. [2017]. At the same time, systematic adoption of existing engineering practices for use in start-ups had attracted little attention [Klotins et al., 2015, Berg et al., 2018].

The empirical data, for instance, Coleman and O'Connor [2008], Klotins et al. [2019a], Klotins et al. [2019b], and Giardino et al. [2015a], show little evidence of anything special, regarding software engineering, in start-ups compared to other market-driven organizations developing innovative software-intensive products. Such results invite to revisit the initial premise.

Understanding to what extent software start-ups are different from established organizations is central to transferring the best engineering practices from other contexts to start-ups. If start-ups are different, the differences need to be explored to develop start-up specific engineering practices. If start-ups are not different, further research needs to emphasize the transfer of the best engineering practices from other contexts to start-ups.

There has been a limited success with formulating a crisp and distinctive definition of a software start-up [Sánchez-Gordón et al., 2016, Unterkalmsteiner et al., 2016].

Ries [2011] broadly defines start-ups as human institutions aiming to deliver new products or services under extreme uncertainty. Carmel [1994a] defines start-ups as new, market-driven companies aiming to launch software product fast with minimal resources. Unterkalmsteiner et al. [2016] define software start-ups as newly founded companies developing software-intensive products under time and resource pressures.

In our earlier study, we define start-ups as small companies created to develop and to market an innovative and software-intensive product and to aim to benefit from economies of scale [Klotins et al., 2018a]. These definitions describe software start-ups, however miss to convey any distinctive features.

Blank [2013b] argues the key difference between start-ups and established organizations is that established organizations aim to execute their business model, while start-ups are searching for one. To software engineers, this difference translates into a focus on iterative development, frequent product releases, and extensive use of customer feedback. A very similar approach is used for market-driven product development in established organizations [Dahlstedt et al., 2003].

Paternoster et al. [2014] compile a list of recurring terms describing software start-ups. The terms are, for example, lack of resources and experience, time pressure, small team, high risk of failure among others. This list is often used by later studies, for example, Gralha et al. [2018], Giardino et al. [2015a], Giardino et al. [2016], and Klotins et al. [2019b], to define what is a start-up and to justify their uniqueness. However, the list is meant to "illustrate how authors use the term software startup", and does not imply any empirical grounding.

The objective of this study is to examine how much empirical support there is for "unique" characteristics of start-ups. We analyze the list of start-up characteristics proposed by Paternoster et al. [2014] and trace the supporting literature. Then, we examine the literature to estimate how much empirical support there is for each characteristic.

The rest of this paper is structured as follows: In Section 2 we examine the terms and the supporting evidence. In Section 3 we discuss our findings. Section 3 concludes the paper.

2 START-UP CHARACTERISTICS

We use the list of recurring terms characterizing software start-ups by Paternoster et al. [2014], to drive our analysis. The original list contains the following characteristics:

1. Lack of resources - Economical, human, and physical resources are extremely limited.
2. Highly Reactive - Startups are able to quickly react to changes of the underlying market, technologies, and product (compared to more established companies)
3. Innovation - Given the highly competitive ecosystem, startups need to focus on highly innovative segments of the market.
4. Uncertainty - Startups deal with a highly uncertain ecosystem under different perspectives: market, product features, competition, people and finance.
5. Rapidly Evolving - Successful startups aim to grow and scale rapidly.
6. Time-pressure - The environment often forces startups to release fast and to work under constant pressure (terms sheets, demo days, investors' requests)

7. Third party dependency - Due to lack of resources, to build their product, startups heavily rely on external solutions: External APIs, Open Source Software, outsourcing, COTS, etc.
8. Small Team - Startups start with a small numbers of individuals.
9. One product - Company's activities gravitate around one product/service only.
10. Low-experienced team - A good part of the development team is formed by people with less than 5 years of experience and often recently graduated students.
11. New company - The company has been recently created.
12. Flat organization - Startups are usually founders-centric and everyone in the company has big responsibilities, with no need of high-management.
13. Highly Risky - The failure rate of startups is extremely high.
14. Not self-sustained - Especially in the early stage, startups need external funding to sustain their activities (Venture Capitalist, Angel Investments, Personal Funds, etc.).
15. Little working history - The basis of an organizational culture is not present initially.

For the brevity of our discussion, we group these terms them into 6 categories as some of the terms appear to be related.

In the following subsections, we examine sources of the characteristics. We look into papers, identified by the review [Paternoster et al., 2014], to find empirical support each start-up characteristic. In our review, we include both papers listed by the mapping study and any relevant papers referenced by the listed papers. In essence, we attempt to trace the original statement, a piece of data that inspired the formulation of each characteristic. In addition, we discuss to what extent each characteristic is relevant in other types of organizations.

2.1 *Lack of resources and dependency on external sponsors*

Lack of human, economic, and physical resources to support product engineering is the most frequently used term to describe software start-ups. It is related to dependencies on 3rd parties for funding and having not enough cash-flow to be self sustainable [Paternoster et al., 2014].

Following the references, we found 24 papers, of which 17 analyze empirical data. We review these 17 papers to understand what exact empirical data was the basis for claiming that lack of resources and dependency of external sponsors are characteristic to start-ups.

Some of the papers discuss the need or intention to allocate resources to support product engineering, and not the lack of resources as a challenge [Yogendra and Sengupta, 2002, Yoffie and Cusumano, 1999, Carmel,

1994b]. Coleman and O'Connor [2008] reference an experience report from a start-up company. The start-up, operating in 1992 was not able to afford then costly Internet connection and had relied on public Internet access elsewhere. May [2012] discusses wasted resources in a start-up due to poor work ethics and using sub-optimal technologies. Mudambi and Treichel [2005] and Yoo et al. [2012] argue that small organizations have lesser resources at hand than larger organizations and may not yet have a sustainable revenue, thus resource allocation is an ongoing issue. Later studies elaborate on the impact of resource shortages.

Giardino et al. [2015a] report allocation of resources as one of the Top 10 challenges in start-ups, and elaborates that a studied company was unable to solve some technical problems in the product due to insufficient resources. Lindgren and Münch [2016] report that start-ups were not able to utilize experimentation to a full extent due to limited resources. Jørgensen [2017] report that shortages in human resources caused delays in product development, and a project was canceled due to an insufficient budget.

Related work on project resource management suggests securing sufficient resources is one of the critical steps in project inception and is linked to project success [Snyder, 2014]. In both plan-driven and agile environments, the presence of a committed sponsor is one of the key denominators for project success [Chow and Cao, 2008]. The trade-offs between features, resources, and quality, are common in any project [Junk, 2000, Elonen and Artto, 2003]. In this aspect, start-ups do not look any different.

Suominen et al. [2017] investigates the impact on venture capital to start-ups prospects and identifies that external funding has no significant effect on start-up outcome. Therefore, the focus of further research and practice should be on better methods for engineering resource planning, control, and risk management, to make the best use of any amount of resources. Hadley et al. [2017] presents similar findings suggesting an association between venture capital and negative long-term consequences.

A report by Harvard Business Review [P. Azoulay, 2018] report that venture capitalists prefer investing in start-ups with younger founders, even though the odds of commercial success are with older and more experienced founders. The report points out that younger founders could be more financially constrained, thus be more willing to cede their business to venture capitalists at a lower price. In other words, young and inexperienced founders could provide higher returns of investment for venture capitalists.

The related work so far does not present any convincing evidence that start-ups would experience the trade-off between resources, scope, and quality differently than other organizations [Junk, 2000]. However, the related work suggests that a potential difference between start-ups and established organizations could be that in an established organization project

sponsor and the project team are from the same organization, thus share the same goals to serve customers, improve internal efficiency, and fulfill organization's mission. However, start-ups are often funded by other organizations, for example, venture capitalists. Thus, their goals may not always be aligned [Suominen et al., 2017, P. Azoulay, 2018].

As shown by P. Azoulay [2018], venture capitalists could aim to maximize their return on investment. Start-up founders, in turn, could be motivated by an intent to bring their ideas to market, desire for autonomy, and need for accomplishment among other factors [Estay et al., 2013].

2.2 *Time pressure*

Time pressure is often used in combination with a lack of resources to describe start-ups [Nguyen-Duc et al., 2017]. The pressure supposedly originates from investors, external deadlines, and contracts. Following the references, we found 13 supporting papers, of which 6 use empirical data [Paternoster et al., 2014].

Examining the papers closer, we found that none of the papers use any data to justify the time pressure in start-ups. However, the papers present a discussion motivating the need for faster delivery time to reduce opportunity cost [Blank, 2013b, Tingling and Saeed, 2007, Payne et al., 1996]. Start-ups aim to spend as little time as possible on activities that have an uncertain contribution to customer value, e.g., building invented features.

Giardino et al. [2016] identifies development speed as the core concept in start-ups. It is motivated by the need to keep the team's morale high and to validate the product idea fast. Another study by Giardino et al. [2015a] links time pressure with available resources and the need to establish a sustainable stream of revenue quickly.

These findings suggest that the time pressure originates from internal considerations and resource limitations, and not from competition or external deadlines. Thus, start-ups may have relative freedom to control the development pace and address the trade-off between quality and speed. Concerning time pressure, established companies face the same opportunity costs. However, they may have more resources at hand to sponsor the product development for longer.

2.3 *Innovation*

Focus on innovative technologies, products, and market segments is another term used to characterize start-ups [Paternoster et al., 2014]. Following the references, we identified in 15 papers, of which 9 uses empirical data, concerning innovation in start-ups.

These studies show that start-ups use innovative offerings primarily to differentiate from other competitors in the market [Yogendra and Sengupta, 2002, Carmel, 1994b]. The innovation in start-ups is to a large extent incremental and adds slight improvements to an existing product [Linda, 2010, Yogendra and Sengupta, 2002]. The innovative aspects can concern product features, quality, packaging, and marketing [Kakati, 2003].

Continuous innovation, driven by the innovation strategy, is essential to maintain a competitive edge [Linda, 2010, Kakati, 2003]. Heitlager et al. [2007] argue, albeit without empirical support, that start-ups start with product innovation to enter the market, followed by process innovation to improve efficiency.

Multi-vocal literature recognizes multiple types of innovation, for example, incremental and process innovation, business model innovation, radical and disruptive innovation [typ]. Incremental, process and business model innovation appears to be most suited for small organizations as they focus on improving already known features, activities, and business models [Kakati, 2003]. However, disruptive and radical innovation requires substantial investments and time to replace existing products and create entirely new markets with new business models [rad]. Thus, these types of innovation could be less suited for resource-strapped start-ups.

Regarding innovation, larger organizations may have the leverage to push more ambitious innovations than small start-ups. For example, Apple had created several disruptive innovations by launching its music platform, iPhone, and AppStore. Such innovations were enabled by their experience within the market, human, organizational and economic resources, and their brand name [West and Mace, 2007]. However, start-ups lack most, if not all, such enablers. Regarding innovation, start-ups may have to be more modest than established organizations [Slinger Jansen, Sjaak Brinkkemper, Ivo Hunink, 2008].

2.4 *Rapidly evolving new company*

Terms such as rapid evolution, a new company, small and flat team, focus on one product, and little working history are supported by 34 papers, 22 of them analyzing empirical data [Paternoster et al., 2014].

There is an agreement among the papers that start-ups are new organizations established by one or a few founders championing the product idea [Yogendra and Sengupta, 2002, Carmel, 1994b, May, 2012]. More people, resources, and processes are brought in to support product development and customer service. More processes and artifacts are introduced as the organization grows [Carmel, 1994b, Coleman and O'Connor, 2008].

Surprisingly, none of the studies present data illustrating the start-up growth. The growth is extrapolated from interviewee reflections (e.g. Cole-

man and O'Connor [2008]), a generalized model ([Carmel, 1994b]), and plans to grow customer volume and market share rapidly (Yogendra and Sengupta [2002]).

Later studies identify evolving engineering practices in start-ups. Gralha et al. [2018] and Melegati et al. [2016b] identify that requirements engineering practices in start-ups develop from informal to more structured as the start-up matures. Giardino et al. [2016] identifies a similar pattern in the adoption of agile practices. Early on, start-ups opt for an ad-hoc approach to engineering and introduce new practices as needed. Introduction of new practices and processes impair development speed, however improve coordination and product quality [Klotins et al., 2018a, Giardino et al., 2016].

Established organizations, compared to start-ups, are per definition more stable. Although organizational changes occur in established organizations, they are supported by processes, infrastructure, and concern one or few aspects of the organization at the time [Zhou et al., 2006]. Therefore, rapid evolution in multiple aspects at once could be the most substantial difference between start-ups and other types of organizations.

2.5 *Lack of experience*

Inexperienced start-up teams are reported as a common theme in literature [Paternoster et al., 2014]. This term is supported by 7 papers. However, by looking at the papers closer, we found that none of them present any empirical data supporting the statement.

By analyzing the papers, we found several studies presenting data and analysis providing a strong link between the experience of the teams and prospects of start-up success [Kakati, 2003, Yoo et al., 2012, Carmel, 1994b]. More experienced people require less management [Coleman and O'Connor, 2008], and are an essential resource for rapid product development [Carmel, 1994b, Ambler, 2002]. However, May [2012] and Giardino et al. [2015a] note that it is not always easy to find skilled and motivated individuals.

A report by Harvard Business Review [P. Azoulay, 2018] analyzing a large sample of founders from the US show that most start-up founders are 30 - 50 years old. The average age of commercially successful start-up founder is 45. Authors of the report emphasize the importance of previous experience and acumen to start a new business that comes with older age. Such findings refute the idea of young and inexperienced start-up founders as a typical case.

Other studies add further support for the importance of technical and business experience to start-up success [Zhang, 2011, Politis, 2008]. Giardino et al. [2016] emphasizes the importance of a small and motivated team of skilled individuals. However, we could not find any evidence that

start-ups would have disproportionally more inexperienced engineers than any other type of organization.

Established organizations put substantial effort into on-boarding new software engineers. For example, by providing on-the-job training, mentoring, employee guides, and so on [Johnson and Senges, 2010]. It could take several months until a recruit reaches full productivity [Ganzel, 1998]. A small start-up may lack the capacity to provide such resources to new engineers. As a consequence, start-ups may aim to hire engineers with relevant technical and domain knowledge to compensate for the lack of on-the-job training.

2.6 *Highly risky*

High risk of failure and uncertainty is identified as characteristic to start-ups is supported 12 studies, of which 8 uses empirical data [Paternoster et al., 2014].

Examining the studies further, we found that none of them present any data on start-up failure rate. Blank [2013b] estimates a 75% failure rate among start-ups and motivates it by a report from Harvard Business School. However, we were not able to find the original report.

Looking further, we found a study reporting small business survival rate of 66% after the first year, and 40% after six years or more [Headd, 2003]. The sample includes all types of recently established small businesses. While exact numbers from different sources vary, they agree that most new companies do not survive past the first few years. That said, we were not able to find any credible source estimating a general failure rate among start-ups.

Carmel [1994a] emphasizes that launching a new venture is inherently associated with the risk of failure. However, estimating success and failure rate of start-ups is difficult. Likely, many start-up initiatives are closed down before they appear on any records. After closure, there is no evidence left behind to be studied. Part of the difficulty to estimate start-up failure rate is lack of a clear definition of what is a start-up, and what are their success and failure conditions.

Traditional project management literature considers a project successful if it is delivered within budget, time, and scope [Snyder, 2014]. The economic perspective on start-ups identifies return of investment as the accurate measure of success [Reid and Smith, 2000]. Customer-centric view proposes to use customer satisfaction to assess the project success [Sauvola et al., 2015]. Carmel [1994a] argue that speed is the essential success metric in start-ups.

So far, the related work does not present any evidence that start-ups would have substantially different survival rate than other types of recently

established ventures. However, as we have discussed earlier, start-ups may have stakeholders with different interpretations of success. For example, the investors could be looking for specific return of investment ratio. The odds of attaining such specific objectives could be much lower than of general survival of the company.

3 DISCUSSION

We perform this inquiry to understand if there is enough evidence to claim that start-ups are different from established companies and need a different approach to software engineering. We examine 15 start-up characteristics that are often used to define and differentiate start-ups from established organizations.

By reviewing the literature, we identify several common shortcomings. Firstly, many studies present an anecdotal characterization of start-ups. Such characterization of start-ups is often placed in the introduction, motivating the study. Meanwhile, the research itself focuses on different aspects that neither add or remove support for the characteristics. Such anecdotes propagate, are generalized by further studies, and cause misconceptions about engineering start-ups.

Secondly, studies investigating start-ups rarely, if at all, discuss their findings in a broader context. As a consequence, some challenges, for example, lack of resources and innovation, are presented as unique to start-ups. Such narrow focus takes away the opportunity to transfer the best engineering practices from other contexts to start-ups, and vice versa.

By evaluating the actual empirical evidence, we find little support for most of the characteristics. For example, we could not find any empirical evidence showing that start-up teams are inexperienced. Quite the opposite, empirical studies show that start-ups are often founded by middle-aged entrepreneurs with substantial experience and business acumen. Furthermore, some of the characteristics that are presented as "unique" to start-ups are common in other types of organizations. For example, the challenge of balancing project scope with available resources is hardly unique to start-ups. In other words, by examining the literature, we could not find convincing empirical evidence that start-ups would be in any way "unique" regarding software engineering. Such results suggest that the focus of further research should be on transferring the best engineering practices from established organizations to start-ups.

We identify several limitations concerning our study. The start-up characteristics discussed in this paper are based on work by Paternoster et al. [2014]. There could be other studies more accurately describing start-ups and emphasizing their distinctive characteristics. However, to our best knowledge, the terms identified by Paternoster et al. are the most com-

monly used, thus serve as a good enough basis to raise the discussion on what is so special about software engineering in start-ups.

The literature analyzed in this paper is identified by following traceability information provided by Paternoster et al. [2014]. There is a threat that this information is incomplete and we may have overlooked some important studies. To address this treat, and explore a concept in a broader context, we perform independent searches for relevant literature.

Our discussion is limited only to software engineering perspective of start-ups. Other perspectives, for example, business, finances, and marketing could present more distinct differences between start-ups and established organizations. Such other perspectives are left out from our discussion.

4 CONCLUSIONS AND FURTHER WORK

In this paper, we examine the commonly used characteristics to distinguish between start-ups and established organizations. We found that most of the frequently used start-up characteristics have little empirical support, and some of the characteristics are present in larger organizations as well. We conclude that the terms characterizing software start-ups, and the definition of software start-ups from software engineering perspective need to be revised.

Such finding has implications to our main question whether or not start-ups are special, and should use different engineering practices than small-medium enterprises and other types of organizations. We could not find convincing evidence that start-ups need a different approach to engineering than other types of organizations. We found that rapid evolution and conflicting stakeholders objectives could be adding extra complexity to software engineering. Such additional complexity suggests that start-ups should be more, not less, structured in following the best engineering practices.

From our analysis, we identify three potential research directions concerning software start-ups.

1. Rapid evolution: Growing an organization from a few people to multiple teams working together in a short time requires an evolution of communication and coordination practices as well. Practices that work with few engineers, customers, and a small product, will not suffice in a larger team, thousands of customers and a complex product. There are plenty of engineering practices aimed at dynamic environments, e.g., agile. However, realizing the need for, selection, and continuous adoption of new practices is a major engineering challenge.

2. Thinner margins of error: Given their small size and dependency on external sponsors, start-ups have little margin for errors. The errors may

concern both product decisions, e.g., what features and quality to build, and process decisions, e.g., determining the most efficient way of delivering the features. Larger organizations could cover losses of one product with profits from another. And, compensate for inefficient practices with more resources. However, in start-ups failure to deliver customer value quickly usually means the closure of the company. To software engineers, this translates into the need for proven engineering methods, continuous process improvement, stricter control over resource utilization, and better risk management.

