Scholarly Horizons: University of Minnesota, Morris Undergraduate Journal

Volume 6 | Issue 1 Article 3

Requirements Practices in Software Startups

John D. Hoff
University of Minnesota - Morris

Requirements Practices in Software Startups

John D. Hoff
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA 56267
hoffx247@morris.umn.edu

ABSTRACT

In a dynamic environment full of uncertainties in software startups, software development practices must be carefully approached. It is vital that startups determine the right time to make advancements and evolve their company to the next level. We will discuss the importance of requirements practices in startups and their impact on company culture, work environments, and product quality.

Keywords

Requirements, software, practices, evolution, startups

1. INTRODUCTION

Millions of people around the world are trying to launch their own businesses, also known as startups, and about one-third succeed in doing so [8]. Paul D. Reynolds, director of the research institute at the Global Entrepreneurship Center suggests that as many as 50 million startups are created each year [8]. Software startups take advantage of modern technologies that allow them to rapidly build and release software products. These companies make use of agile and lean methodologies, which are approaches that encourage flexibility to reduce risks associated with software development and requirements gathering. Because startups work in an environment of extreme uncertainty, 75% will fail in the first few years [6].

Startups can fail for many different reasons, including no market, lack of funds, competition, bad management, or poor teamwork [9]. Proper and timely evolution of software development practices can lead to improved teamwork and a better working environment. There are many studies that talk about the evolution of startups as a whole, but very little research about the evolution of software development practices exists. This paper will talk about practices that are most closely related to requirements gathering in software startups.

In Section 3, we will summarize and discuss a case study that uses Grounded Theory to evaluate requirements practices at different startups. The study shows how each startup treats their requirements practices by categorizing each practice into one of three predefined phases of evolution [5]. In Section 4, we will apply this theory to a software startup, AnswerDash, to determine which phase of evolution each of their practices were at, as well as factors that caused transitions to the next phase. Andrew J. Ko, chief technology officer at AnswerDash, documented daily observations from his company over the span of three years with the goal of understanding startup evolution and learning about developers' behavior [7].

UMM CSci Senior Seminar Conference, November 2018 Morris, MN.

2. BACKGROUND

2.1 Software Startups

A software startup is started by a founder or co-founders who are looking for a scalable business model that meets a marketplace need [11]. Due to significant fluctuation in demand, frequent and major product changes, and minimal resources, startups work in highly uncertain environments [12]. The founder(s) begin by building a prototype for their product to develop their business models. Then, they build their product to release to the market. Although there is no universal definition for what a startup is, a general guideline suggests that startups earn less than $50 million in revenue, have less than 100 employees, and have a net worth under $500 million [14]. In this paper, all observed startups are less than 10 years old and have less than 60 employees.

2.2 Software Requirements

A software requirement for a product defines what it needs to do, but not how [15]. For example, if a bank application needs a "view balance" feature, that feature becomes a requirement for the application. Several different stakeholders contribute to the process of gathering requirements. These stakeholders include the chief executive officer (CEO), chief technology officer (CTO), software developers, quality analysts (QA), and clients. While there are numerous practices in software development, this paper follows Gralha et. al [5] in focusing on 6 practices that affect requirements gathering: requirements artifacts, knowledge management, requirements-related roles, planning, technical debt, and product quality.

2.3 Requirements Practices

2.3.1 Requirements Artifacts

A requirement artifact is a task that says what needs to be done, represented as "a piece of paper, texts in a text processing system, a file in a file system, entries in a tool, or a record in a database." [4] A common template for an artifact is: as a [who], I want a [what], so that [why] [2]. For example,

if a bank application needs a "view balance" feature, the artifact could say: "As a bank user, I want a view balance feature so that I can see how much money I have." Artifacts are supposed to be as short and simple as possible, covering only one area of responsibility [4]. Artifacts are comprised of three components. The first is the task's concepts, which describe its elements and dependencies. Next is its syntax, which explains what programming languages or tools are required to perform the task. Finally, the method of a task illustrates a sequence of steps needed to fulfill it [4].

