Master Thesis
Software Engineering
Thesis no: MSE-2012-99
September 2012

Software development in startup companies

Carmine Giardino

Nicolò Paternoster

This thesis is presented as part of Degree of
European Master in Software Engineering

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

This thesis is submitted to the School of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 2 x 20 weeks of full time studies.

Contact Information:
Authors:
Carmine Giardino [870910-T573]
E-mail: carmine.giardino@gmail.com

Nicoló Paternoster [861218-T753]
E-mail: paternoster.nicolo@gmail.com

University advisors:
Prof. Tony Gorschek
Blekinge Institute of Technology

Prof. Pekka Abrahamsson
Free University of Bolzano

School of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona
Sweden

Internet : www.bth.se/com
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Software startups are newly created companies with no operating history and are extremely fast in producing cutting-edge technologies. These companies develop software under highly uncertain conditions, tackling fast growing markets with severe lack of resources. Startups present an unique combination of characteristics which pose several challenges to the software development activities, creating interesting problems for software engineers. However, despite the increasing economical importance and the high failure rate, there are only a few scientific studies attempting to address software engineering (SE) issues, especially for early-stages startups. In a context where a wrong decision can easily lead the entire business to failure, the support of SE can contribute to foster performances of startups and making a big impact on a large number of companies.

Objective. In view of a lack of primary studies, the first step to attending the software development strategies with scientific and engineering approaches is by an understanding of the startups' behavior. For this reason this research aims to understand how software development strategies are engineered by practitioners, in the period of time that goes from idea conception to the first open beta release of the software product.

Methods. This research combines a systematic review of the state-of-the-art with a cross-sectional case study conducted in 13 web startups recently founded and distributed in different geographic areas and market sectors. A *grounded theory* approach guided the execution of a *systematic mapping study*, integrated with semi-structured interviews and follow-up questionnaires to explore the state-of-practice.

Results. We selected, classified and evaluated 37 relevant studies. The systematic review revealed that the studies which constitute the body of knowledge are generally not rigorously designed and executed, make use of inconsistent terminology, and cover only small samples of startups. Moreover, we extrapolated concepts from the case study to form a theoretical model, explaining the underlying phenomenon of software development in early-stage startups. The model is grounded in the empirical data and its explanatory power was further validated through a systematic procedure. Finally we proposed a multi-faceted evolutionary model to describe the dynamics of the software development after the first product release.

Conclusions. The research provided a wide set of evidences fostering the understanding of how software development is structured and executed, from idea conception to the first release. The results revealed the urgent priority of startups of releasing the product as quickly as possible to verify the product/market fit and to adjust the business and product trajectory according to the early collected user feedbacks. Nevertheless, the initial gain obtained in speeding-up the development by low-precision and product-centric engineering activities is counterbalanced by the need of restructuring the product and the workflows before setting off for further grow. In fact, when user requests and company's size start to increase startups face an initial and temporary drop-down in productivity, creating the need of mitigation strategies to find a sweet spot between being fast enough to enter the market early while controlling the amount of accumulated technical debt. The conclusions can be generalized with a good degree of confidence to the majority of early-stage software startups involved in the production of innovative products, especially for web and mobile applications.

Keywords: Software development; Startups; Theoretical model; Grounded theory.

Acknowledgments

First and foremost, we owe sincere thankfulness to our research supervisors - Profs. Tony Gorschek and Pekka Abrahamsson - who made us believe in ourselves and supported us throughout the research process while allowing us the room to work in our own way.

We are truly indebted and thankful towards the two institutions that contributed to develop our knowledge during our two-years European Master program in Software Engineering: the Blekinge Institute of Technology and the Free University of Bolzano. Working in an environment constituted by top-ranking professors in SE has been an unique opportunity for us. We would like to show our gratitude to Barbara Russo, Darja Šmite, Kai Petersen, Ludwik Kuzniarz, Claes Wohlin, Emilia Mendes, Alberto Sillitti and Gabriella Dodero for their assistance.

We would like to show our gratitude to other researchers, librarians and colleagues for their stimulating discussions and morale support. In particular we thank Michael Unterkalmsteiner for revising the entire thesis and giving precious advices. Moreover we thank for the revisions conducted by Ali Nauman Bin, Tobias Pfeiffer, Jürgen Börstler, Hassan (Gilani), Waqas Rasheed Qureshi, Chaitanya Gurram and Krishna Sandeep Taduri.

We would like to thank all the CTOs and CEOs of the startups that participated to the case study, subtracting personal time out of their busy schedule. This thesis would not have been possible without their effort.

Further, this thesis is completed thanks to the support of our families, who understanded and encouraged the long work on our research.

We would like to show our gratitude to Max Marmer, Paul Grahm, Eric Ries, Steven Blank, Dave Snowden and Gerry Coleman that through their inspirational works enriched and fostered our knowledge of the field.

We are obliged to many of our colleagues who supported us during our studies: Santiago, Olesia, Ali (Demirsson), Tomasz, Sebastian, Tiago, Tony, Daniel (Graziotin), Adam, Hassan, Farnaz and all the EMSE students and alumni.

Last but not the least, we would like to thank our friends for their patience and support throughout difficult moments of our life. Thank you Antonella, Gemma, Bentt, Paolo (Lombardi), Åsa, Massimo, Marco (Valente), Marshed, Selma ,Daniel (Masero), Die Atzen, Aitor, Alicia, Martin, David, Eva, Piazza Verdi WG, the erasmus group of Karlskrona, Carmine (Giardino), Mariagrazia, Manuel, Ermanno, Enrico, Oto, Luca, Vincenzo, Luigi, Giuseppe, Francesco, Marilita, Giusy, Monica, Carmine (Serluca), Stefano, Claudio, Virginio, Giuseppe (Sorce), Valerio (Raco), Mario, Dan, Shen, Akira, Cristina, Mattia, Mirza, Rebecca, Manuele, Cicco, Sandro, Valerio, Riccardo, Flavia, Marco (Soave), Stefania, Arturo (Moleti), Paolo Emilio and all the others that are not mentioned here.

It is a great pleasure to thank everyone who helped us write our thesis successfully and gave us this extraordinary learning opportunity.

"The best way to predict the future is to invent it." - Alan Kay

Contents

List of Figures

List of Tables

Chapter 1

Introduction

An impressive number of new software startups are launched worldwide every day as a result of an increase of large new markets, accessible technologies, and venture capital [9]. With the term *software startups* we refer to those temporary organizations focused on the creation of high-tech and innovative products[1], with little or no operating history, aiming to grow by aggressively scaling their business in highly scalable markets.

New ventures such as *Facebook*, *Linkedin*, *Spotify*, *Pinterest*, *Instagram*, *Groupon* and *Dropbox*, to name a few, are good examples of startups that evolved into successful businesses. But, despite many successful stories, the great majority of startups fail within two years from their creation, primarily due to self-destruction rather than competition [10]. Operating in a chaotic, rapidly evolving and uncertain domain, software startups face intense time-pressure from the market and are exposed to tough competition [11, 12]. In order to succeed in this environment, it is crucial to choose the right features to build and be able to quickly adapt the product to new requests constrained by very limited amount of resources [13].

As Marc Andreessen recently stressed in a widely discussed article on the Wall Street Journal , the role of software in economy has never been as important as it is today (to use his own words, *"software is eating the world"*[14]). In this context, with the tech startup industry expanding at an impressive pace (solely in the US *"startups create an average of 3 million new jobs annually"* [15]), it is easy to understand how a small increase in their performance and success rate can make a big difference on a global scale. Startups are able to produce cutting-edge technologies and quickly transform into large organization by initially operating with early-adopters markets.

From a software engineering perspective startups are extremely interesting as they develop software in a context where processes can hardly follow a prescriptive methodology [13, 16]. Despite startups share some characteristics with similar domains (e.g. small and web companies), the combination of different factors makes the specific development context quite unique [17, 13]. More research is needed to support their engineering activities [16], guiding practitioners in taking decisions and avoiding choices that could easily lead the whole business to failure

[1] In this thesis we use the term *"product"* for both software products and software services.

[18]. However, despite the impressive size of the startup ecosystem [19], the state-of-the-art presents a gap. Through a *Systematic Mapping Study* (SMS) of the literature only few publications were found related to engineering activities in startups. Moreover the majority of the these studies do not possess the attributes required to form a consistent body of knowledge in the state-of-the-art and in the state-of-practice (see Section 5.1).

This thesis aims to understand how software development strategies are engineered by practitioners in startup companies in terms of level of: structure, planning and control of software projects, in the period of time that goes from idea conception to the first open beta release of the software product (see Figure 1.1).

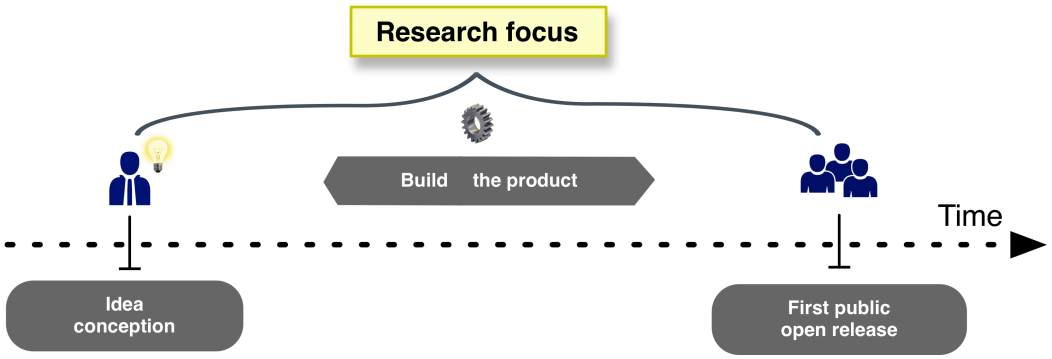


Figure 1.1: Area of interest

With a cross-sectional study we focused the research context in a limited time-frame to highlight the nature of uncertainty in the development activities and time pressure from the market[2], as discussed in [11, 12]. We performed semi-structured in-depth interviews with CEOs and CTOs of startups, covering a wide spectrum of thematics and iteratively adjusting the direction of the research according to the emerging evidences.

The systematic explorative study allowed the researchers to ground the findings in the empirical data, and contribute to the field designing the initial landscape of research with multiple contributions:

- We present a systematic mapping of the existing academic literature, drawing the landscape and evaluating the quality of the state-of-the-art. We identified 37 relevant studies and discuss their results identifying the most the most important contributions to research and practice (see Section 5.1).
- We discuss the surrounding context by analyzing similar areas of SE and reviewing the existing grey literature (see Appendix A.2).

[2]Studies on more mature startups with advanced operating history are presented in Subsection 3 as related work.

- We provide a behavioural model with validated explanatory capabilities which illustrates the phenomenon of software development in early-stage startups, at multiple level of abstractions (see Chapter 6).
- We analyze how the short-term priorities of software development influence the long-term behaviour of startups, extracting an early-stage life cycle model and comparing it to existing models (see Chapter 7).
- We compare the software development approach in early-stage startups with traditional software development methodologies and suggest strategies to improve the company's performance (see Section 7.2).
- We validate our findings through systematic comparison with existing models, frameworks, empirical data, and the state-of-the-art (see Section 6.6).
- We provide to other researcher all the necessary material to reproduce our study with additional companies and validate the study with new empirical data.
- We discuss the limitations of the study by analyzing the most important threats to validity (see Section 8.5) and suggest possible directions for further research (see Section 8.6).

The remains of this document is structured in different chapters with different concerns: first we provide relevant contextual information about startups in *Background* (Chapter 2), then we discuss the *state-of-the-art* in *Related work*(Chapter 3). In *Research methodology* (Chapter 4) we first introduce the research goal (Subsection 4.1.1) and research questions (Subsection 4.1.2) and then we discuss the design and execution of the research process. The initial results are discussed in *Results and analysis* (Chapter 5) which further converge in the presentation of the *Model* (Chapter 6) and *Dynamics and evolution of startups* (Chapter 7). In *Summary* (Chapter 8) we summarize the results of the thesis which are eventually wrapped up in *Conclusion* (Chapter 9). The *Appendix* A contains different sections presenting detailed tables, figures and other information which are integrated within other chapters. Figure 1.2 provides a visual representation of the structure of the document, explaining the main concerns addressed in each chapter.

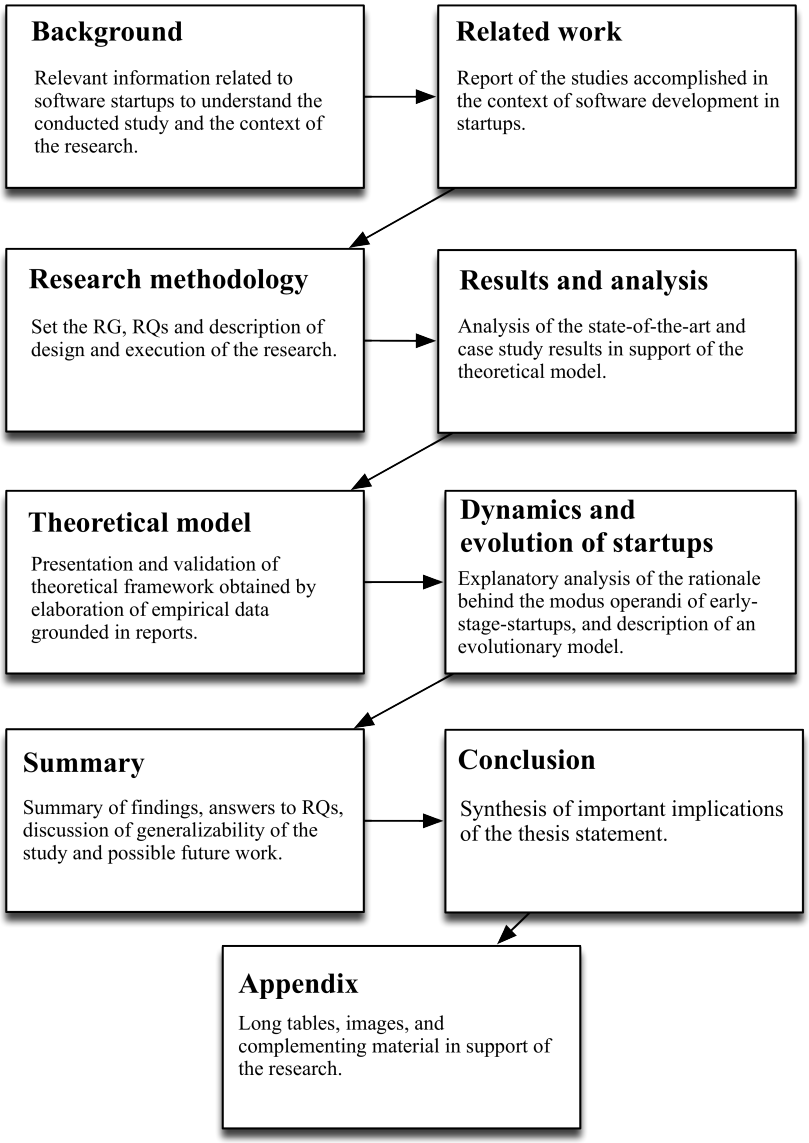


Figure 1.2: Structure of the document

Throughout the thesis we make use of some conventions: the specific terminology is summarized in the Glossary at the end of the document, while graphical conventions are presented in Appendix A.1.

Chapter 2

Background

Since studying entrepreneurship requires multidisciplinary competences, software development in startups cannot be seen as a single unit. In order to apply a proper analysis it is required to master several concepts. Among others the more relevant are: engineering of software processes, technical debt, Agile/Lean methodologies, web development, business models, customer development and venture management. In this chapter we provide a basic knowledge of the concepts listed above, grounding our research work in the surrounding context.

2.1 Overview

Modern entrepreneurship, which was born more than thirty years ago [20], has been boosted by the advent of the consumer internet markets in the middle of the nineties and culminated with the notorious *dot-com bubble burst* of 2000 [21]. Several years later, with the massification of the internet and mobile devices, we are now assisting to an impressive proliferation of software ventures - metaphorically referred as the *startup bubble*. The easy access to vast potential markets and the low cost of services distribution are appealing condition for modern entrepreneurs [22]. Inspired by stories of overwhelming successes, a large number of software businesses are created every day. However, the great majority of these companies fail within two years from their creation [10]. Just by looking at the number of new business incubators which appeared in the last three years one can evaluate the importance of startups [23]. The wave of disruption in new technologies has led companies to be more and more competitive, forcing themselves to radical organizational and innovational renewals, which bring many companies to the attempt of behaving like startups [24]. In this thesis we addressed technical aspects related to software development in startups, exploring their operational dynamics. In view of the lack of agreement on an unique definition of the word *startup* (see Subsection 5.1.3), we delimited our initial contextual boundaries to newly created innovative and single-product software companies, in the time-frame that goes from the idea conception to the release of the first product in highly scalable markets. Moreover, since most startups are web-oriented companies [25, 26], in our case study, we inquired web companies with little or no operating history. Fi-

nally, we didn't consider the size of the company as a discriminant, even though the typical founding team of early-stage startups is likely to be relatively small [22].

2.2 Software development in startups

The implementation of methodologies to structure and control the development activities in startups is a major challenge for engineers [27]. Generally, the management of software development is achieved through the introduction of software processes, which defines what steps the development organizations should take at each stage of production and provide assistance in making estimates, developing plans and measuring the quality [28]. In the last decades, several models have been introduced to control software development activities. However, their application in startup companies doesn't report significant benefits [29, 27, 13].

In such context, software engineering (SE) faces complex and multifaceted obstacles in understanding how to manage development processes. Sutton defines startups as creative and flexible in nature and reluctant to introduce process or bureaucratic measures which may hinder their natural attributes [13]. Also Bach refers to startups as *"a bunch of energetic and committed people without defined development processes"* [30]. In fact, startups have very limited resources and typically wish to use them to support product development instead of establishing processes [27, 3]. Some attempts to taylor lightweight processes to startups, reported basic failure of their application: *"Everyone is busy, and software engineering practices are often one of the first places developers cut corners"* [31]. Rejecting the notion of repeatable and controlled processes, startups prominently take advantage of unpredictable, reactive and low-precision[1] engineering practices [13, 33, 34, 35].

Moreover, as a matter of fact, most startups develop packaged applications rather than software for a specific client [36]. Issues related to this domain are addressed in literature by the area known as *market-driven software development* (sometimes called *packaged software development* or *COTS software development* [37]). Among other results, researchers emphasize the importance of time-to-market as a key strategic objective [38, 38, 39] for companies operating in this domain. Other peculiar aspects which influence the software development are related to requirements, which are reported to be often "invented by the software company" [40], "rarely documented" [41], and can be validated only after the product is released to market [42, 43]. Under these circumstances, failure of product launches are largely due to "products not meeting customer needs" [37].

Accordingly, product-oriented practices help startups in having a flexible team, with workflows that leave them the ability to quickly change the direction according to the targeted market [3, 13]. At this regard, many startups focus on team

[1] The term *"low-precision"* has been derived from [32].

productivity, asserting more control to the employees instead of providing them rigid guidelines [33, 34, 35]. Despite some studies tried to address the above mentioned issues, from the systematic review of the literature we found only a few SE works in this specific area, as confirmed by other studies [27, 29, 2, 13]. Moreover the studies, identified in our systematic review, appear to be highly fragmented and spread across different areas rather than constituting a consistent *body of knowledge* (see Secion 5.1). Notwithstanding, a new interesting area of the SE research, trying to tackle the problem of the *technical debt*, seems to bring interesting implications in studying development in software startups. The metaphoric neologism was originally introduced by Cunningham in 1992 [44] and has recently attracted the attention of SE researchers[2]. The concept of *technical debt* is well illustrated in [48]: *"The idea is that developers sometimes accept compromises a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline), and that such compromises incur a "debt" on which "interest" has to be paid and which the "principal" should be repaid at some point for the long-term health of the project"*. The compromise between high-speed and high-quality engineering is faced daily by software startups, not only in terms of architecture design but in multifaceted aspects (weak project management, testing, process control, . . .).

2.3 Related areas

To fully understand the context in which startups operate, is useful to refer to related areas which can offer relevant contributions. Among others, we identified four boundary domains which are particularly interesting for this research, as depicted in Figure 2.1.

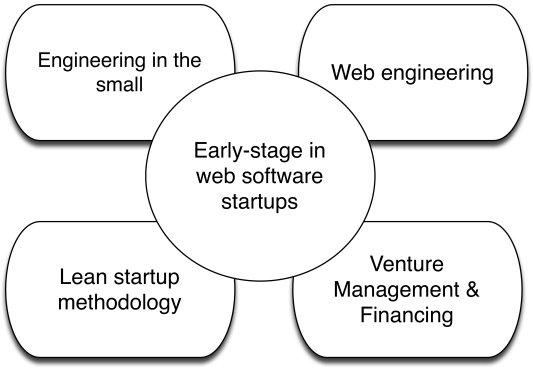


Figure 2.1: Related areas

[2]To attest the fact that *technical debt* is gaining traction among researchers, we mention two important contributions which characterize the *"debt landscape"*: [45, 46] and a dedicated workshop [47] organized by the Software Engineering Institute and ICSE.

First of all, in terms of the number of employees, early stage startups are typically small software companies. Then, it is relevant to understand what research has been provided by *engineering in the small* area. Moreover, since we focus the case study on web startups, another related area is represented by *web engineering*, which presents some peculiar characteristics that differ from traditional approaches. Some other important lessons can be derived by looking at aspects related to *venture management and financing*, which typically are the drivers of startup's decisions. Finally, throughout the thesis document we will refer to the Lean startup methodology [7] which provide good explanatory tools to researchers. In the next sections we review the most important contribution each field can provide to support SE in startup companies. A more detailed discussion is presented in appendix A.2.

2.3.1 Engineering in small companies

A special issue of IEEE Software of 2007 collects a good number of articles entirely dedicated to the topic of *SE in the small* [49] where researchers try to delve into a very interesting research problem, i.e. *"How can small organizations apply software engineering methods, techniques, best practices and tools to improve software quality and productivity without introducing unacceptable overhead?"*. Researchers advocate for effective SE solutions for small software companies which *"aren't just scaled-down version of large firms"* [20] but are usually more flat, extremely flexible, responsive, adopting a free-flowing management style.

The majority of today's software companies are small [49] and present lack of processes and basic documentation [50], despite they are recognized as important aspects of software development [31]. Especially software process improvement (SPI) in small software development organizations is almost neglected, due of the amount of time required to establish all the configuration management and review processes. The emphasis on flexibility and short development schedules, especially in the early-stage of a software company [51], lead small companies to consider SPI models as an obstacle, since the latter aim to achieve repeatability and predictability. SPI requires considerable effort and its value can be recognized only in the long run, when the workflow realistically start to be sustainable for further growth [52] (see Appendix A.2.2 for more details about engineering in small companies).

2.3.2 Web engineering

The great majority of well-funded today's startups are working on web applications [25, 22, 26]. This is consistent with the sample of companies we considered in this research and therefore in this section we briefly discuss the peculiar characteristics of software engineering for a web product, as reported by the literature.

Web engineering is the discipline which makes *"[...] use of scientific, engineering, management principles and systematic approaches with the aim of successfully developing, deploying and maintaining high quality Web-based systems and applications"* [53]. Web development is mainly characterized by small team size, short timelines and agile/rapid development approaches [54]. Another important characteristic of web projects is the constant tradeoff between *"feature slip"*[3], *"time-to-market"* and *"software quality"* [55]. Nevertheless, loosely coupled web-based systems in the long run become larger and more complex, impacting negatively on product quality aspects. Since most of *web-based* system development must be completed in the short term, as described in [56], web engineering generally lacks integrated and formal testing methodologies. Moreover estimating process and product metrics, such as robustness and maintainability, can be challenging since formal requirements, analysis and design are almost neglected [57]. Finally, an interesting recent study, executed a grounded theory research using an approach similar to the one undertaken by our research, to understand SPI initiatives in small web companies providing a detailed conceptual model [58] (see Appendix A.2.3 for more details about web engineering).

2.3.3 Lean startup methodology

One of the modern pioneer of software startups' research is Steven Blank, who has confirmed the profound diversity between startups and smaller versions of large companies from a entrepreneurial and managerial perspectives. Being himself a practitioner and an academic, he developed the *Customer Development* process, extensively described in *The Four Steps to the Epiphany* (2005) [17] and *The Startup Owner's Manual* (2012) [59]. In the course of attracting and keeping customers, Blank suggests a process that has to be place aside of product development, which aims to discover and validate the right market for an idea, building the product features that solve actual customers' needs, testing the correct model and tactics for acquiring and converting customers, and deploying the right organization and resources to scale the business.

"The best student" that Blank ever had was Eric Ries, now successful entrepreneur and engineer, who is recognized for pioneering the *Lean Startup Movement*, which combines the Japanese concept of Lean production with Blank's Customer development, to establish a new sort of discipline. In his bestseller book *The Lean Startup* [7], Ries presents how entrepreneurs in every settings make the same mistakes: they build elaborated products before daring to test them with final users[4], basing their decisions on wrong information. He introduced the concept of *"minimal viable product"* (MVP), which is a strategy used for fast and quantitative market testing of a product that has just those essential

[3]Feature slip is defined as the action of postpone a low-priority feature to a later release.

[4]We use the term "users" to include "customers" throughout the rest of the thesis. See [60] for a discussion on the term "user".

features which allow the product to be released. From a technical perspective he developed the Lean Startup model, which core is the *Build-Measure-Learn* feedback loop. Through this model, he explains how it is important to test early the riskiest elements of a startup's plan, the parts on which everything else depends on (see Appendix A.2.5 for more details about the lean startup methodology).

Although the Lean startup methodology is presented as an innovative tool, from a SE point of view, many concepts discussed in the book can be dated back in time to almost 40 years ago. For instance Basili in 1975 wrote about what was called *Iterative Enhancement technique*, a practical approach to software development which recalls the MVP: *"This technique (Iterative Enhancement) begins with a simple initial implementation of a properly chosen skeletal subproject which is followed by the gradual enhancement of successive implementations"* [61]. Another important study of Carmel, back in 1995, invited companies to *"[...] develop incrementally to improve product design and reduce risks. Evaluate and minimize risk. Evaluate alternatives at each incremental juncture and minimize commitment of capital, time, and labor at each stage of the process. Once the risks are evaluated, employ risk reduction techniques such as: prototyping, mock-ups, proof-of-concepts, simulations, usability labs, and benchmarking"* [62]. However, the Lean startup methodology is gaining high consensus into the startup community, and therefore worth exploring (see Appendix A.2.4 for more details).

2.3.4 Venture management and financing

The increasing economic importance of startups brought a significant management interests to the entrepreneurship. Scientific management, developed in the early 1910's [63], has dramatically contributed in making companies more efficient and effective. However, despite the huge knowledge emerged during these years of research, only a little part of findings has been able to adapt to the context of startups. The reason behind is the chaotic context of the startup's domain, and consequently the difficulties of creating the structures necessary to synthesize a common language and framework to describe and measure innovation and entrepreneurship [64].

Many researchers strived in defining main characteristics in this domain and have principally focused on the commercial risks of a startup. A typically reported challenge is the fact that startups run from project to project *cash-flows*. With very little capital it is hard to gain long term technological knowledge and competences [65]. On the other hand, excessive capital from day 1 might be harmful. As reported in [22] chances of success are statistically correlated to the ability of coherently scaling in activities related to the dimensions of: customer, product, team-size, finance and business model. Also remarked in [66], the strength of mutual cooperation between the entrepreneur and the capitalist is a main factor of success, assuming coordination between product, market and financing [67].

As reported in [68], startups usually rely on third party funding to support their operations. At the very beginning startups have a relatively small initial capital (seed funding) which amount is highly dependent on the kind of product and location. The seed capital can come from founders' personal resources (*bootstrapping*) or from some early *Angel Investors*. In this initial phase most startups are basically burning capital without having any revenue stream. If the product and market is promising, the following financial venture rounds are increasingly more consistent and usually involves Venture Capital (VC) funds. After large rounds of funding, startups try to attack larger and fast-growing markets with the goal of scaling the company [69], being acquired[5] or going public with an IPO[6].

[5]Sometimes startups are acquired by large organizations for strategical reasons related to the talent of the team (*acqui-hire*) rather than for the profitability of the company.

[6]It stands for Initial Public Offering, that is a type of public offering where shares of stock in a company are sold to the general public.

Chapter 3

Related work

In this chapter we present the most relevant studies contributing to the formation of a *body of knowledge* focused on engineering activities in software startups. The materials reviewed here were mostly collected during the execution of a systematic review of the literature (detailed in Section 4.2).

The word *startup* appeared in the SE literature for the first time in 1994 in an article written by Carmel [70] where he studied the *time-to-completion in young package firm*[1]. He noticed how these companies were particularly innovative and successful, advocating for the need of more research investigating software development practices so as to replicate success and try to transfer it to other technology sectors.

Only a few engineering studies in this specific area have been published in the years that followed (see Chapter 5). Moreover the studies we identified in the systematic review appear to be highly fragmented and spread across different areas rather than constituting a consistent *body of knowledge*. In fact, we were able to identify only four empirical SE studies published prior to 2012 which are entirely dedicated to the topic of software development approaches in startups, designed executed and presented in a rigorous way[2]. In the remaining part of this section we provide an overview of the related works that we have identified during our research.

First of all, a research published in 2000 by Sutton [13] confirmed a general lack of studies in this area, claiming that *"software startups represent a segment that has been mostly neglected in process studies"* and it has been further confirmed with the empirical studies of Coleman et al. [27, 29, 2] eight years later.

One of the most prolific SE researcher in the area of startups is Gerry Coleman. He started working with irish startup companies and developing a *"lightweight software process for startups based on agile practices"* [71], which has been presented at a conference in 2004 [72] but apparently the research evolved into a

[1]The software development challenges of a startup have changed dramatically in the last 20 years.

[2]Moreover the studies under consideration [27, 29, 2, 18] have investigated mainly mature startups, whilst the empirical research we performed is focused on early-stages startups. This issue is further discussed in the analysis of the systematic literature review (see Section 5.1)

different direction[3]. Indeed, his attention moved from startups to small enterprises [16]. In fact, he published an article titled *"Using grounded theory to understand software process improvement"* [2], which includes detailed explanation of the research methodology undertaken for his analysis. His results show different factors that influence and hinder the formation of processes in startups and small companies.

First insights reveal how software startups are product-oriented in the first period of their development phase [3]. Despite good achievements at the beginning, software development and organizational management increase in complexity [73, 74] causing deterioration of performance over time. Briefly, the necessity of establishing initial repeatable and scalable processes cannot be postponed forever[4].

A study of Kajko-Mattsson [18], which investigated a Swedish software startup, reported a heavy lack of requirements gathering process, minimal project management, lack of control over the change requests, absence of documentation to track the status and progress of the process and defective releases. Accordingly, Ambler et al. report how two startups approaching to an upcoming IPO started to require processes to focus on scalable solutions, in view of the growing company's size in terms of users and employees [76]. In this regard, Crowne, in [10], specifies different stages through which software startups evolve. Starting without any established workflows, startups grow over time, creating and stabilizing processes to eventually improving them only when sufficiently mature.

As studied in [77], the maturity of a company affects the extent to which processes are adopted. The author reports how introducing Extreme Programming (XP) principles [78] in the development process was challenging because of the need of trained team-members for fully implementing the methodology[5]. In fact, [79] was able to start with all the XP practices in place only after six months of coaching the team, trying to enhance maturity from day-one. Nevertheless, even then, customization of practices were inevitably implemented to adapt the processes to the undertaken startups' context [80].

But when startups have no time for training and orienting activities, as discussed in [13], their main focus remains on team capabilities instead of prescriptive processes hiring people who can *"hit the ground running"* [81]. Empowering the team and focus on methodological attributes of the processes oriented in prototyping, proof-of-concepts, mock-ups and demos, to test basic functionalities, have been the primary priority in startups as described in [70]. Only when they grow,

[3]Coleman has been personally contacted and he confirmed that the lightweight process in question has not been further adopted neither *"developed into the later research"*

[4]This has been confirmed by Peter Thiel, co-founder of *Paypal*, *Asana* and other successful businesses, who declared that *"there is no real chance of setting things up correctly such that the rest unfold easily. But you should still get the early stuff as right as possible"[75].*

[5]According to XP creator Kent Beck, to be effective XP requires to be carefully applied: *"if you follow 80 percent of the process, you get just 20 percent of the benefits"* [78].

formal methodologies arise, followed by a conduction of quality assurance and long-term planning processes [81].

As partially discussed above, contributions to flexibility and reactiveness of the development process has been conducted prominently by means of *Lean* [82] and *Agile* [83] methodologies (also reported in [84, 85]), where the extreme uncertain conditions lead startups to learn fast from trials and errors with a strong customer relationship in order to avoid wasting time in building wrong functionalities and prevent rapidly exhaustion of resources [86, 68, 13]. Customer involvement in software development has also been discussed in [87] as important factor to encourage an early alignment of business concerns to the technology strategies, because both are salient considerations to be successful [77].

When startups take a development approach that is mainly product-oriented rather than process-oriented, it is essential to have a flexible team, with a workflow that helps them quickly to change direction according to the target market [13]. In this regard, many startups have been focused on team productivity, providing more control to the employees instead of providing them rigid guidelines [33, 34, 35].