3. Misaligned stakeholder objectives: When project sponsors and the project team are from the same organization, they share the same high-level goals, e.g., to serve their customers, and fulfill the company's mission. However, in start-ups project sponsors could be from a different organization, thus may have very different goals. For instance, venture capitalists may aim to maximize the returns of investment, while a start-up could aim to pioneer an innovative technology. To software engineers, this implies the need to balance the interests of different stakeholder groups, namely, customers, shareholders, and the start-up itself.

8

A COLLABORATIVE METHOD FOR IDENTIFICATION AND SELECTION OF DATA SOURCES IN MARKET-DRIVEN REQUIREMENTS ENGINEERING

Requirements engineering (RE) literature acknowledges the importance of early stakeholder identification. However, the challenge of identifying and selecting the right stakeholders and the potential of using other inanimate requirements sources for RE activities for market-driven products is not extensively addressed.

Market-driven products are influenced by a large number of stakeholders. Consulting all stakeholders directly is impractical, and companies utilize indirect data sources, e.g. documents and representatives of larger groups of stakeholders. However, without a systematic approach, companies often use easy to access or hard to ignore data sources for RE activities. As a consequence, companies waste resources on collecting irrelevant data or develop the product based on the sub-optimal information sources that may lead to missing market opportunities.

We propose a collaborative method to support identification and selection of the most relevant data sources for MDRE. The method consists of four steps and aims to build consensus between different perspectives in an organization and facilitates the identification of the most relevant data sources. We demonstrate the use of the method with two industrial case studies.

Our results show that the method can support identification and selection of the most relevant data sources for MDRE in two ways: (1) by providing systematic steps to identify and prioritize data sources for RE, and (2) by highlighting and resolving discrepancies between different perspectives in an organization.

1 INTRODUCTION

The success of software projects is determined by the degree of stakeholder satisfaction [Agarwal and Rathod, 2006]. Requirements engineering (RE) aims to elicit stakeholder needs, constraints and wishes to sup-

port the rest of software engineering activities [Hofmann and Lehner, 2001b, IEEE Computer Society, 2014]. Software companies operating in a Market-driven Requirements Engineering context (MDRE) [Regnell and Brinkkemper, 2005a] need to accurately identify many heterogeneous requirements sources, e.g. customers, end-users, prospects, partners, competitors, domain experts, internal engineering, business, marketing, sales representatives, external investors, regulators to name a few. Such a large and diverse group of stakeholders with different interests, inconsistent needs, and varying levels of commitment makes requirements engineering challenging [Hamka et al., 2014, Anwar and Razali, 2016, Regnell and Brinkkemper, 2005a] and greatly contributes to overloaded requirements engineering [Regnell and Brinkkemper, 2005a]. Therefore, timely filtering of requirements sources emerges as one of the crucial factors in MDRE. Digitalization and pervasive connectivity significantly contribute to a paradigm shift towards data-driven user-centered identification, prioritization, and management of software requirements [Maalej et al., 2016].

The Software Engineering Body of Knowledge (IEEE Computer Society [2014]), the Software Engineering Institute (Software Engineering Institute [2006]) and the Rational Unified Process (Kruchten [2000]) acknowledge the importance of stakeholder identification and highlight that relevant stakeholders should be identified early in the product life-cycle process. At the same time, these reports seem to underestimate the challenges associated with it, especially in MDRE. The focus is primarily on categorizing [Software Engineering Institute, 2006, Singh, 1996, Pressman, 2001, Sommerville, 2010, Lauesen, 2001] rather than exploring relationships and dynamic interactions, as only some authors emphasize interactions [Sharp et al., 1999, Preiss and Wegmann, 2001].

Stakeholders in MDRE often are unaware of their roles, have difficulty articulating their needs towards a new product, or could be uninterested to collaborate [Davis et al., 2006b]. Furthermore, requirements engineers may not be aware of all relevant stakeholders [Wnuk, 2017, Paternoster et al., 2014]. Therefore, software requirements often cannot be elicited from stakeholders in a traditional sense, e.g., by arranging elicitation interviews. Instead, the stakeholders and their needs emerge from continuous interactions between the market, developers, and the product [Alspaugh and Scacchi, 2013, Groen et al., 2017]. Frequent releases and analyzing customer feedback are the essential practices to learn about stakeholders' needs [Alves et al., 2006, Dahlstedt et al., 2003, La Rocca et al., 2016].

Interactions among a large number of stakeholders produce a continuous flow of data, such as feature ideas, feedback, problem reports, requests for specific customizations, product usage data, market analysis reports and so on [Dahlstedt et al., 2003]. Documenting, analyzing, and prioritizing input from all these sources is impractical [Alves et al., 2006, Karlsson et al., 2007]. As a consequence, companies often opt for the most natural

path and consult with the most accessible sources of information [Klotins et al., 2017, Pacheco and Garcia, 2012]. For example, by inventing requirements internally [Karlsson et al., 2007], and by responding to customer-specific feature requests [Klotins et al., 2019b]. However, when considering the needs of few stakeholders instead of a broader market, companies may miss the opportunities for growth and revenue that market-driven products can offer [Pacheco and Garcia, 2012, Alves et al., 2006]. Finally, although some consider multiple roles (viewpoints) in requirements prioritization [Xiaoqing Liu et al., 2004], considering and balancing various data sources remains largely unexplored.

In this paper, we present a method for identification and selection of the most relevant data sources for MDRE. The method is aimed to a) support identification of relevant data sources, b) support ranking of data sources by their relevance, c) support prioritization of data sources in changing market needs and contextual factors.

The rest of this paper is structured as follows. Section 2 provides background and summarizes related work in the area, Section 3 describes the steps the research methodology, Section 4 presents the method and its steps, Sections 5.1 and 5.2 describes a static validation of the method, Section 8 concludes the paper.

2 BACKGROUND AND RELATED WORK

RE is an integral part of a software product strategy definition that shapes the management of software products. Software Product Management (SPM) is a collaborative effort comprising of, e.g. marketing, sales, engineering, operations, and business development perspectives [Kittlaus and Fricker, 2015]. The roles involved in product development may differently perceive the importance of different data sources and stakeholders behind them. This contributes to the challenge of reaching a consensus within an organization of which data sources to select.

2.1 *Data sources in MDRE*

Bespoke requirements engineering typically elicits requirements from a limited number of customer representatives via interviews, focus groups, observations or similar methods [IEEE Computer Society, 2014]. In MDRE, companies are part of a loose network of customers, end-users, prospects, technology trends, partners, suppliers, other products, competitors and alike [Regnell and Brinkkemper, 2005a]. Requirements towards the product emerge from interactions between stakeholders and the product context [Groen et al., 2017, Jin et al., 2016].

The main challenges are that there are too many potential requirements sources and not all are equally promising or important [Karlsson et al., 2007, Dahlstedt et al., 2003]. The interactions with a large number of stakeholders produce a continuous flow of feature ideas, feedback, problem reports, requests for specific customizations, and so on [Dahlstedt et al., 2003]. Moreover, connectivity and social media make it very easy to submit feedback about the software products. The consequence is an inevitable evolution of MDRE into collaborative, data-driven and user-centered identification, prioritization and management of software requirements [Maalej et al., 2016]. The emerging importance of implicit requirements feedback from product usage data calls for improved methods for filtering and synthesizing large amounts of user data.

Equally investing in documenting, analyzing, and prioritizing requirements from each data source is impractical [Alves et al., 2006, Karlsson et al., 2007]. As a consequence, companies consult with the most accessible sources [Klotins et al., 2017, Pacheco and Garcia, 2012], invent requirements internally [Karlsson et al., 2007], or respond to customer-specific feature requests [Klotins et al., 2019b,a]. However, for new and innovative products candidate data sources and their relevance may be unknown, with the consequence that companies may miss growth opportunities that market-driven products can offer [Pacheco and Garcia, 2012, Alves et al., 2006].

2.2 *Review of existing methods*

A recent systematic review identifies a number of methods for stakeholder identification and quantification [Hujainah et al., 2018]. These methods propose stakeholder high level classes, e.g. into customers, development team, and business [Babar et al., 2014b, Gu et al., 2011, Pacheco and Garcia, 2012], and triage into mandatory, optional, and nice-to-have stakeholders [Razali and Anwar, 2011]. However, these methods are not specifically designed for MDRE and lack support for identification and classification of inanimate data sources.

We perform a follow-up search based on the results from Hujainah et al. [2018] and Bano et al. [2014]. We use snowball sampling to identify exiting stakeholder identification and classification methods. We evaluate identified methods for suitability for use in MDRE and crowd requirements engineering contexts [Groen et al., 2017, Maalej et al., 2016]. We set forth the following evaluation criteria:

1. Support for identification and classification of different types of data sources such as individual people, large groups of people, documents, artifacts, product usage patterns and alike [Maalej et al., 2016, Alexander, 2005].

2. Support for collaboration between multiple analysts by capturing and illustrating different perspectives, in contrast to capturing only consensus view from the group [Aurum and Wohlin, 2003].
3. Support for adaptation of the method for use in different contexts. Results of a method depends on how well it is suited for use in the given context [Petersen and Wohlin, 2009].

We summarize our results in Table 8.1. We denote a method and criteria with "Yes" if the method description clearly addresses the criteria. We use "Partial" to denote potential method support for the criteria through an extension or adaptation, we use "No" to denote a clear lack of support. Cases where a criterion is not applicable are denoted with "-".

By reviewing the existing methods we observed several patterns. First, we observe that several methods, for example, Babar et al. [2014a, 2015], Razali and Anwar [2011], and Bendjenna et al. [2012], uses predefined generalized criteria for analyzing stakeholders. The proposed criteria, such as interest and communication skills, are relevant for human stakeholders and cannot be applied to artifacts. Thus, these methods are context specific and difficult to are tailor for use in a crowd RE setting.

Ballejos and Montagna [2011] and Lim et al. [2010] proposes methods for analyzing relationships between already known stakeholders. Both professional and personal relationships are relevant only for human stakeholders and cannot be applied to inanimate objects, e.g. product usage data.

None of the reviewed methods with partial exception of Razali and Anwar [2011], support collaboration of multiple analysts. Razali and Anwar [2011] suggest to extend the proposed method with prioritization techniques supporting collaboration, such as AHP [Saaty, 1988].

We conclude that none of the reviewed methods is suitable for use identification and selection ofrelevant data sources for MDRE. Our work occupies this research gap.

2.3 *Decision making scenarios*

Software engineering is a collaborative activity and requires cooperation between multiple individuals [Whitehead, 2007]. Requirements engineering is a decision making activity to determine what functional or non-functional features/requirements are needed [Aurum and Wohlin, 2003].

In general, there are two scenarios of decision making situations [Matsatsinis and Samaras, 2001]:

The group-based approaches aim to achieve consensus by discussion and negotiation between the group members. This approach carries the risk that the final outcome is dictated by more powerful members of the group. The risk is particularly high in groups consisting of individuals from different levels of organizational hierarchy. Negotiations and discussions are

Table 8.1: Evaluation of existing stakeholder selection methods

Method summary	Support for different types of data sources	Support for collaboration	Support for adaptation
Babar et al. [2014a] analyzes skill and level of interest of stakeholders	No	No	No
Razali and Anwar [2011] considers mandatory, optional, and nice to have stakeholders and analyzes their knowledge, interest, and interpersonal skills.	No	Partial	No
Bendjenna et al. [2012] considers stakeholder power, legitimacy, and urgency	No	No	No
Babar et al. [2015] focuses on classifying stakeholders based on their personal and professional characteristics.	No	No	No
Burnay [2016] proposes a taxonomy of requirements elicitation sources comprising of people, organization, artefacts, processes, and environment.	Yes	-	No
Alexander [2005] proposes a model for identifying stakeholders and surrogate requirements sources.	Yes	-	Yes
McManus [2004] presents guiding questions to identify and classify relevant stakeholders	Partial	No	No
Ballejos and Montagna [2011] proposes a model for stakeholder representation comprising of roles, interests, influences, power, and goals.	No	No	Yes
Lim et al. [2010] proposes a method to identify and prioritize stakeholders using social networks analysis methods.	Yes	-	Yes

possible only in relatively small groups, e.g. agile teams. Furthermore, individual views in the group could be conflicting to a degree that a consensus is impossible without compromising the final decision or rejecting conflicting decision items. Thus, group-based approaches are feasible in relatively small and rather aligned groups.

Individual-based approaches aim to support individuals in communicating their perspective and aggregating individual views into a final decision. Such approach treats each individual perspective as equal regardless of rank and negotiation power. However, individual based approaches suffer from the neglect of interactions, exchanges of ideas, and arguments that arise from discussions in the group.

As shown by Tindale and Kameda [2000], the degree to which ideas, preferences, knowledge, etc., are shared in the group determine the acceptance of the final decision, and affects the quality and accuracy of the final decision. However, group discussions are not feasible in MDRE context with a large number of stakeholders, enriched by product-usage data. Thus, eliciting and aggregating individual perspectives is a more promising approach.

3 RESEARCH METHODOLOGY

We have followed the design science research method (DSRM) [Wieringa, 2014]. DSRM describes the steps to design an artifact considering two perspectives: i) a specific real-life situation requiring a practical solution, and ii) an abstract, research view of the problem and solution. The method consists of 6 steps, problem identification and motivation, definition of objectives, design and development, demonstration, evaluation, and communication. In the following subsections we describe each step in detail.

3.1 *Research objectives*

In this section, we formulate the research problem with a tempalate propsed by Wieringa [2014]. The objective of this study is to:

- Improve identification and selection of the most relevant data sources for MDRE
- with a systematic method
- ensuring transparency and building consensus between different perspectives
- to select the most relevant data sources for a given task.

To guide our research we formulate the following research questions:

RQ1: What are the needs towards a method for identification and selection of the most relevant data sources for MDRE?

Rationale: By answering this research formulate specific requirements towards the method.

RQ2: How a method can support identification and selection of the most relevant data sources for MDRE?
Rationale: By answering this research question we aim to propose a method design.

RQ3: What improvements to the method are needed for its use in industry?
Rationale: We aim to validate the method in an industrial setting to collect input for its further development.

The research questions are answered in an iterative process of the design science methodology [Wieringa, 2014]. We identify the specific challenges of selecting stakeholders and data sources, and review existing methods in Section 2. We outline specific objectives of the method in Section 3.3, the design steps are presented in Section 4. Objectives and steps of the validation are outlined in Section 3.5, results of the validation are presented in Section 5. We discuss our results in Section 7.

3.2 *Step 1: Problem identification and motivation*

This research is inspired by related work on innovative software-intensive product engineering in MDRE and known challenges in MDRE [Karlsson et al., 2007]. Several authors, e.g. Klotins et al. [2019b,a], Jin et al. [2016], and Groen et al. [2017], identify requirements engineering, and specifically the identification and selection of data sources representing key stakeholders, as one of the key practices to ensure that the product offers the right set of features and is technically and commercially successful. The similar challenge of identifying the right sources and filtering out the most promising requirements makes MDRE difficult [Karlsson et al., 2007]. As summarized in Table 8.1 there exist a shortage of methods for identification and classification of data sources (both human and machine) in MDRE. In particular, there is a need for developing methods that can support different types of data sources, collaboration between multiple analysts by capturing and illustrating different perspectives and offer adaptation for various contexts. Identification of appropriate data sources is central to understanding the context in which the product is developed and operated [Tovar and Pacheco, 2006].

3.3 *Step 2: Objectives of the method*

The main objective of the method is to support identification and selection of the most relevant data sources for MDRE, e.g. market analytics, people, product usage analytics, sensor data, and similar, for consideration in new,

innovative, market-driven, software-intensive product engineering. We decompose the main objective into sub-goals:

1. Support for adaptation of the method for solving specific problems in different contexts
2. Support for identification and selection of the most relevant data sources
3. Support for different types of data sources such as individuals people, large groups, documents, artifacts, product usage data and alike
4. Support for collaboration and consensus building among multiple analysts but also capturing the individual viewpoints and disagreements
5. Transparency and understandability to comprehend the method and its output.

3.4 *Step 3: Design of the method*

The method was incrementally designed in a series of review rounds. At each round, the authors discussed design concerns, evaluated different potential solutions, and updated the method accordingly. We documented rationales behind our design choices and discussed potential alternative solutions, see Section 4.

3.5 *Step 4: Demonstration*

We perform a static validation of the method with the purpose to evaluate its usability and collect feedback for further development of the method [Gorschek et al., 2006]. The validation consists of the following steps:

1. We start by presenting the aims of our study and establishing a common vocabulary. Throughout the demonstration we avoid posing our own views, rather we elicit participants perspectives on data sources identification and selection, utilized practices, and challenges from their experience.
2. We ask the participants to describe a recent example where multiple data sources were selected and used in crafting ideas for further product development. We ask the participants to list all the data sources that were considered and, with hindsight, rank them according to their contribution to the outcome.
   The purpose of this step is to establish a baseline for comparison with the results from the method.
3. We continue by introducing the method step by step and ask participants to describe the trigger, list criteria, and candidate data sources

that were relevant for the example. This step is done in a discussion format inquiring participants about motivation of selecting or ignoring certain data sources or criteria.
The purpose of this step is to validate constructs of the method, their understandability, and practitioners ability to provide meaningful criteria and data sources.

4. We ask participants to provide scores for criteria, evaluations to data sources, and run the calculations to arrive to the final ranking. We use a spreadsheet for data collection.
The purpose of this step is to arrive at ranking of data sources based on the method for comparison with the baseline.
5. Reflections on the results. At the end, we ask participants to reflect on the method and its results, compare it with the initial ranking, and provide their perspective on the differences.

We conduct two case studies and describe results in Section 5. Demonstrations are handled by the first and second authors with help of presentation materials and a semi-structured interview guide. Two practitioners were involved in each demonstration.

3.6 *Steps 5-6: Evaluation and communication*

Further evaluation and communication of the method is planned as continuation of this work. We aim to follow the technology transfer model by Gorschek et al. [2006] and perform several industry feedback and method improvement rounds.

The design for evaluation is similar to what we outline in Step 4. We start by familiarizing the participants with the challenge of selecting data sources, then we establish a baseline, guide the participants through the method application, and help them to interpret the results. However, as the method matures we would like to both reduce researcher support for the use of the method and analyze new situations.

3.7 *Threats to validity*

We follow the guidelines by Runeson et al. [2012] and discuss the four perspectives of validity threats.

3.7.1 *Construct validity*

Construct validity is concerned with establishing appropriate measures to observe the intended concepts. Key concepts of the method originates from related empirical work on innovative software-intensive product en-

gineering and related work in MDRE and crowd RE. We conducted several review rounds to refine formulations and relationships of the concepts.

We further strengthen the construct validity during the case study when asking participants to discuss their view on practices and challenges associated with stakeholder and data sources selection.

A threat is introduced by us potentially influencing the participants views and responses. Knowing the objectives of our study, practitioners may inflate the challenges of data sources selection. Furthermore, participants may be reluctant to reveal their real opinions about the method because the authors were present. To mitigate this threat we avoid posing specific views, rather we introduce concepts neutrally and ask practitioners to reflect on their experience.

3.7.2 *Internal validity*

Internal validity is concerned with uncontrolled factors affecting causal relationships between concepts. The method infers that criteria are the only yardstick to rank data sources. There could be additional influences to ranking that do not fall under our definition of criteria. To minimize this threat, the method needs to be further validated and operationalized, as well as more work is needed towards understanding decision-making factors in stakeholder selection.