2.3.2 Knowledge Management

Knowledge management is "a method that simplifies the process of sharing, distributing, creating, capturing and understanding of a company's knowledge." [3] Commonly referred to as *documentation*, this practice serves two primary purposes in any software company. The first is to protect developers from forgetting processes, reasons, or logic behind any part of their product's design and development. The second is to reduce on-boarding problems, such as recent hires becoming overwhelmed and confused about their new company's product(s). Since documentation can reach a considerable size in software systems and requirements gathering, it is "essential to adequately structure the set of artifacts produced in the development process." [4]

2.3.3 Requirements-Related Roles

There are three roles that typically participate in the requirements gathering process: product managers, QA, and software developers. [5] All three roles contribute to task creation, but only QA and developers are responsible for completing them. Product managers prioritize tasks, developers select tasks to work on, and QA works with developers to ensure that completed tasks meet its company's quality standards.

2.3.4 Planning

Planning is the practice of prioritizing artifacts and estimating how difficult each artifact will be to complete. For prioritization, engineering teams conduct meetings to discuss which artifacts are the most important. Using their collective judgment, artifacts are organized into a list where the most important ones are at the top. As part of their meetings, engineering teams collectively estimate how difficult each artifact is. Once each artifact has been prioritized and estimated, artifacts are assigned to software developers starting at the top of the prioritization list. Keeping difficulty estimates in mind, assigning artifacts is halted when "enough work" has been handed out [13].

2.3.5 Technical Debt

Technical debt is a concept that "reflects the extra development work that arises when code that is easy to implement in the short run is used instead of applying the best overall solution." [1] Like monetary debt, technical debt accumulates "interest" over time. Technical debt management practices include assessment, communication, and implementation. As an engineering team assesses their technical debt, they decide whether it is worth addressing or ignoring. Technical debt is communicated to the rest of the team via meetings or by documentation. If some sources of debt are assessed as high risk, teams will implement fixes to their code that reduces their overall debt.

C#	Age/founded/ #employees	Role	Data collection method (frequency)
C01	4 / 2013 / 21-30	CEO Product Manager CTO	all-day observations (6) meetings attendance (4) focus groups (4) interviews (3)
C02	6 / 2011 / 41-50	Director of Products CTO	meetings attendance (3) focus groups (2) interviews (1)
C03	4 / 2013 / 11-20	Developer People Operations Customer Support Developer	meetings attendance (2) focus grounp (1) interviews (1)
C04	3 / 2014 / 11-20	Business Analyst COO	interviews (2)
C05	4 / 2013 / 11-20	CTO	interviews (1)
C06	7 / 2010 / 51-60	Director of Products Product Manager Product Designer	meetings attendance (2) interviews (1)
C07	6 / 2011 / 21-30	CTO COO	interviews (1)
C08	1 / 2016 / 1-10	CTO	interviews (1)
C09	4 / 2013 / 21-30	CTO	interviews (1)
C10	6 / 2011 / 21-30	CTO	interviews (1)
C11	9 / 2008 / 11-20	CEO	meetings attendance (3) interviews (1)
C12	10 / 2007 / 51-60	Developer CTO	focus group (1) interviews (2)
C13	10 / 2007 / 51-60	CEO	interviews (1)
C14	5 / 2012 / 51-60	CEO	interviews (1)
C15	2 / 2015 / 1-10	CEO	interviews (1)
C16	4 / 2013 / 21-30	CEO	interviews (1)

Table 1: Age, year founded, number of employees, roles interviewed, and data collection methods for 16 startups [5].

2.3.6 Product Quality

Product quality is "conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software." [10] At the cost of additional time and effort in the short run, higher levels of product quality can reduce negative feedback from customers [5]. QA is responsible for assessing product quality before anything is released.

3. THE EVOLUTION OF REQUIREMENTS PRACTICES

The evolution of software development practices was studied at 16 software startups described in Table 1 by Catarina Gralha et. al using Grounded Theory, which is the discovery of emerging patterns in data [5]. They define these startups as "organizations in search of a scalable, repeatable, profitable business model or a human institution designed to create a new product or service under conditions of extreme uncertainty." [5] After collecting data, the researchers defined three phases of evolution for requirements practices. The goal of that study was to classify every practice of each software startup into one of three phases of evolution.