Then, when startups are mature enough to support software process improvement (SPI), the solutions considered according to the state-of-the-art are oriented towards light-weight processes such as a design of development process based on XP, proposed by Zettel in [88]. The process consists of a set of activities and artifacts (in addition to some important roles) defined in order to identify responsibilities and tools to utilize. But, despite the promising benefits reported by Zettel, we were not able to identify any future evaluation in real-world settings.

Another attempt of SPI in startups has been conducted by Deakins et al. introducing a *Helical Model for managing e-commerce development environment* [89]. Also in this case, the author prescribed broad guidelines for a rapid high-quality development process, which underwent limited testing only in academic settings.

The first publication mentioning the problem of *one-size-fits-all*, related to the SPI representations for startups, is described in [90]. The author reveals the problem in actuating the same *best-practices* criteria for established companies in 10-person software startups. Thoroughly remarked in [13], Sutton states that problems of SPI in software startups arise because of: the dynamism of the development process, which precludes repeatability; organizational maturity, that cannot be maintained from startups in view of lack of corporate direction; severe lack of resources, both human and technological for process definition, implementation, management, training ...; in conclusion, the primary benefits of SPI do not address startups, which instead of promoting product quality, aim to minimize time-to-market.

Additionally, the role of SPI has always been neglected because seen as an obstacle to the development team's creativity and flexibility as described in [29] and to the need of a quick delivering product process environment. In fact,

product quality is left aside in favour of minimal and suitable functionalities to shorten the time-to-market. As reported in two studies of Mater and Mirel [91, 92], quality aspects, mostly taken in consideration in internet startups, are oriented to usability and scalability, even though the market and application type heavily influences the quality-demand [27, 93].

Finally, to maintain the development activities, oriented to limited but suitable functionality, many studies suggest to externalize the complexity of parts of the project to third party solutions by means of outsourcing activities, software reuse and open-source strategies [94, 95, 96, 97].

In conclusion, since *"all decisions related to product development are trade-off situations"* [68], generally startups optimize workflows to the dynamic context they are involved into. In fact they typically adopt any development style that might work to support their first needs in what is called the *"Just do it"* school of software startups [7]. Additionally, as remarked by Coleman, *"many managers just decide to apply what they know, as their experience tells them it is merely common sense"* [27].

To summarize - although a number of studies have been discussed in this section - the existing material appears to be inadequate to deal with the increasing importance of startups' demands. In fact, as confirmed by the analysis of results of the systematic review (see Section 5.1.4) this area appears to be, to some extent, immature. Nonetheless, we are recently assisting an impressive proliferation of books (such as [59, 7, 24, 98]) and pseudo researches [99], which are gaining traction among practitioners. For this reason we have dedicated a subsection of Appendix A.2.5 to review the most relevant part of the *grey literature* which affects our research.

Chapter 4

Research methodology

4.1 Introduction

This chapter looks into the undertaken research methodology process, as well as the rationale behind its selection. The first part provides a general overview followed by dedicated sections detailing the execution of the research.

To support the definition of the research goals (RGs), we make use of context-specific terminologies, which will be used throughout the whole document and are specified in the glossary:

- Software development strategy: the overall approach adopted by the company to carry out the product development.
- Engineering activities: the activities needed to bring a product from idea to market[1].
- Engineering elements: any method/practice/tool/framework/technique/-documentation/artifact contributing and supporting the engineering activities.
- Quality attributes: those overall factors that affect run-time behavior, system design, and user experience. They represent areas of concern that have the potential for applications to impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time or user centric issues[2].
- Operational dynamics: the approaches of the company in making decisions.
- Growth: an increase in the company size respect to the initial conditions in terms of either employees or users/customers, and product complexity in terms of handling an increasing number of feature requests.

[1] Instances of traditional engineering activities are, among others, requirement engineering, design, architecture, implementation and testing.

[2] Definition adapted from [100].

4.1.1 Research goal definition

The aim of this research is to *understand how software development strategies are engineered by practitioners in startup companies in terms of level of: structure, planning and control of software projects, in the period of time that goes from idea conception to the first open beta release of the software product.*

The above mentioned goal is structured according to the GQM paradigm [101], as shown in Table 4.1.

Object of study	Software development strategy
Focus	Level of structures, planning and control of software projects
Purpose	Understand
Perspective	Technical practitioners
Context	Newly founded software startup, from idea conception to first release

Table 4.1: GQM template, five components of the research goal [8]

In order to achieve the research goal a set of subgoals has been defined:

1. Explore the SE literature to understand what has been achieved so far by researchers in studying software startups and contributing to the creation of a scientific body of knowledge.
2. Understand how and why startup practitioners use engineering elements in the creation of a new product.
3. Investigate the perception of a perfect hindsight early-stage development strategy among practitioners, which can foster the productivity in subsequent lifecycle stages.

4.1.2 Research questions

Initially the boundaries of the research domain were set by means of non-systematic *literature surveys*, which provided the initial keywords used further in the research process. Moreover it helped us in defining the research questions (RQs) addressed by this thesis:

- RQ-1: What is the state-of-the-art in the SE literature pertaining to engineering activities in startups?
  - RQ-1.1: How is the body of knowledge distributed in literature?
  - RQ-1.2: What is the industrial relevance and scientific rigor of the published studies?
  - RQ-1.3: What are the features which characterize the context of software development in startups, reported in literature?
- RQ-2: What is the current state-of-practice related to software development strategies in early-stage startups?

  - RQ-2.1: How do startups structure and execute the main engineering activities?
  - RQ-2.2: How are product quality attributes considered by startups?
- RQ-3: What development strategies can be adopted by startups with the aim of facilitating future growth?
  - RQ-3.1: What are the operational dynamics that influence the adherence to adopted development strategies?
  - RQ-3.2: What are the main objectives of software development after the first product is released?
  - RQ-3.3: What development strategies help in reaching long-term objectives?

The above mentioned research questions are used as guidelines to outline the results and the analysis throughout the thesis document, referring to them with their unique identifier. RQ-1 is addressed in the analysis of mapping study results (Section 5.1), RQ-2 in the chapter dedicated to the theoretical model (Section 6.5) and RQ-3 in the chapter dedicated to dynamics and evolutions of startups (Chapter 7). Finally the findings are summarized in Conclusions (Chapter 9).

4.1.3 Research methodology overview

In this subsection we present a general overview of the research methodologies undertaken in this thesis for tackling the above-defined RQs. After the identification and refinement of the research problem with non-systematic literature survey[3], we combined different research methodologies in a *grounded theory* (GT) approach [103] to obtain a model of software development in early-stage startups. The research methodology chapter has been divided in two sections, respectively concerned with:

1. The review of the state-of-the-art through a *Systematic Mapping Study* (SMS) (see Section 4.2).
2. The *case study* (see Section 4.3) conducted in 13 early-stage startups through *field interviews* combined with *follow-up questionnaires*.

Figure 4.1 provides an overview of the overall research process, emphasizing the cross-methodological approach conducted exploring the state-of-the-art and the state-of-practice.

[3]The surveys have been executed and refined multiple times on the Compendex database which is the single database indexing the great majority of articles [102].

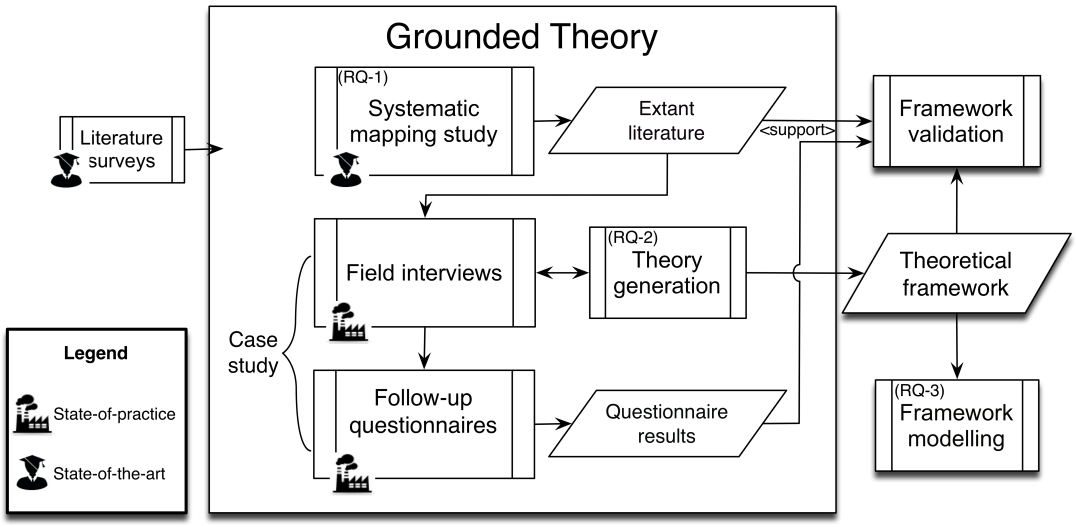


Figure 4.1: Complete Methodology

From the initial definition of the research problem - obtained with *literature surveys* - we moved to the *grounded theory* (GT). GT has been conducted through: the *systematic mapping study* obtaining the *extant literature*; *field interviews* to generate a *theoretical framework*; and *follow-up questionnaires* to triangulate the data in support of the *framework validation*. The main output[4] of this methodological approach [104] is the *theoretical framework* that explains relevant aspects of the underlying phenomenon of software development in early-stage startups (see Chapter 6). As suggested by Glaser [105] and following the example of a similar research carried out by Coleman in 2005 [16], we performed the *framework validation* by systematically comparing our results with the *extant literature* and the empirical data (see Section 6.6). Finally with *framework modelling*, we conducted an in-depth comparative analysis with existing evolutionary and decision-making models, to understand the operational dynamics adopted during the development process (see Chapter 7).

The answer to RQ-1 (state-of-the-art) is discussed in the results and analysis of the SMS (see Subection 5.1.4), RQ-2 (state-of-practice) is discussed in the *Implications* of the theoretical framework (see Section 6.5) and RQ-3 is answered in Chapter 7. The findings are summarized in Chapter 8.

4.1.4 Rationale for methodology selection

Before deciding on which research methodology best would have fitted to the research problems, we had to iterate and change them a number of times in order to come up with a reasonable solution. The main challenge was to find a methodology that would allow us to obtain good and significant results in an

[4]The full list of contribution has been presented in *Introduction*.

arena that is mostly unexplored within the time/effort frame of a master thesis. Under the guidance of our supervisors, we had to get through a difficult trade-off between trying to deepen the understanding of a specific problem or covering a wide spectrum of topics.

Going extensively into a specific problem in software startups context would not have been worth trying because we didn't find a sufficient number of studies that constituted a solid body of knowledge to support the definition of a narrowed research focus. On the other hand, trying to select a broad research topic would have carried the risk of spending a lot of time in achieving a better understanding of the problem without being able to provide any significant or concrete results.

We iterated different approaches before starting the actual research. This led us to explore several possibilities, and gave us the time to perform several literature surveys about both research methodologies and software startups: we started to re-formulate the research problem and research methodology. To complete this process we designed an online survey about engineering practices with the idea of distributing it on a large scale to gather quantitative data. However, when we obtained the first results we immediately realized that what we needed was a more flexible design approach, fine-tuned to the research problem, that allows to gather concrete qualitative results and at the same time can retrieve relevant works from related areas of the SE literature[5].

As we wanted to have the ability to transfer our ideas to startup practitioners, we designed a comparative study of the *state-of-the-art* and the *state-of-practice* following *evidence-based software engineering* principles [106, 107] and providing empirical evidences supporting startups' decisions.

For the review of the existing literature we chose the SMS, which is one of the most appropriate methodologies capable in dealing with wide and poorly-defined areas [1, 108]. A more traditional *Systematic Literature Review* (SLR) [108] would have been a less viable option due to the wide breadth of the research problem and an apparent lack of a strong academic support material (differences between SLR and SMS are thoroughly discussed by Petersen et al. in [1]). Moreover a SMS can be optimally coupled together with an evolutionary *grounded theory* approach, driving the direction of the case study towards most interesting and uncharted areas.

Thus, the rationale which led us to the choice of grounded theory for the case study, can be synthesized with the following statements:

- Given the lack of an integrated theory in the literature, a GT approach allowed a theory to naturally emerge based on experiences gained from technical practitioners.

[5]The inadequacy of surveys in this context has emerged in the empirical study and it is discussed in the comparative analysis (see Section A.2). This argument is partially supported by [18].

- GT is supported with good guidelines that are well documented for conducting theory-generating research [2, 58].
- GT is well known to fit with research problems related to human behavior[6].

We fine-tuned the general guidelines and suggestions of grounded theorists to our specific case, where a lack of existing theories points towards the use of an inductive approach to enhance the comprehension of complex phenomenon. In order to clarify doubts and collect quantitative data that we couldn't capture during interviews, we designed a customized online questionnaire for each company that participated to the interview sessions. Finally, to support the transparence of the research process, we documented the whole procedure in a *diary-like* application constantly updating new ideas and findings as they emerged from interviews and literature, letting our supervisors adjust the study trajectory.

In the process of selecting the methodology, several alternatives have been considered, analyzing their impact on the outcome. The explorative nature of the initial research problem made the possibility of using a purely quantitative approach not suitable for *understanding* this socio-technical phenomenon. On the other hand, the broad research problem forced us in selecting a methodology featuring a *flexible design* approach and to discard those methodologies that require a *fixed* design: we needed to adjust the direction of the research as we progress through, avoiding any early choices that would have limited such flexibility.

A valid alternative approach that could have been undertaken is an observational study. In fact, being a passive observer, into the first phases of the creation of a software product in startup companies, would have been a viable option. However, to achieve the same results we obtained through GT, an observational study would have required observations of a number of different startups for a long-enough timeframe in the creation of their first product and this would have been unreasonably expensive in terms of effort and time for the startups' practitioners.

Additionally, among other traditional qualitative methods, some of them require less effort than a GT approach, but they are not as systematic and rigorous as GT. Then, in view of a lack of studies exploring the topic, to create a solid scientific base for further investigation, the best possible choice is to select the most reliable and documented research methodology [103].

In the following sections of this chapter we detail the research methodology and execution of the systematic mapping of the literature (4.2) and the grounded theory case study (4.3).

[6]Software development is to a great extent characterized by human-intensive activities.

4.2 Systematic mapping study

A first preliminary literature survey in the SE databases revealed a quite wide gap in works addressing our research problem but at the same time it showed us how broad the domain was. In fact, understanding the general approach undertaken by software startups to develop software requires us to explore different research areas.

One of the most appropriate methodologies, which helps researchers to investigate wide and poorly defined fields, is the *Systematic Mapping Study* (SMS), also known as *Scoping Study* [108]. Despite SMS has its roots in medical research, the growing body of knowledge of SE and the tendency towards evidence-based approaches [109] made SMS a technique increasingly adopted by software engineers [110]. SMS provides systematic guidelines to classify existing works obtaining an overview of the area, which can be easily transferred to other researchers and startups' practitioners.

A definition which confirms the suitability of this methodology is provided by Kitchenham et al. in [108]: *"SMSs are designed to provide a wide overview of a research area, to establish if research evidence exists on a topic and provide an indication of the quantity of the evidence [...]"* - and the authors continue - *"[...] with the aim of influencing the future direction of primary research."*

4.2.1 SMS - Process Overview

For executing the *Systematic Mapping Study* we scanned studies throughout the scientific databases following the process suggested by Petersen et al. in [1] as shown in Figure 4.2, where the upper blocks represent an *action* and the underlying block represents its *output*.

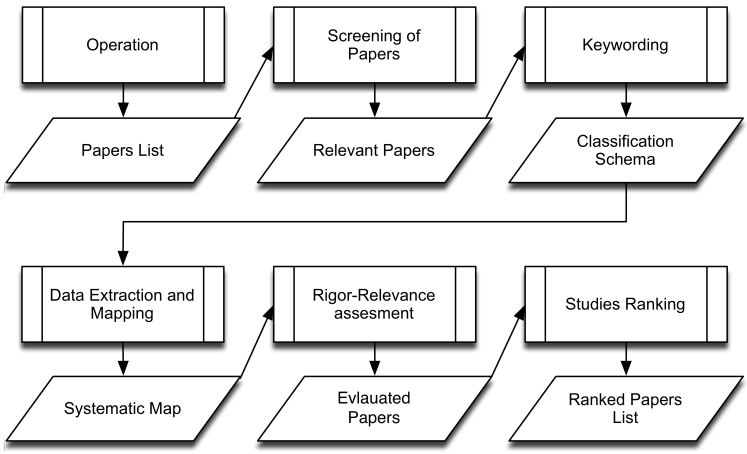


Figure 4.2: Systematic Mapping Study process overview adapted from [1]

Starting from the research problem defined in 4.1.1, the first step of our systematic study was *Operation*. The whole SMS procedure has been executed simultaneously in pair on the same screen, handling conflicts at the end of the *data extraction and mapping* process by reviewing the rationale of similar decisions taken during the *screening of papers*. When necessary we performed an in-depth review of the article[7].

4.2.2 SMS - Operation

The first step for conducting a systematic search was iteratively building a *search string* composed by different *keywords* that emerge by reviewing the scope of the research questions. To reflect our wide research problem we needed a very broad *search string*, which was generated from three *core concepts* identified as fundamental to our problem domain, i.e.:

- Software startup.
- Development.
- Process.

Following Rumsey's guidelines [111], for each core concept we identified synonymous, related concepts, broader concepts, wider concepts, alternative spelling and part of speech that contributed to elaborate the list of terms included in our search string.

The second step was the identification of relevant and significant databases to include in our search process. To increase the internal validity of our research and the chances of finding relevant material we included in our target 5 of the most important peer-reviewed scientific online databases, namely:

- ACM Digital Library [112].
- Compendix/Inspec [102].
- IEEE Xplore [113].
- ISI Web of Science [114].
- Scopus [115].

To be more confident of the results, we included in the set of databases *Google Scholar* [116], which indexes an extremely large set of data, both peer and non-peer reviewed. All the mentioned databases were selected in view of their ability to handle advanced queries.

We then proceeded in customizing the *search string* and iteratively perform the search while refining search *keywords*. The overall process we used is shown in Figure 4.3.

[7] If the conflicts persisted after an in-depth review of the article, we let a third expert person (i.e. our supervisors) take the final decision.

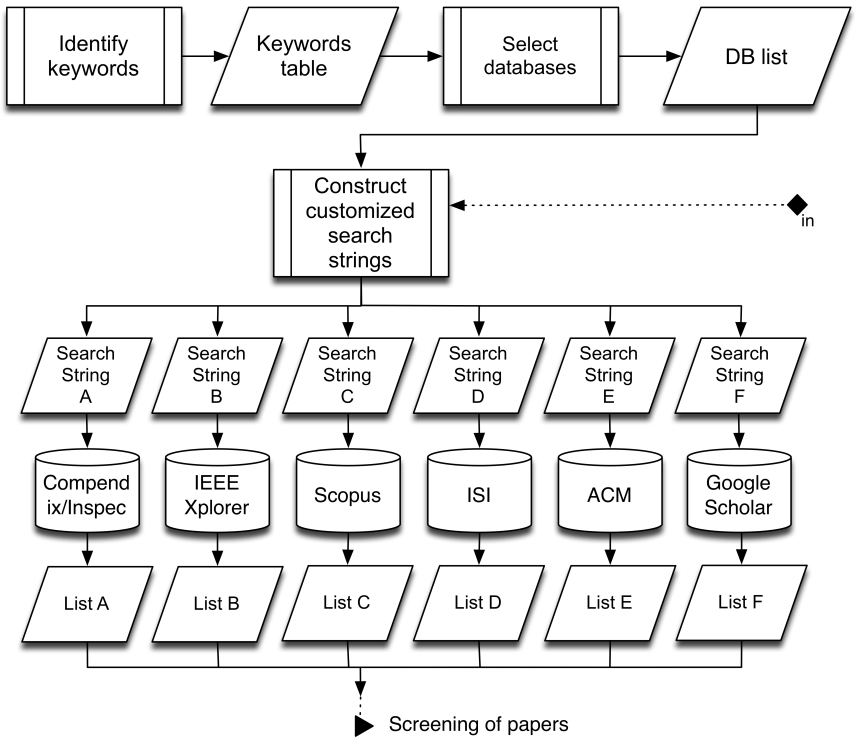


Figure 4.3: Systematic Mapping Study - Operation

The procedure established in the figure above produces six different outputs - one per database - which are further analyzed as shown in Figure 4.4.

The terms which formed our final search string are shown in Table 4.2, next to the core concept they represent[8].

Core Concept	Terms
Software Startups	software startup*; software start-up*; early-stage firm*; early-stage compan*; high-tech venture*; high-tech start-up*; start-up compan*; startup compan*; lean startup*; lean start-up*; software package start-up*; software package startup*; IT start-up*; IT startup*; software product startup*; software start up*; internet start-up*; internet startup*; web startup*; web start-up*; mobile startup*; mobile start-up*;
Development	develop*; engineer*; model*; construct*; implement*; cod*; creat*; build*;
Strategy	product*; service*; process*; methodolog*; tool*; method*; practice*; artifact*; artefact*; qualit*; ilit*; strateg*; software;

Table 4.2: Mapping Study - Search String Keywords

In forming our *search string*, detailed in Appendix A.3 (Table A.2), terms belonging to different core concepts were linked with the logical *and*, while terms on the same row have been linked with the logical *or*.

[8]The symbol * indicates the use a wildcard, mainly for including plural forms and alternative spellings.

After executing the search strings on each database we saved the bibliographic references in 6 separated *BibTeX* files, containing a total number of 1417 items, coming from different databases as shown in Table 4.3.

List	Database	Items
List A	Inspec/Compendex	508
List B	IEEE Xplore	104
List C	Scopus	357
List D	ISI Web of Knowledge	219
List E	ACM Digital Library	71
List F	Google Scholar	158
	Total:	**1417**

Table 4.3: Retrieved papers source overview

4.2.3 SMS - Screening of papers

To execute the screening of papers we defined and followed a rigid step-by-step workflow, shown in Figure 4.4.

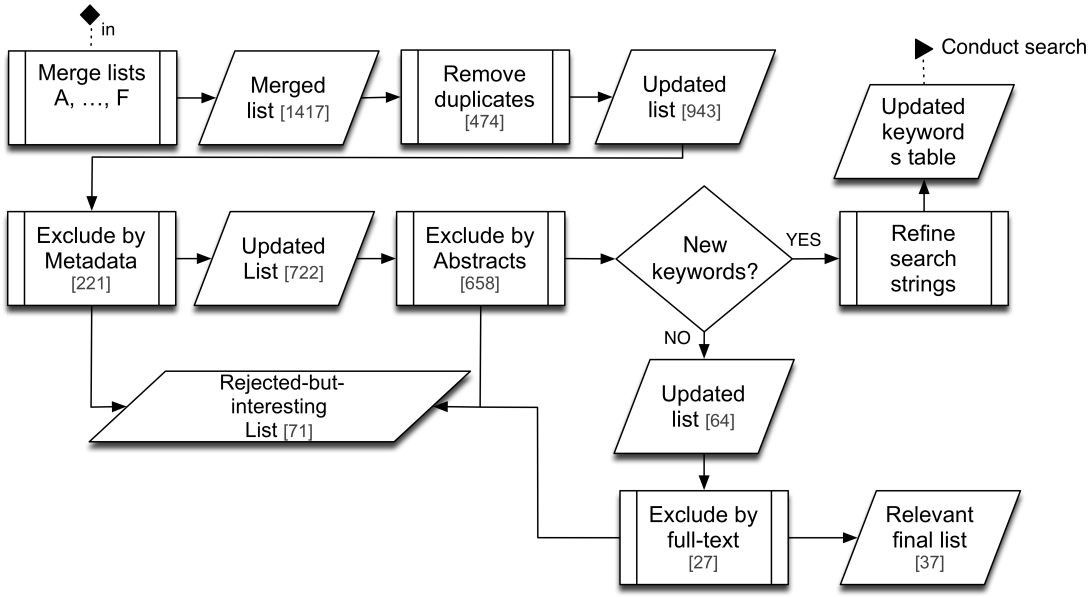


Figure 4.4: Screening of papers

With the support of a tool for bibliographic management [117] we merged the six lists together and removed the duplicate items in two distinct steps: first we used an automated feature to detect and remove items based on the author, year and title of the publication. Then, we manually deleted other duplicates that were not detected by the tool, for a total number of 474 duplicates.

Before proceeding with further exclusions we defined our *inclusion/exclusion criteria*, aiming to include only peer-reviewed papers which brought some contri-

bution to the body of knowledge of software engineering, by explicitly mentioning results related to software development in startups. Additionally we decided to reject articles with the following characteristics:

- Non-peer reviewed (grey literature, books, ...).
- Not written in English.
- Clearly obsolete results (more than 20 years old).
- Related to non-software companies (biotech, manufacturing, electronics, ...).
- Related only to established companies (VSE, SME, research spin-offs).
- Related only to technicalities of startups (algorithms, programming languages, ...).

We started analyzing the metadata of each article in pair (title, venue, keywords and the publication year) to exclude items that clearly didn't satisfied the inclusion criteria (221), leaving important decision for the subsequent steps. Then, going through a more in-depth review, we analyzed abstracts of each paper to determine whether the article satisfied our inclusion criteria, eliminating 658 items. In case of internal conflicts, hard decisions, or incomplete abstracts we read through an entire article, excluding 27 additional items. As shown in Figure 4.4, while screening abstracts we improved the search process by enriching the initial search strings with new keywords identified in retrieved articles and iteratively conducting a new search (Figure 4.3).

During the screening of papers, we have found that some articles, not included, had the potential of contributing to our analysis. For this reasons we created an additional list (*Rejected-but-interesting List*) that we filled during the exclusion process with 71 managerial articles filtered and analyzed in a dedicated section in Appendix A.2.1.

During this review process we kept track of the rationale for each exclusion, visually presented in the right column of the Figure 4.5.

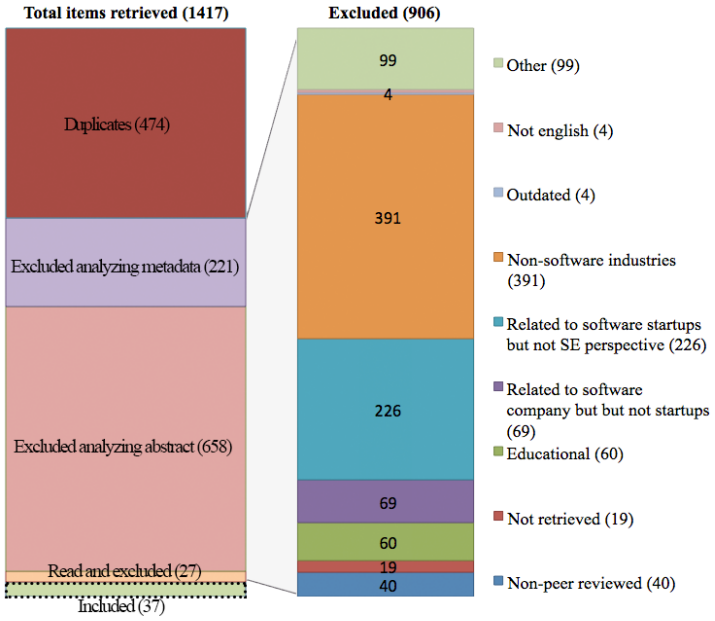


Figure 4.5: Studies selection process overview

The majority of paper we excluded were not related to the software industry (391), not interesting under a SE perspective (226) or related to established companies (69). Other excluded work were off-topic (99), educational (60), non peer-reviewed (40), non-English(4) or outdated (4).

4.2.4 SMS - Keywording

To generate the *classification schema* we used the technique of *keywording*, following the systematic process illustrated in Figure 4.6.

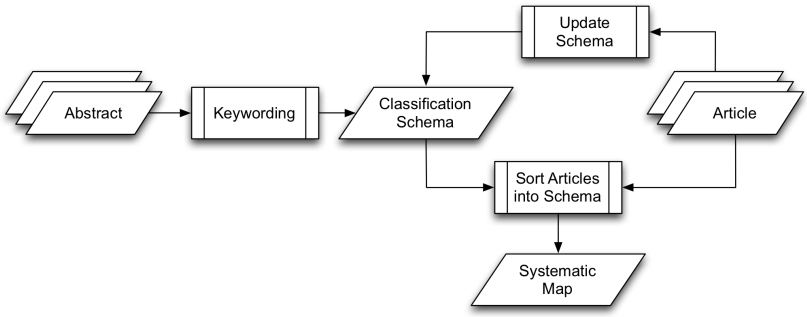


Figure 4.6: Classification schema creation adapted from [1]

The first step consisted in reading abstracts of the relevant studies, assigning them a set of keywords, to spot the main contribution area of a paper. To get a rough understanding of the research area represented by the sample of

collected papers, we combined together the defined keywords forming a high-level set of categories, which initially generated a first representation of the selected articles. Then, by progressively trying to fit the articles into categories, the schema underwent a refinement process by being updated to receive new data.

4.2.5 SMS - Data extraction and mapping

After we defined the *classification schema* (presented in Subsection 5.1.1)we proceeded to systematically extract data from the selected relevant studies. We filled a spreadsheet where, for each article, we assigned a specific category of the *schema*, and additionally we collected other information inspired by other similar studies [118, 119]:

- Article title.
- First author.
- Year of publication.
- Synthesis of results (one-line).
- Keywords.

When executing the actual extraction, we motivated with comments the non-obvious decisions we took in a spreadsheet, writing the rationale behind the belonging of an article to a certain category rather than another. The output of this process is shown in *Results and Analysis* (see Subsection 5.1.1) and represents the final *systematic map*. We took advantage of the data extraction process to identify an additional relevant aspect which emerged while reading abstracts and documents: the recurrent patterns of common attributes among startup companies (resulting *themes* are reported in Subsection 5.1.3). Moreover, the bibliography of each extracted article has been screened to identify other possible relevant articles to our research, adopting the *snowballing* technique[9] [120]. Collecting this information early in the research process helped us to build a conceptual baseline that we utilized to assist the selection criteria for initial convenience sampling of companies in the case study. Additionally, we used *themes* in combination with *grounded theory* results for the creation of the the theoretical model (see Chapter 6).

4.2.6 SMS - Rigor-relevance assessment

The primary challenge of SE is to transfer research results and knowledge to practitioners showing valid and concrete advantages [121]. To assess how results

[9]Note that in our case with snowball sampling we didn't identify any additional relevant article, but in the case that relevant articles were retrieved, a refinement of keywords and reprocess of the *systematic mapping study* would have been required as described in Figure 4.3.

are presented in our studies' selection, we integrated the traditional framework of SMS with an additional step, that is the evaluation of paper's *rigor* and *relevance*.

We used a systematic and validated model developed by Ivarsson and Gorschek [121] to evaluate the scientific rigor and the industrial relevance of each paper. The model provides a complete set of rubrics to measure *rigor* and *relevance* dividing these two factors in different aspects, and quantifying the way in which each aspect is considered in the study (also see [122] for a real application).

As described by the authors, rigor refers to the precision or exactness of the research method used and how the study is presented. Aspects that are considered are:

- Context - description of development mode, speed, company maturity and any other important aspect where the evaluation is performed.
- Study design - description of the variable measured, treatments, control used and any other design aspect considered.
- Validity - description of different types of threats to validity evaluating conclusion, internal, external and construct validity.

On the other hand the relevance of an article consists of the realism of the environment where the study is performed and in the degree to which the research method undertaken has been investigated in several literature reviews. Aspects that are considered are:

- Subjects - use of subjects who are representative of the intended users of the technology.
- Context - use of settings representative of the intended usage setting.
- Scale - use of a realistic size of the applications.
- Research method - use of a research method that facilitates investigating real situations and relevant for practitioners.

Aspects related to the rigor of the study are scored with three score levels: *weak*, *medium* and *strong* description. Whilst, aspects related to relevance are scored 1 if contributing, 0 otherwise. For the sake of brevity we reported only the high-level quantification table (see Table 4.4). The detailed rubrics, used to evaluate the articles, can be found in [121].

Rigor Ri = Ri1 + Ri2 + Ri3	
Context described (Ri1)	
Study design described (Ri2)	
Validity discussed (Ri3)	
Each aspect scored according to following scheme	
Weak presentation	0
Medium presentation	.5
Strong presentation	1

Relevance: Re = Re1 + Re2 + Re3 + Re4
Context (Re1)
Research method (Re2)
User/Subject (Re3)
Scale(Re4)
Each aspect scored 1 if contributing to relevance, 0 otherwise

Table 4.4: Rigor and relevance quantification

Finally to obtain the final score of rigor and relevance the sum is performed according to the number of aspects that are classified as contributing to the industrial rigor and relevance respectively.

With this approach we intended to extend the scope of the traditional *Systematic Mapping Study*, which is limited to paper classification without assessing the quality of underlying studies. In our research, by combining a classic SMS with the study of rigor, relevance and the creation of a simple ranking function (see next subsection), we obtained a quick tool to partially overcome the above-mentioned limitations.

4.2.7 SMS - Ranking of studies

The final step, which concludes the mapping study, is the creation of a ranking function. We assigned a score to each paper, evaluating it in face of its *classification schema*'s categories, *rigor* and *relevance* score, and two additional factors which characterize the venue of the study and the publication year.