A potential limitation of the method is the need to avoid ties between the criteria. Having dependencies between criteria would impair rankings and the final results. We plan to explore different strategies for removing this limitation with further work.

3.7.3 *External validity*

External validity is concerned to what extent the results can be generalized outside the studied cases and remains the main limitation of our work. We cannot claim that the studied cases are representative to all companies operating in MDRE. Thus, the fitness of the method needs to be further validated. In particular, we plan to add support for the theoretical constructs and the usefulness of the method.

3.7.4 *Reliability*

concerns the degree of repeatability of the study. To support traceability and transparency, we described the rational and objective for each step of the research method and the case study. We also provide raw data and our calculations as supplemental material [1]. However, the method embodies ideas, knowledge, and interpretations that remains to some extent subjective.

1 Supplemental material: https://eriksklotins.lv/files/ds-ranking-supplemental-material.pdf

The proposed method is a structured approach to combine input from multiple analysts and produce a ranked list of the most appropriate data sources to analyze and consult in a given decision making situation. The data sources are ranked based on their attributes which are selected and estimated by the analysts. The method consists of four main steps, see Fig 8.1.

In *the Step 1*, the analysts identify a task and formulate a problem statement. The problem statement implies the need to decide and consult multiple stakeholders and data sources. In *the Step 2*, the analysts identify criteria that are relevant for the task (problem) under decision on what data sources and stakeholders to consult. In *the Step 3*, the analysts evaluate the data sources according to the identified criteria. In *the Step 4*, the analysts analyze and interpret the results.

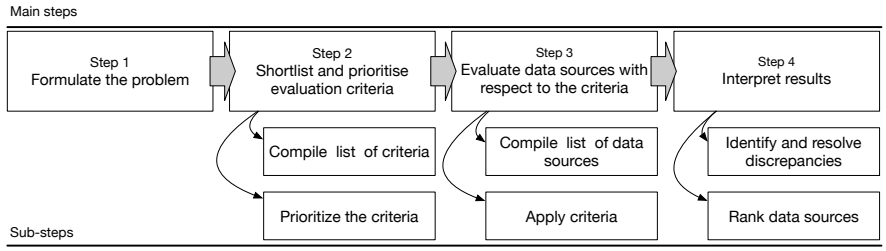


Figure 8.1: Overview of the method. The method consists of 4 main steps with several sub-steps

4.1 *Preconditions*

We start the method design by identifying and outlining the core concepts, described below.

A trigger is circumstances leading up to the formulation of the problem statement. A trigger could be, for example, identification of a new market opportunity, a customer request for a specific feature, market pressure, or technological shift.

A problem statement is brief description of a non-trivial issue to be addressed or a situation to be improved. A problem statement describes the gap between the desired situation and the current, sub-optimal situation [Annamalai et al., 2013]. A problem statement should be quantifiable, e.g., number of lost customers, amount of potential revenue or saved resources.

Data sources may have different forms, for example, individual stakeholders (such as key customers, managers, or product engineers), the

crowd [Groen et al., 2017] consisting of e.g. potential customers, opinion leaders, and users of competitor products, other organizations (such as competitors, partners, or suppliers), analytics (such as product usage data, analysis of customer data, or results from market analysis), industry standards, and applicable laws and regulations. Different data sources may offer different types of information and different perspectives on the decision.

We define a set of all data sources as:

$$D = \{d_1, d_2, \ldots, d_n\}.$$

Criteria are principles how data sources are selected and compared. The exact criteria depends on the problem to be addressed. For example, an internal engineer is easier to access than a customer. In turn, customers have more profound knowledge on their actual needs than engineers but may have a little insight in underlying technology constraints. Another example are the constructs used by Mitchell et al. [1997] in his theory of stakeholder identification: power, legitimacy, urgency and salience.

We define a set of all potentially relevant criteria as:

$$C = \{c_1, c_2, \ldots, c_m\}.$$

An analyst is an individual who collaborates with other analysts on selecting the data sources and on the use of the method. We use the term analyst to refer to a range of people that could be involved in product decisions, such as, software engineers, product managers, financial officers, marketing, sales representatives, and alike. The analysts are involved in the decision making and are the primary stakeholders, that is, the analysts are the decision makers.

We define all analysts involved in the decision making as:

$$E = \{e_1, e_2, \ldots, e_k\}.$$

4.2 *Step 1: Define the problem statement*

The first step is to formulate the problem to be addressed. The problem formulation should contain the following information:

1. Description of the current situation.
2. Characterization of the desired situation.
3. Quantification of the difference between current and desired situation, e.g improve X by Y.
4. Preliminary candidate solutions to address the gap and reach the desired situation. The candidate solutions are to be evaluated and refined by the input from selected stakeholders.

4.3 *Step 2: Shortlist and prioritize criteria*

The second step is to identify and prioritize criteria for comparing potential data sources. This step has three sub-steps: 1) compile the initial list of criteria, 2) shortlist relevant criteria by voting, and 3) prioritize (weight) the criteria.

4.3.1 *Compile a list of criteria*

The analysts put together an initial list of criteria that will be used to compare different data sources. The aim is to identify all relevant characteristics of an optimal data source to resolve the problem defined in Step 1. The proper criteria are domain and problem specific and can be identified through, e.g. discussions and brainstorming.

In order to facilitate the identification of relevant criteria we adapt a taxonomy of requirements criteria proposed by Riegel and Doerr [2015], see Table 8.2. We propose to use the high level criteria from the table as a starting point for the initial analysis and brainstorming to identify additional criteria.

To avoid inconsistencies in further prioritization and estimation, the analysts need to avoid ties between the criteria. For example, if criteria such as knowledge and domain knowledge appear together, analysts may have a difficulty to assign consistent scores [Vigna, 2015]. A counter-strategy could be to identify potential ties early and break down higher level criteria into more fine-grained criteria, thus eliminating the ties.

4.3.2 *Shortlist of relevant criteria*

After the initial list of criteria is compiled, the analysts vote to shortlist more relevant and exclude less relevant criteria, thus reducing the effort in the next steps. Based on the votes and a cut-off threshold, relevant criteria are selected for further consideration.

Each analyst takes a list of all criteria and assigns a binary vote (relevant/not relevant) to each criterion. The votes produced by the different analysts on each criterion are summed. Criteria with votes above a given threshold are included for further consideration. For example, a majority vote can be used to decide which criteria to prioritize, i.e. we can apply a cut-off threshold of $k/2$, where k is the number of analysts.

Shortlisting may be relevant only if at least two analysts are involved. For example, if maximum two analysts are involved then a cutoff threshold of $k/2$ is 1. In this context criteria supported by both analysts should be considered or it may be decided to consider all criteria proposed by the analysts.

Table 8.2: Example criteria of requirements to bootstrap data source criteria

Category	High level criteria	Examples of detailed criteria
Benefits in terms of knowledge	Level of knowledge	Domain, customer needs and wishes, product technologies or features, business processes, laws, regulations, standards
Benefits in terms of experience	Amount of experience	The product, similar products, specific market segments, domain, product technologies
Benefits in terms of financial value	Revenue	Customer life-time value, price per purchase
Costs	Associated costs	Access, establishing access, maintaining access
Penalties	Opportunity cost	Business, market, technical, financial, reputation, dissatisfaction of other stakeholders
Risks	Generic risk	Human factors, technical risks, implementation risks, volatility, business risks, time, budget, project scope, dependencies, reputation, legitimacy
Temporal context	Timing	Lead time, time to access, timeliness of data, frequency of access
Suitability for use	Ease of use	Ease of access, mutual trust, understandability, granularity, analyzability, accuracy, overhead
Behavioral	Suitability for collaboration	Availability, interest to contribute, commitment, volatility, trustworthiness, willingness to experiment, capacity volunteer resources for collaboration, power, leverage

Assume that in this step a list of m candidate criteria and k analysts are involved in the voting. Each analyst e_i ($i = 1, 2, \ldots, k$) casts a vote v_{ij} on criterion c_j ($j = 1, 2, \ldots, m$), where $v_{ij} \in \{0, 1\}$, i.e. a binary vector $v_i = [v_{i1}, \ldots, v_{im}]$ presents the criteria votes for analyst i. The result is a $k \times m$ binary matrix V:

$$V = \begin{bmatrix} v_1 \\ \vdots \\ v_k \end{bmatrix} = \begin{bmatrix} v_{11} \cdots v_{1m} \\ \vdots \quad\quad \vdots \\ v_{k1} \cdots v_{km} \end{bmatrix}.$$

We further count votes V_j on each criterion j ($j = 1, 2, \ldots, m$) by summing the values in each column of V, i.e.:

$$V_j = \sum_{i=1}^{k} v_{ij}.$$

The resulting set of criteria C' can be defined as:

$$C' = \{c_j | V_j > T\}, \text{where } C' \subseteq C \text{ and } T = k/2.$$

4.3.3 *Prioritize the criteria*

The purpose of this step is to prioritize the criteria by their relevance to the problem under decision. Each analyst assigns a score to each criterion assessing its relevance (importance). We propose to use an ordinal scale from 0 to 5, i.e., all scores in $\{0, 1, 2, 3, 4, 5\}$, where 0 indicates no relevance at all, 5 indicates the highest relevance, and the numbers 1, 2, 3, 4 represent intermediate values. In Table 8.3 we provide semantic meaning of each score. To arrive at the final ranking, the scores from each analyst are normalized.

Table 8.3: Semantic meaning of the measurement scale

Score	Meaning
0	Not relevant at all (to the given problem)
1	Marginally relevant
2	Somewhat relevant
3	Moderately relevant
4	Very relevant
5	Most relevant

Each analyst e_i ($i = 1, 2, \ldots, k$) evaluates each criterion $c_j \in C'$ by assigning a score $w_{ij} \in \{0, 1, 2, 3, 4, 5\}$. The result is a vector $w_i = [w_{i1}, w_{i2}, .., w_{im'}]$ of scores produced for each analyst e_i ($i = 1, 2, \ldots, k$), where m' is the cardinality of C' (the set of the selected relevant criteria).

We further calculate weights denoting the relative relevance (importance) of each criterion. Namely, the scores of each analyst e_i are normalized by dividing each value of vector w_i with the maximum given score by the analyst. Thus a vector $w'_i = \left[w'_{i1}, w'_{i2}, .., w'_{im'}\right]$ of weights for each involved analyst e_i ($i = 1, 2, \ldots, k$) is generated, where

$$w'_{ij} = \frac{w_{ij}}{\max_{j=1}^{m'} w_{ij}}.$$

We average weights from all the analysts to calculate the overall weights, i.e., a vector $W = [W_1, W_2, .., W_{m'}]$, where the overall weight W_j of criterion j ($j = 1, 2, \ldots, m'$) is calculate as:

$$W_j = \frac{1}{k} \sum_{i=1}^{k} w'_{ij}.$$

The resulting vector W consists of the relative weights (importance) of the selected evaluation criteria.

The produced individual and overall weights can be studied and compared to better understand and interpret the problem under consideration. It could be useful to analyze whether there is a significant discrepancy in analysts' opinions about the criteria importance. Analysing discrepancies can help to improve transparency, quality and acceptance of the results [Tindale and Kameda, 2000].

4.4 *Step 3: Evaluate data sources with respect to criteria*

In the third step, the analysts compile a list of potential data sources and use the selected criteria to evaluate each data source. This step has two sub-steps: i) compile a list of data sources, and ii) apply criteria

4.4.1 *Compile a list of data sources*

The analysts put together a list of potential data sources. The list can be created by writing down already known data sources, brainstorming, or a combination of both. The aim is to identify a diverse set of potentially informative data sources for further consideration.

The proper data sources are usually domain specific. However, we propose to use the high level data sources given in Table 8.4 as a starting point for the initial analysis and brainstorming to identify additional data

sources. Note that stakeholders can be accessed directly, e.g., by consulting with engineers and indirectly, e.g., by analyzing user behavior through product usage patterns.

4.4.2 *Quantify the data sources (apply criteria)*

The analysts apply the selected criteria to evaluate each data source. Each analyst e_i ($i = 1, 2, \ldots, k$) evaluates each data source d_j ($j = 1, 2, \ldots, n$) on a scale from 0 to 5, where 0 denotes the lowest, and 5 the highest score. Higher scores are awarded to more favorable evaluations, e.g., lower cost and higher accuracy. In Table 8.5 we provide the semantic meaning of each score. Note that we consider only favorable (positive) scores. The scores of each analyst e_i for the data sources given in D on the set of criteria C' produces a $n \times m'$ matrix U_i of weights:

$$U_i = \begin{bmatrix} u^i_{11} \cdots u^i_{1m'} \\ \vdots \quad\quad \vdots \\ u^i_{n1} \cdots u^i_{nm'} \end{bmatrix},$$

where vector

$$U_{i(j)} = \begin{bmatrix} u^i_{1j} \\ \vdots \\ u^i_{nj} \end{bmatrix}$$

represents the assessments of analyst e_i of the data sources w.r.t. criterion c_j. In that way k different matrices are obtained, i.e. one for each analyst e_i ($i = 1, 2, \ldots, k$). We initially normalize the analyst scores in each matrix U_i ($i = 1, 2, \ldots, k$) by calculating the relative importance of each data source w.r.t. each criterion, i.e. the value of each entry u^i_{lj} ($l = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m'$) of U_i is divided by the maximum value in the column the entry appears in. The overall scores of the data sources can also be calculated by averaging over all matrices $\{U_i\}^k_{i=1}$, i.e. a matrix U of overall weights can be obtained as follows:

$$U = \begin{bmatrix} u_{11} \cdots u_{1m'} \\ \vdots \quad\quad \vdots \\ u_{n1} \cdots u_{nm'} \end{bmatrix},$$

where $u_{lj} = 1/k \sum^k_{i=1} u^i_{lj}$ represents the overall assessment of data source d_l ($l = 1, 2, \ldots n$) w.r.t. criterion c_j ($j = 1, 2, \ldots, m'$). Evidently, the vectors $U_{(j)}$ (columns of matrix U, $j = 1, 2, \ldots, m'$) can be used to rank and compare the data sources separately for each criterion. Finally, the vector Y of overall data source ranks can be obtained by taking into account the overall criteria weights given in vector W, i.e.

Table 8.4: Example stakeholders and data sources

Category	High level examples	Detailed examples
Internal stakeholders	Engineers, Product managers, business stakeholders	Product engineers, architects, customer service representatives, sales representatives, managers, company executives
External stakeholders	Users	Premium customers, feemium customers, prospects, end users, partners, competitors, suppliers, lawmakers, regulators
Analytics	Product usage data	Telemetry data, user data, user behavior analysis
Reports	Market research	Market analysis, public surveys, trends, analysis of similar products
Environment	Domain knowledge	Domain experts, Technology standards, laws, regulations, industry conventions, opinion leaders

Table 8.5: Semantic meaning of the measurement scale

Score	Meaning
0	The data source does not favorably contribute to a criteria at all
1	The data source provide marginal favorable contribution
2	The data source provide somewhat favorable contribution to the criteria
3	The data source provide a moderately favorable contribution to the criteria
4	The data source provide a favorable contribution to the criteria
5	The data source most favorably contribute to a criteria

$$Y = W \times U = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix},$$

where $y_l = \sum_{j=1}^{m'} W_{lj} \cdot u_{lj}$ for each l ($l = 1, 2, \ldots, n$). Notice that the resulting vector Y contains overall assessments denoting the importance of different data sources, respectively. These can be used to rank the data sources with respect to their relevance to the problem under decision.

In addition to the above, we can take into account the relative importance of different criteria defined by each analyst e_i (vector w_i') by producing a weighted mean aggregation with the corresponding normalized matrix U_i, i.e. a vector y_i is generated for each analyst e_i ($i = 1, 2, \ldots, k$) as follows:

$$y_i = w_i' \times U_i = \begin{bmatrix} y_{i1} \\ \vdots \\ y_{in} \end{bmatrix},$$

where y_{ij} ($j = 1, 2, \ldots, n$) is the overall relative importance of data source d_j w.r.t. analyst e_i. It is interesting to notice that in the above expression we can use W instead w_i', i.e. $W \times U_i$. The calculated scores present the data sources' evaluation of analyst e_i who has taken into consideration the analysts' group criteria evaluation.

In this context we can calculate the vector Y of overall ranks (weights) of data sources w.r.t. the all involved analysts by calculating the average of the vectors y_i ($i = 1, 2, \ldots, k$), i.e.

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix},$$

where $y_j = 1/k \sum_{i=1}^{k} y_{ij}$ for each j ($j = 1, 2, \ldots, n$).

4.5 *Step 4: Interpret results*

Interpretation of results consists of two steps: i) Identify and resolve discrepancies between analyst perspectives, ii) Rank and select data sources.

4.5.1 *Identify and resolve discrepancies between perspectives*

Breaking down the results and analyzing how different analysts have estimated criteria and data sources can help spotting discrepancies. Analysis of such discrepancies can be useful for better understanding of the problem under consideration, removing ambiguity and improving results of the method. We propose to visualize the vectors y_i ($i = 1, 2, \ldots, k$) to identify discrepancies between the analysts (e.g., see Fig. 8.3).

The values in matrices U_i ($i = 1, 2, \ldots, k$) show how different analysts have evaluated each data source according to each criterion. For example, vector $U_{i(j)}$ represents the assessments of analyst e_i of the data sources w.r.t. criterion c_j. The vectors produced by the different analysts for each single criterion can further be studied in order to understand the cause of differences in vectors y_i ($i = 1, 2, \ldots, k$) containing the assessments of the data sources w.r.t. all the criteria (e.g., see Fig. 8.2, right side). In addition, vectors $U_{i(j)}$ ($i = 1, 2, \ldots, k$) containing the individual evaluations of the analysts can be compared with the corresponding overall (group) assessments of the data sources given in vector $U_{(j)}$. This can provide an additional insight about the data sources and evaluation criteria.

We can also study in the same fashion the analysts' criteria evaluations represented by vectors w'_i ($i = 1, 2, \ldots, k$) versus the overall weights given in W. In addition, we can compare the ranking of the data sources produced based on the analyst individual evaluations ($w'_i \times U_i$) with the ranking he/she will generate by using the overall criteria weights ($W \times U_i$). These can facilitate the analysis and better understanding of the decision scenario and further identification of the reasons of a discrepancy in the analysts' evaluations.

In this study, discrepancy is analysed by using fuzzy concepts like *negligible*, *moderate* and *severe*. For example, a discrepancy can be interpreted as *negligible* if the difference between two weights is below or equal to 0.3 (i.e. in the interval $[0, 0.3]$), *moderate* in case of $[0.3, 0.7]$, and *severe* when

it is above 0.7 (i.e., in the range $[0.7, 1]$). These boundaries can vary for different contexts.

It is important to identify the cause of any discrepancies in order to improve and strengthen the results of the method. We identify the following causes for discrepancies:

1. *Mistakes.* A discrepancy can be caused by a wrongly entered number, miscalculation, or issue in a tool, etc. Such discrepancies can be easily removed by remedying the cause.
2. *Misunderstanding.* Different analysts may have different interpretation of certain criteria or data sources. Misunderstandings can be mitigated by initiating discussions and conducting of analysis of the analysts' estimates. As a result of these the analysts will arrive to shared understanding that will be followed by revising their estimates.
3. *Different perspective.* Discrepancies could be also caused by different and multimodal perspectives on the problem at hand. Genuine different perspectives are handled by the method.

Discrepancies caused by misunderstanding, misinterpretation and perspectives modalities can be addressed by breaking down higher level criteria and data sources into more specific terms. For example, a product could consist of many technologies, thus a criterion such as knowledge of product technology could be too broad to be used for ranking data sources. Breaking down such high level criteria into more specific can help arrive at more consistent evaluations.