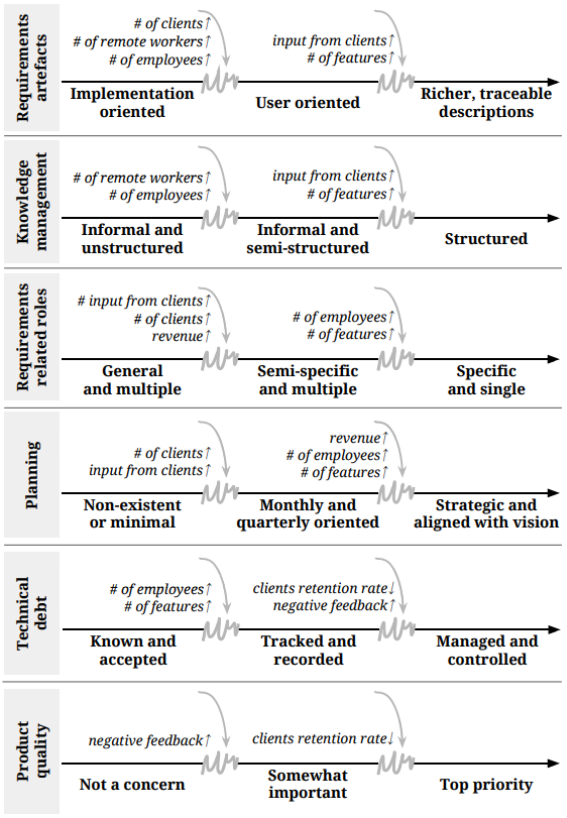

Figure 1: Turning points that cause transitions to the next phase of evolution for each practice [5].

3.1 Research Methods

The data in Figure 1 was collected by conducting interviews, all-day observations, and attending project meetings at 16 different software startups. Each startup was 1-10 years old and had 1-60 employees. To understand the evolution of requirements practices, two questions were asked: how do requirements practices change, and what factors and turning points drive those changes? These questions were used to learn about company growth, requirements gathering, requirements prioritization, features and knowledge management, and tools. The goal of that study was to determine which of the three phases for the six practices listed in Section 2.3 each startup was at [5].

Gralha et. al characterize each practice by 3 phases of evolution. Section 3.2 will explain the phases of evolution for each practice in more detail. Generally, the first phase is the beginning phase, where practices are unstructured or startups may choose to ignore practices entirely. In the second phase, practices are semi-structured, where startups start using some software and web tools. In the third and final phase, practices are formally structured, where startups set down guidelines and rules for how a practice is carried out. Every advance from one phase to the next is caused by at least one of eight turning points as illustrated in Figure 1: number of clients, input from clients, negative feedback, retention rate of clients, revenue, number of employees, number of remote workers or flexible work hours, and number of features or products [5].

3.2 Phases of Evolution for each Requirements Practice

3.2.1 *Requirements Artifacts*

In the first phase, startups begin with little or no user input because their product is still in the process of entering the market, making requirements artifacts *implementation-oriented*. The founders' backgrounds and company culture determine what tasks need to be done. Tasks are informally documented, sometimes on sticky notes. The second phase is *user-oriented*. Startups look at feedback and input from their users to develop new tasks that are commonly documented with project management tools such as Confluence and Jira [5]. In the final phase, "requirements artifacts evolve into *richer, traceable descriptions*." [5] Artifacts are broken down into smaller, simpler tasks that are given difficulty-based estimations for time of completion. These still reside in the previously mentioned project management tools. They are formally prioritized, assigned to developers, and organized into products or releases.

3.2.2 *Knowledge Management*

Knowledge management is informal and unstructured in its first phase. Founders and developers rely on each other to complete tasks and manage the few features of their product. As the number of tasks, features, and employees grow, startups enter the second phase where knowledge management is informal and semi-structured. Knowledge is shared through regular team and company meetings. Online communication tools, such as Slack, are used in place of ad hoc verbal communication. When a startup grows to a size that separates employees into different departments, knowledge management is formally structured. Using tools such as Slack and GitHub, communication and project management are often intertwined. Startups introduce guidelines that explain *how* these tools should be used.

3.2.3 *Requirements-Related Roles*

Startups begin with general and multiple requirements-related roles where "everyone does everything." [5] Once a startup has clients, they typically move to the second phase where roles are semi-specific. Clients receive more attention and marketing staff are brought on, giving developers more room to focus on product development. In the final phase, roles are specific and single. Startups might decide to hire product managers and quality assurance specialists.

3.2.4 *Planning*

Startups do not have any planning in the first phase. Most time and resources are spent developing a working product that looks attractive to the market. Once a startup understands client needs, it moves to the second phase where planning is monthly and quarterly-oriented. Planning is based on client requests without deadlines. In the third phase, planning is strategic and aligned with vision. Companies prioritize features for a broader market and decide what is better for their clients.