We provided a rough estimation of the actual *value* that each paper brought to the research domain, by giving more importance to recent rigorous journal articles entirely devoted to the topic and presenting empirical results relevant to practitioners. To reflect those criteria, we used tables for converting each factor into an arbitrary numerical value in the range between 0 and 10. The conversion tables which were used to quantify the internal score of each dimension are discussed in Appendix A.3.3. The final ranking of the 37 relevant articles is presented in *Results and analysis* (see Subsection 5.1.4).

4.2.8 SMS - Validity threats

We identified potential threats to the validity of the systematic mapping and its results, which are presented in this section together with the mitigation strategies, structured following the example of a recent study [123].

Publication bias

Systematic reviews suffer from a common bias due to the general problem that *positive outcomes are more likely to be published than negative ones* [123]. In our study this threat is moderate, since the conclusions we draw and the *theoretical framework* we present are based on empirical data. The literature review is only used to make a comparison and validation of our results.

Threat to identification of primary studies

The approach we used to construct the *search string* (see Subsection 4.2.2) aimed to collect the larger number of articles related to software development in startups as possible.

However, a limitation of the current search string lies in the non-inclusion of the stand-alone terms *"startup"* or *"start-up"*, to avoid the screening of more than 20 thousands papers, mostly irrelevant because related to the english phrasal verb "to start up", largely used in many disciplines to indicate the commencing moments of an engine. Then, to mitigate the risk of losing some relevant articles, we included in the search string many redundant terms in logical *or*, to increase the chances of catching any paper somehow related with software development in startup companies limiting the risk of obtaining an unreasonable number of articles with an extremely low *precision* (defined as the ratio of retrieved relevant and all retrieved [124]).

Considering the precision rate of the current *search string* (3.92%), the value we obtained is very low anyway, with 37 studies selected from an initial sample of 943. However we were not interested in obtaining high precision as much as we wanted to obtain an high *recall*, that is the expressed by the ratio of retrieved relevant articles and the existing relevant items [124]. Despite the recall is based upon an unknown quantity hard to estimate (especially in an area where no systematic reviews have ever been conducted to the best of our knowledge), to have our evaluated and validated relevant list we submitted our systematic review results to prominent researchers in the area (identified with the mapping study itself) immediately after executing the methodology. The risk of leaving out relevant results is further mitigated by the use of multiple databases, which cover the majority of scientific publication in the field.

We were not able to retrieve 19 items since they were not available neither in online catalogs nor in the three libraries we consulted. However, this is a minor risk as we had access to their titles, keywords and venues, which gave us a good degree of confidence they were not relevant. Additionally, considering our 3.92% precision rate, the number of relevant articles in this small sample would have been statistically around *one*.

Another threat was given by the fact that relevant studies could have been published after the actual execution of the systematic review. To mitigate this risk

we used the alerts featured on *Engineering Village* and *Google Scholar* database which allowed us to receive a weekly newsletter with new items targeted by our *search string*. At the current date (29th October 2012) no new relevant studies emerged.

Furthermore we noticed a high inconsistency in the use of the word *startup* by different researchers, even in the same area. We couldn't identify a single and widely accepted definition of *startup*, but we identified multiple and somewhat conflicting definitions. With the support of thesaurus and librarians specialized in software engineering we considered their suggestions for additional improvements. Under these conditions, the attempt to identify a body of knowledge and restrict the scoping of our research was highly challenging. However, with the systematic study we were able to get a complete overview of how the term is used and sometimes misused, so we could focus the remains of our research on the early stage of startup companies, that is the most neglected stage by empirical primary studies so far.

Since startups and entrepreneurship in general are appealing for many sectors of the economy, an additional threat lies in the fact that some relevant information can be found in other academic sectors not considered in this systematic review (e.g. Finance, Marketing, Management, or in several non-peer reviewed sources such as books, technical reports or essays). To reduce the potential impact of this risk, we integrated our systematic study with a non-systematic review of prominent work in the *grey literature* and in other siblings disciplines (see appendix A.2).

Threats to study selection and data extraction

An important bias to consider - as discussed in [118] - is related to the selection of publications and the data extraction process, that in our case has been mitigated with an up-front definition of the inclusion/exclusion criteria [108]. The extent to which a SMS is able to provide an overview of software engineering topics has been sufficiently reported by the comparison analysis with the systematic literature review, discussed in [1].

The selection of relevant articles can be further biased by the personal opinions of researchers executing the process. To mitigate this threat and partially supply to our junior experience, we defined and documented a rigid protocol for the selection of each paper and we collaborated to mutually adjust each other biases by conducting the selection together and dedicating a reasonable amount of time to review conflicts, as suggested by [108]. Furthermore we documented the rationale behind the exclusions, to support us in the process of consensus creation by coherently consulting the history of previously taken decisions.

Another threat is related to researchers' personal judgments, which can interfere with the evaluation of rigor and relevance of selected studies. However, the model we used to perform the assessment [121] provides rigorous guidelines and

detailed rubric tables which have been observed to express our judgment more objectively.

An additional threat can undermine validity of the *ranking* of selected studies (see Subsection 4.2.7) that we computed, as the scores for each category and the weights for each dimensions have been arbitrarily chosen to reflect our criteria. However, we consider this threat marginal as we didn't draw any important conclusions based on the ranking table and we specified that the score assigned to each paper has been used only to sort lists of relevant studies in the tables along the document. Furthermore we used an automatic spreadsheet to compute the final scores, which allowed us to adjust scores and weights, observing the effect of the final ranking in real time. For validating our ranking, we tried to modify scores/weights values several times, and we observed that the final ranking was not significantly altered by reasonable numerical adjustments, as long as we kept the order of importance between concepts.

Additionally, we used multi-methodological triangulation [125, 104] to confirm the validity of our results, by intersecting the data extracted with *systematic mapping study* to the empirical data which we obtained by *semi-structured interviews* and *follow-up questionnaires*.

4.3 Case study

4.3.1 Case study - Overview

For the case study of this research we investigated the software development approach undertaken by practitioners in early-stage startups. Following a *Grounded Theory* (GT) methodology [105], we executed 13 semi-structured interviews integrated with follow-up questionnaires. From this, we elaborated and extracted a theoretical framework explaining the underlying phenomenon of software development in startups.

Grounded Theory methodology is described by Robson as *"the best approach to answer the question - What is going on here?"* and defined by its creators [103] as *"a set of well-developed categories (e.g. themes, concepts) that are systematically interrelated through statements of relations to form a theoretical framework that explains some relevant social, psychological, educational, nursing or other phenomenon"*.

Despite GT has its roots in the social sciences, in the recent past it has been increasingly used by *Information System* researchers and for many years it has been almost ignored by the SE research [2]. In 1997 Bertelsen advocated for a need of more qualitative research in SE [126], and in the last decade we assisted to an increasing number of publications that used a GT-like approach in SE [29, 2, 58].

Following the *grounded theory* principles, detailed in this section, we at-

tempted to capture the most relevant aspects of software development from startup practitioners, letting a theory emerge from the interviews and adjusting the research hypotheses and questions as we proceeded through. During these interviews we collected data related to engineering activities undertaken by startups in the time frame between the idea conception and the first open beta release of the product. Then, we proceeded to the analysis of the data, finding important relations among concepts with a formal approach.

As suggested by Coleman, in view of the different versions of *grounded theory* which emerged in the last years, researchers should indicate which *"implementation"* of the theory is being used [2]. In fact, after the methodology was initially introduced, its original authors had some divergences regarding how the theory should be formed by analyzing the data (Strauss and Corbin on one side, Glaser on the other). Glaser advocated for a more *puristic* approach, where the theoretical categories should *"naturally emerge from the data"*, whilst Strauss and Corbin formulated an updated version of the methodology, which leaves to the researchers an higher degree of freedom. The latter approach empowers researchers' *"theoretical sensitivity"* [127], and encourages them to outline the research problem beforehand. Since the knowledge obtained from parallel execution of the SMS (see Section 4.2) and our direct experience with startups companies provided a good initial level of knowledge, in this study we use the Corbin and Strauss approach [128].

Performing GT studies requires well-trained researchers with a good level of experience. To partially supply to this lack we trained ourselves for one month in a low-risk environment, by simulating both face-to-face and remote interviews. Hence, we gained a good skill-set which allowed us to face challenges of real-setting interviews by dividing the several tasks between the two of us (recording, taking notes, checking time, mark checklists, elicit answers, extrapolate data, ...). We recorded ourselves during these simulations, and improved our performance by listening the recordings and mutually adjusting each other's parts.

The basics elements which constitute the grounded theory methodology areq *concepts*, *categories* and *propositions*. Concepts are the basic units of analysis: they represent the conceptualization of underlying phenomenon captured by labels called *codes*, which are extracted from the interview transcripts. The second elements of GT are the categories, which are the high level abstraction of the concepts they represent. Categories are generated by comparing similarities and differences among the different concepts. The third element are propositions that describe the relations between a category and its concepts and between discrete categories [127]. Those elements were identified by means of coding processes performed on interviews' transcripts.

The coding process is divided in three parts[10]:

[10]GT does not force researchers to move between these parts in a consecutive manner.

- Open coding.
- Axial coding.
- Selective coding.

Open coding refers to the extrapolation, labeling and categorization of concepts from the raw data. *Axial coding* refers to the activity of putting together different concepts into categories, defining relations between them. Finally *selective coding* defines connections between discrete categories, integrating them into the initial *theoretical framework* [128].

As discussed above, the generation and development of concepts, categories and propositions is conducted with an iterative approach, where researchers constantly adjust the *theoretical framework* to form the *emerging theory*, that is the abstraction of identified conceptual terms, from the understanding of the dynamics of the substantive field of research [127].

To provide the reader with a practical example of the coding process, here we present a typical GT procedure in a fictitious context. Assume that a GT research investigates the use of test-driven development (TDD) in comparison to test-last development (TLD) to understand which approach leads to higher-quality products. Basic units of analysis (*concepts*) might be extrapolated by statements of interviewed practitioners such as: *"TDD helps us to focus on which results should be returned by functions, shaping an initial modularization of the final system"*. From this, the researchers extrapolate another *code*: *"TDD aids system modularization"*. Then another interviewee claims that *"when we moved from TDD to TLD we had problems in de-coupling modules in the system"*. From this we create another *code*: *"TLD made system cohesion harder"*. Afterwards, assuming that these same concepts are significantly reported by the majority of the practitioners in the sample, the two concepts might be abstracted into a *category* labelled *"TDD enhances architectural design"*. The category *propositions* describe its relation to underlying raw *codes*, in this case indicating in which terms TDD enhances architecture and in what context. Assuming that researchers identified other categories such as *"TDD decreased the number of reported bugs"* and *"TLD augments the project complexity"*, these two can be linked with the first one to formulate an emergent *theory*, e.g.: *"By adopting TDD developers improve architectural design controlling the final project complexity, leading the system to higher product reliability in terms of lower number of defects"*[11].

Going back to our study, beside the GT interviews, in order to clarify doubts and collect quantitative data that we were not able to capture during interviews, we designed a customized online follow-up questionnaire for each company which participated in the interviews. In particular, with the *follow-up questionnaire*, we aimed to capture, evaluate and assess aspects related to:

[11]This is only a very expedite example of GT coding process which aims to facilitate the reader in understanding the coding process.

- The fulfillment of quality aspects that have been considered important in the interviews.
- The effort distribution between the different engineering activities of software development.
- The contribution level of engineering elements identified during the interview process.
- The level of satisfaction with the software development approach.
- Important retrospective thoughts about software development approach with a perfect hindsight.

Despite some of these aspects were already partially covered during interviews, by providing to our respondents a different mean of answering similar questions, we could validate and review our *grounded theory* results. Thanks to the increased *process awareness*, which arises after the interview (as reported by many respondents), questionnaire results were used to confirm and adjust the emerging theory and they were not used to draw any important conclusion.

Following principles and guidelines for designing good SE questionnaires (in particular Wholin [129] and Kasunic [104]), we created an online template questionnaire[12] with the basic structure and general questions to be adapted to each company immediately after each interview. The initial design of the questionnaire - divided in four parts with different concerns - is presented in Subsection 4.3.2. We combined multiple choices with open questions and iteratively refined it by collecting feedback with students' companies (the same *safe-environment* companies utilized during interview design process detailed in Subsection 4.3.2).

A complete overview of the overall case study methodology and execution is shown in Figure 4.7, which presents how we tailored the general GT methodology to our specific needs.

[12]We used an online service for designing and executing follow-up questionnaires: `http://www.surveymonkey.com` .

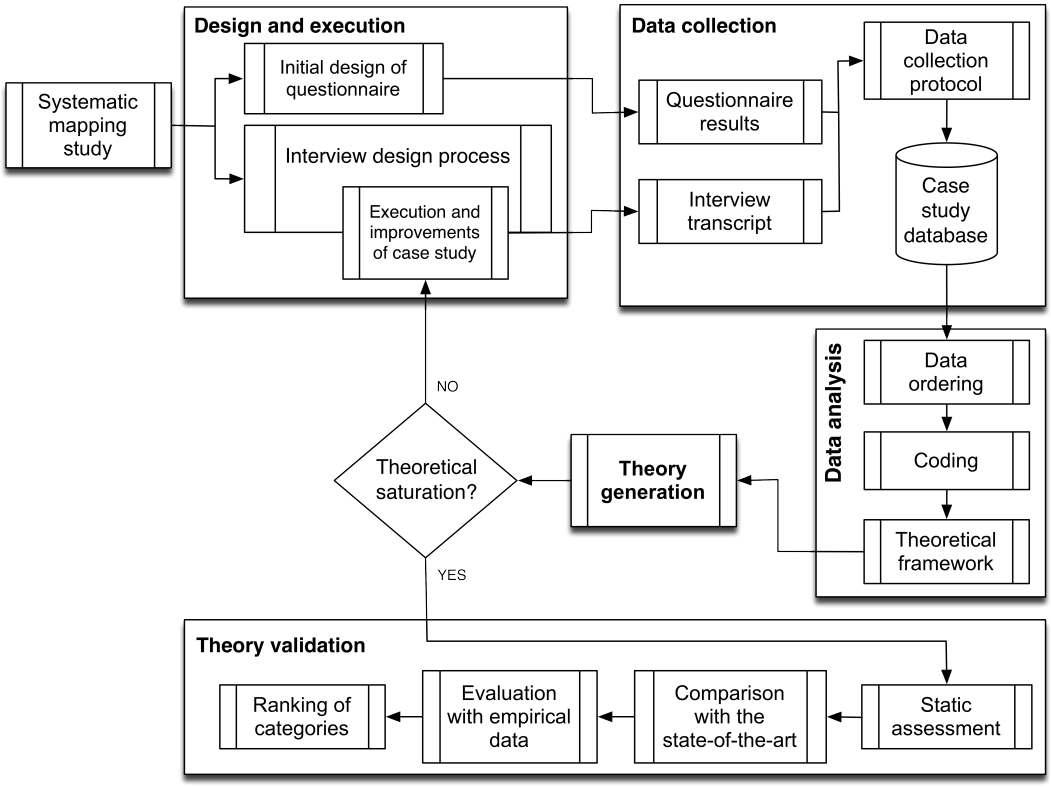


Figure 4.7: Grounded theory study process overview

The initial design of the case study is supported by the results of the *systematic mapping study* which contributed to define the initial questions for both questionnaires and semistructured interviews. The process depicted above (Figure 4.7) is evolutionary and affects the design at each new iteration. In *data collection* the empirical results are stored in a single *case study database* and subsequently processed in *data analysis* to form the theoretical categories. At each iteration the new emergent theory is updated following a formal procedure (*Theory generation*), and after verifying that the *theoretical saturation*[13] of categories has been achieved, we finally proceeded to *theory validation* or performed another complete cycle.

The detail process is thoroughly described in the following subsections which are structured according to the five macro phases depicted in bold in Figure 4.7: *Design and execution* (see Subsection 4.3.2); *Data collection* (see Subsection 4.3.3); *Data analysis* (see Subsection 4.3.4); *Theory generation* (see Subsection 4.3.5); and finally *Theory validation* (see Subsection 4.3.6).

[13]The point at which executing more interviews wouldn't bring any additional value for constructing the theory.

4.3.2 Case study - Design and execution

The first step was supported by the *systematic mapping study* (see Section 4.2), in which the research problem was refined throughout different stages. In fact, in the very first phase the initial problem has been narrowed down to outline the domain of our study, even though still broad enough to allow emerging new theories and to adjust the scope of the research [127]. Then, as shown in Figure 4.8, the design and execution process was conducted by means of two sub-processes: the *interview design process* and the *initial design of questionnaire*.

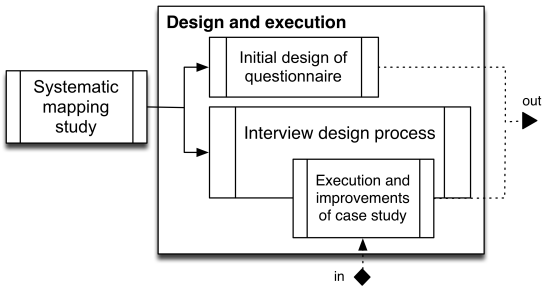


Figure 4.8: Design and execution process (extracted from Figure 4.7)

Interview design process

To design our interviews we defined an iterative approach with different phases illustrated step-by-step in Figure 4.9.

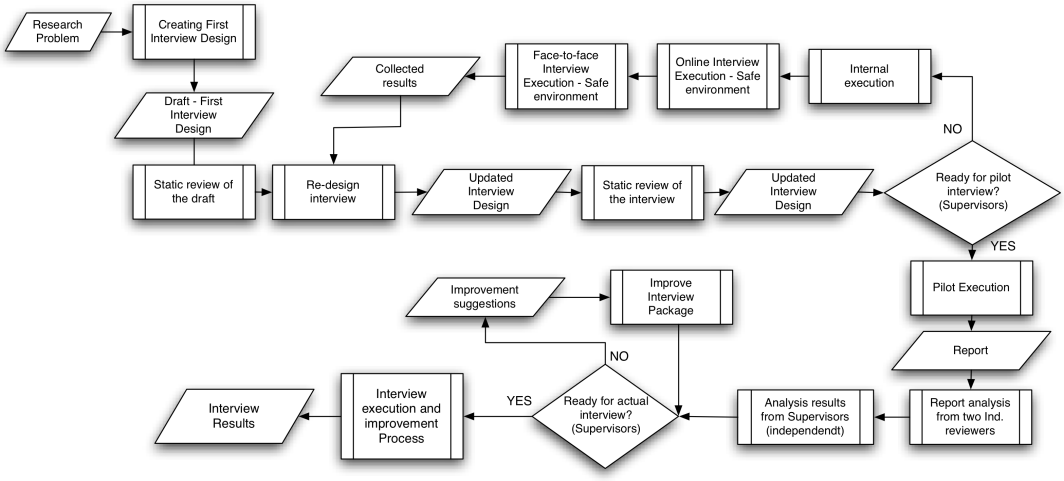


Figure 4.9: Interview package design process

The process depicted above adopts the conventions defined in Appendix A.1 and is thoroughly discussed in this subsection.

The interview design process is centered around the creation of a fully documented *interview package* containing all the material necessary to execute the interviews. The *interview package*, thoroughly described in Appendix A.4.1, underwent several radical changes as its first draft has been subjected to several static reviews, internal executions and both remote and face-to-face mock interviews with colleagues and friends' companies. This allowed us to incrementally refine the quality of the interview, to be able to improve our skills and to learn how to react in different situations before executing field-interviews in the real setting.

After the approval of our supervisors, we tested the *interview package* in a real-world pilot interview, extracting the results into a report. We fine-tuned the design following the improvement suggestions independently provided by our supervisors. Then, we finally started performing the actual interviews. Following a GT approach, we coherently updated the *interview Package* adjusting it to the emerging insights from interviews at each step.

As we were iterating through the interviews, we analyzed new data accordingly, by updating *codes* and *categories* on necessity, and taking notes in the form of memos to adjust the new *emerging theory*. To improve the speed of the coding process (see Subsection 4.3.4), we analyzed all the transcripts one part at-the-time: transcripts followed the semi-structured interview format, and this approach enhanced the chances of finding similar codes across different transcripts, allowing us to quickly iterate and update categories.

To optimize this process we concentrated interviews' execution in a three-weeks time frame, where we worked intensively for transcribing, coding and analyzing the extrapolated reports, while producing the customized follow-up questionnaires (see Subsection 4.3.2). Working with such a fast-paced approach let us be effective and efficient in updating the design asking the right questions, updating the *emerging theory* accordingly and limiting possible interferences of longer time gaps between interviews.

Company sampling – The sampling strategy took place in two distinct phases. At first we executed an initial *convenience sampling* [130], which led to the identification of eight companies. Then we included five additional startups during the theory formation process (*theoretical sampling*) iteratively improving the sample according to the emerging theory. The initial sampling process is shown in detail in Figure 4.10.

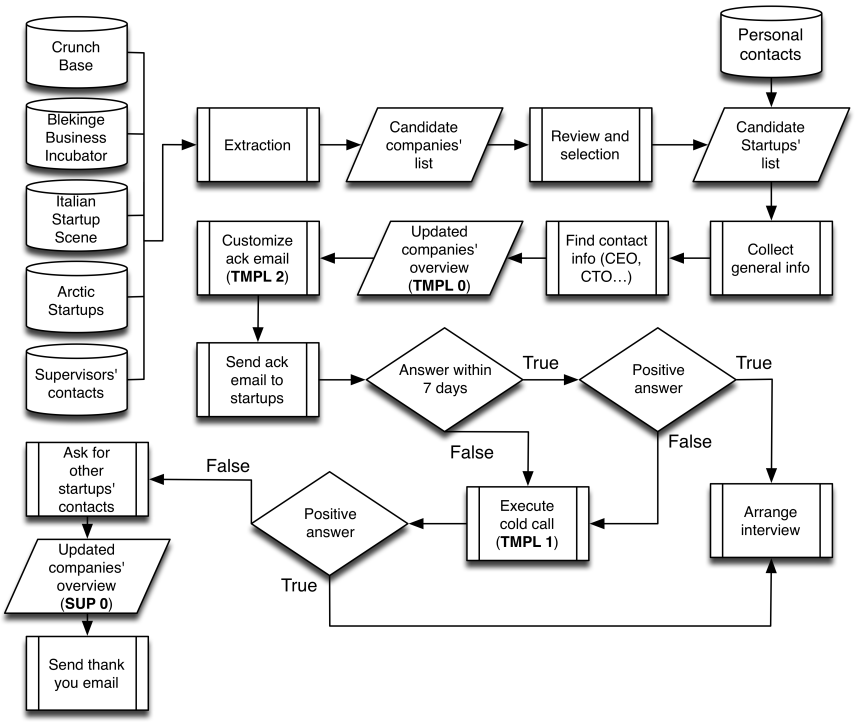


Figure 4.10: Initial company sampling

The blocks in the figure containing a bold identifier in brackets (e.g. *TMPL 0* refer to a specific artifact included in the *interview package* detailed in Appendix A.4.1. The identifiers are used in this chapter in order to refer to specific artifacts. At first we started looking for startups from different sources: public databases, local incubators, online communities and our supervisors' contacts. We created a first list of companies in different market sectors and we refined that list excluding companies which clearly didn't comply with our criteria (established companies, hardware companies, ...) and including only those companies that were clearly satisfying most of the features of software startups as identified with the SMS[14] (see Subsection 4.2.5). In addition, since we wanted to base our case study in early-stage startups, from idea conception to first open beta release, we tried to sample only those companies that were founded in the last two years.

As shown in Figure 4.10, after enriching the list of candidates companies with our personal contacts, we collected some general information (product, size, location, description, founding year) and, where possible, the contact information of their founders CTO/engineers. Then we packaged the data into a spreadsheet (*TMPL.0 Companies overview*) which was used for keeping track of interviews' status throughout the whole process.

We prepared a template email (*TMPL.2 Ack email*) to introduce our research

[14]In particular we prefered companies with: small founding team; one product recently launched; web innovative applications; and working in highly scalable markets.

project and to show to the company our interest in conducting an interview with them about their software development strategy. After customizing the template for each company, making it sound more personal, we sent out our first batch of emails and waited for an answer. In order to improve the response rate we offered to respondents the possibility to choose from a list of rewards [131]. We sent a total of 16 emails receiving 9 responses (6 positive) within one week. To improve the response rate of 37.5% we prepared a script for executing a cold call (*TMPL.1 Cold Call*) to those startups which didn't answer our message, asking if they were not interested in the interview or they just didn't receive the email. After executing the phone calls we were able to schedule interviews with 2 more companies, reaching a total number of 8 startups. Hence, we achieved an initial response rate of 50%, which is satisfactory given the notorious lack of time of startups employees.

As recommended by the GT approach [103], we subsequently integrated the initial sample with five more companies as we were getting insights through the first batch of interviews. The candidate companies have been selected according to the emerging grounded theory, to be able to gather more evidence as new theoretical categories were formed (hence, *theoretical sampling* [127]).

Our final sample was composed by 13 CTOs and founders of software startups, mostly web and mobile, in different stages of their lifecycle[15]. The startups, which participated in our empirical studies, are presented in the following list[16] (randomly sorted): *Mashape* [132], *Blomming* [133], *Searcheeze* [134], *Proliker* [135], *WeddingSnaps* [136], *Podium* [137] , *Mangatar* [138] , *Circleme* [139], *TimeDoctor* [140], *TheBetaFamily* [141], *Next* [142] and *Amen* [143].

Although these startups are all young companies, some of their founders and mentors have an extended experience in the software industry, gained from being directly involved in companies such as *Google*, *Microsoft*, *Amazon* and *Twitter*.

Case study execution – We executed the case study online, supported by tools for video conferencing recording each complete session. A step-by-step workflow have been followed (detailed in Figure 4.11), which allowed us to perform the interviews, collect additional material, prepare the customized follow-up questionnaire and iteratively adjust the *interview package* artifacts.

The interviews took place with a key member of the development team (often the CTO or CEO himself) immediately followed by a preliminary analysis of the collected data aiming to find inconsistencies, catch the most relevant parts of the speech, and identify engineering elements mentioned during the interview. Where possible we collected artifacts (*TAN 0*) to assess and verify the reliability of the respondents statements, by triangulating the sources from which we obtained the information, following suggestions of Creswell and Miller [144].

[15] See Section 5.2 for detailed information about the selected companies.
[16] One company requested to remain anonymous.

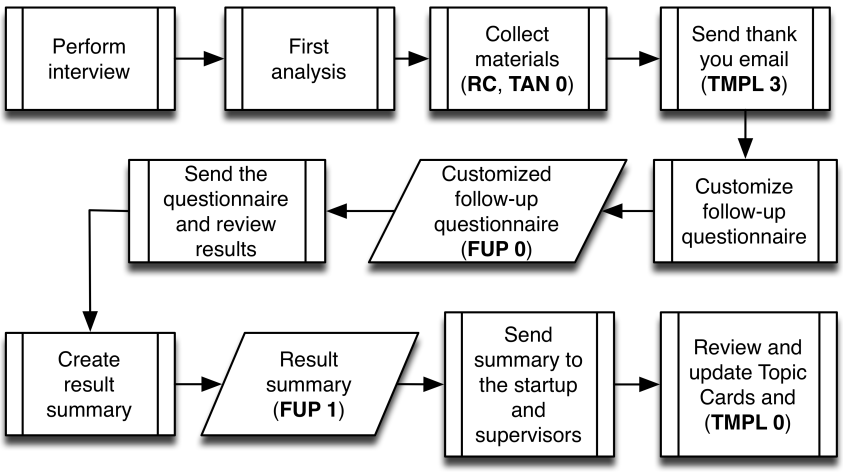


Figure 4.11: Interview execution process

After the interview we sent a first email to show our gratitude to the interviewee (*TMPL 3*) and another one to kindly ask them to answer a short online follow-up questionnaire (*FUP 0*, Subsection 4.3.2) tailored to their interviews. We eventually concluded our process transcribing the interviews and packing the overall results in a summary (*FUP 1*) to be approved by the respondent and peer-reviewed by our supervisors. We executed a full iteration of this process after each interview adjusting the questions of the interview package by the support of information gathered from previous results and comments.

We packaged all documents, files and tools necessary to perform our interviews in an organized collection of artifacts. The *interview package*, which has been incrementally improved and structured, is the output of all the activities involved in the *interview design process* (see Figure 4.9). The package contains a set of files (38) clustered in 9 directories, described in Table 4.3.2.

ID	File Name	Description
TMPL	Templates	Models of documents to be adapted to different companies.
SUP	Support Material	All the materials that supports researchers before and after the interview.
TC	Topic Cards	The actual scripts of the interview, grouped under single *topic cards* (questions, prompts, definitions, verbatims, ...), to be sequentially used during the interview.
CLIST	Checklists	Useful lists of engineering concepts that might be used to help researchers during the interview (ISO standards, methodologies, tools, frameworks, ...) .
HLIST	Hand Lists	Lists that are shown to the respondent to assist in answering specific questions.
TOOL	Tools	Software tools to aid researchers during face-to-face and remote interviews.
FUP	Follow-up	Useful resources for the follow-up questionnaire.
RC	Recordings	All the recordings acquired during the interview (audio, video, notes, ...).
TAN	Data Triangulation	Resources provided by the company to support empirical data. (documentation, models, artifacts, ...).

Table 4.5: Interview package - Structure overview

The detailed content of each category is explained in Appendix A.4.1.

Clustering files and packaging them together helped us in obtaining some benefits:

- Ease of search, organize, update and maintain the file system.
- Separation of concerns of each document, improving the readability during the interview, and letting multiple researchers working on different aspects of the interview at the same time.
- Provide to researchers a tool that helps them conducting a complete and detailed interview covering different important aspects of software development.
- The possibility to refer to the unique identifier assigned to each document throughout this thesis document.
- Provide to other researchers a way to reproduce our interviews, validating our results through experimental replication, and improving the design of the interview[17].
- Ensure coherence between interviews, generating comparable results.

We documented how resources contained in the package have been used during the interviews, displaying them in a chart with *swimming lines* for the different actors involved (Figure 4.12). It is contained in the package file *SUP 4*. In the same document we provide two checklists, pre and post-interview, to help researchers assessing if they are following the procedure described.

[17]The *interview package* has been released under MIT License [145] and it is available for download on Github [146]. We encourage anyone who has interests in pushing this work forward, to fork the repository and contribute to it. If used in a research context, please inform us in order to be able to track where and how the *interview package* has been used.

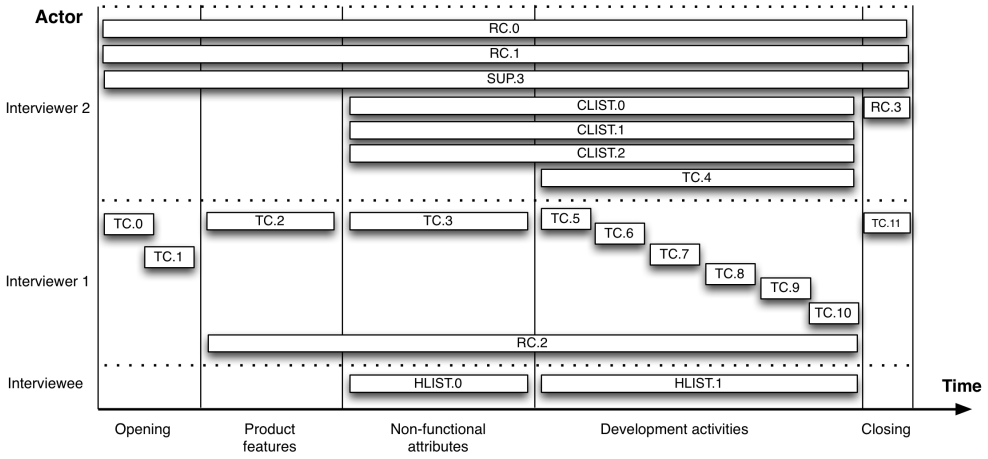


Figure 4.12: Interview package usage

The chart shows identifiers of each resource (identified by a unique label belonging to the interview package) for reproducing the interview process both in face-to-face and remote settings. The time axes has been divided in five distinct phases with different concerns (detailed in Table 4.5), while the vertical axes separates the competences of each actor involved. *Interviewer 1* is in charge of asking questions (detailed in Appendix A.13) while *Interviewer 2* has supervisory responsibilities such as:

- Monitoring and updating the status and execution time of each step.
- Writing notes as the interview proceeds.
- Monitoring the audio recordings of the conversation.
- Taking snapshots of the whiteboard.
- Monitoring the concepts being covered, in case of something important is skipped.
- Adjust interventions of *Interviewer 1*.

The following table presents the five phases in which the interview is divided, both temporally and thematically.

ID	Phase name	Description	Duration est. [mins]
P1	Opening	Introduction to interview, disclaimer, and opening questions.	5
P2	Product features	Elicitation of the main features and characteristics of the product that has being developed by the startup.	10
P3	Non-functional attributes	Elicitation of main non-functional aspects that have been particularly taken into consideration during software development.	20
P4	Engineering activities	Discovery and elicitation of the software development strategy undertaken by the startup, from the idea conception to final deployment of the first public release.	55
P5	Closing	Closing questions, cool-off and appreciations.	5

Table 4.6: Temporal division of interviews

The last column presents an estimation of the duration of each phase, based on the interview we executed. Note that the discovery of engineering activities (P4), requires an extended time[18] (almost an hour).

During the process, it is important that *Interviewer 1* actually drives the interview through different aspects covering all relevant topics, but at the same time he should be flexible letting the respondent moving across different thematic areas with a reasonable degree of freedom. Observe that during the phases *P3* and *P4* the interviewee can support her answers skimming through lists of well-known quality attributes (*HLIST.1*) and SE artifacts (*HLIST.2*).