4.5.2 *Rank and select data sources*

The weights in the vector Y indicate the overall relevance of each data source based on the evaluations of all the involved analysts. That is, higher ranking data sources are more relevant to the problem under consideration and should be consulted with higher priority, e.g., see Fig 8.2 and Fig 8.4. We suggest to interpret the results in parallel with analyzing discrepancies and exploring analysts' views on each data source and criterion. Thus, benefiting both from aggregated individual-based approach, and exchange of ideas and arguments of group-based approach [Matsatsinis and Samaras, 2001].

We propose to interpret values in the vector Y using fuzzy concepts like *low*, *medium*, and *high*. For example, relevance of a data source can be interpreted as *low* if the value is below or equal to 0.3 (i.e. in the interval $[0, 0.3]$), *medium* in case of $(0.3, 0.7]$, and *high* when it is above 0.7 (i.e., in the range $(0.7, 1]$). These boundaries can be adjusted to suit different contexts.

5 CASE STUDIES

In this section, we describe the demonstration and application of the method in two industrial cases along with practitioners reflections on each component of the method, lessons learned from each step and the collected data.

5.1 *Case I - Supply Chain Digitalization*

Stockfiller AB offers a software-intensive service digitalizing the supply chain between sellers (farmers, food producers, and food importers) and buyers (restaurants and grocery stores) of food products in Sweden. The main value created by the service is more efficient and transparent supply chain that historically is based on personal contacts and manual processes. The monetization model of the service is based on the volume of goods traded in the platform. The service manages about 10 000 sellers and buyers. The company is in the market for 5 years. The main objective of the company is to secure the local market before expanding to nearby regions.

Two managers of the company, chief of operations and head of sales, participated in a workshop. The workshop follow the steps described in Section 3.5. We start by eliciting practitioners experience with selecting data sources and a specific example that can be used as an input to the method. We further guide the participants through the steps of our method. Each step of the workshop is detailed in further subsections.

Reflections on the challenge: The participants reflected that they used to elicit feature ideas from potential customers and implement them in the service. However, the quality of such ideas was low due to lack of understanding about the service and potentially sub-optimal processes at the customer side. Many new features suggested by some customers were rejected by others.

According to the participants, customers are often not capable to articulate their actual needs, thus demonstrating a prototype and eliciting feedback on that works much better than interviews. The company uses knowledgeable and experienced customers as a filter to vet new feature ideas, both internal and provided by other customers. The respondents also identified product vision (their view of how the supply chain should be organized), as a key tool to shape the product and gauge customer requests for new features.

Triggers: The company discovered that a substantial amount of sellers have poor sales performance on the service. Lagging sellers present missed revenue, since the monetization model is based on the traded volume. Principals of the company had proposed to explore the matter.

Table 8.6: Before presenting our method, we asked participants from Case I to evaluate the relevance to the given problem of each data source. The participants were asked to assign a score 0-5, where denotes 0 the least and 5 the highest relevance, to each used data source. We compare these baseline results with the results from our method to evaluate its usefulness.

Data source	Scores		Normalized scores		Aggregated score
	Analyst 1	Analyst 2	Analyst 1	Analyst 2	
Key customers	4	5	0.800	0.675	0.738
Similar Products	3	3	0.600	0.405	0.503
Internal engineers	5	1	1.000	0.135	0.568
New Customers	0	4	0.000	0.540	0.270
Technologies	2	2	0.400	0.270	0.335
Technology trends	0	3	0.000	0.405	0.203
Business Needs	5	5	1.000	0.675	0.838
Engineering concerns	1	4	0.200	0.540	0.370
Laws & regulations	0	1	0.000	0.135	0.066
Industry standards	2	5	0.400	0.675	0.538
Product vision	5	5	1.000	0.675	0.838
Average customer	0	2	0.000	0.270	0.135

Analysts: The principals of the company are the main product decision makers. Chief of operations, head of product, and head of sales regularly meet and discuss opportunities for new product features among other concerns. In the demonstration, only chief of operations and head of sales were present.

Problem statement: The principals set forth an objective to support sellers by providing insights on their sales performance. Initially, there were no insights for sellers to assess their performance and spot opportunities

for improvement. As a consequence, the company was losing potential revenue. The ideal scenario would be that the service provide useful information for sellers to monitor their performance, and suggest adjustments to boost their trade volumes.

We used the example to discuss what data sources were useful in hindsight, i.e. to establish a baseline, see Table 8.6. The participants provided their estimates on a scale 0-5, where 0 denotes the lowest, and 5 the highest relevance. We normalize their scores to arrive at scale of $[0, 1]$. The aggregated score is an arithmetic mean of the standardized scores.

Criteria: The participants immediately identified good relationship and knowledge as the primary criteria to select stakeholders for collaboration. The participants also identified knowledge and experience of the product, accessibility, value per purchase, life time value, mutual trust, openness, capacity and eagerness to contribute, e.g by piloting experimental features, as relevant criteria. There was no need to shortlist the criteria because of only two participants.

In Table 8.7 we summarize the criteria and participants (Analyst 1 and Analyst 2) estimates on their importance, standardized scores, and final aggregated score. The participant scores are normalized to arrive to scale $[0, 1]$, the aggregated score is the arithmetic mean of normalized scored.

Table 8.7: Participant estimates on criteria importance from Case I. The participants were asked to evaluate each criterion on a scale 0-5,where 0 denotes lowest and 5 the highest importance to the given problem. We also show normalized scores and the final aggregated score.

Criterion	Scores		Normalized scores		Aggregated score
	Analyst 1	Analyst 2	Analyst 1	Analyst 2	
Knowledge of the product	3	4	0.600	0.661	0.630
Accessibility	2	5	0.400	0.826	0.613
Trust	5	3	1.000	0.496	0.748
Price per purchase	3	4	0.600	0.661	0.630
Life-time value	4	4	0.800	0.661	0.730
Capacity to contribute	2	3	0.400	0.496	0.448

Further discussions revealed that some criteria could be broken down to be more specific, and suit both stakeholders and data sources. For example, accessibility could be interpreted as the ease of access (for meetings, observations, or reading), and understandability (of the actual needs and descriptions).

Data sources: Participants acknowledged using multiple inputs in their decisions. They listed experienced and knowledgeable customers (key customers), similar products, internal engineers, new customers, technologies (used design templates, patterns, frameworks), technology trends (such as new available technologies), business needs, engineering concerns, laws, and industry standards as relevant data sources. In Fig. 8.2 (left side) we show the original baseline estimates on data source relevance by both analysts.

Application of the method: The participants had no difficulty understanding the rationale behind assigning the scores to criteria and evaluate the data sources. However, there were some discussions on the exact definitions of data sources and criteria. For example, the product vision could have no lifetime value at all, if it is considered as a document. However, the vision could have the highest possible life-time value as realizing the vision is the purpose of the company.

Interpretation of the results:

We summarize the results in Table 8.8. Rows in the table are ordered by initial baseline ranking of data sources. We also show the ranking after applying the method and the difference between baseline and final ranking. A positive difference denotes that a data sources has gained relevance compared to the baseline, a negative difference denotes a reduced relevance.

Relevance column shows final and aggregated relevance scores and their interpretation. In the last column, we quantify the disagreement between analysts views.

We observe several tendencies when looking at the results, see also Fig. 8.2. First, results from the method shows mostly negligible differences between analysts views on how relevant each source are. A moderate disagreement concern the relevance of new customers as a source of data. Examining this discrepancy, we found that the root cause is a disagreement on how much value per purchase new customers deliver (see Fig. 8.3). Analyst 1 wished to emphasize that new customers directly contributes to the bottom line of the company. However, the other analyst expressed a view that new customers contribute relatively little compared to established customers.

Secondly, comparing the original estimates and results from the method, see Fig. 8.2, we observe that the method arrives substantially more consistent results between both analysts. Such results demonstrate that the method is useful for consensus building among the analysts. In addition, it enables to conduct a multi-layer analysis of the analysts' assessments

Table 8.8: Aggregated results and their interpretation from Case I. The rows are ordered by the baseline estimates. We also show the resulting ranking from the method and the difference in ranking.

Data source	Ranking			Relevance		Disagreement	
	Baseline	Method	Difference	Score	Interpretation	Score	Interpretation
Product vision	1	4	-3	0.400	medium	0.200	negligible
Business Needs	2	5	-3	0.396	medium	0.210	negligible
Key customers	3	3	0	0.475	medium	0.031	negligible
Internal engineers	4	6	-2	0.266	low	0.146	negligible
Industry standards	5	9	-2	0.180	low	0.028	negligible
Similar Products	6	11	-5	0.118	low	0.100	negligible
Engineering concerns	7	7	0	0.243	low	0.000	negligible
Technologies	8	12	-4	0.106	low	0.138	negligible
New Customers	9	1	+8	0.717	high	0.567	moderate
Technology trends	10	8	+2	0.181	low	0.096	negligible
Average customer	11	2	+9	0.482	medium	0.000	negligible
Laws & regulations	12	10	+2	0.121	low	0.026	negligible

and in that way it further facilitates the interpretation and better understanding of the selected criteria, the used data sources and the connections between them.

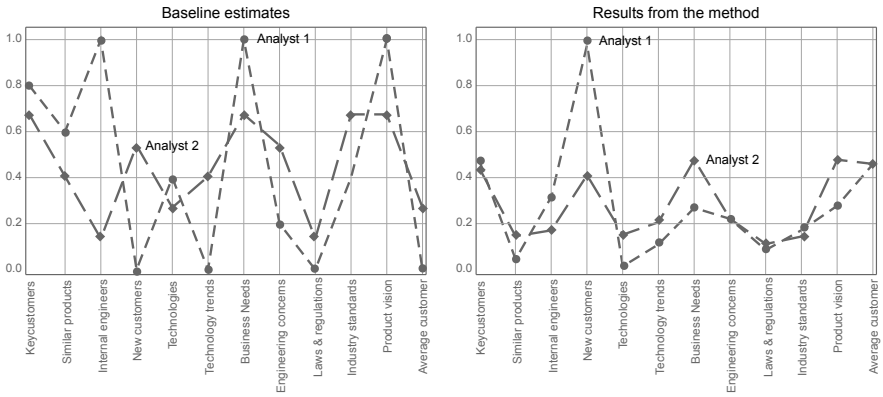


Figure 8.2: Results from the Case I. Figure on the left show baseline estimates, i.e. respondent estimates without the method. Figure on the right show results from applying the method. Y-axis denotes the relative importance of data sources.

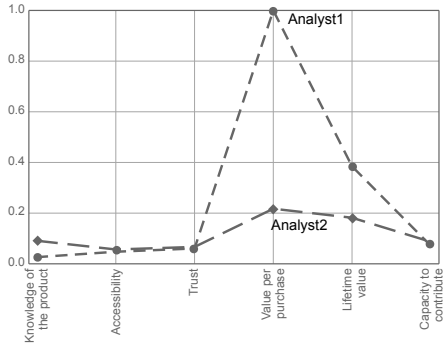


Figure 8.3: Discrepancies in analyst estimates evaluating the relevance of new customers from Case I. Y-axis denote normalized estimates on how much data from new customers contribute to each criterion. Observe the disagreement between analysts on how much value per purchase new customers deliver.

Lessons learned: Feedback from practitioners added support to our hypothesis on the need for a structured method for stakeholder identification. The participants were already familiar with all the concepts and their relationships, thus we did not identify any shortcomings in the construct validity.

The discussions revealed that the main area for improvement could be more refined guidelines for formulating criteria that are applicable to both human stakeholders and other types of data sources.

Application of the method helped participants to reflect on their views on what are the most relevant sources of input and why. Specifically, how use of certain data sources connect to short and long term revenue.

5.2 *Case II - Construction Equipment*

The second case is a company developing and manufacturing construction and mining machinery. The machines are software-intensive and most new features concern updates in the software. The company uses their global partner network to sell and service their machines, and employs over 13 000 people.

Two managers participated in the workshop. The managers are involved and oversee research and development of new product features. We follow the steps described in Section 3.5, and started by presenting the method and then collected the data using a spreadsheet. We did follow-up interviews with each participant to discuss and interpret the results from the method.

Reflections on the challenge: The participants noted that the organization recognises the challenge and had developed an internal framework for gathering information concerning future product development. However, the application of the framework varies per each use.

Example case: The company introduced connectivity to construction machines. The connectivity provides telemetry on machine performance, error reports, maintenance needs, and operator behavior among other data.

Triggers: The opportunity to benefit from the recent IoT development is considered as a trigger.

Analysts: The company has a dedicated product planning group for product development. In addition, they involve key accounts and dealers to product decisions.

Problem statement: A substantial part of company's offering is maintenance of the machines to ensure efficient operation and to reduce downtime. However, there is limited feedback to product development why a machine breaks down and what usage patterns led to it. IoT technology delivers real-time data on how machines are operated, thus enabling the company to actively prevent breakdowns and proactively improve the machines. This reduces maintenance costs and improves availability, leading to overall improvement of service quality and reduced operational costs.

We asked the participants (Analyst 1 and Analyst 2), with hindsight, to list and rank all utilized data sources. Their estimates are shown in Table 8.9. The participants provided their estimates on a scale 0-5, where 0

Table 8.9: We asked participants from Case II to evaluate the relevance to the given problem of each data source, before presenting our method. The participants were asked to assign a score 0-5, where denotes 0 the least and 5 the highest relevance, to each used data source. We compare these baseline results with the results from our method to evaluate its usefulness.

Data source	Scores		Normalized scores		Aggregated score
	Analyst 1	Analyst 2	Analyst 1	Analyst 2	
Key accounts (customers)	5	5	1.000	0.769	0.885
Customer clinics	2	1	0.400	0.154	0.277
Laws & regulations	4	5	0.800	0.769	0.785
Dealers	3	3	0.600	0.462	0.531
Operators (end users)	1	2	0.200	0.308	0.254
Product planning group	1	5	0.200	0.769	0.485
Regional partners	4	5	0.800	0.769	0.785

denotes the lowest, and 5 the highest relevance. We normalize their scores to arrive at scale of $[0, 1]$. The aggregated score is an arithmetic mean of the standardized scores. The estimates are provided on a scale 0 - 5, where 0 denotes the lowest, and 5 the highest contribution to the final solution.

Criteria: The participants needed guidance for selecting criteria, thus they adopted the high level criteria from Table 8.2. We show the exact criteria and participants estimates on their relevance to the problem, normalized scores, and final aggregated score in Table 8.10. The participant scores are normalized to arrive to scale $[0, 1]$, the aggregated score is the arithmetic mean of standardized scores.

Data sources: Participants reflected that the key source for ideas are dealers and key accounts. Dealers and regional partners work closely with end-customers and have the first hand knowledge of their actual needs. Furthermore, key accounts and dealers provide substantial revenue for the company, thus there is an financial incentive to prioritize their input. Other sources of input are industry standards, regulations, engineering, direct

Table 8.10: Participant estimates on criteria importance from Case II. The participatns were asked to evaluate each criterion on a scale 0-5,where 0 denotes lowest and 5 the highest importance to the given problem. We also show normalized scores and the final aggregated score.

Criterion	Scores		Normalized scores		Aggregated score
	Analyst 1	Analyst 2	Analyst 1	Analyst 2	
Knowledge	4	5	0.752	1.00	0.876
Amount of experience	4	4	0.752	0.800	0.776
Revenue	5	4	0.939	0.800	0.870
Cost	5	4	0.939	0.800	0.870
Opportunity cost	4	2	0.752	0.400	0.576
Risk	3	5	0.564	1.00	0.782
Timing	3	3	0.564	0.600	0.582
Ease of use	2	3	0.376	0.600	0.488
Suitability for collaboration	3	1	0.564	0.200	0.382

feedback from customers, workshops with machine operators, and engineers. We show the initial baseline ranking of data sources in Fig. 8.2, left side.

Application of the method: Method application run smooth and with only minor confusion. The first issue was to identify a problem to be studied, but after initial discussion the participants quickly agreed. The effort needed to get them acquainted was moderate and most time was dedicated towards individual estimated of the importance of each data source against the criteria (about 45 minutes on average for participant). Some additional effort was needed during the prioritization to explain the meaning of particular criteria for a particular data source.

Interpretation of the results:

We summarize the results in Table 8.11. Rows in the table are ordered by initial baseline ranking of data sources. We also show the ranking after applying the method and the difference between baseline and final ranking. A positive difference denotes that a data sources has gained relevance compared to the baseline, a negative difference denotes a reduced relevance.

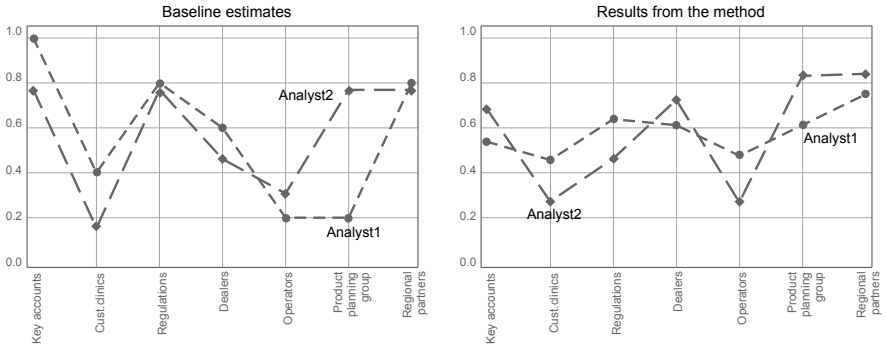


Figure 8.4: Results from the Case II. Figure on the left show baseline estimates, i.e. respondent estimates without the method. Figure on the right show results from applying the method. Y-axis denotes the relative importance of data sources.

The baseline estimates (Fig. 8.4, left side) show a discrepancy between the analysts regarding importance of the product planning group. However, application of the method (Fig. 8.4, right side) helps to arrive at more aligned results. The key accounts were estimated (without the method) to be the primary data source. However, the method suggests that input from regional partners, dealers, and product planning group should be prioritized.

Lessons learned: Participants reflections during the demonstration supported our earlier hypothesis that product decisions are often opinion based. Application of the method builds consensus and helps to reduce discrepancies.

We observed a need to accommodate a scenario where distributed analysts work independently and the method combines their input. This can be achieved by more stringent guidelines on how to select criteria and data sources for evaluation, and tool support to facilitate the method. Our participants saw great value in eliciting the criteria and consensus building among not only them, but also a larger group of stakeholders involved in decision making. In their opinion, that would greatly improve the selection of data sources.

6 Potential Improvements and Limitations

One of the central areas for improvements is providing better support for identifying relevant criteria and candidate data sources. Using overly broad criteria and data sources reduces usefulness of the results. However, identification and evaluations of many very specific terms may require excessive effort. To alleviate this shortcoming, we propose to iteratively ap-

Table 8.11: Aggregated results and their interpretation from Case II. The rows are ordered by the baseline estimates. We also show the resulting ranking from the method and the difference in ranking.

Data source	Ranking			Relevance		Disagreement	
	Baseline	Method	Difference	Score	Interpretation	Score	Interpretation
Key accounts	1	4	-3	0.769	high	0.144	negligible
Regional partners	2	1	+1	0.954	high	0.092	negligible
Regulations	3	5	-2	0.710	high	0.175	negligible
Dealers	4	3	+1	0.828	high	0.113	negligible
Product planning group	5	2	+3	0.881	high	0.219	negligible
Customer clinics	6	7	-1	0.524	medium	0.185	negligible
Operators	7	6	+1	0.532	medium	0.207	negligible

ply the method by increasing the level of detail with each iteration. We suggest to start by high level data sources and criteria, e.g., ones given in Tables 8.2 and 8.4. Then gradually eliminate less relevant criteria and detail the relevant ones, to arrive at specific data instances (e.g., Customer 1 and Customer 2) from high level classes (e.g., key customers).