3.2.5 *Technical Debt*

Technical debt is *known and accepted* in the earliest phase. As products become more complex, startups *track and record technical* debt in the second phase. Hiring additional developers provides more resources to address some, but not all of

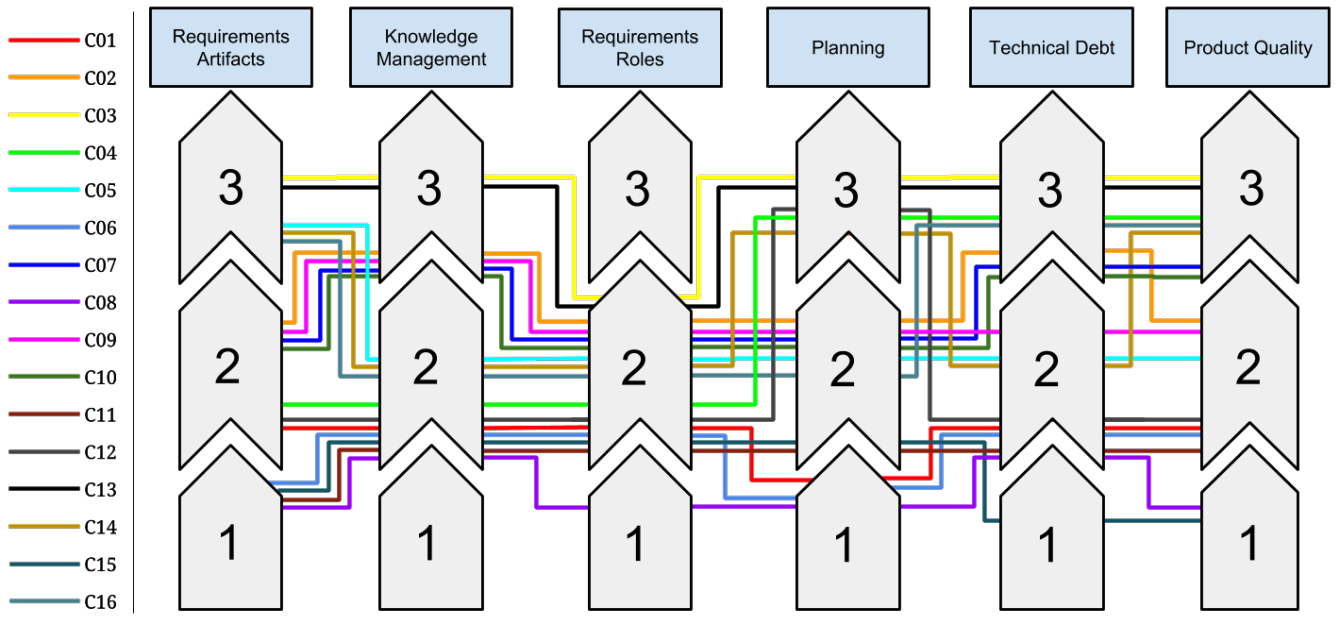

Figure 2: Phases of evolution for each practice at all 16 companies, where each company is represented by a colored line [5].

a startup's technical debt. In the final phase, technical debt is *managed and controlled*. Tasks are specifically created to address this debt and prioritized with their associated features.

3.2.6 *Product Quality*

In the first phase, "speed of release takes precedence over quality." [5] With minimal testing, there are higher rates of negative feedback due to defective features. To avoid negative feedback, startups move to the second phase where product quality is somewhat important. User experience and scalability become critical. Product quality eventually becomes a top priority in the last phase due to its correlation with company reputation.

3.3 Discussion

In a fast-paced and reactive environment, startups usually ignore the long run in order to capture a market, acquire clients, and release products more quickly [5]. Evolution of the six practices are not necessary for success. Factors that cause startups to die can be a combination of the market of its product, human resources, culture, funding, processes, and practices.

Figure 2 shows which phase of evolution of the six practices that each of the 16 companies in Figure **??** was at at the time of the study. No startup reached the third phase of requirements-related roles and all had left the first phase of knowledge management. Company C06, a 10 year-old startup is only at phase one or two in every practice. It was reported that the work environment at C06 was stressful and had long hours. C03, one of the furthest in evolution for all practices, was a 4 year-old startup with 11-20 employees. It was reported that C03 had formal processes for most practices, employees understood the company's vision and felt confident going into product development, and technical debt was low [5]. These contrasting profiles indicate that movement along the three phases of evolution is not always dependent on company age and size.