Questionnaire initial design

The follow-up questionnaires are designed to capture additional data, gather missing information and confirm interview results with triangulation. An initial questionnaire template has been structured in four parts illustrated in this subsection through an example. Starting from this template, questionnaires have been tailored to each startup, partially taking advantage of the repertory grid principles [147]. To obtain better chances of quickly obtaining responses, surveys have been sent to respondents immediately after transcribing the interview results. Two weeks after the conclusion of the last interview, we closed our follow-up questionnaires and collected the data.

Quality achievements – We used a rating scale [103] to quantify the degree to which companies achieved the non-functional attributes they mentioned as

[18]We executed the interview process several times, and from the experience we gained, we found optimal to have a five minutes break between *P3* and *P4* to quickly review partial results, adjust the direction towards the next phases and release the tension. The more interviews we did, the shorter the execution time was, since we gained more experience in eliciting the information we really needed, leaving out marginal information (the first interview lasted around two hours while the last interviews lasted less than one hour).

important for their product during interviews. Figure 4.13 shows an example of how this question has been inserted in a matrix-like fashion.

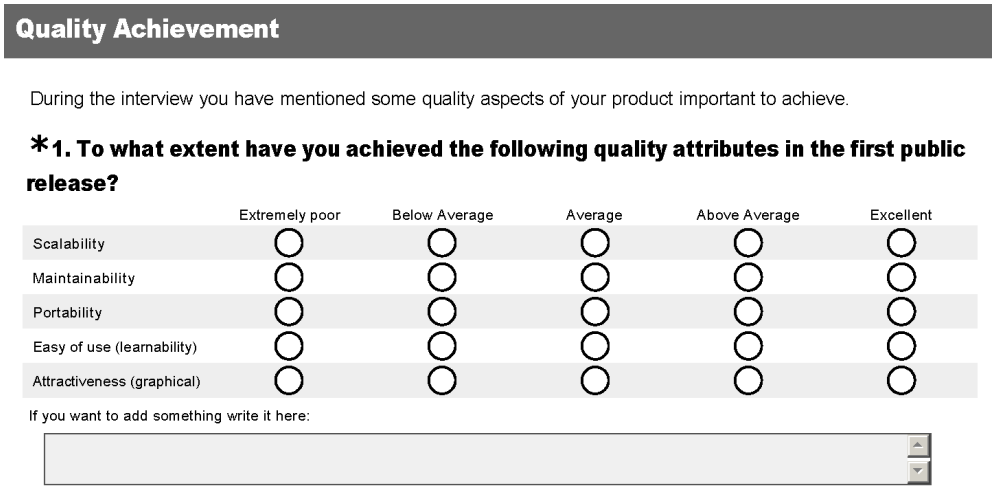


Figure 4.13: Questionnaire template - Quality achievement

With this approach we aimed to understand if the quality strategies undertaken by startups were perceived as effective and eventually identify weaknesses and patterns[19].

Effort distribution – In the second part of the questionnaire, as shown in Figure 4.14, we asked companies to evaluate the effort they have put in each development *phase*, according to what has been discussed during the interview about the release of the first product[20].

[19]Note that the identified quality attributes were initially considered according to the definition of the ISO/IEC 9126. Then, they have been adjusted according to the practitioners' responses.

[20]To limit respondents' bias which could arise from the misconception of the word *effort*, we provided a common definition.

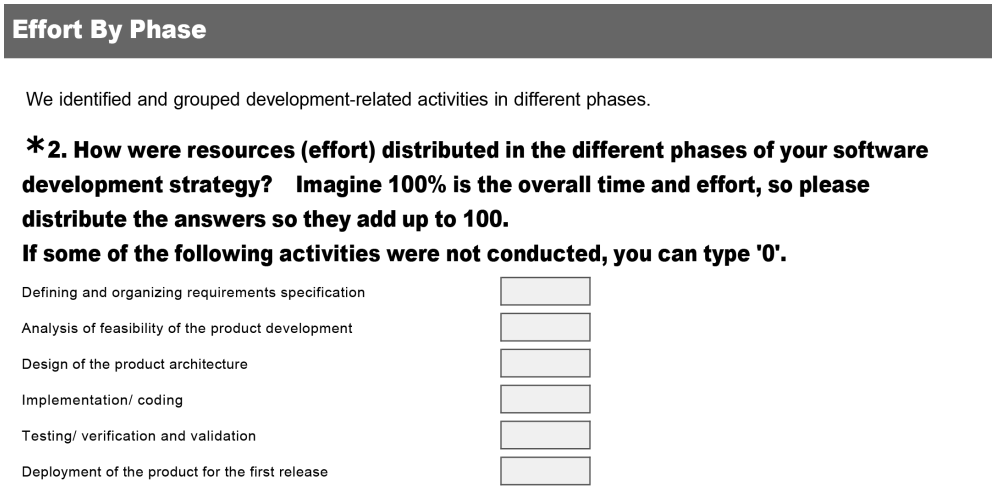


Figure 4.14: Questionnaire template - Effort distribution

Engineering elements – Moreover, in order to understand how engineering elements helped startups in their development process we partially adopted the *repertory grid* principles[21] introduced in 1963 by Kelly [148].

The repertory grid represents a mechanism to aid the elicitation and evaluation of individual's experiences. It relies on the personal construct theory developed by Kelly in the context of his work as a clinical psychologist. The repertory grid technique provides a structured format to help understanding how people construct and evaluate objects. These objects are referred as *elements* (e.g. each row of Figure 4.15 represents an element) and *constructs*, which are the ideas the respondent holds about those *elements*. According to Kelly, individuals hold different constructs based on their experience.

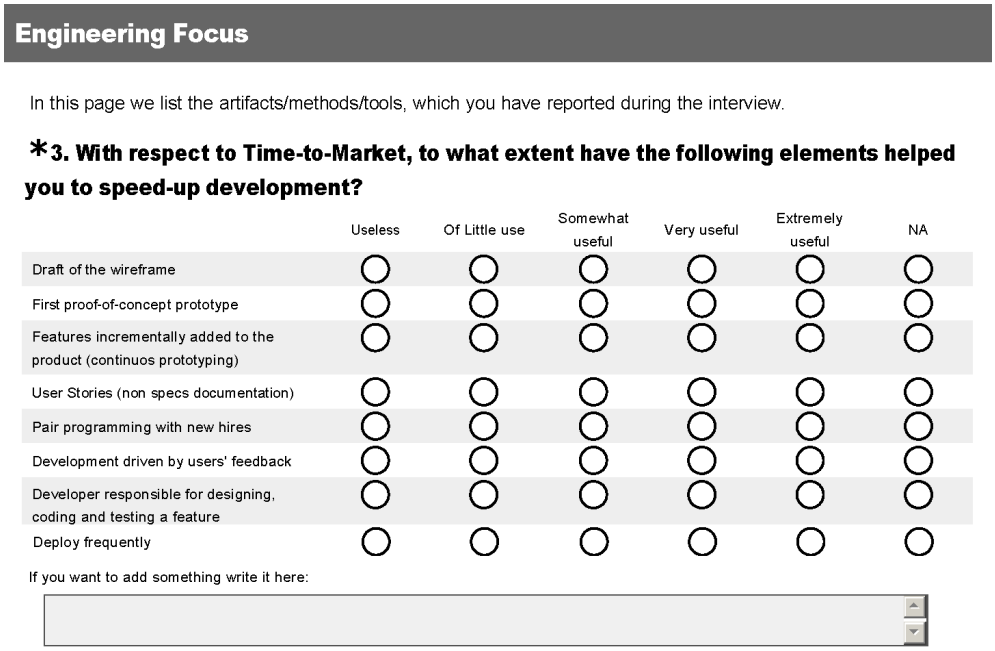
Since we wanted to emphasize respondents' own opinion, without losing rigor, we tailored the repertory grid to our research specific needs. In our case, we built the *elements* using the engineering elements mentioned by practitioners during interviews, and used one fixed construct representing the prominent software development concern - shortening time-to-market. *Elements* have been further adjusted to depict a consistent gri. In fact, when a change in *constructs* appeared during an interview, we explored what the modification meant and adapted the most appropriate type of grid for the identified changes.

A full repertory grid approach should report more than one construct and interactively adjust them with the interviewees. Nevertheless, in this specific case we were only interested in one specific construct. According to concepts emerged with the SMS and the first interviews, the repertory grid focused around

[21]For a practical guideline to use this methodology in SE empirical researches the reader is referred to a recent article of Edwards et al. [147].

time to market as the most prominent construct[22].

As shown in the example extracted from a questionnaire instance (see Figure 4.16), respondents were asked to evaluate each engineering elements within a five-steps rating scale.



Engineering Focus

In this page we list the artifacts/methods/tools, which you have reported during the interview.

***3. With respect to Time-to-Market, to what extent have the following elements helped you to speed-up development?**

	Useless	Of Little use	Somewhat useful	Very useful	Extremely useful	NA
Draft of the wireframe	○	○	○	○	○	○
First proof-of-concept prototype	○	○	○	○	○	○
Features incrementally added to the product (continuos prototyping)	○	○	○	○	○	○
User Stories (non specs documentation)	○	○	○	○	○	○
Pair programming with new hires	○	○	○	○	○	○
Development driven by users' feedback	○	○	○	○	○	○
Developer responsible for designing, coding and testing a feature	○	○	○	○	○	○
Deploy frequently	○	○	○	○	○	○

If you want to add something write it here:

Figure 4.15: Questionnaire template - Engineering lments

Engineering elements have been subsequently mapped to GT categories (see Appendix A.4.5).

Closing questions – In the last part of the questionnaire respondents expressed their degree of satisfaction with the overall software development approach undertaken for releasing the first version of the product. This measure can be used as a proxy for the fulfillment of their way of working. In addition, some open ended questions helped us in clarifying doubts and obtaining perfect hindsight suggestions. An example of the last part of the questionnaire is shown in Figure 4.16.

[22]This choice was taken also according to the time that startups' practitioners were able to grant to the study

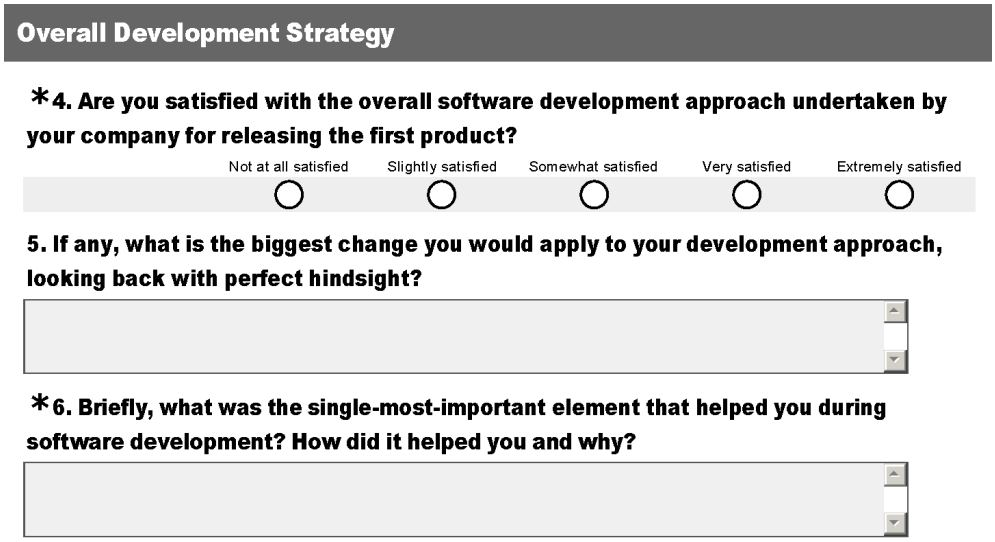


Figure 4.16: Questionnaire template - Closing questions

4.3.3 Case study - Data collection

The approach undertaken to collect the data, known as *data triangulation*, integrates multiple data sources converging on the same phenomenon. As shown in Figure 4.17, *questionnaire results* and *interview transcript* of a company were stored in a single *case study database*.

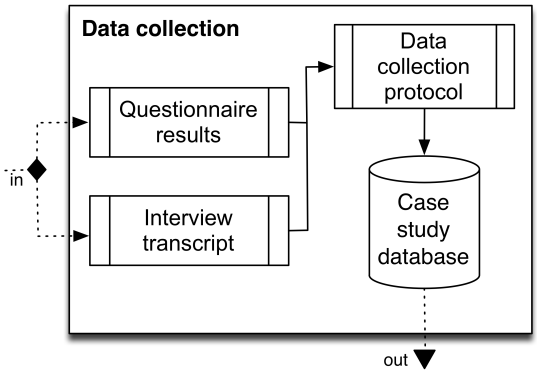


Figure 4.17: Data collection process (extracted from Figure 4.7)

After transcribing the interview, we extracted conceptual relations, and then we integrated them with the survey's results. A well structured case-study database allowed us to easily retrieve and seek for information, assembling the evidence from the case study reports, as described also in [149]. The database has been stored and constructed using the qualitative data analysis software package

AtlasTI[23] - one of the most suitable tools for GT [2].

Entering the field, we overlapped interviews with questionnaire results to adjust the data collection and take advantage of emergent themes and reveal possible inconsistencies. The data was analyzed simultaneously and with flexibility in mind in such a way that adjustments were made according to the emerging findings.

4.3.4 Case study - Data analysis

The raw data were subsequently processed in four steps (see Figure 4.18), namely *data ordering*, *coding*, *theoretical framework* and *ranking of categories*.

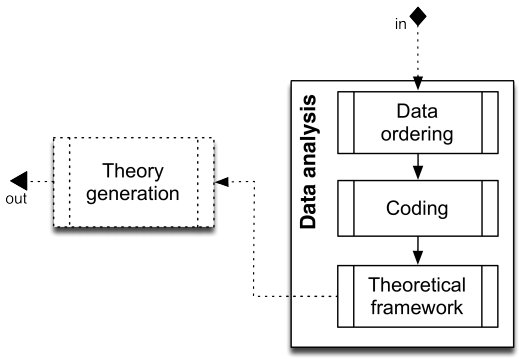


Figure 4.18: Data analysis process (extracted from Figure 4.7)

A *data ordering* procedure was necessary since interviews were spread across a multitude of topics. Therefore transcripts have been structured in thematic areas accordingly to different *topic cards* used during the interviews. We proceeded horizontally between same thematic areas of transcripts to be able to identify a better number of similar concepts, rather than going through an entire transcript at the time.

Once the data were ordered, we executed the process of *coding* interviews, following the steps listed below:

- Labels were assigned to raw data, and a first low-level conceptualization was carried out using both *in-vivo* and *open coding* [103].
- Concepts were grouped together into theoretical categories and subcategories. By means of *axial coding* we first described the different relations between subcategories, and then relations between subcategories and categories.

[23] Available online at `http://www.atlasti.com/` .

- Categories' groups were refined several times in the attempt to create different level of abstractions and adjusting concepts, aided by a simple knowledge management tool[24].
- Consistency among categories were validated by exploring and analyzing links among subcategories by means of the *selective coding*.
- The *core category* - the one with the greatest explanatory power - was identified by analyzing the causal relations between high-level categories.

During data extraction we used the technique of *in-vivo coding* since *"direct quoting from the transcript give more expressive power to the data"* [103] combined with the more descriptive procedure of *open coding* (see Appendix A.4.3). Following the example of other grounded theories, developed in related areas such as Information Systems [150] and Software Process Improvement [151], we performed the high-level conceptualization during creation of categories, in the process of refining axial and selective coding.

Given the large number of codes[25], categories, properties, propositions and related questions that evolved from the analytical process [127], an important activity that helped the coding process was *memoing*. It constituted an important component involved in the formulation and revision of theory during the research process. For this purpose we made use of three different types of memos: code memos, theoretical memos and operational memos. The first type of memo is related to the conceptual labeling of open codes, whilst the second type concerns axial and selective coding, and thus focusing on paradigm features. Finally, operational memos contain directions relating to the evolving *emerging theory*.

After the coding process the first representation of the *experience map* identified in GT is constructed by means of a *theoretical framework*. The theoretical framework is presented in the form of a network of categories and subcategories identified during the coding process. Precisely, categories and subcategories are linked together according to cause-effect relation [127]. During the formation of the theoretical framework the researchers operated by means of a bottom-up approach. From empirical data and coding process, the framework was developed into two different levels: a detailed level representing the network of subcategories (identified mainly by *axial coding* process), and high-level representing the network of main categories (identified mainly by *selective coding* process). When describing the framework (presented in Section 6.2), we made explicit statement of empirical data by reporting most important citations of practitioners, enabling the reader to evaluate categories critically.

The theoretical framework, detailed in Section 6.3, is the structure that can hold or support a theory of a research study, introducing and describing the

[24] Available online at `https://workflowy.com/`.

[25] 630 unique code fully reported in Appendix A.4.3.

experiences gathered from the interviewed startups' practitioners[26]. Presenting the *core category* and its relations, the theoretical framework is able to explain the meaning, nature and challenges of a phenomenon, often experienced but unexplained in the world in which we live, so that we may use the constructed knowledge and understanding to act in more informed and effective ways.

4.3.5 Case study - Theory generation

As mentioned above, emergent theories have been tested integrating the sample with additional companies selected following the principle of *theoretical sampling* [149] (described in Subsection 4.3.2). To verify that we were close to the theory saturation - the point at which executing more interviews wouldn't bring any additional value for constructing the theory - after coding the twelfth interview (the last of the second sampling iteration) we executed an extra interview to assess that no other relevant categories would have emerged. In fact from its transcript we coded only 16 new concepts in the whole transcript, without finding any other relevant category.

The process of *theory generation* took place at each iteration together with interview execution, where we systematically analyzed the *theoretical framework* by means of the *paradigm model* introduced by Corbin [127].

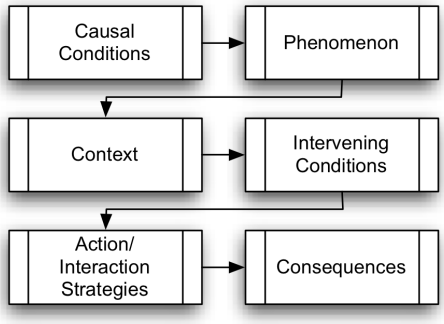


Figure 4.19: Paradigm model

As shown in Figure 4.19 the paradigm model is composed by:

- *Causal conditions*, the events which lead to the occurrence of the *phenomenon*, that is our core category.
- *Context*, set of conditions in which the phenomenon can be extrapolated.
- *Intervening conditions*, the broader set of condition that the phenomenon can be generalized.

[26]Following the example of other studies, categories composed by subcategories grounded in less than 9 interviews, have not been considered in the formation of the final theory.

- *Action/interaction strategies*, the actions and responses that occur as the result of the phenomenon;.
- *Consequences*, specification of the outcomes, both intended and unintended of the actions and interaction strategies.

The role of the generated theory is to explain, predict and understand phenomena, and, in many cases, to challenge and extend existing knowledge, within the limits of the critical bounding assumptions. The final theory is presented in Section 6.4.

4.3.6 Case study - Theory validation

The *theory validation* is conducted in four steps (see Figure 4.20):

- The theory has been evaluated using a systematic procedure presented by [128] (*static assessment*).
- The theoretical framework has been compared with similar frameworks and with results of studies identified with the systematic mapping of the literature (*comparison with the state-of-the-art*).
- The high-level relations between framework's categories have been tested using the empirical data collected during the case study (*evaluation with empirical data*).
- Categories have been ranked by assignment of scores through the use *engineering elements* identified by the follow-up questionnaire.

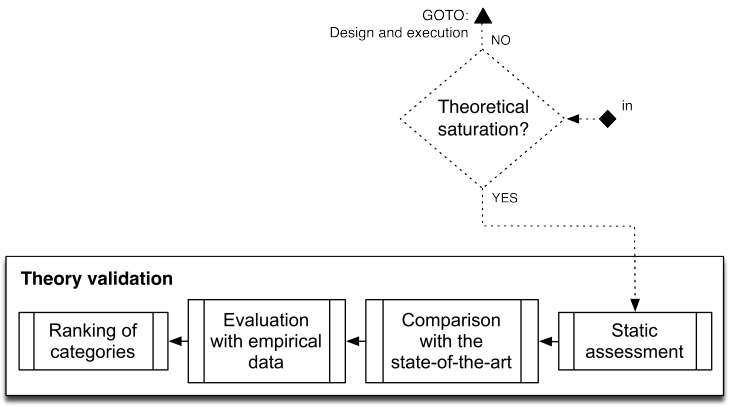


Figure 4.20: Theory validation process (extracted from Figure 4.7)

The *static assessment* is presented in the following part of this subsection while the *comparison with the state-of-the-art*, *evaluation with empirical data* and *ranking of categories* are discussed in Chapter 6 (see Section 6.6), entirely dedicated to the presentation of the theoretical model.

Static assessment

Presenting GT is challenging for a researcher, who must pay attention to structure the level of details included, and to the way data is portrayed to display evidence of the emergent categories. In order to assess our research and determine if the GT is sufficiently grounded, we used a systematic technique to assess the theory, following evaluation lists and factors presented in [128].

First we had to consider the following factors:

- *Fit*, that describes the consistence of the theory with the real data.
- *Understanding*, that describes the degree to which participants agree with the theory.
- *Generality*, that describes the level of abstraction of the theory, and how sufficiently it can be adopted by practitioners without losing its relevance.
- *Control*, that describes how well the theory enables practitioners to fully understand the described situation.

In order to achieve a high-level of fit between the theory and the raw empirical data, we started GT without any assumption on how the emergent patterns would influence the research direction. Ideas and perceptions evolved during the mapping study and interviews. In addition, constant comparative methods, overturning of some early categories as new data came to light, generation and testing of interim hypotheses, and constant re-evaluation of the interview transcripts, ensured that researchers' bias was minimized and theoretical fit maintained.

To represent understandable emergent theories, concepts and categories were carefully developed using appropriate and self-explanatory codes. The developing theory was not immediately presented to the participants to prevent potential bias in the responses. But rather, the theoretical framework has been presented and described to startups practitioners not involved in the data collection, whose reaction was positively expressed.

From generality perspective, we developed a theory from abstract categories, which allowed us to create a general guide to constantly changing situations. Nevertheless, we took attention on maintaining a reasonable level of abstraction, keeping focus on their sensitizing characteristics and providing low-level subgroup of categories.

Finally, we provided a comprehensive set of categories with detailed interrelations to explain how the *core-category* is achieved and the reasons of its consequences, in order to enable practitioners to independently understand and analyze situations, predict changes and their consequences, and to be capable of revising actions or the theory itself. Using both methodological tools and features provided by *AtlasTI*, we explored and tested:

- Each category and the strength of relations between them.
- Hypotheses, derived from and related to the emergent theory.

- Deviant cases to ensure robustness and general applicability.

Additionally, Strauss and Corbin provided a list of questions to assist in determining how well the findings are grounded [128]. They questions are as follows:

1. Are concepts generated, and are the concepts systematically related?
2. Are there many conceptual linkages and are the categories well developed?
3. Is variation[27] built into the theory and are the conditions under which variation can be found built into the study and explained?
4. Are the conditions under which variation can be found built into the study and explained?
5. Has the process been taken into account, and does the theory stand the test of time?
6. Do the theoretical findings seem significant, and to what extent?

Answering the above questions, the concepts were generated according to the coding process described in Subsection 4.3.4. They were systematically related through the use of a network diagram (see figures 6.1 and 6.2), by which we established the linkages and relations between the concepts. At each iteration of the grounded theory process, we considered and examined a concept within different conditions and dimensions, trying to incorporate opinions of a broader as possible range of practitioners.

All the linkages and categories were constructed by the use of Atlas.TI and compared according to the data analysis process defined in Subsection 4.3.4. Variations of the generated theory were applied according to raw grounded data with the aid of codes and memos. Moreover during this study extensive explanation of the concepts were presented in-vivo statements as reported by practitioners.

The process has been designed by the researches in different steps, explaining and making clear the purpose of each of them. Thus, the same processes enable other research to conduct the same study within similar contexts. Moreover *comparison with the state-of-the-art* has been applied in order to validate the theory and enhance odds of its applicability within a wider time-frame (see Subsection 6.6).

Then, we performed an *evaluation with empirical data*, quantifying scores for each company involved in our empirical study by defining metrics based on a set of rubrics and evaluating startups accordingly, through an analysis of interview transcripts and codes. The complete statistical procedure and provided rubrics to quantify the measures are illustrated and discussed in Appendix A.5.

A further validation has been applied by the *ranking of categories* process. We assigned scores to theoretical *categories* - identified with GT interviews (see Subsection 6.6.5) - that most helped respondents to speed-up the development process.

[27] Variation refers to the variety of contexts which theory can be applied.

Then, the *ranking of categories* process has allowed to compute metrics of fitting between the *theoretical categories* and the *engineering elements*, contributing to the validation of the suitability of the model. Results of this process are presented in Section 6.6.5.

4.3.7 Case study - Framework modelling

The last step described in the research methodology (see Figure 4.1) is the *framework modelling* process. It consists of an in-depth comparison with well-known frameworks to analyze similarities and differences with the case study results. The three main steps of the *modelling* process have been:

1. The identification of the operating-context of startups in comparison with existing methodologies applied in the software industry (see Section 7.2).
2. The analysis of a decision-making framework to validate implications of the generated theory (see Section 7.3).
3. The identification of an evolutionary model, fitting the results derived through the case study (see Section 7.4) to help in defining objectives that startups should achieve to support a future growth (see Section 7.5).

Therefore, through the above described *modelling* process, we analyze the applicability of the theoretical framework in view of a future growth[28], while discussing the startup's distinctive operational dynamics (see Chapter 7).

4.3.8 Case study - Validity threats

Since the case study is composed by two distinct research methods (interviews and questionnaires) this section will consider the threats separately.

Interviews related threats

First of all, in our initial theoretical sampling there is a *selection bias*, i.e. we didn't contact startups that are *out of business* (failed), but only those startups which are still operating at the moment. Interviewing ex-founders of failed startups would improve the completeness of results, because they might have precious insights about the mistakes they made. However, startups rarely fail for merely technical reasons, but mostly for problems related in finding the right product/-market fit [10, 152, 37]. Despite certain kind of development approaches can help in realizing the right product/market fit early, respondents of failed startups, who

[28]Note that by definition startups are those temporary organizations focused on the creation of high-tech and innovative products, with little or no operating history, aiming to grow by aggressively scaling their business in highly scalable markets.

are currently not working on a new venture, would drive interviews in a direction which is not of much relevant from a SE perspective, talking mostly about a business perspective[29]. We strongly believe that a good strategy, to mitigate the above mentioned selection bias, is including in the initial sample founders who are currently working in a healthy startup and have experienced failure in the past (most of the founders we interviewed failed multiple times before succeeding). We realized that these kind of *serial entrepreneurs* have a natural tendency to give valuable advices regarding mistakes they have done before, contributing in filling the lack of failed startups data in our sample.

Another threat lies in the choice of using long descriptive codes. Social scientists discourage the use of descriptive codes to characterize a social phenomenon in favour of more conceptual and general codes [153]. Yet, our research falls into a radically different domain - that is - a first attempt of empirically understand the software development approach in early-stage startups. In this context, descriptive details are very important to let other researchers comprehend the underlying phenomenon that is mostly unexplored to the date[30], unlike most social-science-related concepts.

Additionally, to limit the introduction of the authors' *personal bias*, we used the same approach we adopted for screening articles in SMS (see Subsection 4.2.3) performing the coding of transcripts in pair, creating mutual consensus using memos and discussions.

The grounded theory study, as qualitative research, is based on respondent's opinions and it is filtered by their personal judgment. As a consequence, the findings and the theory we generated stick to the data gathered in the field. Through a deep description of phenomenon extrapolated and abstracted from the participants' interviews, data has been encapsulated into *categories* and *subcategories*. Additionally, to support qualitative data, a triangulation of quantitative results has been conducted by means of a follow-up questionnaire. However, the participant's perception of what is taking place may be at odds with reality. Moreover, participants may have reported what they believe the researchers wished to hear (acquiescence bias). This may be particularly true in larger companies, which are reluctant to admit the lack of practices and standard to the public [2], while in startups the risk is moderate.

Furthermore, despite offering the rewards to companies for participating in the case study helped us establishing an initial trust with potential respondents, one could infer that it has affected the *company sampling* process by attracting respondents particularly interested in receiving the prizes regardless of the scientific interest in the research. However, since most of the respondents didn't claim the reward afterwards[31], we can reject this hypothesis with a good degree

[29] We tried to have informal conversations with failed startups in the past.

[30] In terms of grounded theories.

[31] We repeatedly invited all the respondents to redeem the reward via email, and only two respondents out of thirteen actually did so.

of confidence.

Another threat to the validity is the natural inclination of employees with a managerial role (in the case of this research CTOs and CEOs) towards methods and practice, which they propose to the team. Yet, this threat is very low in the specific domain of early-stage startups, since the roles of managers and technicians are blurred and most of the time are not defined at all.

The observer consistency (i.e. measuring the same behaviour at different time intervals [103]) was not an important concern in view of the cross-sectional nature of the study. On the other hand, the inter-observer agreement (i.e the extent to which two or more observers obtain the same result when measuring the same behaviour [103]) has been achieved by working in pair on every single task.

A reasonable heterogeneity of subjects emerged from the sampling process, which contributed to increase the external validity of the findings without compromising the internal validity [129]. The instrumentation used for the interview, consisting of a minimal set of online tools for video conference, had only a little interference with the observation of the phenomenon itself. All the tools have been tested and evaluated against alternatives, with the only scope of reducing the friction introduced by the physical distance between researchers and respondents.

Finally, the validity of any results based on field interviews is undermined, to a certain extent, by the possibility of both intentional and unintentional lies in the answers. To mitigate this risk, every time we doubted about the actual use of an engineering element we asked if they could show to us the item in question, when possible, to further triangulate the results. However, in case of the interviews we performed, these kind of situation were very unlikely to happen since respondents have been extremely clear about the fact that they adopt a very informal way of working, and they don't need to guarantee or proof quality standards required by more traditional process assessments.

Questionnaires related threats

The main risk of adopting a follow-up questionnaire within a startup's context is based on the evidence of the limited time offered by practitioners. In fact, startups work under constant time-pressure, as confirmed by [16]. In view of this constraint, the design decision of a minimal repertory grid[32] was accordingly generated. But, although we may not be able to provide a high statistical reliability for a minimal repertory grid, we can state that the elicited construct and elements are trustworthy since result of an in-depth study based on the GT study [147].

The basic constructs of our repertory grid were firmly grounded during the interviews. Moreover, when respondents had specific believes that important constructs would have been related to the elements, we adapted the follow-up

[32] A repertory grid composed by one construct only.

questionnaire accordingly. Furthermore we always left an extra free-form field in case the respondents wanted to add comments, and a *N/A* option for closed questions.

Moreover, the extent to which results are related to the *real-case* scenario has been attested by triangulating the expressed opinions with analysis of sample of SE artifacts and documents (*TAN 0*) provided by respondents, when possible. The accuracy and adherence of engineering activities have been firmly grounded by interviews and by discussion with experts and supervisors.

In order to provide meaningful constructs, care of wording has been applied by means of reviews in the literature, and when possible, using respondents' own vocabulary. Further adjustments were applied to specific cases when new constructs were considered extremely important from the analyzed interview transcripts. Finally the results suffer, to a certain extent, from the cognitive biases commonly associated with questionnaires [154], especially *response bias*, *acquiescence-bias* and *social desirability bias*.

However, all of the above mentioned threats are mitigated by the fact that the results obtained from questionnaires constitute only a side-element of this research, and they were never used alone to draw conclusions, which are instead based on systematic processes of literature review and interview analysis.

In Subsection 5.2.4 we present other limitations of the online questionnaires, while more general threats to the validity of the research results are finally presented in Section 8.5.

Chapter 5

Results and analysis

This chapter is structured in two parts: the analysis of the *systematic mapping study* results (see Section 5.1) followed by an analysis of the case study results obtained through interviews and questionnaires (see Section 5.2). The analysis of results is further protracted in Chapter 6 in which a *theoretical model*, obtained with a cross-methodological combination, is presented, discussed and validated.

5.1 Systematic mapping study

This section presents the results of the *systematic mapping study*. From an initial sample of 943 articles, we selected 37 relevant studies pertaining engineering activities in startups. From them, we extracted a brief *one-line* sentence which summarizes the content of the article and can be used by the reader to get a rough idea of the study without reading each full article. For the sake of readability, the detailed table is presented in Appendix A.3 (see Table A.3).

The section is structured according to the RQ-1 and its sub-questions:

- Publication distribution (see Subsection 5.1.1).
- Rigor and relevance (see Subsection 5.1.2).
- Contextual features of startups (see Subsection 5.1.3).
- State-of-the-art: summary (see Subsection 5.1.4).

5.1.1 Publications distribution

Figure 5.1 shows the frequency distribution of the publication year, from 1994 to 2011.