We use ordinal scale from 0 to 5 in order to facilitate analysts in the task of ranking the criteria [Labovitz, 1970]. Alternative and more sophisticated ranking methods might also be used, for example, ones based on pairwise comparison [Barzilai, 1997] and large preference relation [Fodor et al., 1998]. However, evaluations in ordinal scale are more intuitive, faster, and require relatively fewer steps than the other two methods.

Another limitation is the use of arithmetic mean, which is well-known to be sensitive to extreme values. In general, the choice of aggregation operator is critical since some aggregation operators can lead to a significant loss of information since their values can be greatly influenced by extreme scores (e.g., the arithmetic mean), while others are penalizing too much for low-scoring outliers (e.g., the geometric mean and the harmonic mean) [Bullen et al., 2013]. A possible solution to the described problem is to use different aggregation operators in order to find some trade-off between their conflicting behaviour, e.g., hybrid aggregation operators proposed in [Tsiporkova and Boeva, 2004, 2006].

A potential scalability issue emerges when inputs from many analysts, a large number of criteria or data sources need to be considered. A large number of criteria (C) and data sources (D) require each analyst to provide $C + D \times C$ inputs. However, this limitation remains theoretical until full-scale validation of the method.

7 DISCUSSION

We structure our discussion around the research questions. We discuss the method with our initial objectives stated in Section 3.3 and lessons learned from the workshops.

7.1 *RQ1: What are the needs towards a method supporting selection of data sources for MDRE?*

We identify and formulate specific needs towards the new method in Sections 2 and 3.3. In this section, we revisit the needs and discuss to what extent our method addresses each specific requirement.

The review of related work in Section 2.2 reveals that most stakeholder selection practices consider requirements engineering as a fixed activity at the beginning of a project. Stakeholders are selected based on generalized characteristics, such as importance and influence on the project [Babar

et al., 2014b, 2015, Ballejos and Montagna, 2011]. Such an approach ignores the continuous nature of MDRE.

Discussions in the workshops confirmed that requirements engineering in a market-driven context is a continuous series of micro-decisions. The decisions pertaining to how to respond to market fluctuations, customer requests, new business objectives, to name a few. Different kinds of data may be needed to support each decision. Thus, the selection of data sources needs to be specific to each situation (*Objective 1*).

Our proposed method is flexible and context-independent. We specifically avoid coupling the method with any specific context, criteria, or data sources. Instead, we provide instructions on how to identify inputs for the method from its context of use. Thus, the method can be tailored for use in a wide range of situations. However, we exemplify some criteria and data sources to describe the method.

In a crowd requirements engineering context, analysts may benefit from a wide range of data sources, both within companies' control and beyond [Groen et al., 2017, Alexander, 2005]. Identification of new and unrealized sources is a crucial step in selecting an optimal set of data sources (*Objective 2*).

Our method addresses this objective by detailing the context of use and identifying data sources through a series of steps, namely describing the problem and setting forth the evaluation criteria of the sources. Such steps help to understand the context of use, thus supporting the identification of unrealized data sources.

An alternative approach could be to develop a taxonomy of data sources for crowd requirements engineering. However, at this stage of our research, we specifically avoid prescribing any specific sources to mitigate threats to construct validity.

Literature characterizes stakeholders in MDRE and crowd contexts with limited motivation to participate in product engineering and unable to articulate their actual needs. Thus, directly consulting specific stakeholder groups may not be worth the investment. In turn, stakeholder needs could be elicited indirectly, for example, by examining documents, artifacts, and data generated by stakeholders interacting with the product. The method should be able to handle a broad range of data sources to support the selection of data sources in such a context (*Objective 3*).

There are no inherent limitations on what types of data sources the method can handle. The difficulty could be to formulate evaluation criteria to suit both individuals and inanimate data sources. Formulation of criteria and data sources can be done iteratively. The initial set of criteria can help to understand the needs of data sources, thus supporting the identification of new data sources. Knowledge of what data sources need to be evaluated can help to arrive at more generalized criteria.

Requirements engineering is a collaborative activity, and product decisions could be cross-cutting and require consideration of multiple perspectives [Gorschek and Wohlin, 2006, Barczak et al., 2010]. Aggregating multiple perspectives and especially reaching consensus between opposing views is challenging. The method should support collaboration between multiple analysts (*Objective 4*).

The method has built-in support for collaboration and consensus development among multiple analysts. We use the arithmetic mean to combine views from multiple analysts. To explore different viewpoints, we propose to estimate the discrepancies and to use plots visualizing the differences.

A review of discrepancies and visualizing immediate results with plots contribute to the transparency and understandability of the results produced by the method (*Objective 5*).

An important concern is to what extent the benefits of using a method outweigh the resources needed to use it. Using our method requires several analysts to meet, set forth relevant criteria and candidate data sources, run the method, and discuss the results. We argue that such meetings already take place, and the use of the method adds structure to the meetings.

In both workshops, practitioners mentioned a committee that regularly meets and steers product decisions. Thus, using the method to support such meetings would require minimal additional effort. Not utilizing any structured approach could lead to unproductive discussions and suboptimal decisions. At a minimum, a suboptimal decision requires an immediate extra effort to correct it. However, it could also lead to wasted development effort and missed market opportunities. Therefore, potential benefits from using the method substantially outweigh the additional effort.

7.2 *RQ2: How to support selection of data sources for MDRE?*

Related work on decision-making scenarios identifies two main approaches. Group-based approaches use discussions and negotiations to arrive at the final decisions. However, such unstructured discussions could be unproductive, and the outcome could be skewed towards the views of more powerful group members. Individual-based approaches aggregate views from individuals regardless of their power. However, such an approach substantially limits the exchange of ideas and arguments in the group [Saaty, 1988, Tindale and Kameda, 2000].

Our method proposes to collect individual preferences in multiple steps and at each step, analyze the discrepancies between the preferences. In this way, we mitigate the adverse effects of group-based approaches and encourage focused discussions on specific disagreements.

7.3 *RQ3: What improvements are needed to use the method in industry?*

To identify opportunities for improvements, we summarize participant feedback and our lessons learned from the workshops. In both workshops, we identified the need for additional guidance for identifying and interpreting the evaluation criteria. Even though we exemplify some criteria in the method description, the selection of criteria was a difficulty. The support for criteria selection could be provided with, for example, a taxonomy of criteria and their descriptions.

In both workshops, the practitioners were able to quickly list relevant data sources based on their experience and our examples. However, the identification of candidate data sources could be supported further by a more extensive taxonomy.

For the workshops, we implemented the method using spreadsheets and a simple script. During the application of the method, participants were guided and supported by researchers. However, we identify the need for a robust tool guiding analysts through the steps of the method and minimizing the need for guidance from researchers.

8 CONCLUSIONS AND FURTHER WORK

In this paper, we have proposed and demonstrated a group-based decision support method for the selection of data sources in MDRE. The method comprises of systematic steps to collaboratively identify candidate data sources, evaluate them according to agreed criteria, and to produce a ranking of data sources by their relevance to the decision. We have paid particular attention to consensus building and provided means to analyze and understand the final results.

We have demonstrated the method on two case studies where we evaluated its usability and gather data for further refinement of the method. We learned that the method helps to build consensus between analysts and to arrive at a more consistent view of the importance of individual data sources. The systematic steps of the method enable analysts to analyze and remedy their disagreements.

Based on the results from demonstrations, we believe that our method can improve the selection of the data sources for MDRE by highlighting and helping to resolve discrepancies in the analyst's views. Thus, improving data quality for following requirements engineering activities and improving the overall quality of the final product.

Further work focuses on additional extensive validations and improvement rounds. We plan to: a) develop a tool to facilitate the use of the method and data collection on its application, and b) extend our support towards criteria and data sources identification.

BIBLIOGRAPHY

Crunchbase. https://www.crunchbase.com/. Accessed: 2015-10-01. (Cited on page 40.)

Radical innovation. https://www.innovation-creativity.com/radical-innovation.html. Accessed: 2018-10-05. (Cited on page 189.)

15 types of innovation. https://thegentleartofsmartstealing.wordpress.com/types-of-innovation/. Accessed: 2018-10-05. (Cited on page 189.)

The State of European Tech 2017. Technical report, Atomico, 2017. URL https://2017.stateofeuropeantech.com/. (Cited on pages 73 and 87.)

ABPM. *Guide to the business process management common body of knowledge (BPM CBOK®): ABPMP BPM CBOK®-[version 2.0-second release]*. ABPMP, 2009. (Cited on page 38.)

Nitin Agarwal and Urvashi Rathod. Defining 'success' for software projects: An exploratory revelation. *International journal of project management*, 24(4):358–370, 2006. (Cited on page 195.)

Agile Alliance. Agile glossary. https://www.agilealliance.org/agile101/agile-glossary/, 2018. [Online; accessed 20-April-2018]. (Cited on pages 162, 166, 173, 174, 175, 176, and 177.)

Alan Agresti. *An introduction to categorical data analysis*, volume 135. Wiley New York, 1996. (Cited on pages 98, 141, and 166.)

Hiva Alahyari, Richard Berntsson Svensson, and Tony Gorschek. A study of value in agile software development organizations. *Journal of Systems and Software*, 125:271–288, 2017. (Cited on pages 120 and 121.)

Ian F Alexander. A taxonomy of stakeholders: Human roles in system development. *International Journal of Technology and Human Interaction (IJTHI)*, 1(1):23–59, 2005. (Cited on pages 198, 200, and 229.)

Thomas A Alspaugh and Walt Scacchi. Ongoing software development without classical requirements. In *Requirements Engineering Conference (RE), 2013 21st IEEE International*, pages 165–174. IEEE, 2013. (Cited on page 196.)

Carina Alves, Silvia Pereira, and Jaelson Castro. A study in market-driven requirements engineering. *Workshop em Engenharia de Requisitos WERo6*, pages 2–3, 2006. (Cited on pages 7, 37, 58, 112, 117, 196, 197, and 198.)

Nicolli SR Alves, Thiago S Mendes, Manoel G de Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100–121, 2016. (Cited on page 138.)

SW Ambler. Lessons in agility from Internet-based development. *Software, IEEE*, 19(2):66 – 73, 2002. (Cited on pages 53 and 190.)

Marc Andreessen. Why software is eating the world. *Wall Street Journal*, 20 (2011):C2, 2011. (Cited on page 1.)

Nagappan Annamalai, Shahrul Kamaruddin, Ishak Abdul Azid, and TS Yeoh. Importance of problem statement in solving industry problems. In *Applied Mechanics and Materials*, volume 421, pages 857–863. Trans Tech Publ, 2013. (Cited on page 206.)

Fares Anwar and Rozilawati Razali. Stakeholders selection model for software requirements elicitation. *American Journal of Applied Sciences*, 13(6): 726–738, 2016. (Cited on page 196.)

Aybüke Aurum and Claes Wohlin. The fundamental nature of requirements engineering activities as a decision-making process. *Information and Software Technology*, 45(14):945–954, 2003. (Cited on page 199.)

Jim Azar, Randy K. Smith, and Davis Cordes. Value-oriented requirements prioritization in a small development organization. *IEEE Software*, 24(1): 32–37, 2007. ISSN 07407459. (Cited on pages 38, 43, and 66.)

Muhammad Imran Babar, Masitah Ghazali, and Dayang NA Jawawi. A bi-metric and fuzzy c-means based intelligent stakeholder quantification system for value-based software. In *SoMeT*, pages 295–309, 2014a. (Cited on pages 199 and 200.)

Muhammad Imran Babar, Masitah Ghazali, Dayang NA Jawawi, and Abubakar Elsafi. Stakeholder management in value-based software development: systematic review. *IET Software*, 8(5):219–231, 2014b. (Cited on pages 198 and 228.)

Muhammad Imran Babar, Masitah Ghazali, Dayang NA Jawawi, and Kashif Bin Zaheer. Stakemeter: Value-based stakeholder identification and quantification framework for value-based software systems. *PloS one*, 10(3), 2015. (Cited on pages 199, 200, and 229.)

Deepika Badampudi, Claes Wohlin, and Kai Petersen. Software component decision-making: In-house, oss, cots or outsourcing-a systematic literature review. *Journal of Systems and Software*, 121:105–124, 2016. (Cited on page 125.)

Sohaib Shahid Bajwa, Xiaofeng Wang, Anh Nguven Duc, and Pekka Abrahamsson. How do software startups pivot? empirical results from a multiple case study. In *International Conference of Software Business*, pages 169–176. Springer, 2016. (Cited on pages 91 and 154.)

Sohaib Shahid Bajwa, Xiaofeng Wang, Anh Nguyen Duc, and Pekka Abrahamsson. "failures" to be celebrated: an analysis of major pivots of software startups. *Empirical Software Engineering*, 22(5):2373–2408, 2017a. (Cited on page 34.)

Sohaib Shahid Bajwa, Xiaofeng Wang, Anh Nguyen Duc, Rafael Matone Chanin, Rafael Prikladnicki, Leandro Bento Pompermaier, and Pekka Abrahamsson. Start-ups must be ready to pivot. *IEEE Software*, 34(3): 18–22, 2017b. (Cited on page 82.)

Luciana C Ballejos and Jorge M Montagna. Modeling stakeholders for information systems design processes. *Requirements engineering*, 16(4): 281–296, 2011. (Cited on pages 199, 200, and 229.)

Muneera Bano, Didar Zowghi, and Naveed Ikram. Systematic reviews in requirements engineering: A tertiary study. In *2014 IEEE 4th International Workshop on Empirical Requirements Engineering (EmpiRE)*, pages 9–16. IEEE, 2014. (Cited on page 198.)

Gloria Barczak, Felicia Lassk, and Jay Mulki. Antecedents of team creativity: An examination of team emotional intelligence, team trust and collaborative culture. *Creativity and Innovation Management*, 19(4):332–345, 2010. ISSN 09631690. (Cited on page 230.)

Dorothy Leonard Barton. *Wellsprings of knowledge: Building and sustaining the sources of innovation*. Harvard Business School, 1995. (Cited on page 89.)

Jonathan Barzilai. Deriving weights from pairwise comparison matrices. *Journal of the operational research society*, 48(12):1226–1232, 1997. (Cited on page 228.)

Richard Baskerville, B. Ramesh, L. Levine, J. Pries-Heje, and S. Slaughter. Is internet-speed software development different? *IEEE Software*, 20 (November 2015):70–77, 2003. ISSN 0740-7459. (Cited on pages 34, 37, 86, and 128.)

Kênia P. Batista Webster, Káthia M. De Oliveira, and Nicolas Anquetil. A Risk Taxonomy Proposal for Software Maintenance. *IEEE International Conference on Software Maintenance, ICSM*, 2005:453–464, 2005. ISSN 1063-6773. (Cited on page 71.)

Kent Beck, Mike Beedle, Arie Van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al. Manifesto for agile software development. 2001. (Cited on pages 161, 163, and 178.)

Hakim Bendjenna, Pierre-Jean Charre, and Nacer Eddine Zarour. Using multi-criteria analysis to prioritize stakeholders. *Journal of Systems and Information Technology*, 14(3):264–280, 2012. (Cited on pages 199 and 200.)

Vebjørn Berg, Jørgen Birkeland, Anh Nguyen-Duc, Ilias Pappas, and Letizia Jaccheri. Software startup engineering: A systematic mapping study. *Journal of Systems and Software*, 2018. (Cited on pages 2, 8, 86, 87, 88, and 184.)

E Bjarnason, K Wnuk, and B Regnell. Overscoping: Reasons and consequences—A case study on decision making in software product management. *Software Product Management (IWSPM), 2010 Fourth International Workshop on*, pages 30–39, 2010a. (Cited on page 58.)

Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. Overscoping: Reasons and consequences—a case study on decision making in software product management. In *Software Product Management (IWSPM), 2010 Fourth International Workshop on*, pages 30–39. IEEE, 2010b. (Cited on page 119.)

SG Blank. *The four steps to the epiphany*. K&S Ranch; 2nd edition, 2013a. ISBN 0989200507. (Cited on pages 7, 37, 90, 113, and 129.)

Steve Blank. Embrace failure to start up success. *Nature*, 477(7363):133, sep 2011. ISSN 1476-4687. (Cited on pages 87 and 120.)

Steve Blank. Why the Lean Start Up Changes Everything. *Harvard Business Review*, 91(5):64, 2013b. ISSN 00178012. (Cited on pages 2, 8, 38, 73, 90, 111, 135, 159, 184, 188, and 191.)

NMP Bocken. Sustainable venture capital–catalyst for sustainable start-up success? *Journal of Cleaner Production*, 108:647–658, 2015. (Cited on page 130.)

Barry Boehm. Value-based software engineering: reinventing. *SIGSOFT Softw. Eng. Notes*, 28(2):3–, 2003. ISSN 0163-5948. (Cited on pages 38, 43, and 119.)

Kathleen Boies, John Fiset, and Harjinder Gill. Communication and trust are key: Unlocking the relationship between leadership and team performance and creativity. *The Leadership Quarterly*, 26(6):1080–1094, 2015. (Cited on page 109.)

Jan Bosch, Helena Holmström Olsson, Jens Björk, and Jens Ljungblad. The early stage software startup development model: a framework for operationalizing lean principles in software startups. In *International Conference on Lean Enterprise Software and Systems*, pages 1–15. Springer, 2013. (Cited on pages 8, 35, 36, and 129.)

M Broy. The 'Grand Challenge' in Informatics: Engineering Software-Intensive Systems. *Computer*, 39(10):72–80, 2006. ISSN 0018-9162. (Cited on page 37.)

Abdulaziz A Bubshait and Gulam Farooq. Team building and project success. *Cost engineering*, 41(7):34–38, 1999. (Cited on page 111.)

J Buchman and Christian Harsana Ekadharmawan. Barriers to sharing domain knowledge in software development practice in smes. In *Proceedings of the 3rd international workshop on knowledge collaboration in software development (KCSD2009)*, pages 2–16. Citeseer, 2009. (Cited on page 118.)

Barbara Budgen, David Turner, Mark Brereton, Pearl Kitchenham. Using Mapping Studies in Software Engineering. In *Proceedings of PPIG, 2008*, volume 2, pages 195–204, 2008. ISBN 978-1-86220-215-3. (Cited on page 38.)

Peter S Bullen, Dragoslav S Mitrinovic, and Means Vasic. *Means and their Inequalities*, volume 31. Springer Science & Business Media, 2013. (Cited on page 228.)

Corentin Burnay. Are stakeholders the only source of information for requirements engineers? toward a taxonomy of elicitation information sources. *ACM Transactions on Management Information Systems (TMIS)*, 7 (3):8, 2016. (Cited on page 200.)

P. Carlshamre and B. Regnell. Requirements lifecycle management and release planning in market-driven requirements engineering processes. *Proceedings - International Workshop on Database and Expert Systems Applications, DEXA*, 2000-Janua(September):961–965, 2000. ISSN 15294188. (Cited on page 57.)

Curtis R Carlson and William W Wilmot. *Innovation: The five disciplines for creating what customers want*. Crown Business, 2006. (Cited on pages 51 and 121.)

E Carmel. A discussion of special characteristics for software package development life cycle models. *ACM SIGSOFT Software Engineering Notes*, 1993. (Cited on page 1.)