4. PARTICIPANT OBSERVATIONS FROM A SOFTWARE STARTUP CTO

Andrew J. Ko was a CTO at AnswerDash, a software startup that he and CEO Jacob O. Wobbrock founded in 2012 in Seattle, WA. AnswerDash facilitates customer service for e-commerce businesses by using machine learning to create databases of support answers for websites, thus reducing the need for human customer service. From 2012 to 2015, he documented everyday happenings at their company, where he amassed 15,000 emails, 9,000 hours of direct experience, and a diary that he used to write down personal observations [7]. With this information, Ko's goal was to study software startup evolution and developers' behavior without bias by acting as a participant. He made several claims about software developers and startup evolution, most of which included requirements practices and hints of evolution along the three phases. We will look at developers' behavior towards each requirements practice and apply the phases of evolution described in Section 3.2 to see where AnswerDash started and which phases of evolution their requirements practices were at the end of Ko's study.

4.1 Requirements Artifacts

The CEO, whose primary job was marketing and strategy, gathered customer feedback to create requirements artifacts

for the engineering team. The CEO and CTO had talked with customers before making their startup, so requirements gathering was not *implementation-oriented*. Ko's startup skipped the initial phase of requirements artifacts, which puts them in the second phase of evolution where requirements artifacts are *user-oriented*.

4.2 Knowledge Management

Ko's startup began in the first phase where documentation was *ignored*. Ko describes *design rationale debt*, which occurs when no one remembers why a component behaved the way it did [7]. To deal with this, developers started documenting components upfront. For example, he recalled a time when they added a feature a year earlier and "could only vaguely remember why we thought it was so critical at the time ... removing the feature risked breaking an undocumented customer requirement." [7] As more problems arose from lack of documentation, the company practiced knowledge management more frequently to be mindful of the long run. By the end of the study, the company transitioned to the second phase of evolution where knowledge management was *informal and semi-structured*.

4.3 Requirements-Related Roles

Ko's startup began with only two employees. Ko became the CTO because of his technology experience. Wobbok, whom he had met in college, was the CEO. Together, they started in the first phase of evolution where each role was responsible for doing everything. They made sales pitches, gathered funds, and collaborated with potential customers. Once they were able to release a product that collected revenue, they hired a team of software developers, a team for customer support, and account managers. Wobbok was able to focus more on marketing and Ko could focus more on their product's technology. So, at the time that the study was completed, they had moved on to the second phase of evolution where roles were *semi-specific*.

4.4 Planning

When the startup began, Ko was the sole developer who worked on releasing a functional product without planning, which is the first phase of evolution. As their startup grew, an engineering team was hired. This entered them into the second phase of evolution where planning is *based on customer feedback without deadlines*. During planning sessions, however, developers found it unethical to accelerate feature development and product releases at the cost of low quality, low security software [7]. Also, developers "wanted work that was technically interesting and would develop their skills, but the business needed them to focus on increasing product value, and these tasks were often straightforward and boring." [7] Ko describes *planning debt*, in which developers maintain plans in their minds, handwritten notes, and partially completed tasks, making it difficult to transfer tasks between developers. Because of these issues, assigning tasks to developers was a challenge.

4.5 Technical Debt

Technical debt was *not a concern* for Ko in the beginning. His startup began in the first phase of evolution where technical debt was *known and accepted*. He said that "responding to client feature requests meant taking on more technical debt in service of more sales pitches, more fundraising pitches, and better product customer fit." [7] He felt that taking on all of this debt would be "worth accruing if we could close funding, so we could hire a larger engineering team to build a better infrastructure." [7] As they earned more revenue and hired more developers, Ko's startup moved to the second phase of evolution where they could "frequently refactor to pay down technical debt." [7]

4.6 Product Quality

They began in the first phase of evolution where product quality was not a concern. As the sole developer, Ko's goal was to release a product that worked and grabbed the attention of potential clients. Then, as they earned more revenue and hired an engineering team, Ko's startup moved into the second phase of evolution where product quality was *somewhat important*. The CEO frequently talked to Ko and the engineering team about the importance of product quality while being able to meet deadlines at the same time. The software developers "valued the pressure to ship as a motivator, but also felt demoralized by having to compromise on quality," making it difficult for their company to find a good balance between the two [7].

4.7 Wrap-up

In 2015, at the end of Ko's three years of observations, AnswerDash was in the second phase of evolution for every practice. Excluding requirements artifacts, all of their practices evolved into the second phase due to one or several turning points mentioned in Section 3.1. Revenue and number of employees accounted for the evolution of requirements-related roles, planning, technical debt, and product quality. Number of features moved knowledge management into the second phase of evolution. Requirements artifacts began in the second phase of evolution because AnswerDash already knew their clients upfront. By the end of the study, AnswerDash had 12 employees and was earning steady revenue.