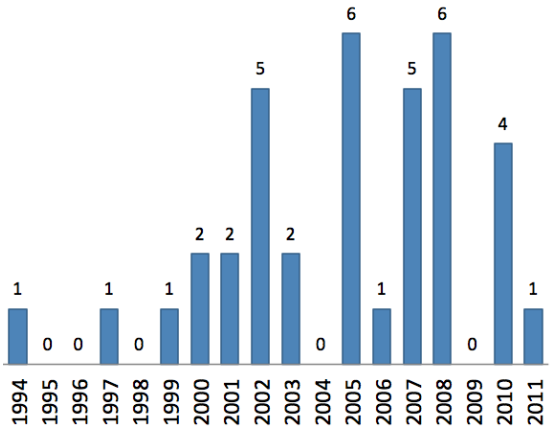


Figure 5.1: Publication distribution-year

More than 80% of the studies in the selected sample have been published in the last ten years, which compared to the long-standing history of the SE discipline and more in general software companies (half of a century), is a quite short time-frame. Only 7 relevant articles dated prior to 2002 discuss software startup related issues[1]. To make a comparison, when looking at one of the closest domain such as software engineering in small companies, we can observe how the specific literature started many years before 2002. For example, in 1994 Brodman was already discussing how to adapt the CMM methodology to small organizations [155]. Only 10 years later we find an empirical study inquiring startups and development methodologies [79].

The yearly distribution of publications attest the novelty of the startup phenomenon, which has been basically enabled and amplified by the potentially huge markets and distribution channels offered by the internet and mobile devices [24, 35]. This context posed a set of new problems and challenges which can hardly be faced using traditional approaches [13].

Publications distribution - Topics

To characterize what are the main topics covered within the area of engineering activities in software startups, we assigned to each article a set of keywords based on the content and the article's own keywords (when available in the metadata), during the process of *keywording* (see Subsection 4.2.4). From the 37 relevant studies we extracted a total number of 327 keywords (117 uniques) averaging about 9 keywords per article. To provide a visual representation of the content of the studies, we show the frequency of occurrence of each keyword in a *tag-cloud* fashion[2] in Figure 5.2. The table containing the raw data, which was used to

[1]Other 4 articles were discarded in the selection process because they didn't comply with the selection criteria and then considered as obsolete (see screening process in Subsection 4.2.3).

[2]The tag cloud was generated with the online service (`http://tagcrowd.com/`).

generate it, is shown in Appendix A.3 (see Table A.3).

Figure 5.2: Keywords cloud overview

The *tagcloud* (see Figure 5.2) was obtained by assigning keywords to each study, reflecting the most important topics covered by the authors. Then, counting the occurrence of each keyword, the reader can grasp an overview of the main concern of the studies in the sample.

Beside the obvious high-frequency of the keyword *software-startup*, a pronounced interest for topics such as *management* (19 times), *software-process* (17), *agile-methodologies* (10) and *web-development* (7) is visible. For software-process, it is worth noticing how studies are concentrated around *process-improvement* (4) and *process-formation* (2). Among the different agile methodologies, a particular consideration is given to *extreme programming* (7), while other methodologies

such as *RUP* (2) and *Scrum* (1) received little attention. This represents a first evidence of our initial assumptions that, in the general meaning of the word, startups are *agile* companies.

Among others, we can observe some topics that are quite peculiar for this area, such as *time-to-market* (6), *rapid-development* (3) and *founder* background (14).

The particular attention given by the literature to those topics, especially aspects related to the *team*, *agile-methodologies*, *time-to-market*, and *web-development*, helped us in refining the specific research domain, contributing to adjust the trajectory of the *case study* conducted with startups' practitioners (see Section 4.3) and driving the formation of a *theoretical model*, which is presented and explained in Section 6.2. However, the wide variety of topics, combined with the very-low frequency in which each topic appears, represents a symptom of a general scarcity of primary studies investigating specific issues.

Furthermore, by reading the articles and analysing their references, we observed that the selected studies are poorly interconnected to each other. They represents low intra-referencing among authors which makes it harder to ground findings to existing works.

Publications distribution - Classification schema

The above identified keywords, together with existing taxonomies, contributed to the formation of a formal *classification schema*. The final schema, which emerged from the process specified in Subsection 4.2.5, consists of four dimensions or *facets*:

- *Research type*: to represent the type of study undertaken.
- *Focus*: to describe the main focus of the research.
- *Contribution type*: to map the different types of outcome of the study.
- *Pertinence*: to distinguish between articles entirely devoted to engineering activities in startups and the ones which are only mentioning something related to it.

Each facet is formed by different categories described in the following four tables - one per dimension - which were used to map the selected studies and obtain a systematic map of the area.

Research type Facet	
Evaluation Research	Methodology is implemented in practice and an evaluation of it is conducted. That means, it is shown how the research is implemented in practice (solution implementation) and what are the consequences of the implementation in terms of benefits and drawbacks (implementation evaluation). This also includes to identify problems in industry.
Solution Proposal	A solution for a problem is proposed, the solution can be either novel or significant extension of an existing methodology. The potential benefits and the applicability of the solution is shown by a small example or a good line of argumentation.
Philosophical Papers	These papers sketch a new way of looking at existing things by structuring the field in form of a taxonomy or conceptual framework.
Opinion Papers	These papers express the personal opinion of somebody whether a certain technique is good or bad, or how things should been done. They do not rely on related work and research methodology.
Experience Papers	Experience papers explain on what and how something has been done in practice. It has to be the personal experience of the author.

Table 5.1: Classification schema - Research type facet

The first dimension, *Research type* facet (see Table 5.1), can be used to distinguish between different types of studies abstracting from the specific underlying research methodology, unlike other types of classification [121]. The research types have been adapted from [156]. The second dimension describes the categories of the *focus facet*, which were obtained by clustering together sets of keywords previously identified (see Table 5.2).

Focus facet	
Type	Description
Software development	Engineering activities used to develop software.
Process management	Engineering activities used to manage the development activities.
Tools and technology	Instruments used to create, debug, maintain and support development activities.
Managerial/organizational	Aspects that are related to software development, by means of: resource management and organizational structure.

Table 5.2: Classification schema - Focus facet

We separated studies concerning software development practices from articles focused on the high-level process management. Furthermore we provide two additional categories to classify articles focused on specific *tools and technologies* and more *managerial/organizational* studies. The third dimension, *Contribution facet*, is presented in Table 5.4.

Contribution facet	
Type	Description
Model	Representation of an observed reality by concepts or related concepts after a conceptualisation process.
Theory	Construct of cause-effect relationships of determined results.
Framework/Methods	Models related to constructing software or managing development processes.
Guidelines	List of advices, synthesis of the obtained research results.
Lesson learned	Set of outcomes, directly analysed from the obtained research results.
Advice/Implications	Discursive and generic recommendation, deemed from personal opinions.
Tool	Technology, program or application used to create, debug, maintain or support development processes.

Table 5.3: Classification schema - Contribution Facet

Similarly to the taxonomy used in [157], this facet expresses which kind of contribution a study brings to the fields. Contribution types can be divided in *weak* (*advices and implications*, *lesson learned*, *tools* and *guidelines*) and *strong* (*theory*, *framework/methods* and *model*). Finally the *pertinence facet* (see Table 5.3) helps in distinguish between studies fully pertaining to engineering activities in software startups (*full* pertinence), studies partially focused on some engineering activities in startups (*partial* pertinence), and finally studies only mentioning some relevant information pertained to software development in startups (*marginal* pertinence).

Pertinence Facet	
Type	Description
Full	Entirely related (main focus) to engineering activities in software startups.
Partial	Partially related to engineering activities in software startups. Main research focus related to engineering activities.
Marginal	Marginally related to engineering activities in software startups. Main research focus different from engineering activities.

Table 5.4: Classification schema - Pertinence Facet

Although the above presented *classification schema* was shaped around our data sample, it can be used by other studies by applying some case-specific adjustments.

Publications distribution - Systematic map

Finally, by fitting the selected studies into the classification schema, we built the *systematic map*, shown in Table 5.5.

Author (year)	Pertinence	Focus	Research Type	Contribution Type	Ref.
Coleman (2008)	Full	Process Management	Evaluation Research	Model	[27]
Coleman (2007)	Full	Process Management	Evaluation Research	Theory	[2]
Coleman (2008)	Full	Process Management	Evaluation Research	Theory	[29]
Kajko (2008)	Full	Process Management	Evaluation Research	Model	[18]

Table 5.5 – *Continued on next page*

Table 5.5 – *Continued from previous page*

Author (year)	Pertinence	Focus	Research Type	Contribution Type	Ref.
Häsel (2010)	Marginal	Managerial & organizational	Evaluation Research	Model	[158]
Hanna (2010)	Marginal	Managerial & organizational	Evaluation Research	Model	[94]
Deakins (2005)	Partial	Managerial & organizational	Experience Paper	Model	[89]
Camel (1994)	Full	Software Development	Evaluation Research	Lesson Learned	[70]
Silva (2005)	Full	Software Development	Evaluation Research	Lesson Learned	[79]
Midler (2008)	Partial	Managerial & organizational	Evaluation Research	Framework & Methods	[86]
Taipale (2010)	Full	Software Development	Experience Paper	Advice & Implications	[84]
Chorev (2006)	Marginal	Managerial & organizational	Evaluation Research	Model	[34]
Zettel (2001)	Full	Software Development	Solution Proposal	Framework & Methods	[88]
Jansen (2008)	Partial	Software Development	Evaluation Research	Lesson Learned	[95]
Sutton (2000)	Full	Process Management	Opinion Paper	Advice & Implications	[13]
Heitlager (2007)	Full	Process Management	Solution Proposal	Tool	[3]
Tingling (2007)	Full	Software Development	Evaluation Research	Advice & Implications	[77]
Deias (2002)	Full	Software Development	Experience Paper	Advice & Implications	[80]
Stanfill (2007)	Marginal	Managerial & organizational	Solution Proposal	Advice & Implications	[159]
Wood (2005)	Partial	Software Development	Experience Paper	Advice & Implications	[160]
Steenhuis (2008)	Marginal	Managerial & organizational	Evaluation Research	Lesson Learned	[161]
Yogendra (2002)	Partial	Managerial & organizational	Evaluation Research	Guidelines	[87]
Ambler (2002)	Full	Software Development	Experience Paper	Lesson Learned	[76]
Crowne (2002)	Full	Software Development	Solution Proposal	Advice & Implications	[10]
Mater (2000)	Partial	Managerial & organizational	Evaluation Research	Model	[91]
Kakati (2003)	Marginal	Managerial & organizational	Evaluation Research	Model	[35]
Kuvinka (2011)	Partial	Software Development	Experience Paper	Advice & Implications	[85]
Su-Chuang (2007)	Marginal	Software Development	Evaluation Research	Advice & Implications	[162]
Sau-ling Lai (2010)	Marginal	Managerial & organizational	Evaluation Research	Lesson Learned	[163]
Mirel (2000)	Partial	Managerial & organizational	Solution Proposal	Advice & Implications	[92]
Himola (2003)	Marginal	Managerial & organizational	Solution Proposal	Advice & Implications	[68]
Kim (2005)	Marginal	Managerial & organizational	Evaluation Research	Model	[93]
Wall (2001)	Partial	Software Development	Experience Paper	Advice & Implications	[96]

Table 5.5 – *Continued on next page*

Table 5.5 – *Continued from previous page*

Author (year)	Pertinence	Focus	Research Type	Contribution Type	Ref.
Yoffie (1999)	Marginal	Managerial & organizational	Evaluation Research	Guidelines	[81]
Bean (2005)	Marginal	Tools and technology	Philosophical Paper	Advice & Implications	[97]
Tanabian (2005)	Marginal	Managerial & organizational	Opinion Paper	Advice & Implications	[33]
Fayad (1997)	Marginal	Process Management	Philosophical Paper	Advice & Implications	[90]

Table 5.5: Systematic map overview

The above table quickly reveals the nature of the study conducted, the extent of pertinence with the research problem, the type of contribution produced and the major research area investigated. Although the table is complete and contains the entire classification schema, it is still hard to extract relevant information from it. For this reason we computed the frequency of publications in each category using an electronic spreadsheet. In this way it is easier to emphasize what has been achieved in the past by researchers in the area, identifying gaps to drive the direction of our *grounded theory* case study, and suggesting possible future researches.

As suggested by Peterson et al. [1], to provide a good overview on how the topic is structured, we present our *systematic map* using multi-dimensional bubble charts (*"x-y scatter plots with bubbles in categories intersections"*) where the size of the bubble is determined by the number of publications corresponding to the *x-y* coordinates. Differently from other similar studies [164, 109], in our *classification schema* each data point is represented by four features. Thus, we created three plots to visualize all the six possible combinations of *facets*, giving a complete overview of the systematic map.

The charts depicting the systematic map are presented in figures 5.3, 5.4 and 5.5. They have been obtained by joining together multiple facets in different quadrants to allow the reader to consider different *dimensions* simultaneously. Finally we added some statistical data on the chart for individual dimensions.

For example, the big bubble in the top-left part of Figure 5.3 indicates that 11 studies (29.73% of the total) are focused on *managerial and organizational factors* and conducted through an *evaluation research*. Simultaneously in the same figure is possible to observe, for instance, how 7 items with *managerial and organizational* focus contributed to the body of knowledge with a *model*. However, by looking at Figure 5.4, one can quickly notice that 5 out of the total 9 *models* have only a *marginal* pertinence with engineering activities in software startups.

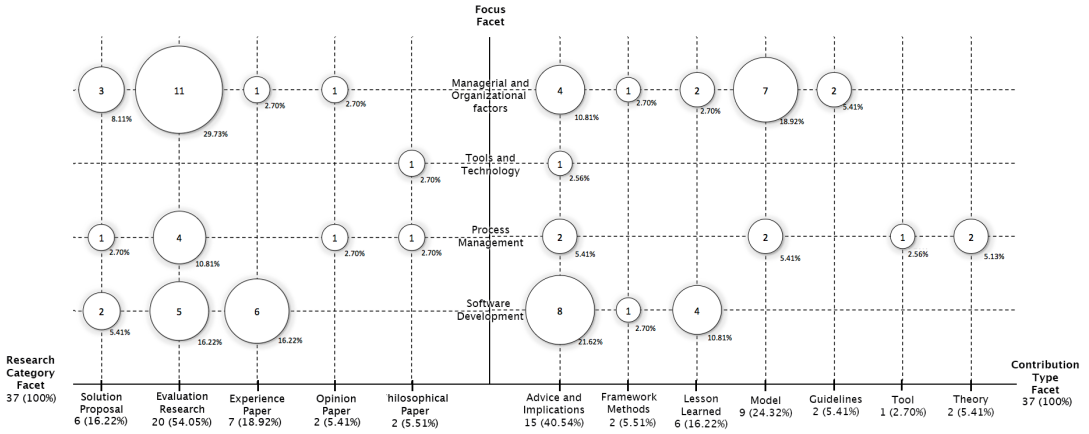


Figure 5.3: Systematic map - *Focus*, *contribution* and *research type*

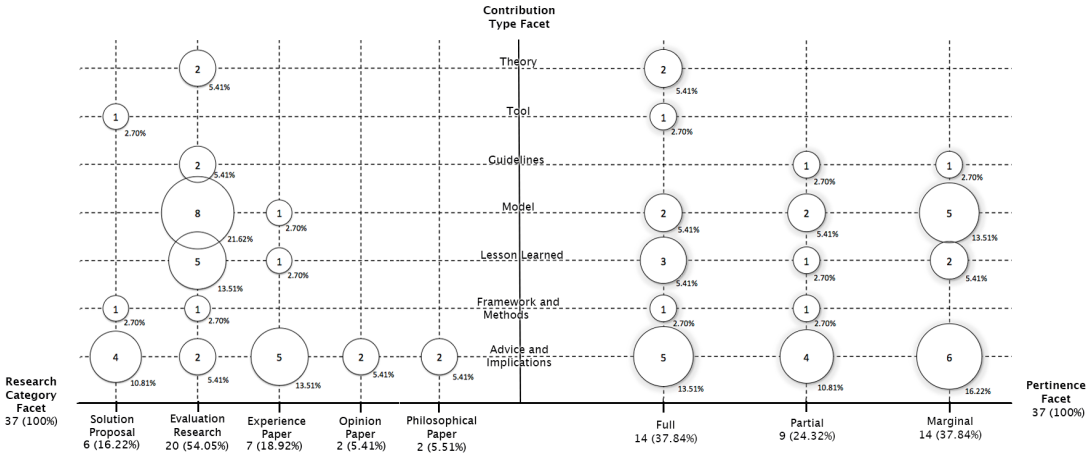


Figure 5.4: Systematic map - *Contribution*, *pertinence* and *research type*

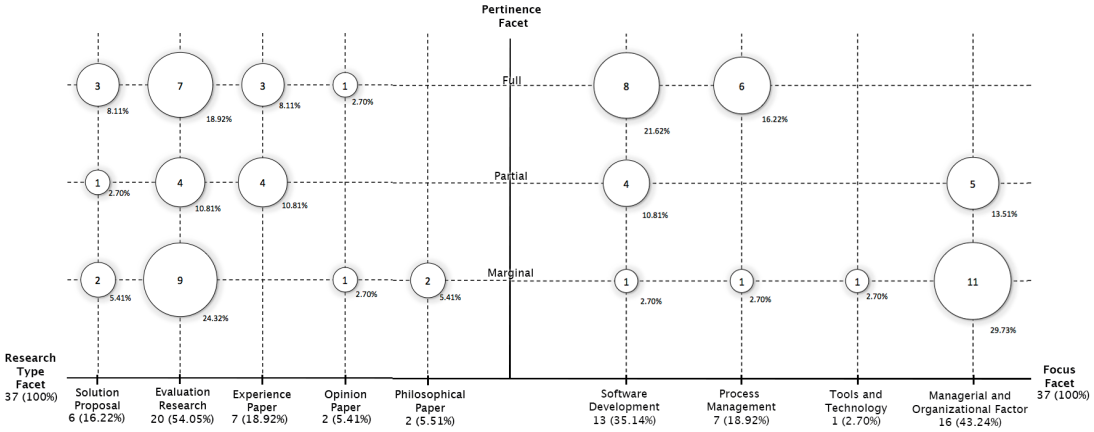


Figure 5.5: Systematic map - *Pertinence*, *focus* and *research type*

By analyzing in depth the three bubble charts representing the *systematic map* of the selected studies, we report a list of most significant observations:

- First of all, by looking at the *pertinence* facet we can observe that only 14 papers (37.84%) are entirely dedicated to software development in startups, and 9 of those produced a *weak contribution* (*advice and implications (5)*; *lesson learned (3)*; *tool* (1)).
- Observing the *focus* facet, it is easy to see how 16 articles (43.24%) are focused on *Managerial and organizational factors*, which are only relatively interesting under a SE perspective. In fact, none of those 16 articles have a *full pertinence*.
- The overall *contribution types* produced by the studies are for the greater part *weak* (24 items, 64.86%). Of the 13 remaining studies (35.13%), which produces a *strong contribution type*, only 4 have the *focus* on what is considered fundamental for our research problem (*software development* and *process management*).
- A good portion of the selected studies were carried out using an *evaluation research* (20 items, 54.05%) which is the only *research type* which involves a field study. However, by simultaneously looking at other facets, we can observe how 11 of these evaluation researches are related to *managerial and organizational factors* , and only 7 out of 20 have a *full pertinence* with engineering activities in software startups. On the other hand the *contribution types*, produced by *evaluation researches*, are widely distributed on the map, whilst other - less empirical - *research types* generally brought only to *weak contribution types*.
- 14 out of 20 studies, with *focus* on *process management* (7) and *software development* (13), have a *full pertinence* with our research area.

So, if we don't consider items which produced a *poor contribution type*, items

which do not have *full pertinence* with software development in startups and items with a non-empirical research approach, only four studies remain ([27, 29, 2, 18]), and represent the most prominent contribution to the field. In the remains of this document and in Related Work (Section 3), the content of these four article is discussed in detail.

Summarizing the implications regarding the RQ-1.1 (*How is the body of knowledge distributed in literature?*) we have found that:

- The literature lacks of relevant primary studies which investigate engineering activities in software startups. The 37 selected studies are researches spread across different scientific areas and journals. In addition, they are weakly interrelated by mutual references. This confirms what was partially revealed by the first non-systematic literature survey and, at the same time, discloses a complex underlying field.
- About 80% of the studies have been published in the last decade, opening a set of new issues brought by the advent of internet and mobile devices which enabled fast and wide proliferation of software startups. The state-of-the-art under a SE perspective is quite recent, especially when compared to siblings areas such as SE in small companies, which is much more advanced in terms of number of primary studies and provided evidences.
- The selected studies focus on a wide variety of topics of which we provided a complete overview. Despite more attention is given to some sort of lightweight methodologies for startups and some discussion about process formation and improvement, the evidence produced are definitely undersized to support the dimension of the actual startup phenomenon.
- To asses the detailed distribution of the body of knowledge we mapped the identified studies into a classification schema which considers four dimensions, namely: research type, contribution type, main focus and pertinence. Of a total of 37 selected studies, 16 of them (43.24%) are focused on *managerial and organizational factors*, which are only partially interesting under a SE perspective. Additionally, only 14 studies (37.84%) are entirely dedicated to software development in startups and 9 of those produced weak contribution types.
- Overall, we identified only 4 important contributions to the field that are published in scientific journals, are entirely dedicated to engineering activities in startups, provide strong contribution type and are conducted through an evidence-based research approach [27, 29, 2, 18]. Three of these studies are from the same author, Gerry Coleman.

5.1.2 Rigor and relevance

Figure 5.6 shows a *pie chart* representing the distribution of the relevant sample in the three *venue* categories.

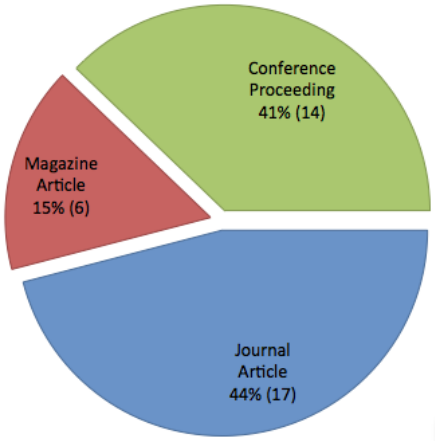


Figure 5.6: Publication distribution - Venue

Despite the scientific validity of a study cannot be a mere consequence of the venue where it has been published, the peer-review process, required for publishing a journal article, is generally much more rigorous and formal than the procedure to get an article published on a scientific magazine or accepted to a conference [103]. As can be seen in Figure 5.6, only 17 publications (44%) in the sample are journal articles, while the remaining 56% consist of conference proceedings and magazine articles. Although this feature alone is not enough to represent a direct implication on the quality[3], it can be interpreted as a first indicator of the scientific quality of the sample, formally assessed with the rigor and relevance process here discussed.

Following the process described in Subsection 4.2.6, we assessed the *rigor* and *relevance* of each paper by summing up the contribution of individual *aspects* defined in the evaluation model [121]. The detailed results of this process are shown in Table 5.6. Observe that the maximum possible value for rigor ($Ri = Ri1 + Ri2 + Ri3$) is 3, and for relevance ($Re = Re1 + Re2 + Re3$) is 4 (see the detailed quantification Table 4.4).

First author (year)	Ri1	Ri2	Ri3	**Ri**	Re1	Re2	Re3	Re4	**Re**	Ref.
Coleman (2007)	1.0	1.0	1.0	**3.0**	1.0	1.0	1.0	1.0	**4.0**	[2]
Coleman (2008)	1.0	1.0	1.0	**3.0**	1.0	1.0	1.0	1.0	**4.0**	[29]
Coleman (2008)	1.0	1.0	1.0	**3.0**	1.0	1.0	1.0	1.0	**4.0**	[27]
Camel (1994)	0.5	1.0	1.0	**2.5**	1.0	1.0	1.0	1.0	**4.0**	[70]

Table 5.6 – *Continued on next page*

[3]The publication criteria are determined by the specific editor of the journal/magazine or the committee of a conference. There is a vast multitude of excellent quality studies presented in conference proceedings and magazines, and many example of poor-quality journal articles.

Table 5.6 – *Continued from previous page*

First author (year)	Ri1	Ri2	Ri3	**Ri**	Re1	Re2	Re3	Re4	**Re**	Ref.
Häsel (2010)	0.0	1.0	1.0	**2.0**	1.0	1.0	1.0	1.0	**4.0**	[158]
Hanna (2010)	0.0	1.0	1.0	**2.0**	1.0	1.0	1.0	1.0	**4.0**	[94]
Deakins(2005)	1.0	0.5	1.0	**2.5**	1.0	1.0	0.0	1.0	**3.0**	[89]
Chorev (2006)	1.0	1.0	0.0	**2.0**	1.0	1.0	0.0	1.0	**3.0**	[34]
Kajko (2008)	1.0	0.5	0.0	**1.5**	1.0	1.0	0.0	1.0	**3.0**	[18]
Jansen (2008)	0.5	0.0	0.5	**1.0**	1.0	1.0	0.0	1.0	**3.0**	[95]
Midler (2008)	0.5	0.5	0.0	**1.0**	1.0	1.0	0.0	1.0	**3.0**	[86]
Steenhuis (2008)	0.0	1.0	0.0	**1.0**	1.0	1.0	0.0	1.0	**3.0**	[161]
Yogendra (2002)	1.0	0.0	0.0	**1.0**	1.0	1.0	0.0	1.0	**3.0**	[87]
Wood (2005)	0.0	0.5	0.0	**0.5**	1.0	1.0	0.0	1.0	**3.0**	[160]
Tingling (2007)	0.0	0.5	0.0	**0.5**	1.0	1.0	0.0	1.0	**3.0**	[77]
Su-Chuang (2007)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	1.0	**3.0**	[162]
Sutton (2000)	1.0	0.0	0.0	**1.0**	1.0	1.0	0.0	0.0	**2.0**	[13]
Kakati (2003)	0.0	1.0	0.0	**1.0**	0.0	1.0	1.0	0.0	**2.0**	[35]
Mater (2000)	0.5	0.0	0.0	**0.5**	1.0	1.0	0.0	0.0	**2.0**	[91]
Yoffie (1999)	0.5	0.0	0.0	**0.5**	1.0	1.0	0.0	0.0	**2.0**	[81]
Deias (2002)	0.5	0.0	0.0	**0.5**	1.0	1.0	0.0	0.0	**2.0**	[80]
Silva (2005)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[79]
Wall (2001)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[96]
Kuvinka (2011)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[85]
Mirel (2000)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[92]
Zettel (2001)	0.5	0.0	0.5	**1.0**	1.0	0.0	0.0	0.0	**1.0**	[88]
Ambler (2002)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[76]
Stanfill (2007)	0.5	0.5	0.0	**1.0**	0.0	1.0	0.0	0.0	**1.0**	[159]
Taipale (2010)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[84]
Sau-ling Lai (2010)	0.0	0.0	0.0	**0.0**	1.0	1.0	0.0	0.0	**2.0**	[163]
Kim (2005)	0.5	0.0	1.0	**1.5**	0.0	0.0	0.0	0.0	**0.0**	[93]
Crowne (2002)	1.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0	**0.0**	[10]
Heitlager (2007)	1.0	0.0	0.0	**1.0**	0.0	0.0	0.0	0.0	**0.0**	[3]
Himola (2003)	0.5	0.0	0.0	**0.5**	0.0	0.0	0.0	0.0	**0.0**	[68]
Bean (2005)	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	0.0	**0.0**	[97]
Fayad (1997)	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	0.0	**0.0**	[90]
Tanabian (2005)	0.0	0.0	0.0	**0.0**	0.0	0.0	0.0	0.0	**0.0**	[33]

Table 5.6: Mapping Study - Rigor-relevance results

The table is sorted by placing at the top the studies, which received the highest scores in the two dimensions summed together. It is straightforward to observe that Coleman's studies received the maximum score for both scientific rigor and industrial relevance. By contrast, there are many papers which report a 0 score (11 zeros for rigor, 7 zeros for relevance), providing a first confirmation of what was only a suspect about the generally low reliability of a good part of these studies.

To further extract useful insight, it is more convenient to adopt a graphical representation of the rigor and relevance. We use the same approach previously utilized to depict the *systematic map*: a bubble chart where the bubble size is proportional to the number of publications, corresponding to the coordinates x-y, along with some statistical data shown for each dimension (Figure 5.7).

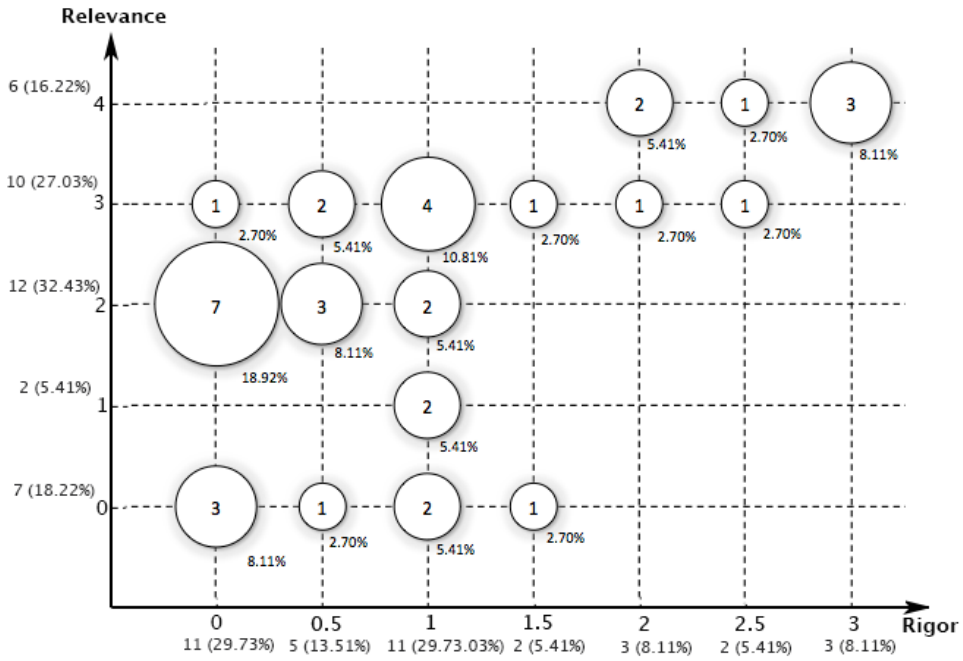


Figure 5.7: Rigor-relevance overview

If we don't consider studies in the highest region of scores (see the top-right part of Figure 5.7) for industrial *rigor* (≥ 2) and *relevance* (≥ 3) from the rest, it is easy to observe how the greater part of the selected studies (78.38%) received mediocre scientific rigor and industrial relevance scores. Only 8 items belong to the *high-score region*.

Moreover, the biggest bubble in the chart shows 7 studies (almost 19% of the total) which received an average score for industrial relevance (2) but only a 0 for the scientific rigor. This kind of studies, according to the authors of the rigor-relevance model [121], presents a major issue: although their findings appear to be somewhat appealing for practitioners (average relevance), the extremely poor scientific rigor of the study will make a possible knowledge transfer to the industry highly unlikely or highly dangerous. Indeed Kitchenham states that one of the most important factors for having academic results applied in the industry is by providing proper scientific evidences of the implications [106, 107].

Finally, by looking at how bubbles are distributed on the chart (see Figure 5.7) with no need of sophisticated statistical methods, it is clear the relation between items which received the highest scores for *rigor* and the ones which received the highest score for *relevance*. Since the two dimensions represent two independent variables the result is not straightforward. In this particular sample, it indicates that results of studies, which have been rigorously reported, are generally more relevant for the industry (and vice-versa). By thoroughly analyzing the studies in the high rigor-relevance region, we observed that the authors have strong academic backgrounds but at the same time they have worked for many years in

direct contact with startups. This is even more visible when comparing them to the other authors in the sample with lower value of rigor and relevance[4]. This could somehow explain the reason why, in our sample, the rigorous studies are also highly relevant.

Summarizing the implications regarding the RQ-1.2 (*What is the industrial relevance and scientific rigor of the published studies?*) we have found that:

- Only 17 publications (44%) in the sample are journal articles. The remaining 56% consists of conference proceedings and magazine articles, which usually require a less rigorous peer-review process than a scientific journal publication.
- By assessing the rigor and relevance of the selected studies with a systematic procedure we attested that the majority of works are poorly relevant for startups and are presented with low scientific rigor. By looking at the detailed rigor-relevance chart, it can be observed how most of the studies are located in the lower region of the rigor-relevance map. Additionally, eleven articles received 0 for scientific rigor, and seven articles received 0 for rigor.
- Coleman's studies [27, 29, 2], which appears to be a prominent researcher in this area, received also the highest score both for scientific rigor and industrial relevance. This provided to our research a solid starting point. In fact, these references are used a considerable amount of times throughout the thesis document.
- Seven studies, which received an average score for industrial relevance, were marked with 0 for the scientific rigor. Thus, although their results could be potentially relevant (2) for the industry, the low rigor makes a possible adoption in startups highly-risky or highly-unlikely. These studies represent almost 19% of the total.
- The most rigorous studies in our sample are highly relevant for practitioners, and vice-versa. We observed that in those cases, the authors are academics with a deep connections with the industry or direct experiences in startups.

5.1.3 Contextual features of startups

In this part we present the results of the data extraction process described in Subsection 4.2.5, where we identified peculiar characteristics of startup companies as described in the literature. The results confirms that there is no agreement on a standard definition which specify what exactly a startup is. Different authors provide different definitions and they use the term *startup* referring to context which are often quite different. This makes any attempt to identify a solid and

[4]Since it is not possible to obtain a complete background study of each author, we based our assumption on a small biographic review.

coherent body of knowledge very challenging.