Erran Carmel. Rapid development in software package startups. In *Proc. 27th Hawaii Int'l Conf. System Sciences*, 1994a. (Cited on pages 4, 36, 87, 121, 184, and 191.)

Erran Carmel. Time-to-completion in software package startups. In *1994 Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences*, 1994b. (Cited on pages 186, 189, and 190.)

Wayne F Cascio and Ramiro Montealegre. How technology is changing work and organizations. *Annual Review of Organizational Psychology and Organizational Behavior*, 3:349–375, 2016. (Cited on page 1.)

CBInsights. Startup Failure Post-Mortems, 2015. URL `https://www.cbinsights.com/blog/startup-failure-post-mortem/`. Accessed: April 15, 2015. (Cited on pages 39, 45, and 75.)

Lianping Chen. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, 32(2):50–54, 2015. (Cited on page 123.)

Ron Chernow. *Titan: The Life of John D. Rockefeller, Sr.* Vintage, 2007. (Cited on page 1.)

Schaul Chorev and Alistair R. Anderson. Success in Israeli high-tech startups; Critical factors and process. *Technovation*, 26(2):162–174, feb 2006. ISSN 01664972. (Cited on page 6.)

Tsun Chow and Dac-Buu Cao. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6):961–971, jun 2008. ISSN 01641212. (Cited on pages 37, 64, 106, 107, 109, 113, 130, 161, 163, 164, and 187.)

N Churchill and V Lewis. Five Stages of Small Business Growth. *Harvard Business Review*, 61(3):30–40, 1983. (Cited on pages xxi, 53, 88, and 90.)

M Hammar Cloyd. Designing user-centered web applications in web time. *IEEE software*, 18(1):62–69, 2001. (Cited on page 79.)

Paola Cocca and Marco Alberti. A framework to assess performance measurement systems in SMEs. *International Journal of Productivity and Performance Management*, 59(2):186 – 200, 2009. (Cited on pages 68 and 69.)

Alistair Cockburn and Jim Highsmith. Agile software development, the people factor. *Computer*, 34(11):131–133, 2001. (Cited on page 111.)

J Cohan. *Statistical power analysis for the behaviour sciences*. Hillsdale, nj: Erlbaum, 1988. (Cited on pages 98, 141, and 166.)

Gerry Coleman and Rory V. O'Connor. An investigation into software development process formation in software start-ups. *Journal of Enterprise Information Management*, 21(6):633–648, 2008. ISSN 1741-0398. (Cited on pages 8, 184, 187, 189, and 190.)

Eliane Figueiredo Collins et al. Software test automation practices in agile development environment: An industry experience report. In *Proceedings of the 7th International Workshop on Automation of Software Test*, pages 57–63. IEEE Press, 2012. (Cited on pages 124 and 163.)

Efthymios Constantinides, Carlota Lorenzo-Romero, and Miguel a. Gómez. Effects of web experience on consumer choice: a multicultural approach. *Internet Research*, 20(2):188–209, 2010. ISSN 1066-2243. (Cited on page 61.)

Juliet M. Corbin and Anselm Strauss. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology*, 13(1):3–21, 1990. ISSN 01620436. (Cited on pages 16, 42, 44, and 97.)

John W Creswell and J David Creswell. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017. (Cited on page 12.)

Giuseppe Criaco, Tommaso Minola, Pablo Migliorini, and Christian Serarols-Tarrés. "to have and have not": founders' human capital and university start-up survival. *The Journal of Technology Transfer*, 39(4):567–593, 2014. (Cited on page 65.)

Mark Crowne. Why software product startups fail and what to do about it. In *Engineering Management Conference*, pages 338–343, Cambridge, UK, 2002. IEEE. ISBN 0780373855. (Cited on pages xxi, 34, 36, 53, 71, 74, 86, 88, 90, 108, 136, 144, 155, and 169.)

Anna S Cui and Fang Wu. Utilizing customer knowledge in innovation: antecedents and impact of customer involvement on new product performance. *Journal of the academy of marketing science*, 44(4):516–538, 2016. (Cited on page 118.)

Åsa G Dahlstedt, Lena Karlsson, Anne Persson, J Natt och Dag, and Björn Regnell. Market-Driven Requirements Engineering Processes for Software Products - a Report on Current Practices. In *International Workshop on COTS and Product Software, RECOTS 2003*, 2003. (Cited on pages 7, 37, 38, 43, 52, 53, 54, 58, 112, 117, 155, 185, 196, and 198.)

Wayne W Daniel. Spearman rank correlation coefficient. *Applied nonparametric statistics*, pages 358–365, 1990. (Cited on page 168.)

Alan Davis, Oscar Dieste, Ann Hickey, Natalia Juristo, and Ana M. Moreno. Effectiveness of requirements elicitation techniques: Empirical results derived from a systematic review. *Proceedings of the IEEE International Conference on Requirements Engineering*, pages 176–185, 2006a. ISSN 1090705X. (Cited on page 56.)

Christopher J Davis, Robert M Fuller, Monica Chiarini Tremblay, and Donald J Berndt. Communication challenges in requirements elicitation and the use of the repertory grid technique. *Journal of Computer Information Systems*, 46(5):78–86, 2006b. (Cited on pages 56 and 196.)

Claudia O. De Melo, Daniela S. Cruzes, Fabio Kon, and Reidar Conradi. Interpretative case studies on agile team productivity and management. *Information and Software Technology*, 55(2):412–427, 2013. ISSN 09505849. (Cited on pages 65 and 155.)

Eric Deakins and Stuart Dillon. A helical model for managing innovative product and service initiatives in volatile commercial environments. *International Journal of Project Management*, 23(1):65–74, jan 2005. ISSN 02637863. (Cited on pages 35 and 36.)

Philipp Diebold and Marc Dahlem. Agile practices in practice: a mapping study. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 30. ACM, 2014. (Cited on page 161.)

Torgeir Dingsøyr, Tor Erlend Fægri, Tore Dybå, Børge Haugset, and Yngve Lindsjørn. Team performance in software development: research results versus agile principles. *IEEE Software*, 33(4):106–110, 2016. (Cited on page 104.)

A Dorling. SPICE: Software process improvement and capability dEtermination. *Information and Software Technology*, 35(6-7):404–406, jun 1993. ISSN 09505849. (Cited on page 38.)

Jorge Rômulo Frota Dos Santos, Adriano Bessa Albuquerque, and Plácido Rogério Pinheiro. Requirements prioritization in market-driven software: A survey based on large numbers of stakeholders and requirements. In *Quality of Information and Communications Technology (QUATIC), 2016 10th International Conference on the*, pages 67–72. IEEE, 2016. (Cited on page 117.)

Anh Nguven Duc, Yngve Dahle, Martin Steinert, and Pekka Abrahamsson. Towards understanding startup product development as effectual entrepreneurial behaviors. In *International Conference of Software Business*, pages 199–204. Springer, 2017. (Cited on page 184.)

Tore Dybå and Torgeir Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, 50 (9-10):833–859, 2008. (Cited on pages 163, 164, and 180.)

Tore Dybå, Torgeir Dingsøyr, and Nils Brede Moe. Agile project management. In *Software project management in a changing world*, pages 277–300. Springer, 2014. (Cited on page 132.)

Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008. (Cited on page 12.)

Evgenia Egorova, Marco Torchiano, Maurizio Morisio, Claes Wohlin, Aybuke Aurum, and Richard Berntsson Svensson. Stakeholders' perception of success: An empirical investigation. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 210–216. IEEE, 2009. (Cited on page 6.)

Kathleen M Eisenhardt. Building theories from case study research. *Academy of management review*, 14(4):532–550, 1989. (Cited on pages 92, 93, 94, and 99.)

Bill Elliott. Anything is possible: Managing feature creep in an innovation rich environment. In *Engineering Management Conference, 2007 IEEE International*, pages 304–307. IEEE, 2007. (Cited on pages 114 and 118.)

Suvi Elonen and Karlos A Artto. Problems in managing internal development projects in multi-project environments. *International journal of project management*, 21(6):395–402, 2003. (Cited on page 187.)

Martin J Eppler and Oliver Sukowski. Managing team knowledge: core processes, tools and enabling factors. *European Management Journal*, 18 (3):334–341, 2000. (Cited on page 111.)

Christophe Estay, François Durrieu, and Manzoom Akhter. Entrepreneurship: From motivation to start-up. *Journal of International Entrepreneurship*, 11(3):243–267, 2013. (Cited on page 188.)

Aleksander Fabijan, Helena Holmström Olsson, and Jan Bosch. Customer Feedback and Data Collection Techniques in Software R&D: A Literature Review. In *Icsob*, volume 114, pages 139–153, 2012. ISBN 978-3-642-30745-4. (Cited on pages 53, 55, and 56.)

Fabian Fagerholm and Jürgen Münch. Developer experience: Concept and definition. In *Proceedings of the International Conference on Software and System Process*, pages 73–77. IEEE Press, 2012. (Cited on page 64.)

Fabian Fagerholm, Marko Ikonen, Petri Kettunen, J??rgen M??nch, Virpi Roto, and Pekka Abrahamsson. Performance Alignment Work: How software developers experience the continuous adaptation of team performance in Lean and Agile environments. *Information and Software Technology*, 64:132–147, 2015. ISSN 09505849. (Cited on page 65.)

Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, and Jürgen Münch. The right model for continuous experimentation. *Journal of Systems and Software*, 123:292–305, 2017. (Cited on pages 7 and 130.)

Fabian Fagerholm, Arto Hellas, Matti Luukkainen, Kati Kyllönen, Sezin Yaman, and Hanna Mäenpää. Designing and implementing an environment for software start-up education: Patterns and anti-patterns. *Journal of Systems and Software*, 146:1–13, 2018. (Cited on page 5.)

Robert Feldt, Lefteris Angelis, Richard Torkar, and Maria Samuelsson. Links between the personalities, views and attitudes of software engineers. *Information and Software Technology*, 52(6):611–624, 2010. ISSN 09505849. (Cited on page 44.)

Janos Fodor, Sergei Orlovski, Patrice Perny, and Marc Roubens. *The Use of Fuzzy Preference Models in Multiple Criteria Choice, Ranking and Sorting*, pages 69–101. Springer US, Boston, MA, 1998. ISBN 978-1-4615-5645-9. doi: 10.1007/978-1-4615-5645-9_3. (Cited on page 228.)

Rebecca Ganzel. Putting out the welcome mat. *Training*, 35(3):54–60, 1998. (Cited on page 191.)

P Garengo, S Biazzo, and U Bititci. Performance measurement systems in SMEs: A review for a research agenda. *International Journal of Management Reviews*, 7(1):25–47, 2005. ISSN 1460-8545. (Cited on page 68.)

Vahid Garousi, Michael Felderer, and Mika V Mäntylä. The Need for Multivocal Literature Reviews in Software Engineering: Complementing Systematic Literature Reviews with Grey Literature. *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering*, pages 26:1—–26:6, 2016. (Cited on pages 35 and 40.)

Carmine Giardino, Michael Unterkalmsteiner, Nicolo Paternoster, Tony Gorschek, and Pekka Abrahamsson. What Do We Know about Software Development in Startups? *IEEE Software*, 31(5):28–32, sep 2014a. ISSN 0740-7459. (Cited on pages 4, 34, 36, 38, 57, 69, 74, 86, 87, 88, 118, 127, 130, 135, 137, 143, 159, 161, 164, 168, and 180.)

Carmine Giardino, Xiaofeng Wang, and Pekka Abrahamsson. Why Early-Stage Software Startups Fail : A Behavioral Framework. pages 27–41, 2014b. (Cited on pages 8, 37, and 184.)

Carmine Giardino, Sohaib Shahid Bajwa, and Xiaofeng Wang. Key Challenges in Early-Stage Software Startups. In *Agile Processes, in Software Engineering, and Extreme Programming*, volume 212, pages 52–63, 2015a. ISBN 978-3-319-18611-5. (Cited on pages 34, 35, 36, 86, 135, 137, 143, 159, 160, 161, 168, 184, 185, 187, 188, and 190.)

Carmine Giardino, Sohaib Shahid Bajwa, Xiaofeng Wang, and Pekka Abrahamsson. Key challenges in early-stage software startups. In *International Conference on Agile Software Development*, pages 52–63. Springer, 2015b. (Cited on pages 8 and 73.)

Carmine Giardino, Nicolò Paternoster, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. Software Development in Startup Companies: The Greenfield Startup Model. *IEEE Transactions on Software Engineering*, X(September):233, 2016. ISSN 0098-5589. (Cited on pages 2, 8, 34, 35, 64, 65, 70, 86, 88, 89, 100, 104, 107, 110, 121, 125, 126, 131, 132, 136, 137, 138, 139, 160, 161, 162, 163, 164, 178, 179, 184, 185, 188, and 190.)

Anandasivam Gopal, J Alberto Espinosa, Sanjay Gosain, and David P Darcy. Coordination and performance in global software service delivery: The vendor's perspective. *IEEE Transactions on Engineering Management*, 58(4):772–785, 2011. (Cited on page 109.)

T. Gorschek, C. Wohlin, Per Garre, and Stig Larrson. A model for technology transfer in practice. *Learning and Leading with Technology*, 30(4):88–95, 2006. (Cited on pages 203 and 204.)

Tony Gorschek and Claes Wohlin. Requirements abstraction model. *Requirements Engineering*, 11(1):79–101, 2006. ISSN 09473602. (Cited on page 230.)

Dorothy Graham. Requirements and testing: Seven missing-link myths. *IEEE Software*, 19(5):15–17, 2002. (Cited on page 70.)

Catarina Gralha, Daniela Damian, Anthony Wasserman, Miguel Goulão, and João Araújo. The evolution of requirements practices in software startups. In *Proceedings of the 40th International Conference on Software Engineering*, pages 823–833. ACM, 2018. (Cited on pages 7, 86, 89, 179, 184, 185, and 190.)

Lucas Gren, Richard Torkar, and Robert Feldt. The prospects of a quantitative measurement of agility: A validation study on an agile maturity model. *Journal of Systems and Software*, 107:38–49, 2015. (Cited on page 178.)

Eduard C Groen, Norbert Seyff, Raian Ali, Fabiano Dalpiaz, Joerg Doerr, Emitza Guzman, Mahmood Hosseini, Jordi Marco, Marc Oriol, Anna

Perini, et al. The crowd in requirements engineering: The landscape and challenges. *IEEE software*, 34(2):44–52, 2017. (Cited on pages 7, 196, 197, 198, 202, 207, and 229.)

Qing Gu, Michael Parkin, and Patricia Lago. A taxonomy of service engineering stakeholder types. In *European Conference on a Service-Based Internet*, pages 206–219. Springer, 2011. (Cited on page 198.)

Shelby J Haberman. The analysis of residuals in cross-classified tables. *Biometrics*, pages 205–220, 1973. (Cited on pages 98 and 141.)

Irit Hadar, Pnina Soffer, and Keren Kenzi. The role of domain knowledge in requirements elicitation via interviews: an exploratory study. *Requirements Engineering*, 19(2):143–159, 2014. (Cited on pages 111 and 119.)

Beth Hadley, Peter A Gloor, Stephanie L Woerner, and Yuhong Zhou. Analyzing vc influence on startup success: They might not be good for you. In *Proceedings of COINs Conference, 2017*, 2017. (Cited on page 187.)

Paul N Hague, Nicholas Hague, and Carol-Ann Morgan. *Market research in practice: a guide to the basics*. Kogan Page Publishers, 2004. (Cited on page 52.)

Fadly Hamka, Harry Bouwman, Mark De Reuver, and Maarten Kroesen. Mobile customer segmentation based on smartphone measurement. *Telematics and Informatics*, 31(2):220–227, 2014. (Cited on page 196.)

Geir Kjetil Hanssen and Tor Erlend Fægri. Agile customer engagement. *Proceedings of the 2006 ACM/IEEE international symposium on International symposium on empirical software engineering - ISESE '06*, page 164, 2006. (Cited on page 60.)

Nils C. Haugen. An empirical study of using planning poker for user story estimation. *Proceedings - AGILE Conference, 2006*, 2006:23–31, 2006. (Cited on page 69.)

Brian Headd. Redefining business success: Distinguishing between closure and failure. *Small business economics*, 21(1):51–61, 2003. (Cited on page 191.)

Geoffrey Hecht, Omar Benomar, Romain Rouvoy, Naouel Moha, and Laurence Duchien. Tracking the software quality of android applications along their evolution (t). In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 236–247. IEEE, 2015. (Cited on page 155.)

Ilja Heitlager, Remko Helms, and Sjaak Brinkkemper. A tentative technique for the study and planning of co-evolution in product. In *Software*

Evolvability, 2007 Third International IEEE Workshop on, pages 42–47. IEEE, 2007. (Cited on page 189.)

H F Hofmann and F Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, 18(4):58–66, 2001a. ISSN 07407459. (Cited on page 53.)

Hubert F Hofmann and Franz Lehner. Requirements engineering as a success factor in software projects. *IEEE software*, (4):58–66, 2001b. (Cited on page 196.)

Laura Hokkanen and Kaisa Väänänen-Vainio-Mattila. Ux work in startups: Current practices and future needs. In Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 81–92, Cham, 2015a. Springer International Publishing. ISBN 978-3-319-18612-2. (Cited on page 86.)

Laura Hokkanen and Kaisa Väänänen-Vainio-Mattila. Ux work in startups: current practices and future needs. In *International Conference on Agile Software Development*, pages 81–92. Springer, 2015b. (Cited on page 127.)

Laura Hokkanen, Kati Kuusinen, and Kaisa Väänänen. Minimum viable user experience: A framework for supporting product design in startups. In *International Conference on Agile Software Development*, pages 66–78. Springer, 2016. (Cited on pages 89, 127, and 128.)

Adery CA Hope. A simplified monte carlo significance test procedure. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 582–598, 1968. (Cited on pages 98, 141, and 166.)

Debra Howcroft. After the goldrush: deconstructing the myths of the dot. com market. *Journal of Information Technology*, 16(4):195–204, 2001. (Cited on page 1.)

Wayne D Hoyer, Rajesh Chandy, Matilda Dorotic, Manfred Krafft, and Siddharth S Singh. Consumer cocreation in new product development. *Journal of service research*, 13(3):283–296, 2010. (Cited on pages 113 and 118.)

Fadhl Hujainah, Rohani Binti Abu Baka, Basheer Al-Haimi, and Mansoor Abdullateef Abdulgabber. Stakeholder quantification and prioritisation research: A systematic literature review. *Information and Software Technology*, 2018. (Cited on page 198.)

Antti Hyrkäs et al. Startup complexity: Tracing the conceptual shift behind disruptive entrepreneurship. *Publications of the Faculty of Social Sciences*, 2016. (Cited on page 2.)

IEEE Computer Society. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0*. IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014. ISBN 0769551661, 9780769551661. (Cited on pages 9, 53, 54, 56, 59, 60, 63, 65, 67, 69, 70, 88, 107, 111, 196, and 197.)

Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an agile methodology implementation. In *Proceedings. 30th Euromicro Conference, 2004.*, pages 326–333. IEEE, 2004. (Cited on page 162.)

ISO/IEC 42010. ISO/IEC/IEEE 42010 Systems and Software engineering - arcthitecture description, 2011. (Cited on pages 5 and 38.)

Martin Ivarsson and Tony Gorschek. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16(3):365–395, oct 2010. ISSN 1382-3256. (Cited on pages 5, 20, and 22.)

Clemente Izurieta and James M Bieman. A multiple case study of design pattern decay, grime, and rot in evolving software systems. *Software Quality Journal*, 21(2):289–323, 2013. (Cited on page 156.)