5. CONCLUSIONS

Categorizing requirements practices into one of the three phases of evolution is useful for analyzing organizational progress in software startups. It shows companies' capabilities to hire more employees, increase revenue, boost product quality, and improve their work environment. In Section 3.3, we showed that being an older startup with more employees is not necessary for advancement along the phases of evolution. Also, we found evidence that advancements gives employees more confidence, trust, and reduced stress in the workplace.

In Figure 2, we noticed trends showing that companies lean towards certain phases of evolution for different practices. For requirements artifacts, there is a mostly equal distribution. 4 companies are in the first phase, 7 in the second phase, and 5 in the third phase, indicating that the practice of gathering requirements artifacts might not be too vital. However, none of the 16 startups were still in the first phase of knowledge management. Neglecting to document processes and product details slows down development in the long run, leads to accidentally removing important features, and makes the onboarding process of new hirees more difficult. Thus, having at least some level of knowledge management helps startups reduce risks associated with losing knowledge. For requirements-related roles, no startup entered the third phase of evolution. We believe that this is

attributed to the fact that most software startups utilize agile and lean methodologies, which means that no role is responsible for one specific area of work. In the second phase of evolution, each role is capable of performing the work of other roles, which keeps companies flexible in the event that an employee leaves.

Acknowledgments

I would like to express my deep gratitude to Professor Elena Machkasova and Professor Nic McPhee, my research supervisors, as well as UMM alumnus Scott Steffes for their patient guidance, enthusiastic encouragement and useful critiques of this research work.

6. REFERENCES

[1] R. Barrett. Why not all technical debt is created equal, Jun 2018. https://medium.com/@richb_/why-not-all-technical-debt-is-created-equal-253d727302b1, retrieved 30 Nov 2018.

[2] M. Berteig. User stories and story splitting, Mar 2014. http://www.agileadvice.com/2014/03/06/referenceinformation/user-stories-and-story-splitting, retrieved 30 Nov 2018.

[3] T. H. Davenport and L. Prusak. *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, 2000.

[4] D. M. Fernandez, A. Vogeslang, J. Mund, M. Kuhrmann, and T. Weyer. Artefacts in software engineering: A fundamental positioning. *International Journal on Software and Systems Modeling*, Sep 2018.

[5] C. Gralha, D. Damian, A. I. T. Wasserman, M. Goulão, and J. A. Araújo. The evolution of requirements practices in software startups. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 823–833, New York, NY, USA, 2018. ACM.

[6] P. Henry. Why some startups succeed (and why most fail), Feb 2017. https://www.entrepreneur.com/article/288769, retrieved 30 Nov 2018.

[7] A. J. Ko. A three-year participant observation of software startup software evolution. In *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*, ICSE-SEIP '17, pages 3–12, Piscataway, NJ, USA, 2017. IEEE Press.

[8] M. K. Mason. Worldwide business start-ups. http://www.moyak.com/papers/business-startups-entrepreneurs.html, retrieved 28 Nov 2018.

[9] E. McGowan. The 13 top reasons why startups fail, Dec 2017. https://www.startups.co/articles/why-do-startups-fail, retrieved 30 Nov 2018.

[10] R. S. Pressman. *Software Engineering: A Practitioners Approach: 6th Edition*. McGraw-Hill, 2004.

[11] N. Robehmed. What is a startup?, May 2015. https://www.forbes.com/sites/natalierobehmed/2013/12/16/what-is-a-startup/#1030164a4044, retrieved 30 Nov 2018.

[12] A. Schmitt, K. Rosing, S. X. Zhang, and M. Leatherbee. A dynamic model of entrepreneurial uncertainty and business opportunity identification: Exploration as a mediator and entrepreneurial self-efficacy as a moderator. *SAGE Journals*, Sep 2017.

[13] V. Tasheva. The importance of prioritizing and sizing your backlog for agile planning, May 2018. https://www.telerik.com/blogs/the-importance-of-prioritizing-and-sizing-your-backlog-for-agile-planning, retrieved 30 Nov 2018.

[14] A. Wilhelm. What the hell is a startup anyway?, 2014. https://techcrunch.com/2014/12/30/what-the-hell-is-a-startup-anyway/, retrieved 30 Nov 2018.

[15] S. Withall. *Software Requirement Patterns*. Microsoft Press, Redmond, WA, USA, first edition, 2007.