Defining what makes a software startup unique is an interesting problem: apparently is not strictly related to the size of the company. For instance, some authors call *"startups"*, companies with 6 employees [92], whilst others refer to *startups* with more than 300 employees [81, 27]. It is not about the age of the company either: some authors studied *startups* which have been operating for many years [163], while others are more strict about companies recently founded [18]. For other authors *start-up* is a phase, and it is not clear where to draw the line anyway. Others claim that startups works on *innovative* products, but they don't give the exact definition of innovation, which makes this factor a little confusing (a recent systematic study identified *"41 definitions of innovation in 204 selected primary SE studies"* [64]).

To illustrate how authors use the term *software startup*, we systematically extracted *themes* which characterize companies inquired in the selected relevant studies. We were able to identify 15 main *themes* which are reported in Table 5.7. The last two columns on the right side indicate respectively the frequency of occurrence of a specific theme and its references.

ID	Theme	Description	Frequency	Ref.
T.1	Lack of resources	Economical, human, and physical resources are extremely limited.	13	[87], [81], [10], [18], [70], [29], [2], [27], [94], [13], [76], [159], [33]
T.2	New company	The company has been recently created.	4	[10], [81], [70], [34]
T.3	Small Team	Startups starts with a small numbers of individuals.	7	[88], [81], [10], [18], [13], [34], [33]
T.4	Uncertainty	Startups deal with a highly uncertain ecosystem under different perspectives: market,product features, competition, people and finance	8	[3], [93], [29], [2], [27], [90], [86], [33], [68]
T.5	Little working history	The basis of an organizational culture are not present initially.	2	[81], [76]
T.6	Higly Risky	Failure rate of startups is extremely high.	4	[3], [18], [70], [33]
T.7	Not self-sustained	Especially in the early stage startups need external funding to sustain their activities (Venture Capitalisst, Angel Investments, Personal Funds, ...)	2	[88], [94]
T.8	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.	5	[70], [29], [2], [76], [35]
T.9	Flat organization	Startups are usually founders-centric and everyone in the company has big responsabilities. No high-management.	4	[81], [18], [79], [33]
T.10	One product	Company's activities gravitate around one product/service only.	7	[76], [85], [97], [80], [79], [27], [84]

Table 5.7 – *Continued on next page*

Table 5.7 – *Continued from previous page*

ID	Theme	Description	Frequency	Ref.
T.11	Innovation	Given the highly competitive ecosystem, startups need to focus on highly innovative segments of the market.	11	[3], [87], [92], [161], [95], [163], [13], [158], [158], [86], [35]
T.12	Time-pressure	The environment often forces startups to release fast and to work under constant pressure (terms sheets, demo days, investors' requests)	8	[88], [70], [27], [77], [13], [89], [68], [91]
T.13	Highly Reactive	Startups are able to quickly react to changes of the underlying market, technologies, and product. (compared to more established companies)	11	[88], [18], [70], [29], [27], [77], [13], [90], [76], [85], [80], [79]
T.14	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, ...	7	[81], [160], [96], [95], [163], [94], [13], [76]
T.15	Rapidly Evolving	Successful startups aim to grow and scale rapidly.	8	[87], [81], [70], [27], [162], [13], [76], [85], [89]

Table 5.7: Mapping Study - Recurrent themes

When discussing software startups, thirteen authors reported a general lack of human, physical and economical resources (T.1). For this reason, startups deeply depend upon external software solutions such as third party APIs, COTS and OSS (T.14). Other studies refer to companies which are able to quickly react to changes in the market and technologies (T.13), under remarkably uncertain conditions (T.4). Some authors indicate that these companies are focused on highly innovative segments of the market (T.11), generally working on a single core-product (T.10) under extremely high time-pressure (T.12). Furthermore, eight authors write about startups as fast growing companies (T.15) designed to rapidly scale-up. Other researches mention a very small founding team (T.3), which is often composed by low-experienced people (T.8) with a very flat organization structure (T.9) where the CEO is sometimes a core developer itself. Finally, other studies agree on the highly risky nature of these businesses (T.13) newly created (T.2) and therefore with no or little working history (T.5).

It is important to understand that the above mentioned contextual factors have a relevant impact on software development activities[5], making them different from established companies [13]. And since there is no common agreement on the use of the term *startup* and its implications, it should be an important responsibility of the authors to specify which kind of companies the study actually refers to, in order to avoid misleading and ambiguous results. In the selected studies, most of the authors used the term *startup* without explicitly mentioning what exactly they meant.

[5]To further attest the validity our theoretical model, in Section 6.6 we use the above mentioned *themes* to execute a comparative analysis with the theoretical framework we developed with the *grounded theory* case study.

For this reason in Chapter 2, we specified that our case study and consequently the results of the grounded theory regard to *newly created and product-centered companies, in the time-frame that goes from the idea conception to the release of the first product in highly scalable markets* (this formulation was initially shaped around the definitions used in [59, 7, 22]). We finally dedicated a subsection to discuss the generalizability of results to similar domains (see Section 6.7).

Summarizing the implications regarding the RQ-1.3 (*What are the features which characterize the context of software development in startups, reported in literature?*) we have found that:

- Different authors use the word *startup* actually referring to different kind of companies. It is very hard to identify an explicit definition of the context in which the companies operate, often given for granted. Under these conditions, trying to identify a coherent body of knowledge is even more challenging and is even harder for practitioners to somewhat adopt the results without having a proper context. For instance, some results about *"startups"* have been obtained by studying companies with more than 200 employees and others from companies with 6 employees. To make an example, from a SE engineering point of view, the size of the company actually have several implications on the software development activities [165].
- To capture the implicit contextual features that authors used when referring to software startup companies we identified a set of 15 main *themes* and counted their frequency in the selected studies. The most frequent reported themes concern the general lack of resources, highly reactiveness and flexibility, intense time-pressure, uncertain conditions and fast growth.
- Since the contextual boundaries of startups resulted to be highly blurred, it is responsibility of the researchers who refer to *"startups"* to explicitly mention which are the features of the company that the study is actually concerned with (in most of selected studies an explicit contextualization has been neglected). For this reason we explicitly declared the area of interest of our case study in Introduction (see Chapter 1).

5.1.4 State-of-the-art: summary (RQ-1)

In this subsection we provide the overall answer to the RQ-1: *What is the current state-of-the-art in the SE literature pertaining to engineering activities in startups?*.

To emphasize the importance of articles, which brought prominent contributions to the area under investigation, we followed the procedure described in Subsection 4.2.7, computing a score (in the range $[0-10]$) for each selected study. Table 5.8 lists the dimensions which were used to assign the final score, next to the arbitrary weight to balance the ranking criteria.

W	Dimension (id)	Weight
w_1	Pertinence (P)	.25
w_2	Rigor (Ri)	.175
w_3	Relevance (Re)	.175
w_4	Age (A)	.15
w_5	Venue (V)	.1
w_6	Contribution (C)	.05
w_7	Research type (R)	.05
w_8	Focus (F)	.05
	TOTAL:	**1**

Table 5.8: Ranking weights

The most important dimension, counting for the 25% alone, is *Pertinence*, followed by *rigor* and *relevance*, respectively counting 17.5% each. The remaining dimensions are increasingly less important to the final score, which is computed for each paper i by adding up the contributions as shown in the formula below.

$$Score_i = (w_1 \cdot P_i) + (w_2 \cdot Ri_i) + (w_3 \cdot Re_i) + (w_4 \cdot A_i) + (w_5 \cdot V_i) + (w_6 \cdot C_i) + (w_7 \cdot R_i) + (w_8 \cdot F_i)$$

The final score is presented in Table 5.9. The dimensions, which contributed to the final score, are the columns representing: (A)ge, (Ri)gor, (Re)levance, (V)enue, (P)ertinence, (C)ontribution type, (R)esearch type and (F)ocus.

First author (year)	**Score**	A	Ri	Re	V	P	C	R	F	Ref.
Coleman (2008)	**9.70**	1.20	1.75	1.75	1.00	2.50	0.50	0.50	0.50	[27]
Coleman (2007)	**9.70**	1.20	1.75	1.75	1.00	2.50	0.50	0.50	0.50	[2]
Coleman (2008)	**9.70**	1.20	1.75	1.75	1.00	2.50	0.50	0.50	0.50	[29]
Kajko (2008)	**8.09**	1.20	0.88	1.31	0.70	2.50	0.50	0.50	0.50	[18]
Häsel (2010)	**7.47**	1.50	1.17	1.75	1.00	0.75	0.50	0.50	0.30	[158]
Hanna (2010)	**7.47**	1.50	1.17	1.75	1.00	0.75	0.50	0.50	0.30	[94]
Deakins(2005)	**6.87**	0.90	1.46	1.31	1.00	1.25	0.50	0.15	0.30	[89]
Camel (1994)	**6.61**	0.15	1.46	1.75	0.70	1.25	0.30	0.50	0.50	[70]
Silva (2005)	**6.58**	0.90	0.00	0.88	1.00	2.50	0.30	0.50	0.50	[79]
Midler (2008)	**6.55**	1.20	0.58	1.31	1.00	1.25	0.40	0.50	0.30	[86]
Taipale (2010)	**6.53**	1.50	0.00	0.88	0.70	2.50	0.30	0.15	0.50	[84]
Chorev (2006)	**6.43**	0.90	1.17	1.31	1.00	0.75	0.50	0.50	0.30	[34]
Zettel (2001)	**6.32**	0.60	0.58	0.44	1.00	2.50	0.40	0.30	0.50	[88]
Jansen (2008)	**6.25**	1.20	0.58	1.31	0.60	1.25	0.30	0.50	0.50	[95]
Sutton (2000)	**6.11**	0.60	0.58	0.88	0.60	2.50	0.30	0.15	0.50	[13]
Heitlager (2007)	**6.08**	1.20	0.58	0.00	0.70	2.50	0.30	0.30	0.50	[3]
Tingling (2007)	**5.99**	1.20	0.29	0.00	0.70	2.50	0.30	0.50	0.50	[77]
Deias (2002)	**5.92**	0.60	0.29	0.88	0.70	2.50	0.30	0.15	0.50	[80]
Stanfill (2007)	**5.74**	1.20	0.88	1.31	0.70	0.75	0.30	0.30	0.30	[159]
Wood (2005)	**5.70**	0.90	0.29	1.31	1.00	1.25	0.30	0.15	0.50	[160]
Steenhuis (2008)	**5.65**	1.20	0.58	1.31	0.70	0.75	0.30	0.50	0.30	[161]
Yogendra (2002)	**5.55**	0.60	0.58	1.31	0.70	1.25	0.30	0.50	0.30	[87]
Ambler (2002)	**5.53**	0.60	0.00	0.88	0.60	2.50	0.30	0.15	0.50	[76]
Crowne (2002)	**5.48**	0.60	0.58	0.00	0.70	2.50	0.30	0.30	0.50	[10]
Mater (2000)	**5.45**	0.60	0.29	1.31	0.70	1.25	0.50	0.50	0.30	[91]
Kakati (2003)	**5.41**	0.90	0.58	0.88	1.00	0.75	0.50	0.50	0.30	[35]
Kuvinka (2011)	**5.28**	1.50	0.00	0.88	0.70	1.25	0.30	0.15	0.50	[85]
Su-Chuang (2007)	**5.26**	1.20	0.00	1.31	0.70	0.75	0.30	0.50	0.50	[162]
Sau-ling Lai (2010)	**5.23**	1.50	0.00	0.88	1.00	0.75	0.30	0.50	0.30	[163]
Mirel (2000)	**4.98**	0.60	0.35	0.88	1.00	1.25	0.30	0.30	0.30	[92]
Himola (2003)	**4.57**	0.90	0.58	0.44	1.00	0.75	0.30	0.30	0.30	[68]
Kim (2005)	**4.53**	0.90	0.88	0.00	0.70	0.75	0.50	0.50	0.30	[93]

Table 5.9 – *Continued on next page*

Table 5.9 – *Continued from previous page*

First author (year)	**Score**	A	Ri	Re	V	P	C	R	F	Ref.
Wall (2001)	**4.28**	0.60	0.00	0.88	0.60	1.25	0.30	0.15	0.50	[96]
Yoffie (1999)	**4.22**	0.60	0.29	0.88	0.60	0.75	0.30	0.50	0.30	[81]
Bean (2005)	**3.50**	0.90	0.00	0.00	1.00	0.75	0.30	0.15	0.40	[97]
Tanabian (2005)	**3.10**	0.90	0.00	0.00	0.70	0.75	0.30	0.15	0.30	[33]
Fayad (1997)	**2.45**	0.15	0.00	0.00	0.60	0.75	0.30	0.15	0.50	[90]

Table 5.9: Mapping Study - Ranking of selected studies

It can be observed that the same 4 papers which occupy the first 4 positions of the above presented *ranking* ([27, 29, 2, 18]), are in the most interesting regions of the *systematic maps* described before, and at the same time are rated as highly rigorous and relevant (Table 5.9). These studies are, in fact, the ones which mostly guided and supported us in the research process. In particular Coleman's studies received the highest scores mainly because his works are actually focused on software process in startups from a SE perspective, undertaking rigorous and well documented approaches, and producing meaningful outcomes published on relevant journals. However, we observed that the three publications are, in fact, derived from the same study with 21 companies.

Although a little part of the companies inquired in his publications had an extremely limited number of employees involved in software development, the greatest part of them had more than 20 developers/engineers, making its results hard to generalize to actual early-stage startups with 3 or 4 founding members.

The results of the systematic mapping of the literature, fostered our motivation in filling the gap by executing a case study on early-stage startups. Thus, our case study is clearly distinct from what has already been investigated by Coleman. In fact, the companies, which we inquired in the interviews at the time of the first release, employed an average number of about 8 employees, contrary to Colman's companies which employed up to 190 employees.

Moreover, as Coleman himself admits, his results suffer from a limitation which is not present in our study due to the fact that his respondents were covering high-managerial roles (Managers/CTOs/Head of development) which are in practice *"one or more steps removed"* from actual carrying out of the software development [2]. Therefore their opinions are strongly biased towards methodologies they are themselves proposing to their teams. By contrast, in the early stage of the studied startups, the totality of our respondents were directly involved within all the activities of software development, giving to us the possibility to actually attest more reliable data[6].

Finally, we summarize what are the results of the overall systematic mapping, conducted through: creation of a classification schema which was used to map the existing literature; a systematic assessment of the rigor and relevance of the

[6] As remarked by Coleman, interviewing engineers in mature companies can be misleading to understand the higher-level dynamics, as they might not be aware of all process issues. By contrast, in our research, respondents have been wearing multiple hats during the first stages, acting as managers and engineers at the same time.

selected studies; and finally identifying contextual features which characterize software startups. The overall results of the analysis are summarized as follows (answering RQ-1):

- The evidences provided by the 37 selected studies are, for the most part, inadequate to understand the underlying phenomenon of software development in startups. To the current date, twelve years after Sutton assessed that startups have been neglected from process studies [13], the gap has been only partially filled.
- From the provided *systematic map*, it can be observed that only 13 studies out of 37 selected, are entirely dedicated to the study of software development in startups. The remaining articles are only partially or marginally mentioning relevant contributions to the area. Additionally, by visualizing the four dimensions of the systematic map with the technique proposed by Petersen et al. [1], we were able to provide detailed insights by simultaneously analyzing multiple facets of the literature distribution.
- The creation of a coherent body of knowledge about software startups is restrained by the fact that different authors use the word *startup* in different contexts (above summarized by showing the more cited contextual *themes*). Some authors use the word *startup* for small new companies while others considered startups companies with hundreds of employees. Others refer to innovative companies or operating in uncertain markets, and for others, *startup* is the name of a phase in a software project. Given this lack of consensus and consistency, when investigating software startups, it is responsibility of the researcher to make explicit mention to the particular context affected by the study. In the selected studies an explicit mention was often neglected, affecting the generalizability of results.
- The results of selected studies are mediocrely relevant for the industry. Furthermore, considering the general lack of high scientific rigor, it will be very unlikely that the findings can have a real impact on startup companies. Since transferring the knowledge to the companies should be one of the most important concern of SE research [121], the next generation of studies should provide more relevant and scientifically rigorous evidences. However, in our sample, we identified a small portion of studies with both very high relevance and very high rigor. We observed that the authors of those papers have generally academic background with strong connections or past experiences with startup companies. This might infer that the combination of excellent academics skills (providing rigor) with practical experience in the field (providing relevance) fosters the quality of a startup-related study.
- We identified only four prominent contributions to the field by ranking the studies according to a defined score, considering different features of the study. Furthermore we realized that three of these studies, conducted from the same author, are based on the same empirical data collected in 21

companies with different profiles.

- Some of the features, which characterize startups, are common to other SE domains (innovation, market-driven development, small companies, short time-to-market, ...). However, the unique combination of elements, which coexists in modern startup companies, poses a new series of issues which need to be addressed with primary studies in the specific domain. A new and consistent body of knowledge should support activities and decisions of the fast growing number of startup companies, through evidence-based research [106, 129]. The need of more studies is attested by the impressing proliferation of (non-peer reviewed) books dedicated to startups, which quickly became best-sellers (see Appendix A.2.5 for detailed books review).
- Finally, all of the above mentioned results fostered our motivation in pursuing research in this field. In particular, we observed that most studies focused on *mature* startups and we couldn't identify any relevant empirical evidences discussing engineering activities in the very early-stage of the startup creation. This factor contributed in driving the direction of our case study towards a cross-sectional analysis in the time frame that goes from the idea conception to the first release of the product.

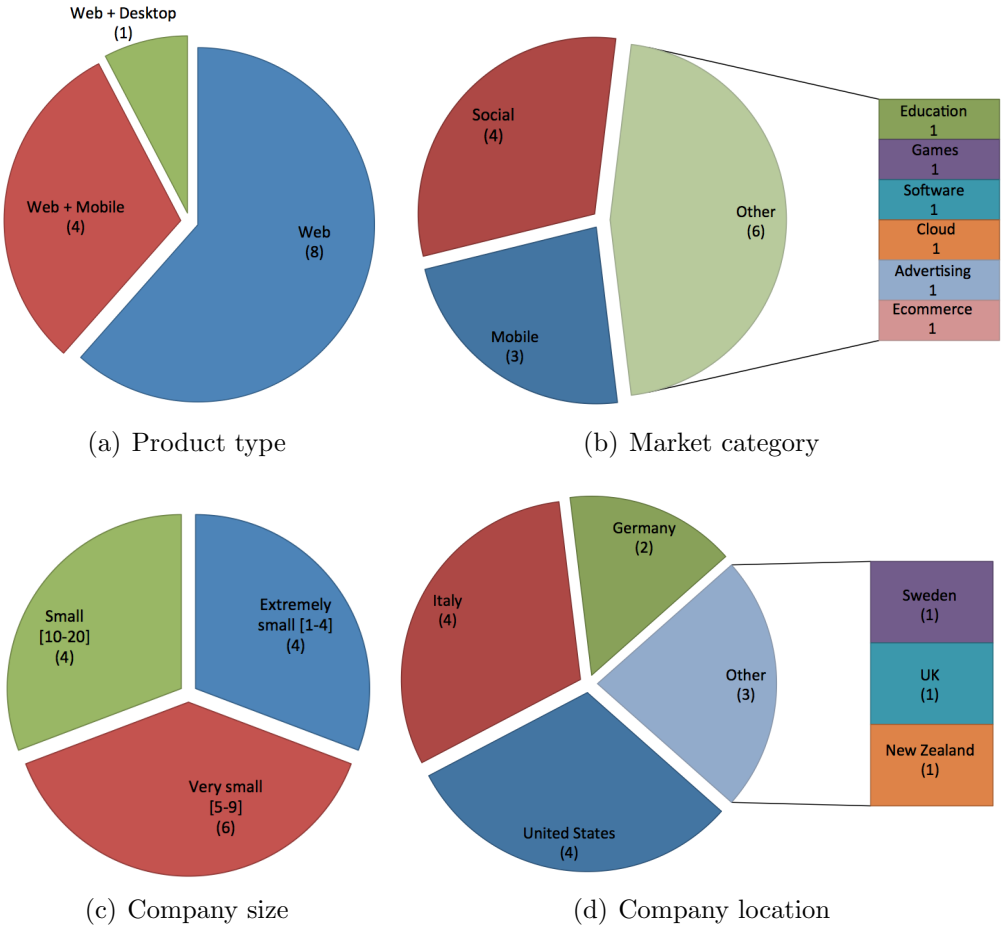
5.2 Case study

In this section we present and discuss the initial outcomes of the case study conducted through semi-structured interviews and follow-up questionnaires. Since the research process followed a grounded theory approach, the results outlined in this section underwent a further process of analysis which converged in the creation of a theoretical model, which is fully presented and discussed in the next chapter (see Chapter 6) where the RQs are answered.

In the following part of this section we: discuss the sample of companies involved in the study (Subsection 5.2.1); provide an overview of the coding process (Subsection 5.2.2); display the results of the questionnaires (Subsection 5.2.3); and finally execute a comparison of adopted methodologies between interviews and questionnaires (Subsection 5.2.4).

5.2.1 Companies distribution

To understand how the population of 13 companies, which participated to the case study is structured, we show in Figure 5.8 the distribution of the sample in terms of *product type*, *business category*, *location* and *size*.



(a) Product type (b) Market category

(c) Company size (d) Company location

Figure 5.8: Sample companies distribution

The totality of the startups[7] has at its core a web product (see Subfigure 5.8(a)): 4 of them with a mobile component and 1 with a desktop application. By looking at the market category[8] (see Subfigure 5.8(b)), the companies are well distributed across different domains: 4 *social*, 3 *mobile* and the remaining 6 working in *education*, *games*, *software*, *cloud*, *advertising* and *ecommerce*. As can be observed in Subfigure 5.8(c), all the companies are quite small: four startups employ between 10 and 20 people (*small*), five of them between 5 and 9 people (*very small*) and four between 2 and 4 people (*extremely small*). Furthermore the sample is distributed across different geographic areas, worldwide (see Subfigure 5.8(d)). Thus, the sample considered here is composed by small web-oriented startups spread in different countries and market sectors.

Additional information about the companies were elicited during the inter-

[7]The companies are listed in Subsection 4.3.2.

[8]The business category has been assigned according to the profile of the company on *CrunchBase* [166].

views and reported in Table 5.10. The first column represent an identifier assigned to each company, that is further used in Chapter 6.

ID	Company age (months)	Founding team	Initial developers	Current employees	First product building time (months)
C1	11	4	2	11	6
C2	5	2	2	6	3
C3	18	4	4	4	6
C4	17	3	2	11	6
C5	20	2	1	4	12
C6	30	3	2	4	1
C7	12	2	1	7	4
C8	24	4	3	16	4
C9	5	5	4	5	1
C10	43	6	4	9	4
C11	36	3	3	6	2
C12	12	3	3	3	3
C13	24	2	2	20	3

Table 5.10: Companies overview

As shown in column *Company age*, most of the companies were founded within the last three years with only one exception (C10, which age was 3 years and 6 months). As can be observed by looking at the column *Founding team*, the companies have been founded by an average number of 3 members, which in majority are developers, as can be seen by looking at the column *Initial developers*. Moreover, the number of *Current employees* shows how, to different degrees, companies expanded the initial teams, being able to bring the product to the market in less than 6 months from the idea conception[9] (see *First product building time*). Summarizing the above identified features, the sample of the case study is mainly composed by recently founded companies, working with web applications in different nations and market sectors, with very small development teams. Moreover, the growing team size and and the publicly available data suggest an healthy status of the businesses.

5.2.2 Coding process overview

By analyzing the interview transcripts with the process explained in Subsection 4.3.4 we extracted the information necessary to form the theoretical model presented in Chapter 6. The initial coding process (open and in-vivo) provided 630 unique codes, grounded to 1295 citations in the transcripts, divided according to the thematic areas. Table 5.11 summarizes the distribution of *unique codes* and the frequency in which they have been *grounded* into the data.

[9] C5 building time is extended to 1 year mainly because the product has been developed part-time by one person only.

Thematic Area	Unique codes	Grounded
Opening questions	11	83
Product discussion	120	242
Process discussion	169	332
Ideation and Requirements	63	122
Analysis	25	47
Design and architecture	49	99
Implementation	75	159
Verification and Validation	59	109
Deployment	13	36
Closing questions	46	66
Total	630	1295

Table 5.11: Number of codes

The thematic areas were divided according to the development phases designed for the interview process. The complete list of raw codes is presented in appendix A.4.3 and constitutes the empirical foundation of the theoretical categories, which has been used to form the theoretical framework presented in Chapter 6.

5.2.3 Follow-up questionnaires results

In this subsection we present the results obtained with the follow-up questionnaires, structured according to the design process discussed in Section 4.3.2. Two weeks after the execution of the last interview we closed the surveys and collected the data, obtaining a response rate approximated to 70% (9 companies out of 13 completed it). When triangulating the data, we promptly observed some inconsistencies between what has been said during interviews and what has been answered in the questionnaires. Therefore, we asked to respondents to clarify inconsistent results, validating that the answers provided during interviews were, in all instances, correct. In these cases we adjusted the survey result consequently. To discuss this issue, in the last part of this chapter we compare the two methodologies and discuss the reasons behind such inconsistencies (see Subsection 5.2.4). However, for the sake of completeness and transparency we decided to briefly show in this section results of the questionnaires. Therefore we reduced the level of analysis performed on them. In fact, none of the conclusions are drawn solely based on the questionnaire results in the generation of the theoretical categories, entirely based on the interviews. The questionnaire results are used only as support material in the validation of the theory (see Section 6.6).

Quality achievements

All the quality aspects discussed during interviews have been evaluated by the respondents and presented[10] in Table 5.12.

Company	UX	Efficiency	Interoperability	Portability	Functionality	Scalability
C1	5	3	2	–	–	–
C2	3	3	4	–	–	–
C3	3	3	–	–	–	–
C4	N/A	N/A	N/A	N/A	N/A	N/A
C5	5	–	–	4	4	5
C6	4	–	–	–	–	–
C7	4	–	–	3	3	–
C8	N/A	N/A	N/A	N/A	N/A	N/A
C9	3	–	–	5	5	–
C10	4	–	5	–	–	4
C11	N/A	N/A	N/A	N/A	N/A	N/A
C12	–	5	–	3	–	-
C13	N/A	N/A	N/A	N/A	N/A	N/A
Count	8	4	3	4	3	2
Average	3.88	3.50	3.67	3.75	4.00	4.50
1=Extremely poor; 2=Below average; 3= Average; 4=Above average; 5=Excellent;						

Table 5.13: Qualities achievement

The *Count* row represents the number of those companies which have considered the specified quality attributes very important in their first release. It can be observed how 8 out of 9 startups have considered aspects related to user experience (UX) as very important. By looking at the *Average* values selected by companies, it is possible to note how respondents declared to achieve their quality aspect with an evaluation that is above the average of the presented cases.

Effort distribution

Table 5.14 shows how respondents declared to distribute the total effort on the different development phases involved in the creation of the first product.

[10]The hyphen indicates that the specific quality attribute was not an important concern and therefore not included in the customized questionnaire while the companies that didn't submitted their response are crossed. Also note that the presented quality attributes initially considered, were defined according to the ISO/IEC 9126, and subsequently adjusted according to practitioners' responses. This is the case of the user experience, which has not been defined by the ISO/IEC 9126, but considered as the most important aspect to accomplish within the studied startups.

Company	Requirements	Analysis	Design	Implementation	Testing	Deployment
C1	10%	15%	5%	30%	30%	10%
C2	10%	10%	10%	30%	20%	20%
C3	15%	5%	10%	60%	8%	2%
C4	N/A	N/A	N/A	N/A	N/A	N/A
C5	10%	10%	15%	45%	18%	2%
C6	10%	10%	30%	30%	10%	10%
C7	10%	5%	5%	50%	25%	5%
C8	N/A	N/A	N/A	N/A	N/A	N/A
C9	20%	10%	20%	30%	10%	10%
C10	20%	10%	15%	35%	10%	10%
C11	N/A	N/A	N/A	N/A	N/A	N/A
C12	15%	15%	20%	40%	8%	2%
C13	N/A	N/A	N/A	N/A	N/A	N/A
Average	13%	10%	14%	39%	15%	8%

Table 5.14: Questionnaire results - Effort by phase

The average values across startups, which are displayed in Figure 5.9, show how the majority of resources are, as expected, dedicated to *implementation*[11].

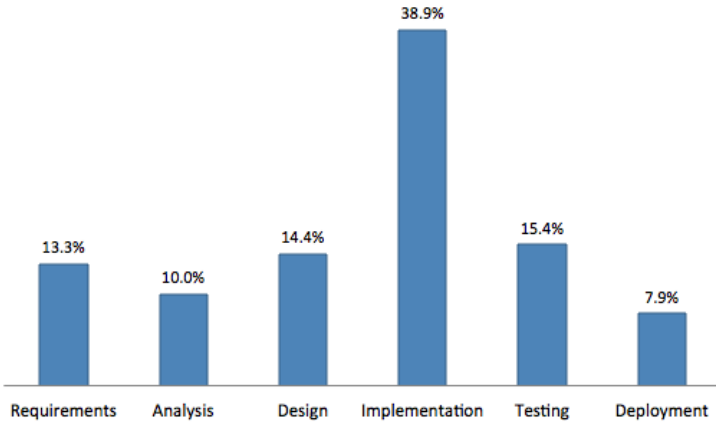


Figure 5.9: Development effort

Engineering elements fostering speed

As described in methodology (see Subsection 4.3.2) respondents assigned scores on a five-steps rating scale evaluating those engineering elements which contributed to foster time-to-market, as mentioned during the interviews. The results of the repertory grid are summarized in Table 5.15, where elements (separated by semi-colon) have been grouped according to the opinion expressed by respondents[12] in *Extremely useful*, *Very useful*, *Somewhat useful*, *Little use* and *Useless*.

[11]Due to identified inconsistencies (further discussed in Subsection 5.2.4), we do not speculate much on this plot.

[12]Multiple recurrences (N) are indicated by X N.

Time-to-market contribution level	Engineering elements
Extremely useful	Amazon EC2; Analyzing competitors' feedbacks; Basecamp's tasklist; CEO solving conflicts in development decisions; Chef for deployment; Clean-code; Co-located team members; Collecting feedback from pre-launch through landing page; Competitor's analysis; Consulting available libraries before start implementing; Cross-functional team (full stack developers); Daily stand-ups; Dashboard of the Content Manager; Database model; Deploy workflow (feature brench - pull req - Capistrano); Development experience; Early identification of functional bugs with Unit tests; Feature meetings; Flat hierarchy of the team; Get early feedback from customers; Git for code-base; Github for having review of the code; Google analytics; Having senior developers; Heroku for deployment; High-experience developers; HTML5, CSS3; Hype of media for increasing moral of developers; Illustrator for icons; Informal meetings for discussing biggest changes only; Initial feature list (whiteboard); Initial survey to collect user feedbacks; Integration tests (Selenium for browser tests); Mind mapping instead of text writing communication; MySQL as DBMs; Only in-line comments to document the code; Oral communication; Personal experience for story estimation; Post-it notes for tracing tasks and bugs; Prototype an Hybrid Django/Php; RESTful API; Scrum board (by means of whiteboard with post-it notes); Self-imposed informal deadlines (Google spread-sheet); Skype for communication; Using an MVP approach; Using whiteboard for main focus of the product/vision; Whiteboard for tracing the progress (modules implemented);
Very useful	Analysis of critical/important use case scenarios; Assembla for managing tickets/tasks; Automatic testing; Basecamp for bugs/issues; Break-down of big tasks in smaller tasks; Build APIs to export functionalities to mobile; Class diagrams; Continuos integration; Create ticket on-the-fly without any analysis and design; Customer development and Lean startup methodology; Deployment on Amazon infrastructure; Developer responsible for designing,coding and testing a feature.; Developing the product without having schemas/diagrams; Dropbox for sharing documents (x3); Electronic Kanban Wall for managing features (Agile Zen); Evolutionary prototyping/MVP (X4); Focus Group for assessing graphical aesthetic; Focus Group for setting the main functionalities; No need of formal analysis (past experience); Having Mentors in early stage; Hip-chat for internal communication; Initial architectural diagram (communication interfaces); Lack of a structured framework from the beginning; Lack of detailed documentation; Lack of formal and automatic testing; List of features (paper notes); Mock-ups of the UI; Naive diagrams (disposable) instead of UML communication diagrams; Postpone potential choices which could "limit"; Structure the app in a self-explanatory way with Rails; Collecting initial feedbacks before coding; Separation of technical from non-technical documents; Setting informal deadlines between co-founders; Short release time (weekly deployment); Sketches (wireframe); Skype for assigning bugs; Slides for the presentation of the product; Starting from a previously developed technology; SVN for the codebase; Let user test secondary functionality; Treating bugs as user stories; Trying the product internally to identify bugs/issues; Unit testing; Use Cases (Assembla); User feedbacks (by means of the "super circle" of users); UserVoice for collecting users' feedback; Using a Whiteboard; Using Linode to deploy the application;

Table 5.15 – *Continued on next page*

Table 5.15 – *Continued from previous page*

Time-to-market contribution level	Engineering elements
Somewhat useful	Analysis of competitors; Analyzing only parts of the system; Assembla for measuring project progress; Co-located team; Communication diagram and Middleware specification; Cucumber to test and document the code; Email for collecting feedbacks; Emails for communication with developers; ER diagrams for model data; Estimating task effort; Feedback forum; Github for managing of the codebase (X2); Github for assign bugs and stories; Google Documents to collect customer-feedback; Having little/no documentation (X2); Lack of formalized design (X2); Having one-month milestones; In-line comments; Internal/in-house testing; Lack of formal analysis of possibilities; Focusing only on implementation (no management); Launch video; List of issues and bugs; List of user stories on Trello; Mock-ups of the product UI; Not writing automatic tests for client-side; Oral discussions with co-founders; Photoshop for web-design; Previous experience with Scrum; Prioritizing stories in three levels (urgent/to do/ideas); Rails standards instead of formal documentation of the code; Scheduling milestones/deadlines; Shared vision of the product; Show and tell meetings; Stand-up meetings (every 2 days); Using Hackpad for features categories; Writing clean-code; Wunderlist as tool for task management;
Little use	Wanderlist; Uservoice for user feedbacks; Two-weeks sprints schedule; Timedoctor; Test reports; Starting from a functioning prototype; Skype for communication; Ruby on Rails as web-framework; Ruby and Mongo as technologies; Refactoring the code base; Phpdoc; Oral discussion instead of formal design (X2); Not having formal automatic testing (X2); Mysql Workbench for modeling db; Let friends test the product before the release; Lack of measurements; Lack of in-line comments in the code; In-line comments only for important/complex functions; Having a free and elastic development process; Github logs for traceability of stories; Github for managing issues and bugs; Github for deploy; Focus Group for choosing the app name; Evernote; Emails for communication; Business Model Canvas for brainstorming;
Useless	Analyzing competitors' projects; Not having formal automatic testing; Refactoring the code base;

Table 5.15: Questionnaire results - Elements fostering time-to-market

The engineering elements specified by practitioners and reported in Table 5.15 are further used to validate the theoretical model (see Subsection 6.6.5).