Samireh Jalali and Claes Wohlin. Global software engineering and agile practices: a systematic review. *Journal of software: Evolution and Process*, 24(6):643–659, 2012. (Cited on page 161.)

S Jansen, K M Popp, and P Buxmann. The Sun also Sets: Ending the Life of a Software Product. *Software Business*, 80:154–167, 2011. (Cited on page 69.)

Antero Järvi, Ville Taajamaa, and Sami Hyrynsalmi. Lean software startup–an experience report from an entrepreneurial software business course. In *International Conference of Software Business*, pages 230–244. Springer, 2015. (Cited on page 5.)

Jian Jin, Ying Liu, Ping Ji, and Hongguang Liu. Understanding big consumer opinion data for market-driven product design. *International Journal of Production Research*, 54(10):3019–3041, 2016. (Cited on pages 197 and 202.)

Maggie Johnson and Max Senges. Learning to be a programmer in a complex organization: A case study on practice-based learning during the onboarding process at google. *Journal of Workplace Learning*, 22(3):180–194, 2010. (Cited on page 191.)

Mona Erfani Joorabchi, Ali Mesbah, and Philippe Kruchten. Real challenges in mobile app development. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 15–24. IEEE, 2013. (Cited on page 155.)

Kristine Jørgensen. Newcomers in a global industry: Challenges of a norwegian game company. *Games and Culture*, page 1555412017723265, 2017. (Cited on page 187.)

William S Junk. The Dynamic Balance Between Cost, Schedule, Features, and Quality in Software Development Projects. *Computer Science Dept., University of Idaho*, SEPM-001, 2000. (Cited on pages 56, 90, 112, 130, and 187.)

Veerapaneni Esther Jyothi and K Nageswara Rao. Effective implementation of agile practices-incoordination with lean kanban. *International Journal on Computer Science and Engineering*, 4(1):87, 2012. (Cited on page 161.)

M. Kakati. Success criteria in high-tech new ventures. *Technovation*, 23(5): 447–457, may 2003. ISSN 01664972. (Cited on pages 189 and 190.)

Lena Karlsson, Åsa G Dahlstedt, Björn Regnell, Johan Natt och Dag, and Anne Persson. Requirements engineering challenges in market-driven software development–an interview study with practitioners. *Information and Software technology*, 49(6):588–604, 2007. (Cited on pages 38, 43, 56, 113, 196, 197, 198, and 202.)

Rilla Khaled, Pippin Barr, James Noble, and Robert Biddle. System metaphor in extreme programming: A semiotic approach. In *7th Internat. Workshop Organ. Semiotics*. Citeseer, 2004. (Cited on page 163.)

Hammad Khalid, Emad Shihab, Meiyappan Nagappan, and Ahmed E Hassan. What do mobile app users complain about? *IEEE Software*, 32(3): 70–77, 2014. (Cited on page 81.)

Rao Aamir Khan and Konrad Spang. Critical success factors for international projects. In *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2011 IEEE 6th International Conference on*, volume 2, pages 879–883. IEEE, 2011. (Cited on page 64.)

Mahvish Khurum, Tony Gorschek, and Magnus Wilson. The software value map—an exhaustive collection of value aspects for the development of software intensive products. *JOURNAL OF SOFTWARE-EVOLUTION AND PROCESS*, (July 2010):481–491, 2012. ISSN 20477481. (Cited on pages 57, 118, 119, and 121.)

D Kirk and S G MacDonell. Categorising software contexts. *20th Americas Conference on Information Systems, AMCIS 2014*, (April), 2014. (Cited on page 94.)

Barbara Kitchenham and Shari Lawrence. Software quality: the elusive target. *IEEE Software*, 1(January):12–21, 1996. (Cited on pages 6 and 65.)

Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: the elusive target [special issues section]. *IEEE software*, 13(1):12–21, 1996. (Cited on page 162.)

Hans-Bernd Kittlaus and Samuel A Fricker. *Software Product Management*. Springer, 2015. (Cited on page 197.)

E Klotins, M Unterkalmsteiner, and T Gorschek. Software engineering in start-up companies: an exploratory study of 88 startups. *Empirical Software Engineering*, 2016. (Cited on page 143.)

E Klotins, M Unterkalmsteiner, and T Gorschek. Software Engineering Anti-patterns in start-ups. *In review by IEEE Software*, 2017. (Cited on pages 87, 89, 136, 137, 161, 184, 197, and 198.)

Eriks Klotins. Using the case survey method to explore engineering practices in software start-ups. In *Proceedings of the 1st International Workshop on Software Engineering for Startups*, pages 24–26. IEEE Press, 2017. (Cited on pages 140 and 165.)

Eriks Klotins. Software start-ups through an empirical lens: are start-ups snowflakes? In *SiBW*, pages 1–14, 2018. (Cited on page 161.)

Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. Software Engineering Knowledge Areas in Startup Companies : a mapping study. In *Lecture Notes in Business Information Processing*, pages 245–257. Springer, 2015. (Cited on pages 2, 8, 34, 35, 36, 88, 94, and 184.)

Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladnicki, Nirnaya Tripathi, and Leandro Pompermaier. Exploration of technical debt in start-ups. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 75–84. IEEE, 2018a. (Cited on pages 86, 89, 124, 126, 164, 184, and 190.)

Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. Software-intensive product engineering in start-ups: a taxonomy. *IEEE Software*, 35(4), 2018b. (Cited on pages 2, 8, 13, 86, 89, 94, 95, and 184.)

Eriks Klotins, Michael Unterkalmsteiner, Panagiota Chatzipetrou, Tony Gorschek, Rafael Prikladniki, Nirnaya Tripathi, and Leandro Pompermaier. A progression model of software engineering goals, challenges, and practices in start-ups. *IEEE Transactions on Software Engineering*, 2019a. (Cited on pages 160, 164, 165, 178, 184, 198, and 202.)

Eriks Klotins, Michael Unterkalmsteiner, and Tony Gorschek. Software engineering in start-up companies: An analysis of 88 experience reports.

Empirical Software Engineering, 24(1):68–102, 2019b. (Cited on pages 160, 164, 168, 184, 185, 197, 198, and 202.)

Pavneet Singh Kochhar, Ferdian Thung, Nachiappan Nagappan, Thomas Zimmermann, and David Lo. Understanding the test automation culture of app developers. In *Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on*, pages 1–10. IEEE, 2015. (Cited on pages 150 and 155.)

Gerald Kotonya and Ian Sommerville. *Requirements engineering: processes and techniques*. Wiley Publishing, 1998. (Cited on page 7.)

Mayuram S Krishnan. The role of team factors in software cost and quality: An empirical analysis. *Information Technology & People*, 11(1):20–35, 1998. (Cited on page 108.)

Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2000. ISBN 0201707101. (Cited on page 196.)

Philippe Kruchten, Robert L. Nord, and Ipek Ozkaya. Technical Debt : From Metaphor. *IEEE Software*, pages 18–22, 2012. (Cited on pages 136 and 138.)

Justin Kruger and David Dunning. Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments., 1999. ISSN 0022-3514. (Cited on page 107.)

S. Kujala. Effective user involvement in product development by improving the analysis of user needs. *Behaviour & Information Technology*, 27(6): 457–473, 2008. ISSN 0144-929X. (Cited on page 56.)

Antonella La Rocca, Paolo Moscatelli, Andrea Perna, and Ivan Snehota. Customer involvement in new product development in b2b: The role of sales. *Industrial Marketing Management*, 58:45–57, 2016. (Cited on pages 114 and 196.)

Sanford Labovitz. The assignment of numbers to rank order categories. *American sociological review*, pages 515–524, 1970. (Cited on page 228.)

Erkki K Laitinen. A dynamic performance measurement system: evidence from small finnish technology companies. *Scandinavian journal of management*, 18(1):65–99, 2002. (Cited on page 6.)

R. Larsson. Case Survey Methodology: Quantitative Analysis of Patterns Across Case Studies. *Academy of Management Journal*, 36(6):1515–1546, 1993. ISSN 0001-4273. (Cited on pages 91, 92, 93, 100, 140, and 165.)

Soren Lauesen. *Software Requirements: Styles and Techniques*. Pearson Education, 1 edition, 2001. ISBN 0201745704. (Cited on page 196.)

Lucas Layman, Laurie Williams, and Lynn Cunningham. Exploring extreme programming in context: an industrial case study. In *Agile Development Conference*, pages 32–41. IEEE, 2004. (Cited on page 162.)

L Lehtola, M Kauppinen, and S Kujala. Linking the business view to requirements engineering: long-term product planning by roadmapping. *13th IEEE International Conference on Requirements Engineering RE05*, pages 439–443, 2005. ISSN 1090705X. (Cited on page 57.)

Timothy C Lethbridge, Susan Elliott Sim, and Janice Singer. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering*, 10(3):311–341, 2005. (Cited on pages 14, 15, and 40.)

Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220, 2015. ISSN 01641212. (Cited on pages 136, 138, 139, 142, 147, and 148.)

Soo Ling Lim, Daniele Quercia, and Anthony Finkelstein. Stakenet: using social networks to analyse the stakeholders of large-scale software projects. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 295–304. ACM, 2010. (Cited on pages 199 and 200.)

Sau-ling LAI Linda. Chinese entrepreneurship in the internet age: Lessons from alibaba. com. *International Science Index, Economics and Management Engineering*, 4(12), 2010. (Cited on page 189.)

Eveliina Lindgren and Jürgen Münch. Raising the odds of success: the current state of experimentation in product development. *Information and Software Technology*, 77:80–91, 2016. (Cited on page 187.)

David C Logan. Known knowns, known unknowns, unknown unknowns and the propagation of scientific enquiry. *Journal of experimental botany*, 60(3):712–714, 2009. (Cited on page 9.)

Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Forging high-quality user stories: towards a discipline for agile requirements. In *2015 IEEE 23rd international requirements engineering conference (RE)*, pages 126–135. IEEE, 2015. (Cited on page 163.)

Walid Maalej, Maleknaz Nayebi, Timo Johann, and Guenther Ruhe. Toward data-driven requirements engineering. *IEEE Software*, 33(1):48–54, 2016. (Cited on pages 196 and 198.)

Mika Mantyla, Jari Vanhanen, and Casper Lassenius. A taxonomy and an initial empirical study of bad smells in code. *International Conference on Software Maintenance, 2003. ICSM 2003. Proceedings.*, (OCTOBER):381–384, 2003. ISSN 1063-6773. (Cited on pages 148 and 163.)

Steven Maranville. Entrepreneurship in the business curriculum. *Journal of Education for Business*, 68(1):27–31, 1992. (Cited on page 125.)

Martin N Marshall. Sampling for qualitative research. *Family practice*, 13 (6):522–526, 1996. (Cited on page 22.)

Nikolaos F Matsatsinis and Andreas P Samaras. Mcda and preference disaggregation in group decision support systems. *European Journal of Operational Research*, 130(2):414–429, 2001. (Cited on pages 199 and 216.)

Beverly May. Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran: Lessons Learned in Creating & Launching a Complex Consumer App. In *Agile Conference*, pages 141–147. Ieee, aug 2012. ISBN 978-1-4673-2622-3. (Cited on pages 8, 37, 61, 187, 189, and 190.)

John McManus. A stakeholder perspective within software engineering projects. In *2004 IEEE International Engineering Management Conference (IEEE Cat. No. 04CH37574)*, volume 2, pages 880–884. IEEE, 2004. (Cited on page 200.)

Jorge Melegati, Alfredo Goldman, and S Paulo. Requirements engineering in software startups: A grounded theory approach. In *2nd International Workshop on Software Startups, Trondheim, Norway*, 2016a. (Cited on page 36.)

Jorge Melegati, Alfredo Goldman, and São Paulo. Requirements Engineering in Software Startups : a Grounded Theory approach. In *2nd International Workshop on Software Startups, Trondheim, Norway*, 2016b. (Cited on pages 79, 86, 89, 94, 118, and 190.)

Andreas Metzger and Klaus Pohl. Software product line engineering and variability management: achievements and challenges. In *Proceedings of the on Future of Software Engineering*, pages 70–84. ACM, 2014. (Cited on page 137.)

Granville Miller and Laurie Williams. Personas: Moving Beyond Role-Based Requirements Engineering. *Microsoft and North Carolina State ...*, pages 1–10, 2006. (Cited on page 55.)

Subhas Misra, Vinod Kumar, Uma Kumar, Kamel Fantazy, and Mahmud Akhter. Agile software development practices: evolution, principles, and criticisms. *International Journal of Quality & Reliability Management*, 29(9): 972–980, 2012. ISSN 0265-671X. (Cited on pages 161 and 164.)

Ronald K. Mitchell, Bradley R. Agle, and Donna J. Wood. Toward a theory of stakeholder identification and salience: Defining the principle of who and what really counts. *The Academy of Management Review*, 22(4):853–886, 1997. ISSN 03637425. (Cited on page 207.)

Ian I Mitroff. *Stakeholders of the organizational mind*. Jossey-Bass San Francisco, 1983. (Cited on page 55.)

Øystein Moen, Morten Gavlen, and Iver Endresen. Internationalization of small, computer software firms: Entry forms and market selection. *European Journal of Marketing*, 38(9/10):1236–1251, 2004. (Cited on page 1.)

K. Molokken and M. Jorgensen. A review of software surveys on software effort estimation. *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, (1325), 2003a. (Cited on page 69.)

Kjetil Molokken and Magne Jorgensen. A review of software surveys on software effort estimation. In *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on*, pages 223–230. IEEE, 2003b. (Cited on page 130.)

Raimund Moser, Pekka Abrahamsson, Witold Pedrycz, Alberto Sillitti, and Giancarlo Succi. A case study on the impact of refactoring on quality and productivity in an agile team. In *IFIP Central and East European Conference on Software Engineering Techniques*, pages 252–266. Springer, 2007. (Cited on page 163.)

Ram Mudambi and Monica Zimmerman Treichel. Cash crisis in newly public internet-based firms: An empirical analysis. *Journal of Business Venturing*, 20(4):543–571, 2005. (Cited on page 187.)

Hussan Munir, Krzysztof Wnuk, and Per Runeson. Open innovation in software engineering: a systematic mapping study. *Empirical Software Engineering*, 21(2):684–723, 2016. (Cited on page 79.)

Mohd Hairul Nizam Nasir and Shamsul Sahibuddin. Critical success factors for software projects: A comparative study. *Scientific research and essays*, 6(10):2174–2186, 2011. (Cited on page 130.)

Anh Nguyen-Duc, Xiaofeng Wang, and Pekka Abrahamsson. What influences the speed of prototyping? an empirical investigation of twenty software startups. In *International Conference on Agile Software Development*, pages 20–36. Springer, 2017. (Cited on page 188.)

Berrt Ogawa, R. T.; Malen. Towards Rigor in Reviews of Multivocal Literatures: Applying the Exploratory Case Study Method. *Review of Educational Research*, 61(3):265–286, 1991. ISSN 0034-6543. (Cited on page 40.)

Chitu Okoli and Suzanne D Pawlowski. The delphi method as a research tool: an example, design considerations and applications. *Information & management*, 42(1):15–29, 2004. (Cited on page 11.)

Helena Holmström Olsson and Jan Bosch. From opinions to data-driven software r&d: A multi-case study on how to close the'open loop'problem. In *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, pages 9–16. IEEE, 2014. (Cited on page 115.)

Helena Holmström Olsson and Jan Bosch. Towards Continuous Customer Validation: A Conceptual Model for Combining Qualitative Customer Feedback with Quantitative Customer Observation. In *Icsob*, volume 114, pages 261–266, 2015. ISBN 978-3-642-30745-4. (Cited on pages 37, 60, and 68.)

Aamir Omair et al. Sample size estimation and sampling techniques for selecting a representative sample. *Journal of Health specialties*, 2(4):142, 2014. (Cited on page 22.)

Alexander Osterwalder, Yves Pigneur, and Christopher Tucci. Clarifying business models: origins, present, and future of the concept. *Communications of the Association for Information Systems*, 15:1–43, 2005. ISSN 15293181. (Cited on page 38.)

English Oxford. *Oxford english dictionary*. 1989. (Cited on page 5.)

Muammer Ozer and Doug Vogel. Contextualized relationship between knowledge sharing and performance in software development. *Journal of Management Information Systems*, 32(2):134–161, 2015. (Cited on page 178.)

J. D. Kim J. Miranda P. Azoulay, B. Jones. Research: The average age of a successful startup founder is 45. *Harvard Business Review*, Jul 2018. (Cited on pages 115, 187, 188, and 190.)

Carla Pacheco and Ivan Garcia. A systematic literature review of stakeholder identification methods in requirements elicitation. *Journal of Systems and Software*, 85(9):2171–2181, 2012. ISSN 01641212. (Cited on pages 55, 197, and 198.)

Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrea De Lucia. Do they really smell bad? A study on developers' perception of bad code smells. *Proceedings - 30th International Conference on Software Maintenance and Evolution, ICSME 2014*, (November 2016): 101–110, 2014a. ISSN 1063-6773. (Cited on page 148.)

Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrea De Lucia. Do they really smell bad? a study on developers'

perception of bad code smells. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 101–110. IEEE, 2014b. (Cited on page 163.)

Jevgenija Pantiuchina, Marco Mondini, Dron Khanna, Xiaofeng Wang, and Pekka Abrahamsson. Are software startups applying agile practices? the state of the practice from a large survey. In *International Conference on Agile Software Development*, pages 167–183. Springer, Cham, 2017. (Cited on page 160.)

Chetankumar Patel and Muthu Ramachandran. Agile maturity model (amm): A software process improvement framework for agile software development practices. *International Journal of Software Engineering, IJSE*, 2(1):3–28, 2009. (Cited on page 30.)

Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. Software development in startup companies: A systematic mapping study. *Information and Software Technology*, 56(10):1200–1218, oct 2014. ISSN 09505849. (Cited on pages 8, 15, 27, 34, 35, 86, 88, 94, 100, 109, 131, 135, 160, 183, 185, 186, 188, 189, 190, 191, 192, 193, and 196.)

John W Payne, James R Bettman, and Mary Frances Luce. When time is money: Decision behavior under opportunity-cost time pressure. *Organizational behavior and human decision processes*, 66(2):131–152, 1996. (Cited on page 188.)

Kai Petersen and Claes Wohlin. Context in industiral software engineering research. . . . *Symposium on Empirical Software Engineering . . .* , 2009. (Cited on pages 8 and 199.)

Kai Petersen, Deepika Badampudi, Syed Shah, Krzysztof Wnuk, Tony Gorschek, Efi Papatheocharous, Jakob Axelsson, Severine Sentilles, Ivica Crnkovic, and Antonio Cicchetti. Choosing Component Origins for Software Intensive Systems: In-house, COTS, OSS or Outsourcing? – A Case Survey. *IEEE Transactions on Software Engineering*, 44(2):237–261, 2017a. ISSN 0098-5589. doi: 10.1109/TSE.2017.2677909. URL http://ieeexplore.ieee.org/document/7870688/. (Cited on pages 13, 77, and 87.)

Kai Petersen, Deepika Badampudi, Syed Shah, Krzysztof Wnuk, Tony Gorschek, Efi Papatheocharous, Jakob Axelsson, Severine Sentilles, Ivica Crnkovic, and Antonio Cicchetti. Choosing component origins for software intensive systems: In-house, cots, oss or outsourcing?–a case survey. *IEEE Transactions on Software Engineering*, 2017b. (Cited on pages 91, 125, 128, and 140.)

Raphael Pham, Stephan Kiesling, Leif Singer, and Kurt Schneider. Onboarding inexperienced developers: struggles and perceptions regarding automated testing. *Software Quality Journal*, 25(4):1239–1268, 2017. (Cited on page 124.)