Satisfaction

Answers to open questions have been used by researchers to analyze the interview results and are not shown in this section. Answers related to the satisfaction level with the undertaken development approach, all characterized by high values, are presented in Table 5.16, confirming what has been vastly discussed in the interviews.

Company	Satisfaction level
C1	Somewhat satisfied
C2	Extremely satisfied
C3	Somewhat satisfied
C4	N/A
C5	Very satisfied
C6	Very satisfied
C7	Very satisfied
C8	N/A
C9	Very satisfied
C10	Very satisfied
C11	N/A
C12	Somewhat satisfied
C13	N/A
Average	3.67

Table 5.16: Questionnaire results - Development approach satisfaction

5.2.4 Comparison of methodologies: interviews and questionnaires

As already mentioned in Subsection 5.2, in questionnaire results we observed small inconsistencies between what some respondents declared in the interview and the answers they gave in the survey. We contacted the respondents and made a more in-depth comparison, and what we've found is that the questionnaires' results were much more unreliable than the data obtained with the interviews. However, we decided to display the raw data we obtained from questionnaires (see Subsection 5.2.3) without drawing any important conclusion from the results alone.

First of all, despite all the companies agreed beforehand to answer the follow-up questionnaire and they have been solicited multiple times, yet we lack the answer of 4 startups, which makes a more in depth comparative study unfeasible. We usually sent to companies the survey within 24-hours from the interview execution and we paid attention in using respondents own words and expression to represent the concepts, always providing a free form option to answer the question in case of doubt and lack of clearness.

Furthermore, for some answer which appeared inconsistent with interviews' results, we contacted the respondents asking for clarification. What we found is that the reason behind such inconsistencies is not the question's formulation, but mainly lack of time to answering them: entrepreneurs are constantly flooded with surveys from market research and they are not much inclined to answer them. Therefore, as explicitly declared by some of the respondents, they *"didn't put much effort"* in the task as they don't have time to dedicate to it. In fact, when we asked the same inconsistent questions over the phone, responses were consistent with the interviews.

Note that the degree of reliability varies among different responses we received:

some of them are more adherent than others to what has been said during the interview. For instance a respondent, who during the interview explicitly expressed that they didn't use any analysis procedure (paraphrasing, *"we didn't have any analysis, formal or informal, we just jumped into the code after taking the ticket from the whiteboard"*), when asked to distribute 100 points of effort among different phases in the online survey (see Subsection 5.2.3) he assigned 30 points to *analysis* phase. He declared they spent 30% of their time and resources in analysis, in contrast with what we discussed during the interview and with the data triangulation (Q: *"do you have any document or picture of the whiteboard for the analysis?"*, A: "no"). For similar cases, when the inconsistency was clear, we asked for clarification and when possible adjusted the answer according to what they actually meant.

Behind the limited time and attention whereby respondents answered the survey, another cognitive bias could have affected them when answering questions. When posed in front of an official and formal questionnaire, respondents acknowledged the fact that the result they were inserting would have been individually evaluated, so they did their best to provide a good image of their company (*social desirability bias*) and at the same time try to accommodate researchers expectations (*response bias*) [167]. On the other hand, during the interview the settings were much more informal and colloquial, and respondent were free to talk and express whatever they wanted to say without the pressure of being judged by two young students.

Another lesson that we learned is that not only questionnaires but even semi-structured interviews can be limiting instruments when exploring complex problems domain (confirming the arguments of Robson [103]). Following a script and only asking predefined questions without adapting them to respondents' answers and further delve into the problem can severely affect correctness and validity results[13]. We experienced this issue during our interviews, in particular when starting inquiring the development process with the question *"Did you use any specific development methodology?"* (Q.3.1 in Table A.13). Indeed, 10 companies out of 13 answered this question declaring that they used some sort of ad-hoc lightweight version of *Scrum* or *XP*, when in fact, none of them applied any development methodologies at all[14]. Although we observed this kind of inconsistency early during our preliminary interviews, we decided to keep asking the same question throughout all interviews to be able to witness similar and comparable behaviours. We believe that the repeated observations, made during online in depth interviews, could hinder the validity of some software process studies, especially those which investigated the adoption of development methodologies

[13] In our study this risk has been mitigated by the implicit use of the *5 whys* technique [130].

[14] Throughout all the interviews we explored in depth, phase by phase, the development activities. What we have found is that startups only adopt some rough high-level concept of methodologies, without using any of the practices prescribed. Only a few exceptions can be made for pair programming and test-driven development, yet seldom and never systematic.

and conducted only by means of massive online questionnaires.

What we've learned from this result is that for understanding the true reasons behind a complex phenomenon such as software development, an in depth discussion is always needed to obtain the necessary full attention of the respondents and dig into the underlying *"whys"*. This is especially true for startups that work up to *"16 hours per day"* to put the product online as soon as possible and do not have time for focusing on an online questionnaire, which they constantly receive from different researchers and companies. Answers can hardly be mere numbers: every issue involved is much more complex and deserves its own in-depth discussion.

In the next chapter we present the theoretical model which emerged from the systematic analysis of GT data.

Chapter 6

Theoretical model

6.1 Introduction to the model

In this section we present a theoretical framework attempting to model and capture the underlying phenomenon of software development in early-stage startups. The framework, here presented, is based on a systematic procedure (see Subsection 4.3.4) which is grounded into empirical concepts obtained by conduction of the case-study. Starting from 630 raw *codes* extrapolated from the interview transcripts[1], the framework is formed by 128 sub-categories, clustered in 35 *groups*, and finally in 7 categories at the highest level of abstraction.

By the means of a conceptual framework we want to provide explanations of the development strategies and engineering activities undertaken by startups. The knowledge, which we attempted to pack into a conceptual model, can be used by researchers and practitioners to improve their understanding of the phenomenon. Thus, the activities and the strategies undertaken during the early-stage can be rationalized and designed acknowledging the surrounding contextual elements.

The theory is presented in this section is the form of an *experience map* [27], illustrating important challenges faced by practitioners in startups and fully answering to the research questions related to the state-of-practice (RQ-2). In particular, the answer to the sub-questions related to how startups consider quality attributes (RQ-2.2) is discussed in Subsection 6.3.4. The discussion on how practitioners structure and execute the main engineering activities (sub-question RQ-2.1) is spread across most of the subsections presenting and analyzing the model, summarized in *Implication* (see Section 6.5).

The remains of this section is structured in six parts: the description of the network formed by relations between the high-level categories (see Section 6.2), followed by the detailed explanation of the theoretical framework (see Section 6.3) and the generation of the theory (see Section 6.4). The implications of the model and the answers to RQs are discussed in Section 6.5. In Section 6.6 we perform a validation of the model by analysing it in comparison to: existing frameworks;

[1]The complete list of raw codes extracted from interview transcript is shown in Appendix A.4.3.

selected relevant studies; and empirical data. Finally in the last Section 6.7 of this chapter, we discuss the generalizability of results to similar domains.

6.2 High-level framework

As mentioned above, the main concepts representing underlying phenomena have been grouped together to form high-level categories. Figure 6.1 shows the network of causal relationships (represented by arrows) between categories (represented by blocks)[2].

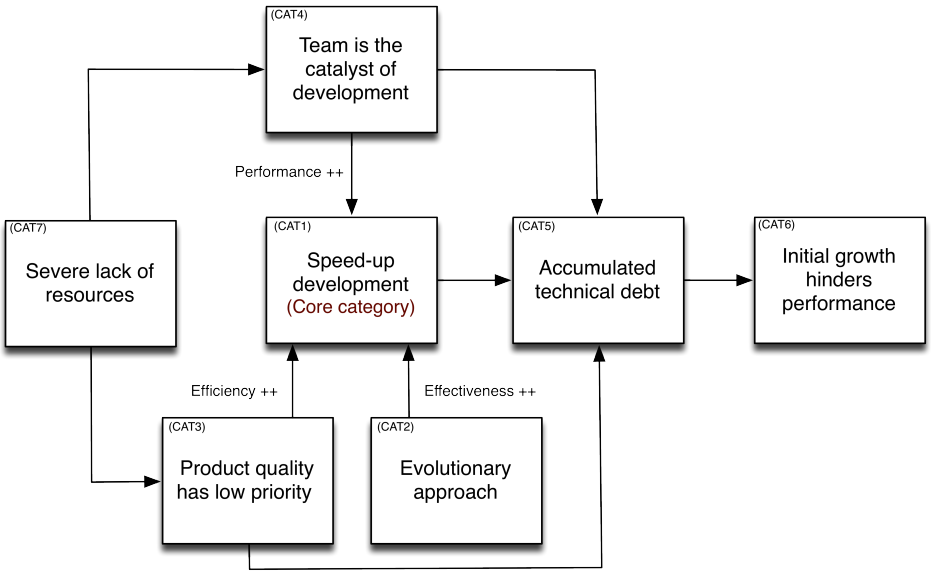


Figure 6.1: High-level theoretical framework

In the explanation of the framework in Figure 6.1, we emphasize labels of theoretical categories using the *italic* font style. The network is centered around the core category, *speed-up development*, which is the most interconnected node in the framework reflecting the fact that *"it is the one (category) with the greatest explanatory power"* [128]. The meaning of the labels on the three arrows that reach the core category (*efficiency*, *effectiveness* and *performance*) is discussed in detail in Subsection 6.6.4). The high-level framework, depicted in Figure 6.1 above, is presented starting from the left side block (*severe lack of resources*) and concluding on the right side (*initial growth hinders performance*).

A contextual condition which characterizes, to some extent, each and every startup, is the *severe lack of resources*. In fact, the capabilities of an early-stage

[2]Each node is linked to subsequent nodes which are called successors, and preceding nodes which are called predecessors. The relations between them are denoted by the direction of the interconnecting arrows. The network is read from left to right.

startups' to support its development activities are constrained by an extremely limited access to human, time and intellectual resources, eventually leading to a general absence of structures and processes.

The *severe lack of resources* forces the company to focus on implementing an essential set of functionalities and it is one of the main reasons why the *product quality has low priority*[3] respect to other more urgent needs[4]. At the same time, to be able to deal with such constraints, startups must depend on a small group of capable and motivated individuals.

As it has been unanimously expressed by respondents, from a software perspective, the most urgent priority is to *speed-up the development* as much as possible by adopting an extremely flexible and effective *evolutionary approach*. On the other hand the efficiency of teamwork is facilitated by the low attention initially given to architectural aspects related to product quality. This allows startups to have a faulty but functioning product, which can quickly be introduced to the market beginning with the in-house implementation of the prototype from day-one.

The initial employees are the ingredients which enable high levels of performances in software development. To support such a fast-paced production environment, engineers are required to be highly committed, co-located, multi-role, and self-organized. In other words *team is the catalyst of development*[5]. With an essential and flexible workflow, which relies on tacit knowledge instead of formal documentation, startups are able to achieve very short time-to-market. However, each line of code, written without following structures and processes, contributes to grow the *accumulated technical debt*[6], which is further increased by having almost non-existing specifications, a minimal project management and a severe lack of automated tests.

Despite the fact that the consequences of such debt are not clearly perceived in the initial stages (where finding the product/market fit as quickly as possible is the most important priority), startups, which survive to subsequent phases, will likely increment their user-base size, the product size, and the number of developers. This will require the company to eventually pay the *accumulated technical debt*, confronting the fact that an *initial growth hinders productivity*.

[3] To fully comprehend the actual phenomenon behind this theoretical category, it is highly suggested to read the discussion of its subcategories in the detailed framework (see Subsection 6.3.4).

[4] There are some exceptions where the quality aspects actually matter and such cases will be discussed later in Subsection 6.6.3.

[5] Observe in Figure 6.1 relationships between the core category and its three *ancestors*: the *team is the catalyst of development* contributes to performance, *product quality has low priority* contributes to efficiency and *Evolutionary approach* to effectiveness, as depicted in Figure 6.7 (see Subsection 6.6.4).

[6] As specified in Background (see Chapter 2) the term *technical debt* is *"a neologistic metaphor referring to the eventual consequences of poor or evolving software architecture and software development within a codebase"* [48].

6.3 Detailed framework

Moving from the high-level of Figure 6.1 to the detailed framework, we can focus on each group of subcategories describing the existing relationships as depicted in Figure 6.2.

In the network we used small arrows between subcategories to represent a strong causal relationship among them, where relationships were grounded directly in the interview transcripts. Specifically, small arrows represent a direct correspondence of cause-effect relation between subcategories, whilst the absence of a small arrow indicates that the correspondence with other subcategories is indirectly created through the relation between higher level categories (i.e. bold arrows in Figure 6.2). In this regard, an example is the subcategory *working overtime to meet deadlines* that doesn't have a strong relation with any other subcategories in CAT5: the relation exists only when it is wrapped into the concept of CAT1. In fact, when it is combined with other subcategories through the axial coding process, the subsequent process of selective coding creates an indirect relation between the two high-level main categories (i.e. CAT1 $\rightarrow$ CAT5)[7].

In order to structure the description of the detailed framework, each category is presented in a different subsection, which begins with the list of subcategories belonging to it. To further support the description of the framework, we make use of quotations extracted from transcripts, which refer to the interviewed company along with their identifier[8].

[7]Coding processes are explained in Subsection 4.3.4. For the sake of brevity in this chapter we don't describe all the small arrow relations. However, we made use of self-explanatory labels to let the reader understand the causal relations only by reading the codes.

[8]Identifiers are assigned to companies in Table 5.10.

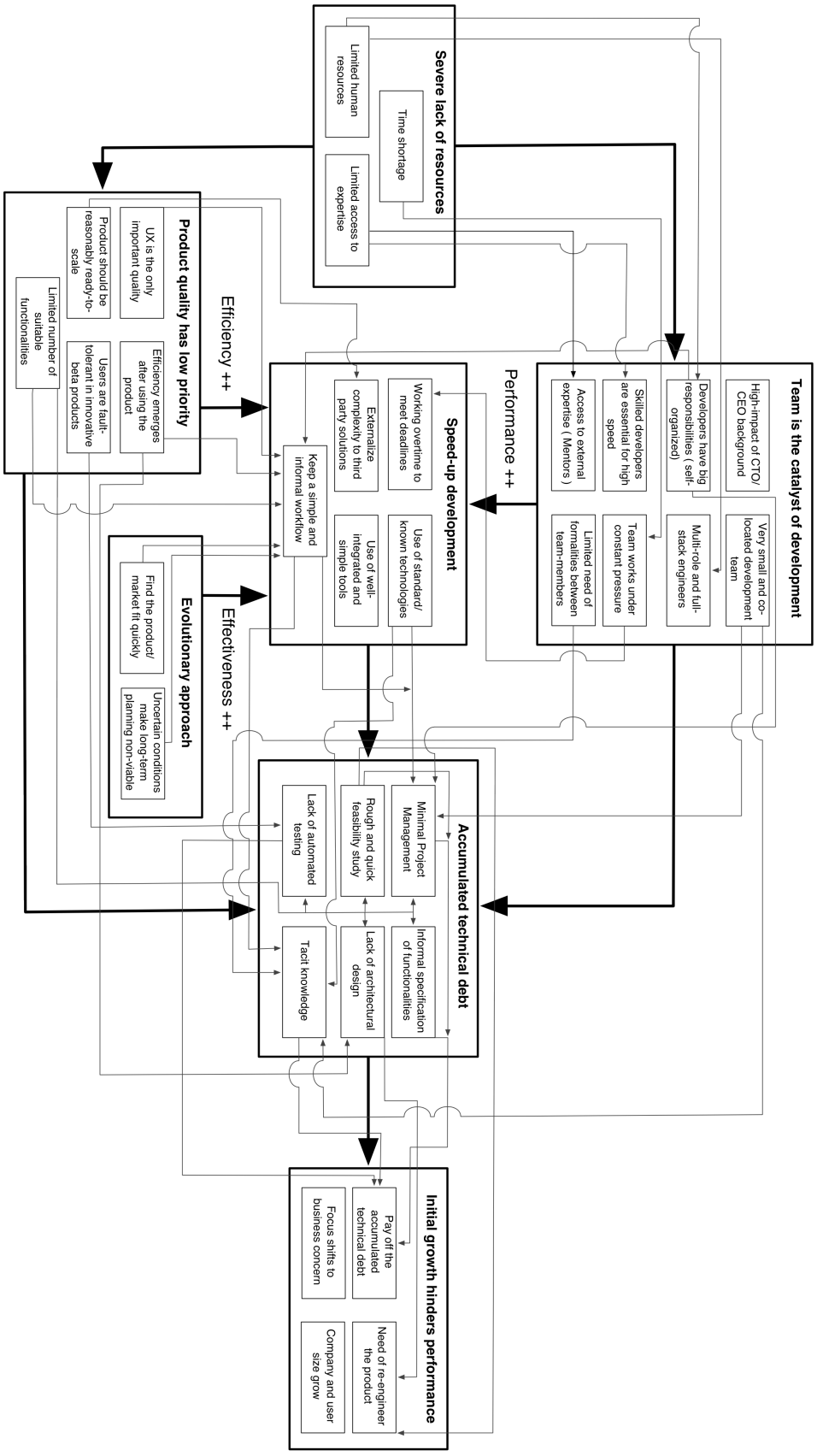


Figure 6.2: Detailed theoretical framework

6.3.1 Severe lack of resources (CAT7)

The first theoretical category and its subcategories are discussed in this subsection. The concept of *severe lack of resources* characterizes the uncertainty of the development strategies in startups and it is composed by three subcategories: *time-shortage*, *limited human resources* and *limited access to expertise*.

Subcategories:

- Time shortage:
  - Investor(s) pressure.
  - Business pressure.
  - Internal final deadline.
  - Demo presentations at events.
- Limited human resources.
- Limited access to expertise.

Since startups want to bring the product to market as quickly as possible, the resource they are the most deprived of is the *time*. Startups operate under a constant time pressure, mainly generated by external sources (*investor(s) pressure*, *business pressure*) and sometimes internal necessities such as *internal deadlines* and *demo presentations at events*. In this regard, C3 commented: *"Investors wanted to see product features, engineers wanted to make them better. Finally the time-to-market was considered more important and the team interests was somehow sacrificed."*

In addition, to compensate the *limited human resources*, practitioners need to empower *multi-role and full stack engineers*, as confirmed by C1: *"Everyone was involved in any tasks, from mobile to web development, organizing themselves in choosing the part to implement"*.

The extent to which startups have access to specialized knowledge - both internal and external to the company - is extremely reduced when compared to traditional established software companies. Therefore, in order to partially mitigate the *limited access to expertise*, startups rely on the external aid of mentors or advisors. Under these strict limitations, most of the decisions related to software development are fundamentally trade-off situations (confirming Himola's results [68]).

6.3.2 Team is the catalyst of development (CAT4)

Among the different aspects fostering the speed of the development process, the startups' main focus is on the characteristics of the initial team. This category comprehends the following subcategories:

Subcategories:

- Developers have big responsibilities (self-organized).
- Team works under constant pressure.
- Very small and co-located development team.
- Limited need of formalities between team-members:
  - Positive impact of high co-location.
  - Previous working experience.
  - Knowing each other before starting the company.
- Multi-role and full-stack engineers:
  - Engineers are responsible for marketing/sales/development (flat structure).
  - Generalists developers instead of specialists (full-stack).
- Skilled developers are essential for high speed.
- High-impact of CTO/CEO background.
- Access to external expertise (mentors).

In startups *developers have big responsibilities*. In fact, the *limited human resources*, discussed in CAT7, causes the team-members to be active in every aspect of the development process, from the definition of functionalities to the final deployment.

Early engineers in the founding team of a startups are typically *multi-role and full-stack engineers*: they are full-stack for being able to tackle different problems at different level of the technology stack (*generalists developers instead of specialists*) and multi-role since usually *engineers are responsible for marketing/sales/development*. As remarked by C11: *"Instead of hiring gurus in one technology, startups should hire young developers, generalists, that know how to quickly learn new technologies, and quickly move among a huge variety of tasks"*.

Moreover, having a *very small and co-located development team*, enables members to operate with high coordination, control and communication, which heavily rely on *tacit knowledge*, replacing most of the documentation with informal discussions. Practitioners reported that keeping the development team small helps startups in being fast and flexible, as remarked by C8: *"If you have more than 10 people, it is absolutely impossible to be fast"*. Then, also *previous working experience* and *knowing each other before starting the company* enforce the efficiency of activities by *limiting th need of formalities between team-members*.

In every software company, *skilled developers are essential for high speed* in development. Especially in startups, the "hacking culture" and a tendency to the "just-do-it" approach allow the team to quickly move from the formulation of a feature idea to its implementation. In this regard, C1 comments: *"we had a hacker culture/environment, people hacking stuff without formally analyzing it, but breaking it down and finding a way around. Ticket on trello and go."*

A *limited access to expertise* forces the team to rely mainly on their personal abilities, even though asking opinions to mentors is reported to be a viable practice to maintain feasible objectives. Furthermore *teams work under constant pressure*, *"up to 16 hours per day"*, mainly constrained by a tight *time shortage*.

Finally, startups present founders-centric structures, and especially in the early-stage, it is clear that *CTO/CEO background has high-impact* on the company's development approach. For instance, in case of an academic background, the CTO encourages the introduction of some architectural design before the development phase. Moreover, even though the CTO/CEO starts guiding initially the development process, most of the decision are taken consensually by all members of the team. Then, the CTO/CEO decides only in particular situations where some conflicts occur.

6.3.3 Evolutionary approach (CAT2)

Among different approaches to development, startups prefer to build an initial prototype and iteratively refine it over time, similarly to the concept of *"evolutionary prototyping"*[9]. The reason of using this approach is that startups want to validate the product against the market as soon as possible, finding the proper product/market fit. Indeed, they can focus on developing only parts of the system they want to validate instead of working on developing a whole new system. Then, as the prototype is released, users detect opportunities for new functionalities and improvements, providing their feedbacks to developers.

Subcategories:

- Find the product/market fit quickly:
  - Flexibility and reactiveness are the main priorities.
  - Build a functioning prototype and iterate on it (minimum viable product).
  - Progressively roll-out to larger number of people.
  - Small iterations (release frequently).
  - Validate the product:
    - Find what is valuable for customers.
    - Direct contact and observation of users.
    - Automated feedback collection.
    - Analysis of product metrics.
- Uncertain conditions make long-term planning not viable:
  - Changes in technologies.
  - Dynamic competition landscape.
  - User requests.
  - Changes in market.
  - Changes in team.

[9]Evolutionary prototyping is a software development approach, which aim is to build a robust prototype to form the core of a new system, to further add improvements and new requirements.

Since *flexibility and reactiveness are the main priorities*, the most suitable class of software development approaches are clearly highly evolutionary in nature. In fact, as *uncertain conditions make long-term planning not viable*, startups cannot base their work on assumptions without rapidly validating them, releasing the product to market. The uncertainty is first of all in the team composition. In fact, since the teams are typically very-small and the project knowledge is mostly not written, even a little change in their composition (e.g. a developer falls in ill) can have a very high impact on the overall product development. Furthermore, startups operate in a continuous evolving environment in terms of competitors and targeted market sectors. Then, to obtain a competitive advantage in the market, startups typically make use of cutting-edge solutions which evolution cannot be foreseen in the long run. However, a special role in the day-by-day managerial decisions is covered by user feedbacks and requests, which represent the main drivers of what product features should be provided in the short-term.

Indeed, to obtain fast user responses and quickly *validate the product*, startups *build a functioning prototype and iterate on it* over time. In fact, paraphrasing C4, *"[. . .] you should start with something that is really rough and then polish it, fix it and iterate. We were under constant pressure. The aim was to understand as soon as possible the product market/fit iterating quickly, adjusting the product and releasing fast."*

The company focus on building a small set of functionalities to include in the first version, and *progressively rolls-out to larger number of people* with very *small iterations* (confirmed again by C4: *"we deploy from 5 to 20 times a day"*).

The scope of this evolutionary approach is to avoid wasting time in "over-engineering the system" and building complex functionalities that have not been tested on real users. By releasing a very small number of good-enough functionalities (see CAT3) the startup can verify the suitability of the features and understand how to adjust the direction of product development towards actual users' needs. The first version of the product is typically a prototype which contains the basic rough functionalities developed with the least possible effort that can validate critical features which can undermine the startup's survival in the short term (recalling Ries' definition of *Minimum Viable Product*).

Thanks to a *direct contact and observation of users*, *automated feedback collection* and *analysis of product metrics*[10], startups attempt to *find what is valuable for customers* (these activities are in support of Blank's *Customer validation* stage [17]).

Observe that we grouped these subcategories together under the category *find the product/market fit quickly*, which contains several concepts similar to the *Lean*

[10]The product metrics mainly refer to the UX of the product to find any kind of frictions that might reduce the satisfaction level of the final users. Basic metrics are traced automatically by means of tools (e.g. *CrazyEgg*) or embedded scripts to track users' behaviors.

startup methodology pioneered by Eric Ries [7]. However, especially in the early stage, the above mentioned principles are only part of the complete methodology[11] (hence, we refer to them as *light* principles).

6.3.4 Product quality has low priority (CAT3)

The main interests of software startups, related to the product, are mainly concentrated on building a *limited number of suitable functionalities* rather than fulfilling restrictive non-functional requirements. This strategy allows them to quickly release simple products with less need of preliminary architectural studies (see RQ-2.2).

Subcategories:

- Limited number of suitable functionalities.
- UX is the only important quality:
  - Ease of use (operability).
  - Attractiveness UI.
  - Smooth user-flow without interruptions.
- Efficiency emerges after using the product.
- Product should be reasonably ready-to-scale.
- Users are fault-tolerant in innovative beta products.

Exploring quality aspects considered during the development process, the only important concerns are expressed in favour of the user experience[12], in terms of *ease of use, attractiveness of the UI* and most of all, *smooth user-flow without interruptions*. The fact that *UX is the only important quality* is remarked by C11: *"When a user needs to think too much on what action should be done next, he will just close the application without returning"* and confirmed by C3: *"If the product works, but it is not usable, it doesn't work"*.

As discussed in Subsection 6.6.3, the extent to which quality aspects are taken into account might heavily depend upon the market sector and the type of application. Nevertheless, realizing high level of UX is often the most important attribute to consider for customer discovery of evolutionary approaches in view of

[11] Lean methodologies on theory are well coupled with these kind of approaches (see Section 7.2 for a comparison between existing methodologies and the one we identified in early-stage startups).

[12] According to the ISO 9241-210 (Ergonomics of human-system interaction), UX is defined as *"a person's perceptions and responses that result from the use or anticipated use of a product, system or service"*. Even though this definition leaves so much to interpretation, we can refer to it as presented in the glossary.

the *limited human resources* and *time shortage*, presented in CAT7. C4 confirms: *"None of the quality aspects matter that much as the development speed does."*

To achieve a good level of UX while dealing with lack of human resources and time shortages, startups can analyze similar products of bigger companies which can afford more rigorous usability studies. Then, the users' feedback and product metrics[13] begins to have a central role in determining the level of UX achieved.

Other than UX, some other factors can influence the quality concerns of development:

- The *efficiency emerges after using the product*, letting engineers avoid wasting time in excessive improvements of not-validated functionalities. In fact, the level of efficiency can be optimized after attesting the effectiveness of the minimal set of functionalities, obtained according to CAT2 and to the concept of MVP.
- The *product should be reasonably ready-to-scale* to be able to accommodate a potential growth of the user-base[14]. Thanks to modern cloud services, startups can *externalize complexity to third party solutions* achieving a good level of scalability with a reasonable effort.
- Realizing high reliability is not an urgent priority since *users are fault-tolerant in innovative beta products*. In these cases, users have typically a positive attitude towards the product, even though it presents unreliable behaviours. In this regard, the main focus of beta testing is reducing frictions between the product and the users, often incorporating usability testing. In fact, the beta release is typically the first time that the software is available outside of the organization that developed it. [15].

Finally, startups implement a *limited number of suitable functionalities*, which though limited by reasons of *limited human resources* and *time shortage*, represent a viable strategy to focus on shortening time-to-market.

6.3.5 Speed-up development (CAT1)

As remarked above in the description of the high-level model, *speed-up development* represents the core category of our theoretical framework. Firmly grounded as the primary objective of startups, it shows the most important characteristic of developing software in the early stage.

[13]Product metrics are typically web-based *statistical hypothesis testing*, such as A/B testing. Moreover, basic metrics are traced automatically by means of tools (e.g. *CrazyEgg*) or embedded scripts to track users' behaviors.

[14]Note that startups tackle fast growing markets which are particularly subject to sudden user growths

[15]Detailed discussion of the impact of innovative products on the user satisfaction is presented in Subsection 6.6.3 according to the Kano model.

Subcategories:

- Keep simple and informal workflow:
  - Informal and frequent discussion to take quick decisions.
- Externalize complexity to third party solutions:
  - Deploy on cloud service infrastructure.
  - Use external services (SaaS).
  - When possible adopt COTS.
- Use of well-integrated and simple tools:
  - Ticket-based tools to manage stories/features:
    - Partial use of online and collaborative tools.
    - Offline low-precision tools (whiteboard, post-it notes, ...).
  - Collaborative version control systems to manage the codebase.
  - Automatic tools for collecting/report user feedbacks.
  - Automatic tools for deployment on the infrastructure.
  - Excess of non-integrated tools overloads the team.
- Working overtime to meet deadlines:
  - Create disruptive technologies.
  - Demonstrate personal abilities.
  - Be able to meet deadlines.
  - Have the product used in the market.
  - Business success (increased ROI).
  - Media coverage generates hype.
- Use of standard/known technologies:
  - Past experience with same technology.
  - Strong community.
  - Easier to hire new developers.
  - Availability of documentation.

As mentioned before, in order to *speed-up development*, startups adopt evolutionary approaches supported by a solid team focusing on implementing a minimal set of suitable functionalities. Startups *keep simple and informal workflows* to be flexible and reactive, adapting to the fast changing environment. While a rigorously defined workflow generally consists of an established sequence of well-defined steps to follow, startups, by contrast, adopt informal workflows. This choice is mainly dictated by contextual factors characterized by the general conditions of unpredictability and uncertainty. The adoption of informal workflows is facilitated by the fact that teams are typically self-organized and *developers have big responsibilities*.

Additionally possible planning activities are refrained by the the aim to shorten time-to-market, as reported by C8: *"Speed was the essence so we didn't plan out too many details"*. To deal with such unpredictability, startups prefer to take decisions as fast as possible, mainly by means of informal and frequent verbal discussions.

Despite Agile principles appear to be suitable to embrace changes, especially in the early-stage, development practices are often perceived as time-wasters and ignored to accommodate the need of releasing the product to market as quickly as possible. In this regard, to maintain the simplest and most informal approach, startups aim to *find the product/market fit quickly* and implement a *limited number of suitable functionalities*. This approach is considered possible also in view of a lack of systematic quality assurance activities, since only the user experience is considered as important and other quality aspects, such as efficiency, can be postponed after the first release.

Another strategy, which supports startups in quickly delivering products, is the *externalization of complexity on third party solutions* by: making use of third party components (COTS) and open source solutions (both for product components either development tools and libraries); taking advantage of external services; and deploying the product on external infrastructures, for the sake of presenting a *product reasonably ready to scale* for a possible future growth[16].

Even though *the use of well-integrated and simple tools* allows startups to automate many activities and reduce their completion time, drawbacks of such approaches are the increased concerns of interoperability issues. A category of tools, worth , is represented by advanced version control systems, which are not only used to manage the code-base, but for many other purposes such as: assigning tasks; trace responsibilities; configuration management; issue management; automatic deployment; discussions and code reviews.