Inc. PitchBook Data. European Middle Market Report 2H 2015. 2015. (Cited on pages 34, 35, 137, and 161.)

PitchBook Data, Inc. U.S. Middle market report Q4 2015. Technical report, PitchBook, 2015. (Cited on pages 34, 35, 137, and 161.)

Diamanto Politis. Does prior start-up experience matter for entrepreneurs' learning? a comparison between novice and habitual entrepreneurs. *Journal of small business and Enterprise Development*, 15(3):472–489, 2008. (Cited on pages 106, 110, and 190.)

O. Preiss and A. Wegmann. Stakeholder discovery and classification based on systems science principles. In *Proceedings of the Second Asia-Pacific Conference on Quality Software*, APAQS '01, pages 194–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1287-9. (Cited on page 196.)

Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, 2001. ISBN 0072496681. (Cited on page 196.)

E. Pronin, D. Y. Lin, and L. Ross. The Bias Blind Spot: Perceptions of Bias in Self Versus Others. *Personality and Social Psychology Bulletin*, 28(3): 369–381, 2002. ISSN 0146-1672. (Cited on pages 21 and 44.)

John Pruitt and Jonathan Grundin. Personas : Practice and Theory. *Proceedings of the 2003 conference on Designing for user experiences*, pages 1–15, 2003. (Cited on pages 55 and 56.)

Usman Rafiq, Sohaib Shahid Bajwa, Xiaofeng Wang, and Ilaria Lunesu. Requirements elicitation techniques applied in software startups. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 141–144. IEEE, 2017. (Cited on page 36.)

Paul Ralph and Paul Kelly. The dimensions of software engineering success. In *Proceedings of the 36th International Conference on Software Engineering*, pages 24–35. ACM, 2014. (Cited on pages 6 and 125.)

Balasubramaniam Ramesh, Lan Cao, and Richard Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, nov 2007. ISSN 13501917. (Cited on page 37.)

Rozilawati Razali and Fares Anwar. Selecting the right stakeholders for requirements elicitation: a systematic approach. *Journal of Theoretical and Applied Information Technology*, 33(2):250–257, 2011. (Cited on pages 198, 199, and 200.)

John S Reel. Critical success factors in software projects. *IEEE software*, 16 (3):18–23, 1999. (Cited on page 130.)

Björn Regnell and Sjaak Brinkkemper. *Market-Driven Requirements Engineering for Software Products*, pages 287–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005a. ISBN 978-3-540-28244-0. doi: 10.1007/3-540-28244-0_13. URL https://doi.org/10.1007/3-540-28244-0_13. (Cited on pages 196 and 197.)

Björn Regnell and Sjaak Brinkkemper. Market-driven requirements engineering for software products. In *Engineering and managing software requirements*, pages 287–308. Springer, 2005b. (Cited on page 7.)

Björn Regnell, Richard Berntsson Svensson, and Thomas Olsson. Supporting roadmapping of quality requirements. *IEEE Software*, 25:42–47, 2008. ISSN 07407459. (Cited on pages 66, 70, 79, 122, and 124.)

Gavin C Reid and Julia A Smith. What makes a new business start-up successful? *Small Business Economics*, 14(3):165–182, 2000. (Cited on page 191.)

Helen Reijonen and Raija Komppula. Perception of success and its effect on small firm performance. *Journal of Small Business and Enterprise Development*, 14(4):689–701, 2007. (Cited on page 6.)

Randall W Rice, CSTE CSQA, and LLC Rice Consulting Solutions. Surviving the top ten challenges of software test automation. *CrossTalk: The Journal of Defense Software Engineering*, pages 26–29, 2003. (Cited on page 155.)

Norman Riegel and Joerg Doerr. A systematic literature review of requirements prioritization criteria. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 300–317. Springer, 2015. (Cited on page 208.)

Eric Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business; First Edition, 2011. (Cited on pages 2, 4, 7, 8, 37, 129, and 184.)

Linda Rising and Norman S Janoff. The scrum software development process for small teams. *IEEE software*, 17(4):26–32, 2000. (Cited on pages 59, 82, 161, and 164.)

C Robson. Real world research: A resource for social scientists and practitioner-researchers., 2nd edn.(blackwell publishing: Oxford, uk.). 2002. (Cited on pages 11 and 16.)

Jorge L Romeu. A simulation approach for the analysis and forecast of software productivity. *Computers & industrial engineering*, 9(2):165–174, 1985. (Cited on page 11.)

P Runeson, C Andersson, T Thelin, a Andrews, and T Berling. What do we know about defect detection methods? [software testing]. *Software, IEEE*, 23(3):82–90, 2006. ISSN 0740-7459. (Cited on page 70.)

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, dec 2008. ISSN 1382-3256. (Cited on pages xxv, 11, 12, and 16.)

Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case study research in software engineering*. John Wiley & Sons, Inc., mar 2012. ISBN 9781118181034. (Cited on pages 14, 21, 43, 44, 45, 91, 142, 143, 167, 168, and 204.)

Thomas L Saaty. What is the analytic hierarchy process? In *Mathematical models for decision support*, pages 109–121. Springer, 1988. (Cited on pages 199 and 230.)

Johnny Saldaña. *The coding manual for qualitative researchers*. Sage, 2015. (Cited on pages 21, 40, 41, 42, and 48.)

Outi Salo and Pekka Abrahamsson. An iterative improvement process for agile software development. *Software Process: Improvement and Practice*, 12(1):81–100, 2007. (Cited on page 30.)

Mary-Luz Sánchez-Gordón, Ricardo Colomo-Palacios, Antonio de Amescua Seco, and Rory V O'Connor. The route to software process improvement in small-and medium-sized enterprises. In *Managing Software Process Evolution*, pages 109–136. Springer, 2016. (Cited on page 184.)

Tanja Sauvola, Lucy Ellen Lwakatare, Teemu Karvonen, Pasi Kuvaja, Helena Holmström Olsson, Jan Bosch, and Markku Oivo. Towards customer-centric software development: a multiple-case study. In *Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on*, pages 9–17. IEEE, 2015. (Cited on page 191.)

C.B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 25(4):557–572, 1999. ISSN 0098-5589. (Cited on pages 15, 35, 38, 40, and 41.)

Bran Selic. Agile documentation, anyone? *IEEE software*, 26(6), 2009. (Cited on pages 148 and 163.)

Pertti Seppänen, Kari Liukkunen, and Markku Oivo. Little big team: Acquiring human capital in software startups. In *International Conference on Product-Focused Software Process Improvement*, pages 280–296. Springer, 2017. (Cited on page 107.)

Arash Shahin and M. Ali Mahbod. Prioritization of key performance indicators. *International Journal of Productivity and Performance Management*, 56(3):226–240, 2007. ISSN 1741-0401. (Cited on pages 68, 69, and 129.)

Helen Sharp, Anthony Finkelsteiin, and Galal Galal. Stakeholder identification in the requirements engineering process. In *10th International Workshop on Database & Expert Systems Applications*, DEXA '99, pages 387–, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0281-4. (Cited on page 196.)

Sheri Sheppard, Anne Colby, Kelly Macatangay, and William Sullivan. What is engineering practice? *International Journal of Engineering Education*, 22(3):429, 2007. (Cited on page 99.)

Michael Shermer. How the survivor bias distorts reality. *Scientific American*, 1, 2014. (Cited on page 106.)

Forrest Shull, Janice Singer, and Dag IK Sjøberg. *Guide to advanced empirical software engineering*. Springer, 2007. (Cited on page 11.)

MA Sicilia, Juan-José Cuadrado, Elena García, Daniel Rodríguez, and José R Hilera. The evaluation of ontological representation of the swebok as a revision tool. In *29th Annual International Computer Software and Application Conference (COMPSAC), Edinburgh, UK*, pages 26–28, 2005. (Cited on page 38.)

Janice Singer and Norman G. Vinson. Ethical issues in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, 28(12): 1171–1180, 2002. (Cited on page 16.)

Raghu Singh. International standard iso/iec 12207 software life cycle processes. *Software Process: Improvement and Practice*, 2(1):35–50, 1996. (Cited on page 196.)

Dag I K Sjoberg, Aiko Yamashita, Bente C D Anda, Audris Mockus, and Tore Dyba. Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering*, 39(8):1144–1156, 2013. ISSN 00985589. (Cited on page 136.)

Dag IK Sjøberg, Jo Erskine Hannay, Ove Hansen, Vigdis By Kampenes, Amela Karahasanovic, N-K Liborg, and Anette C Rekdal. A survey of controlled experiments in software engineering. *IEEE transactions on software engineering*, 31(9):733–753, 2005. (Cited on page 10.)

Cetin Demir Slinger Jansen, Sjaak Brinkkemper, Ivo Hunink. Pragmatic and Opportunistic Reuse in Innovative Start-up Companies. *IEEE Software*, pages 42–49, 2008. (Cited on page 189.)

Darja Šmite, Claes Wohlin, Zane Galviņa, and Rafael Prikladnicki. An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering*, 19(1):105–153, 2014. (Cited on page 38.)

Cynthia Stackpole Snyder. A guide to the project management body of knowledge: Pmbok (®) guide. *Project Management Institute: Newtown Square, PA, USA*, 2014. (Cited on pages 128, 129, 187, and 191.)

Software Engineering Institute. CMMI for Development, Version 1.2. Technical report, Software Engineering Institute, Carnegie Mellon University, 2006. (Cited on pages 38 and 196.)

Adam Solinski and Kai Petersen. Prioritizing agile benefits and limitations in relation to practice usage. *Software quality journal*, 24(2):447–482, 2016. (Cited on page 161.)

Ian Sommerville. *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition, 2010. ISBN 0137035152, 9780137035151. (Cited on page 196.)

Bradley R Staats, Katherine L Milkman, and Craig R Fox. The team scaling fallacy: Underestimating the declining efficiency of larger teams. *Organizational Behavior and Human Decision Processes*, 118(2):132–142, 2012. (Cited on pages 109 and 155.)

Startup Compass Inc. The Global Startup Ecosystem Ranking 2015. Technical Report August, Start-up Genome, 2015. (Cited on pages 34, 35, 36, and 47.)

Christoph Johann Stettina and Egbert Kroon. Is there an agile handover? an empirical study of documentation and project handover practices across agile software teams. In *Engineering, Technology and Innovation (ICE) & IEEE International Technology Management Conference, 2013 International Conference on*, pages 1–12. IEEE, 2013. (Cited on page 154.)

TJ Stiles. *The first tycoon: The epic life of Cornelius Vanderbilt*. Knopf, 2009. (Cited on page 1.)

Goparaju Purna Sudhakar. A model of critical success factors for software projects. *Journal of Enterprise Information Management Iss Industrial Management & Data Systems Iss Charalambos Spathis Journal of Enterprise Information Management*, 19(1):83–96, 2012. (Cited on page 64.)

Arho Suominen, Sami Hyrynsalmi, Marko Seppänen, Kaisa Still, and Leena Aarikka-Stenroos. Software start-up failure an exploratory study on the impact of investment. In *CEUR Workshop Proceedings*, volume 2053, pages 55–64, 2017. (Cited on pages 130, 187, and 188.)

Stanley M Sutton. The role of process in software start-up. *IEEE software*, 17(4):33–39, 2000. (Cited on pages 2 and 8.)

Stanley M Sutton, E C Cubed, and Mario Andretti. The Role of Process in a Software Start-up. *IEEE Software*, 17(4):33–39, 2000. (Cited on pages 4, 36, 37, 38, 86, 87, 120, 125, 130, 131, 137, 161, 164, 179, and 184.)

Henri Terho, Sampo Suonsyrjä, Aleksi Karisalo, and Tommi Mikkonen. Ways to cross the rubicon: pivoting in software startups. In *International Conference on Product-Focused Software Process Improvement*, pages 555–568. Springer, 2015. (Cited on pages 68 and 136.)

R Scott Tindale and Tatsuya Kameda. 'social sharedness' as a unifying theme for information processing in groups. *Group Processes & Intergroup Relations*, 3(2):123–140, 2000. (Cited on pages 201, 211, and 230.)

Peter Tingling and Akbar Saeed. Extreme programming in action: a longitudinal case study. In *HCI International*, pages 242–251, 2007. (Cited on pages 53 and 188.)

Donald D Tippett and James F Peters. Team building and project management: How are we doing?, 1995. (Cited on page 111.)

Edith Tom, Aybüke Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, jun 2013. ISSN 01641212. (Cited on pages 40, 71, 81, 136, 138, 139, 142, 148, 149, 162, and 163.)

Edmundo Tovar and Carla Pacheco. Stakeholder identification in requirements engineering: Comparison of methods. In *Proc. of Software Engineering Applications (SEA)*, 2006. (Cited on page 202.)

George Tovstiga and Henning Grossmann. Strategic innovation: Exploring the link between differentiation, learning and innovation failure in start-up enterprises. In *2nd Annual International Conference on Innovation and Entrepreneurship (IE 2012)*, number Ie, page 76, 2012. (Cited on page 34.)

Nirnaya Tripathi, Eriks Klotins, Rafael Prikladnicki, Markku Oivo, Leandro Bento Pompermaier, Arun Sojan Kudakacheril, Michael Unterkalmsteiner, Kari Liukkunen, and Tony Gorschek. An anatomy of requirements engineering in software startups using multi-vocal literature and case survey. *Journal of Systems and Software*, 2018. (Cited on pages 7, 86, and 184.)

Elena Tsiporkova and Veselka Boeva. Nonparametric recursive aggregation process. *Kybernetika, Journal of the Czech Society for Cybernetics and Information Sciences*, 40:51–70, 2004. (Cited on page 228.)

Elena Tsiporkova and Veselka Boeva. Multi-step ranking of alternatives in a multi-criteria and multi-expert decision making environment. *Information Sciences*, 176:2673–2697, 2006. (Cited on page 228.)

M Unterkalmsteiner, R Feldt, and T Gorschek. A taxonomy for requirements engineering and software test alignment. *ACM Transactions on . . .*, V(212):1–39, 2014. (Cited on pages 86 and 125.)

Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, et al. Software startups–a research agenda. *e-Informatica Software Engineering Journal*, 10(1), 2016. (Cited on pages 36, 86, 89, 183, and 184.)

Muhammad Usman, Emilia Mendes, and Jürgen Börstler. Effort estimation in agile software development: a survey on the state of the practice. In *Proceedings of the 19th international conference on Evaluation and Assessment in Software Engineering*, page 12. ACM, 2015. (Cited on page 69.)

Axel Van Lamsweerde. From system goals to software architecture. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 25–43. Springer, 2003. (Cited on page 112.)

Ronald J. Vetter, Chris Spell, and Charles Ward. Mosaic and the world wide web. *Computer*, 27(10):49–57, 1994. (Cited on page 1.)

Sebastiano Vigna. A weighted correlation index for rankings with ties. In *24th international conference on World Wide Web*, pages 1166–1176, 2015. (Cited on page 208.)

Jeffrey Voas and Lora Kassab. Using assertions to make untestable software more testable. *Software Quality Professional*, 1(4):31, 1999. (Cited on page 155.)

Xiaofeng Wang, Dron Khanna, and Pekka Abrahamsson. Teaching lean startup at university: an experience report. In *International Workshop on*

Software Startups (IWSS) co-located with 22nd ICE/IEEE International Technology Management Conference, 2016. (Cited on page 5.)

Karl Wennberg and Dawn R DeTienne. What do we really mean when we talk about 'exit'? a critical review of research on entrepreneurial exit. *International Small Business Journal*, 32(1):4–16, 2014. (Cited on page 6.)

Joel West and Michael Mace. Entering a mature industry through innovation: Apple's iphone strategy. In *DRUID Summer Conference*, pages 18–20, 2007. (Cited on page 189.)

Jim Whitehead. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*, pages 214–225. IEEE, 2007. (Cited on page 199.)

Roel J Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014. (Cited on pages 201 and 202.)

Yoram Wijngaarde, Daryia Paliksha, Julien Puls, Marco Squarci, and Irina Anihimovskaya. Annual European Venture Capital Report, 2017. Technical Report February, Dealroom.co, 2018. (Cited on page 86.)

Chauncey E Wilson. Triangulation: the explicit use of multiple methods, measures, and approaches for determining core issues in product development. *interactions*, 13(6):46–ff, 2006. (Cited on page 56.)

Krzysztof Wnuk. Involving relevant stakeholders into the decision process about software components. *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, pages 129–132, 2017. ISSN 1336-9563. doi: 10.1109/ICSAW.2017.68. (Cited on page 196.)

Claes Wohlin, Martin Höst, and Kennet Henningsson. Empirical research methods in software engineering. In *Empirical methods and studies in software engineering*, pages 7–23. Springer, 2003. (Cited on page 11.)

Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012. (Cited on pages xxv, 10, and 12.)

Claes Wohlin, Darja Šmite, and Nils Brede Moe. A general theory of software engineering: Balancing human, social and organizational capitals. *Journal of Systems and Software*, 109:229–242, 2015. ISSN 01641212. (Cited on page 107.)

Eoin Woods. Aligning Architecture Work with Agile Teams. *IEEE Software*, 32(5):24–26, 2015. ISSN 0740-7459. (Cited on page 62.)

Xiaoqing Liu, C. S. Veera, Y. Sun, K. Noguchi, and Y. Kyoya. Priority assessment of software requirements from multiple perspectives. In *Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004. COMPSAC 2004.*, pages 410–415 vol.1, Sep. 2004. doi: 10.1109/CMPSAC.2004.1342872. (Cited on page 197.)

Chen Yang, Peng Liang, and Paris Avgeriou. A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, 111:157–184, 2016. ISSN 01641212. (Cited on pages 60 and 62.)

Alex Yau and Christian Murphy. Is a Rigorous Agile Methodology the Best Development Strategy for Small Scale Tech Startups? Technical report, University of Pennsylvania Department of Computer and Information Science, 2013. (Cited on pages 8, 34, 36, 89, 160, 161, 178, and 179.)

RK Yin. *Designing case studies*. 2003. (Cited on pages 11 and 21.)

David B Yoffie and Michael A Cusumano. Building a company on internet time: Lessons from netscape. *California Management Review*, 41(3):8–28, 1999. (Cited on page 186.)

S Yogendra and S Sengupta. Aligning business and technology strategies: a comparison of established and start-up business contexts. In *Engineering Management Conference, 2002. IEMC'02. 2002 IEEE International*, volume 1, pages 2–7. IEEE, 2002. (Cited on pages 186, 189, and 190.)

Changsok Yoo, Dongwoo Yang, Huykang Kim, and Eunnyeong Heo. Key value drivers of startup companies in the new media industry - the case of online games in korea. *Journal of Media Economics*, 25(4):244–260, 2012. (Cited on pages 187 and 190.)

John A Zachman. The zachman framework for enterprise. *Zachman International*, 2003. (Cited on page 38.)

Jörg Zettel, Frank Maurer, Jürgen Münch, and Les Wong. Lipe: a lightweight process for e-business startup companies based on extreme programming. In *International Conference on Product Focused Software Process Improvement*, pages 255–270. Springer, 2001. (Cited on pages 8 and 36.)

Junfu Zhang. The advantage of experienced start-up founders in venture capital acquisition: evidence from serial entrepreneurs. *Small Business Economics*, 36(2):187–208, 2011. (Cited on page 190.)

Kevin Zheng Zhou, K Tse David, and Julie Juan Li. Organizational changes in emerging economies: Drivers and consequences. *Journal of International Business Studies*, 37(2):248–263, 2006. (Cited on page 190.)