Startups can further improve development speed by making *use of standards and known technologies* which are widely recognized, well tested, and supported by strong communities. Moreover, the use of widely adopted standards and frameworks reduces the need of a formal architectural design since most of implementation solutions are well documented and ready-to-use. This is confirmed by C1: *"as long as you use Ruby standards with the Rails framework, the language is clean itself and doesn't need much documentation"*.

Other important factors that positively impact the speed of development are the team's desire to: *create disruptive technologies*; to *demonstrate personal abilities*; and to *have the product used in the market*. As reported by practitioners, these factors are essential to enhance the morale of developers and therefore to achieve higher team performances. By contrast, especially in the typical sprint-based environments of Agile, when a developer is not *able to meet deadlines* the morale goes down, hindering the development speed.

Finally the constant pressure under which the company regularly operates, leads the team to often *work overtime to meet deadlines*. But as reported by practitioners, such way of working can be an effective strategy only in the short

[16] A good initial degree of scalability, at least infrastructural, can be reached with modest costs and time. However, this is not a necessary condition for realizing a functioning MVP, but only a recommendation.

term since can lead to poorly maintainable code and developers burnouts in the long run.

6.3.6 Accumulated technical debt (CAT5)

Startups achieve a very high development speed by radically ignoring aspects related to documentation, structures and processes. However, although these aspects are not considered important in the very first stages, they will become increasingly more important for the productivity in the long-term, as we illustrate in CAT6.

Subcategories:

- Informal specification of functionalities:
  - Informal definition of business objectives.
  - Low-precision list of features to implement initially.
  - Self-explanatory user stories.
  - Features prioritization based on personal experience.
- Rough and quick feasibility study:
  - Lack of formal Analysis:
    - Exploring alternative solutions to mitigate limited knowledge.
    - Hard to analyze risks with cutting-edge technologies.
    - Learn from competitors' solutions and mistakes.
    - Past experience in similar contexts helps assess feasibility.
  - Keeping the product as simple as possible.
  - Naive estimations based on personal experience.
- Lack of architectural design:
  - Only high-level mockups and low-precision diagrams.
  - Maintaining architecture only when strictly necessary.
  - Using modularization and frameworks from day 1.
  - CTO's academic background lead to a first attempt of architectural design.
  - Describing critical interactions with third-party components only.
- Lack of automated testing:
  - Automatic and partial testing of critical parts only.
  - Let users report non-critical bugs.
  - Major bugs emerge by trying the product internally.
- Minimal Project Management:
  - Uncertainty makes formal scheduling pointless:
    - Only a final release milestones is viable.
    - Keep internal milestones short and informal.
  - Naive metric to trace project progress.
  - Naive task assignment mechanism.
  - Ticket-based systems to control to-do lists.
  - No dedicated person for project management.
  - Simplify issue management by integrating it with feature management.
- Tacit Knowledge:
  - No need of recording taken decisions (informal discussions).
  - Only minimal and not updated initial documentation.

– Clear and self-explanatory code (standards).
– Documentation external to source-code perceived as waste.

Instead of traditional requirement engineering activities, startups make use of *informal specification of functionalities* by means of ticked-based tools to manage low-precision lists of features to implement, written in form of self-explanatory user stories[17]. Practitioners intensively use physical tools such as post-it notes and whiteboards, which help in making functionalities visible and prioritizing stories based on personal experiences. C4 comments *"[. . .] it is the only way. Too many people make the mistake of sitting down and write big specs and then they build it for four months, realizing the product is not valuable only at the end."*

Since startups are risky businesses by nature, even less attention is given to the traditional phase of *analysis*, which is replaced by a *rough and quick feasibility study "only sometimes and however informally"*, as stated by C5. First of all it is generally *hard to analyze risks with cutting-edge technologies*. To find out the feasibility of such cutting-edge projects, startups attempt a first implementation with rough and informal specifications, assuming that project's complexity will remain limited to a small number of functionalities, as discussed in CAT3. Additionally by *keeping the product as simple as possible* and learning from competitors' solutions and mistakes, practitioners can use their *past experiences in similar contexts to help assessing feasibility* of the project. Finally, to avoid restrictions on the flexibility of the team, potential limiting decisions are taken only when strictly necessary and anyhow as late as possible.

Another important factor which contributes to *accumulate the technical debt* is the general *lack of architectural design*, substituted by *high-level mockups and low-precision diagrams*, *using modularization and frameworks from day 1* and *describing critical interactions with third-party components only*. In particular, the use of well-known standards, frameworks and conventions limits the need of formal UML[18] diagrams and documentation, and provide a minimum level of maintainability. Additionally, since quality aspects are not a main concern (see *limited number of functionalities* and efficiency emerges after using the product in CAT3) having a well-structured architecture remains a secondary priority, and design is conducted only when strictly necessary.

A similar attitude towards verification and validation brings startups to a severe *lack of automated testing*, which is mainly replaced by manual smoke tests executed by: trying the product internally; seeking for critical faults; and letting

[17]User stories are very high-level definitions of the project requirements, written by the users' point of view. They contain just enough information so that the developers can produce a reasonable estimate of the effort to implement them.

[18]Unified Modeling Language (UML) is a standard to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.

early adopters report non-critical bugs. Paraphrasing C3, *"Trying the product internally allows us to get rid of 50% of bugs of important functionalities. Meanwhile users report bugs of secondary functionalities, eventually allowing us to mitigate the lack of testing. Indeed, staying one week in production enables us to identify 90% of bugs"*. The lack of complete automated tests is partially also motivated by the fact that *users are fault-tolerant in innovative beta products* and by the limited number of functionalities, which allow the team to easily control critical bugs. However, in certain cases where components of the system might cause loss of data or severe damages to the product or users, engineers realize a reasonable level of automatic testing. In such cases, aided by modern automatic tools, they can quickly assess the status of the system integration as they add new functionalities to the product.

Furthermore, a rigid project management is perceived as *"waste of time"* which hinders the development speed since the *uncertainty makes formal scheduling pointless* (C9 reports that *"initial chaos helps to develop faster"*). Startups' *minimal project management* is supported by keeping: *internal milestones short and informal*, *low-precision task assignment mechanism* and a very low attention for project metrics (paraphrasing C13, *"the only track of progress was made by looking at closed tickets"*). In this context *only a final release milestone is viable*, which helps practitioners to remain focused on short term goals and put new features in production. To further *speed-up development*, startups *simplify issue management by integrating it with feature management*. Then, the *limited number of functionalities* and the use of standard/known technologies with a *simple workflow* are the main reasons for not establishing a heavy project management process since the project and technologies can be autonomously managed by the development team (with characteristics of: co-location, self-organization and very small size).

Finally, one of the categories, which most contributes in growing the *accumulated technical debt*, is the substantial use of informal and verbal communication channel on daily basis. The high co-location and the fast paced development approach increase the volume of the *tacit knowledge* and the consequent severe lack of any kind of documentation. In fact, C4 observes in this regard that: *"[...] the issue of having documentation and diagrams out of the source code is that you need to update them every time you change something. There is no time for it. Instead, there is a huge pay off in having a code that is understandable itself."* This approach is supported by the fact that startups make use of *simple and informal workflow*, *standard/known technologies*, *very small and co-located development team* with *limited need of formalities*.

6.3.7 Initial growth hinders productivity (CAT6)

The lack of attention given in the first phases to engineering activities allows startups to ship code extremely quickly (see CAT5). However, if successful, the

initial product becomes more and more complex overtime, the number of user increases and the company size starts to grow. Under these circumstances the need of controlling the initial chaos forces the development team to return the *accumulated technical debt* instead of focusing on new users' requests. Hence, *initial growth hinders performance*[19].

Subcategories:

- Company and user size grow:
  - Business event:
    - New round of funding.
    - Acquisition.
    - Competing product released to market.
  - Open the first public release.
  - Increasing number of users.
  - Current team not able to manage increased complexity.
  - Hiring new staff members.
- Focus shifts to business concerns.
- Need of re-engineer the product:
  - Portion of codes needs to be rewritten.
  - Substantial refactoring of the codebase.
  - Increase scalability of the product/infrastructure.
  - Standardize codebase with well-known frameworks.
- Pay off the accumulated technical debt:
  - Fear of changing a product which is working.
  - Growing necessity of having a release plan.
  - Need of control initial chaos with more structured workflow:
    - Partially replace informal communication with traceable systems.
    - Introduce basic metrics for measuring project and team progress.

When the number of users increases, customers become more quality demanding and also scalability issues start to arise.

Usually, *company and user size grow* when new business events occur, such as: a *new round of funding*, a possible *acquisition*, the release of a *competing product on the market*, or when the project is *open for the first public release*. As a consequence, the *increasing number of users* creates growing number of requests and expectations. Therefore, whereas the project lacks of minimal processes, suddenly, *the current team is not able to manage increased complexity* of new functionalities to implement and maintain the codebase.

Subsequently, practitioners start considering the need of project management activities, also in view of *hiring new staff members*, as discussed by C13: *"(project*

[19]Performance is defined in terms of number of new user stories, which brings new suitable functionalities to the users, implemented in a certain amount of time.

management) is strictly necessary if you radically change the team or when team grows. The informal communication and lack of documentation slow down the process afterwards". Project management becomes further important when the *focus shifts to business concerns*. In fact part of the effort, which was initially almost entirely dedicated to product development, is required to be moved to the business. Moreover the availability of project information becomes an important issue because of the accumulated *tacit knowledge*, which hinders the ability of new hires to quickly start the development of project tasks.

When the company faces growth, the partial and informal engineering activities that have been conducted during the first phases of *rush* development (refer to CAT5 subcategories *minimal project management*, *informal specification of functionalities*, *rough and quick feasibility study*, *lack of automated testing*, *tacit knowledge*), force the company to *pay off the accumulated technical debt*. In fact, under these circumstances, startups need to be able to cope with the renewed needs and expectations of both the company (internal necessities) and customers (product).

Another factor that slows down performance is caused by the fact that *portion of codes needs to be rewritten* and *substantial refactoring of the codebase* is required by the increasing product's demand.

In fact practitioners realize that some decisions taken (or not taken) during the *rough and quick feasibility study* before starting the implementation, have led to negative consequences on the long term performance and maintainability of the product. Additionally, the quick product design decision initially taken, might not be able to satisfy the increased demands of the product's users and developers (*lack of architectural design*). The combination of these factors leads to the need of *re-engineer the product*.

By re-engineering the systems, startups aim to *increase the scalability of the product/infrastructure* and starting to *standardize the codebase with well-known frameworks*.

C7 reports that: *"To mitigate this (lack of frameworks) I had to make a schema for other developers when we hired them. We had to do a big refactoring of the codebase, moving it from custom php to Django, normalizing the model and making it stick with the business strategy. I had the code in different php server communicating via JSON, some engineering horror. Now that we are fixing it, it's really painful. We had to trash some code. However I don't regret that I didn't make this choice sooner, it was the only way"*.

The *fear of changing a product, which is working*, arises when product complexity increases. In fact, the changes to the codebase, to support bug fixing, become highly interrelated with other functionalities and difficult to manage because the product is poorly engineered (see CAT5). Therefore, the fear arises from changing a validated product that might bring changes to the user responses.

The increasing number of feature requests causes the *growing necessity of having a release plan*, rationalizing the rollout of new features.

Therefore, startups begin to *partially replace informal communication with traceable systems* and *introduce basic metrics for measuring project and team progress* in order to establish an initial structured workflow. In conclusion, because the increasing number of users causes issues in scalability and reliability, the *need of more structured workflow* becomes of major relevance. C11 states: *"Yet, it is still better to have a reasonable drop-down in performance when team grows than lose time in the beginning"*.

In Chapter 7 we dedicated Section 7.5 to discuss the evolution of startups.

6.4 Theory generation

In order to explain and understand the development strategies in early-stage software startups we construct the theory generated and supported by the theoretical framework presented above. As discussed in Subsection 4.3.5, we applied the *paradigm model* to form the final theory, here presented:

Focusing on limited number of suitable functionalities, and adopting partial and rapid evolutionary development approaches, early-stage software startups operate at high development speed, aided by skilled and highly co-located developers. Through these development strategies, early-stage software startups aim to find early a product/market fit within extremely uncertain conditions and severe lack of resources. Nevertheless, by speeding-up the development process, they accumulate technical debt, causing an initial and temporary drop-down in performance before setting off for further growth.

The theory above is composed by different elements reflecting the *paradigm model* shown in Figure 4.19. Here we structure the list of elements presented in the theory:

- *Causal conditions* are represented by three main conceptual categories: *product quality has low priority*, *evolutionary approach* and *team is the catalyst of development*.
- the *Phenomenon* is represented by the core category *speed-up development*.
- the *Context* is limited to web early-stage software startups operating in conditions of severe lack of resources aiming to early find product/market fit.
- *Intervening conditions* are summarized by the extremely uncertain development environment[20].

[20]Refer to subcategory *uncertain conditions* in Subsection 6.3.3 in terms of business and technology management, and companies whose main development objective is shortening time-to-market. Limitations of the theory are further discussed in Section 8.5.

- *Action and interaction strategies* are represented by the accumulation of technical debt (see Subsection 6.3.6).
- *Consequences* lead to a temporary performance drop-down (see Subsection 6.3.7).

6.5 Theory implications (RQ-2)

In this subsection we present a list of the most relevant implications which emerge from the behaviour of early-stage startups, formally expressed in the model above, answering to RQ-2 (*What is the current state-of-practice related to software development strategies in early-stage startups?*) through its sub-questions. Although the startups we inquired in the case study were spread across different nations and market sectors, we have found several patterns that emerged very clearly:

- **The most urgent priority of software development is to shorten time-to-market.** When developing a new product, which has not been attempted before, it is crucial for the company to release the product soon and observe the users reactions. The company's chances to survive, are strictly related to the ability to find the right product/market fit as quickly as possible. Developing software without releasing it for a long period of time, coming up with very complex solutions often is not a viable option. Business demands, investors interests, market pressures and the general uncertain conditions force startups to quickly iterate on the product by exploring and experimenting new functionalities progressively rolled-out to early-adopters.
- **Startups do not apply any standard development methodology.** When focused on building the first version of the product, startups do not observe any specific and standard development methodologies or processes[21]. From an accurate and in-depth investigation into day-by-day activities, it emerged that the number of the observed development *practices* is extremely small and never systematic (only little *pair programming* sessions and rarely *test driven development*). Despite the wide number of publications, which discuss the adoption of lightweight methodologies, we have found that especially in this first stage not only methodologies but even individual practices are basically neglected.
- **The closest development approach undertaken by early-stage startups is the *Lean startup methodology*.** A highly evolutionary development approach, centered around the quick production of a functioning

[21] Is worth noticing however that when asked the first question *"Did you use any specific development methodology?"* 10 startups out of 13 referred to some kind of customized light version of SCRUM or XP (see Subsection 5.2.4 for more details).

prototype and guided by customer feedbacks is a central aspect of the *Lean startup methodology* pioneered by Ries [7]. However, startups do not formally follow the cycle *build-measure-learn* as it is proposed by the methodology, where they should set-up an experiment to *test the riskiest element of the business idea*, nor make explicit use of the good practices suggested in the book. For example, when it comes to user feedbacks, startups do not explicitly follow the step-by-step process of *customer development* formally defined by [17]. Instead, startups absorb and implement the high-level principles that outlines the methodology, reflected in the model by the theoretical category *find the product/market fit quickly* and its subcategories.

- **The greatest part of engineering activities of startups are focused on the implementation while only little attention is given to more conventional activities (project management, requirement specifications, analysis, architecture design, automatic testing, ...).** Especially when bringing the first product to the market, the attitude of startups towards traditional engineering activities is strongly characterized by a general feeling of unnecessity. Every document, artifact or process, which does not produce a short-term tangible benefit to the software production is perceived as a waste of time. The most visible downside of this kind of approach is a growing *accumulated technical debt* (Subsection 6.3.6), which is partially mitigated by different elements supporting the development. Answering RQ-2.1, project management is substituted by the use of informal short-term deadlines supported by online collaborative tools with a kanban-board-like fashion; feature list are managed through ticket based systems supporting low-precision lists of requirement specifications in the form of user stories; after the essential set of functionality have been defined, making estimations and prioritization of features is not of much help; the traditional analysis process is, to a great extent, neglected (analysis is informally conducted for specific critical functionalities, usually by studying similar software solutions and rarely documenting decision-making processes); the design of the architecture is usually replaced by the use of well-known frameworks[22] and naive diagrams on the whiteboard; advanced and modern version control systems work well for the configuration management; high-volumes of informal communication and the use of coding standards reduce the need of formal documentation; in the majority of startups writing automated test cases is perceived as a waste too (since the product reliability is not a primary concern, engineers identify major bugs by testing the product internally and letting early adopters identify secondary bugs); finally,

[22]The usage of well-known frameworks has been recognized by practitioners to provide many benefits with a relatively short learning time: frameworks have limited need of documentation, facilitate hiring, simplify the architecture, improve interoperability and are supported by active online communities. These elements are very important for startups dealing with sever shortage of resources.

the infrastructural complexity of deployment is typically left to third party cloud applications[23]. Thus, by radically ignoring activities related to documentation, structures and processes, startups achieve a very high initial development speed in the first stage, but on the other hand they accumulate a technical debt that needs to be at least partially fulfilled when the company and the product grow.

- **The first release of the product includes only a limited set of well suitable functionalities focused on user experience.** The beta version is usually focused on a minimal and essential set of functionalities providing a reasonable user experience (UX). As confirmed by practitioners, the user must be able to accomplish a task on the product without major interruptions in the flow, which would cause frustration before being able to capture his attention. To realize UX in a short time frame, developers focus on a limited number of suitable functionalities, especially in the first version(s) of the product. Other product qualities are regarded as less important[24]: for instance, obtaining a high-reliability is not a priority since early adopters of innovative product are keen to accept minor bugs in exchange of new functionalities[25]. Furthermore, thanks to cloud computing and more in general high availability of third party components, aspects such as performance, security and scalability can be achieved with a reasonable effort. Thus, given the general lack of quality concerns[26], and supported by the fact that the optimal strategy to improve the UX is by having the product used by real customers, startups can effectively focus on achieving short time-to-market (see RQ-2.2).

[23]Note that in this thesis the described development activities are directly derived from the case study as result of positive experiences. On the contrary, the conduction of "traditional" engineering practices was reported as a waste (examples of past experiences in conducting architectural design and extensive testing have been reported by the interviewees as useless in order to find an early product/market fit, concept further discussed in Chapter 7). Indeed, only the development strategies described by the theoretical framework (see CAT2, CAT3 and CAT4 in Section 6) resulted as most important factors to enhance the development speed.

[24]Note that from a business perspective, one might argue that established companies might take advantage of the new idea and develop a better version of the product internally. Despite this might represent a risk, established companies usually cannot afford to operate in a market of *innovators* and *early adopters* group of users - described by the Roger's bell curve in [168] - because of the little return of investment they would gain, as discussed by Christensen in [169]. Instead, established companies have more interests in acquiring a startup and all the knowledge it has gained during the *customer development* process (see Appendix A.2.5 for more details) while focusing effort on acquiring larger shares of a "mass market" sector.

[25]These results further highlight differences between software development in startups and traditional web engineering, where quality aspects are rated as very important, even more than time-to-market [170].

[26]The extent to which quality aspects are considered by the company, is strongly influenced by the market sector and the application type, as discussed in Subsection 6.6.3. However our case study has been carried out in startups working on innovative web application with very few quality concerns.

- **Engineering activities are supported by low-precision artifacts and collaborative tools integrated with the development environment.** Startups take advantage of the simplicity of cutting-edge tools, which require a minimum learning and maintenance effort and provide a large payoff. Tools are used for a vast number of tasks: ticked-based boards to manage user stories; systems to collect user feedback; tools for tracing product metrics; and scripts for managing deployments. The most prominent families of tools, which support software development in startups are modern version control systems. They are used by practitioners not only to manage the code base and the repositories, but for many other purposes such as: task assignment; tracing responsibilities; configuration management; issue management; automatic deployment; discussions and code reviews. On the other hand, an excessive usage of *out-of-the-code* tools can negatively affect the development speed, as significantly reported by C11: *"We realized we were losing too much time moving tickets around boards than actually working on stories. You need to find the balance between the time spent with tools and the amount of actual coding"*. This risk is often mitigated with an intense usage of offline manual tools: especially post-it notes and whiteboards are reported to be particularly effective to foster internal communication within the team.
- **The founding team is the most important determinant of speed.** More than techniques and methodologies, the features of the team are the essential factors to achieve high-development speed. Discussed in the theoretical category the *team is the catalyst of development* (see Subsection 6.3.2), this result has been confirmed in multiple occasions both in the interviews and in the questionnaires. What is truly important for the startup, among other factors, is the extremely high-colocation which enables continuous informal and colloquial coordination and discussions. The initial (small) development team usually spends most of the hours of the week working closely to implement the product, often extending the conventional office hours. The large amount of communication, spontaneously held in informal places, has been proven to be an extremely effective element of several successful software projects [171]. Hence, the vastly *informal* development environment of startups acts as a catalyst of speed. Other important elements, which contribute in achieving such an initial high-speed, are: minimal set of functionalities with low concerns on product quality; evolutionary prototyping approach; smart use of advanced tools and frameworks; low precision artifacts; and the close colocation and capability of the team.
- **The initial lack of structures and processes negatively affects the performance when the company starts to grow.** One of the primary objectives of startups is to expand their business into fast growing markets. In fact, if successful, the company will face a growth in terms of number of customers, employees and product functionalities. Then, the

increased complexity arises the necessity of controlling the initially chaotic software development environment. Under delicate circumstances, where an increased market demand meets a poorly engineered product and process, the development team need to start returning the accumulated technical debt. As reported by practitioners, introducing more structures, documentation and standard procedures, cause an initial drop-down in performance which severity depends on the speed of growth, the amount of technical debt and the capacity of the team to absorb it.

- **Startups bring the first product to market in a very short-time and practitioners are satisfied with the software development strategy.** The companies inquired in the case study were typically able to release the first beta of the product in a short time frame. Although practitioners acknowledge the initial negative consequences of the technical debt on subsequent performances, they confirm that it is better spending some effort in later stage to recover the debt than losing time in over-engineering the production initially with the risk of never achieving any significant growth.

All of the above observations are extracted from the analysis of the model, which is systematically derived from empirical data of the case study. Indeed, with slightly different levels of adherence, the implication we presented are reflected in the behaviour of the great majority of companies in the sample. The results of this analysis, which has been focused on the software development strategy from the idea conception to the first release, indicate that startups are far from adopting standard or ad-hoc development methodologies, especially in this phase. The typical tendency is to focus on the team capability to implement and quickly iterate on a prototype, which is released very fast. Thus, in a context where even for the most *lightweight* agile methodologies is hard to penetrate, we believe that it is premature to discuss issues related to SPI. Researchers should rather focus on the trade-off between development speed and accumulated technical debt [48], which appears to be the most important determinant for the future of software development in the company, as further discussed in Chapter 7.

6.6 Theory validation

In this section we discuss the validity of the implications of this study, obtained by means of cross-methodological observations:

- We perform a comparison between our framework and similar models and frameworks (see Subsection 6.6.1).
- We perform a validation of the theoretical framework by triangulating the results of different methodologies with the existing literature. We highlight the areas which have been neglected by existing studies, providing possible

directions for future studies (see Subsection 6.6.2).
- We identify and discuss confounding factors which could interfere with the theoretical model (see Subsection 6.6.3).
- We validate the correctness of the GT procedure by attesting that high-level relations between categories, evaluating if they are compliant with the empirical data we collected (see Subsection 6.6.4).
- Finally we demonstrate how engineering elements independently reported in the follow-up questionnaire by practitioners, can be mapped to theoretical categories in the model (see Subsection 6.6.5).

6.6.1 Comparison with other frameworks

To validate the generalization of the theory, in this section we describe conceptualizations defined from the framework (see Section 6.3) that are in support of a previous model developed by Coleman in [2, 29, 27]. We refer to Coleman's framework since it is the only author which has conducted a similar research in startups' context, even though with a different focus. The author investigated factors in software development that hinder initiatives of software process improvement (SPI) in much more mature companies.

In this subsection we illustrate the most important aspects considered by Coleman, examining similar and contrast factors in face of our *theoretical framework*. We will consider only those categories strictly related to the core category, which relations, as described in GT methodology, generate complete explanatory power of the generated theory.

With his framework Coleman aims to highlight how managers consider two distinct kind of processes: *essentials* and *non-essentials*. The *essential* processes are the most closely linked to the product development such as requirements gathering, design and testing. The *non-essential* processes are those that might be omitted such as planning, estimating and staging meetings.

In relation to the the factors influencing process formation, Coleman considers the background of software development manager and the market requirements the main contributors. In particular, he discusses how practices are routinely removed: *"With most methodologies and approaches, very few stick to the letter of them and they are always adapted, so we adapted ours to the way we wanted it to work for us, for our own size and scale"*.

Our theoretical framework explores the same challenges in software startups, where the act of tailoring processes leads developers to adopt only minimal practices, which are most suitable for the startups context. Furthermore, CTOs and CEOs' background has a great impact on the speed of the development process as described in Subsection 6.3.2. This is in accordance to Coleman's framework, which presents the background of founders and development managers as main factors that affect the management style of the development process.

Differently from Coleman's, our theoretical framework doesn't present how *market requirements* can affect the conduction of processes. In this regard, Coleman describes how the more the definition of requirements are predictable the more well-defined workflows are established. This particular aspect is considered as a confounding factor[27] in our study since we were not able to ground this concept in the data. Therefore we discuss it with additional details in Subsection 6.6.3. In the next part we present the most important comparisons between our framework and Coleman's to highlight similarities and differences. As shown in Figure 6.3, small rectangles represent the main categories of Coleman's framework, whilst big rectangles with bold labels represent ours.

Figure 6.3 depicts Coleman's network of *cost of process* (core category) and all the factors that in management contributed to the lack of software process improvements (SPI)[28].

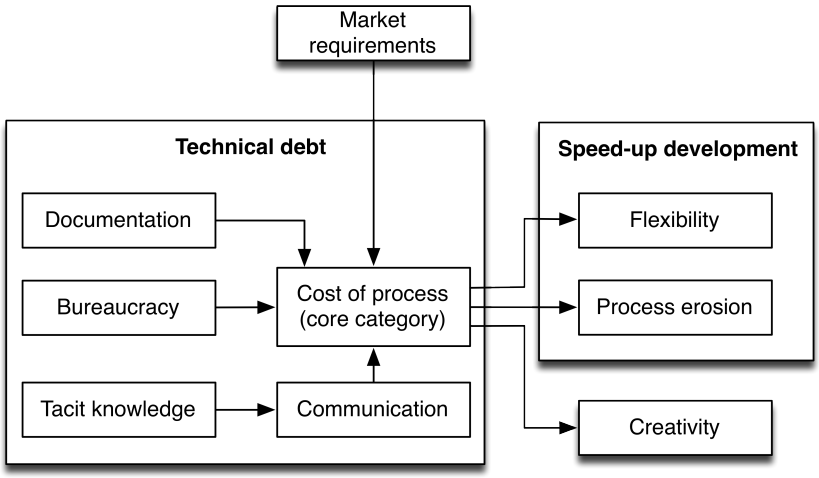


Figure 6.3: Network for the core category of Coleman's framework, adapted from [2]

The *cost of process* represents the lack of formal and prescriptive workflows in development, mainly conducted by means of verbal *communication* limiting heavy *documentation* and *bureaucracy*.

The author reports practitioners' perception that *documentation* alone would not have ensured a complete shared understanding of project requirements. Moreover, defined processes are perceived by managers as additional items with a negative impact on the *Creativity*[29] and *Flexibility* of the development team. This is in accordance with the generated theory (see Section 6.4), which bases the reasons

[27] The identification of confounding depends on the context in which the study is conducted and on the background knowledge of the researchers [172].

[28] The reported network is not complete. For the sake of brevity we present only the conceptual categories related to the core category.

[29] Creativity is considered as confounding factor to this research (see Subsection 6.6.3).

of adopting evolutionary and low-precision engineering elements to the flexibility and reactivity attributes of the development process.

At this regard we can paraphrase a comment recorded by Coleman during an interview: *"When we set up we had more supervisory and managerial roles in that group than we have now and we had to scale that back which has made things a lot more flexible. I do think you have to be nimble, quick and capable being responsive in our position. That works well and I don't want to lose it."* [2].

As shown in Figure 6.3, verbal *communication*, lack of heavy *documentation* and *bureaucracy* can easily be reconducted to the *accumulated technical debt* category (see Subsection 6.3.6) that is consistent with Coleman's framework, which describes how startups experience the lack of main engineering activities and documentation. Additionally, the *speed-up development* category expresses the same concept of *flexibility* and *process erosion* since they have direct relation to the subcategory of *keep a simple and informal workflow*, as discussed in Subsection 6.3.5.

As also reported in our framework, the definition of a *"minimum process"* is not a matter of poor knowledge and training, but rather it is the necessity of operating with solutions that let the company move faster. *"One-size-fits-all"* solutions have always found difficult to penetrate small software organizations [173]. In fact, also when startups began establishing any SPI process they experienced *process erosions*, which resolved to a barely sufficient workflow to satisfy the organizational business needs.

Then, software startups privilege the use of agile principles in support of *creativity* and *flexibility* instead of SPI methods, whereby processes need to be predictable and repeatable. Nimble and ad-hoc solutions prevent the use of heavy bureaucracy and fast communication strategies, even though the accumulated tacit knowledge is hard to manage and to transfer to new hires as also discussed by the category *CAT6*.

Also in [27], the author describes how the *"management approach"* is oriented towards *"embrace and empower"* solutions in contrast with *"command and control"* style, where there is an evidence of trust in development staff to carry out tasks with less direct supervision, greater delegation of responsibilities and a more generally consensual environment. Nevertheless, software development manager and founders have still impact on management style and indirectly on software development process. In the case of early-stage startups, founders are mainly the software development managers as CEOs/CTOs and technical practitioners at the same time[30]. Then, as Coleman identified that background of founders and managers highly influence the software development process, also in our framework the background of CEOs and CTOs shapes the high-level strategies adopted to develop the initial product. Notwithstanding, team members

[30]Note that Coleman, in his studies, refers to more mature software startups, as discussed in Chapter 3.

remain self-organized, able to intervene in all the aspects of the development process without any direct supervision, as discussed in Subsection 6.3.2.

In conclusion, by the comparison of the theoretical framework with Coleman's framework we have found that despite Coleman's study contributes to define which factors hinder the introduction of SPI, his results are coherent with the theory generated within our research[31] (see Section 6.4), and further consideration regarding market and application context of startups need to be explicit into the provided results. Then in addition to *market requirements*, as shown in Figure 6.3, also the creativity category will be further discussed in Subsection 6.6.3 since both couldn't be directly correlated to our theoretical framework.

With wider research focus, another study, developed in 1986 by Brooks [176], discussed what challenges are involved in constructing software products. In his study the author divides difficulties in development into essence (inherent to the nature of the software), and accidents (difficulties attending software production but which are not inherent). In other words essence concerns the hard part of building a software through activities such as specification, design, testing. Accidents refer to the labor of representing the software or testing its representation.

The author claims that the major effort applied by engineers in the last decades was dedicated towards accidents problems, trying to exploit new strategies to enhance software performance, reliability and simplicity of development, such as the introduction of high-level languages for programming. Despite the great achievements in improving development performance, the *essence* property of the software remained unaltered.

Essence difficulties are inherent to: complexity, conformity, changeability and invisibility. Complexity mainly refers to interaction of software modules and elements between each others in some non-linear fashion, which increases accordingly to the project size. Conformity refers to the duty of software to adapt to human institutions and systems. Changeability refers to the constant pressure of software products to accommodate culture, market, laws and hardware transitions. Invisibility refers to the difficulty of representing software in space[32].

[31]Note that a wide variability of processes is a key factor for startups. In contrast with some SPI processes, such as *Six Sigma* which objective is minimizing unpredictability in the definition of workflows [174], startups seem to follow the *Fischer's Fundamental Theorem of natural selection* [175]. In fact, moving towards flexible and variable processes (easy to adapt to the uncertain conditions), increase odds of "natural selection" only when some "restrictive conditions" are met (i.e. CAT1, CAT2, CAT3 and CAT4, described in Section 6.3). In other words, even though ignoring SPI practices is advisable, not following SPI does not predict major adaptiveness of startups development to the market. But, to imply a major evolutionary product/market fit, certain conditions explained in Section 6.3 need to be considered. Further discussion related to changes in flexibility and reactiveness are presented in Chapter 7.

[32]In fact, attempts to diagram software structures brought to constitute not one, but several and general graphs. Despite progress in restricting and simplifying the structures by software models they remain inherently more complex than geometric representation in the way that land has maps, chips have diagrams, computers have connectivity schemas.

Basic attacks (or mitigation strategies) on the conceptual essence proposed by the author are:

- Buy versus build- the most radical possible solution for constructing software is not to construct it at all, taking advantage of what others have already implemented.
- Requirements refinement and rapid prototyping- avoid deciding precisely what to build but rather iteratively extract and refine the product requirements with customers and users.
- Incremental development- starting from simple solutions allows organizations to early prototype and control complexity overtime.
- Great team- people are the center of a software project and it is profoundly important to empower and liberate their creative mind.

What we observed is that the basic attacks presented in 1986 by Brooks, can be easily ascertained within the theoretical framework presented in this thesis.

- *Buy versus build* is the main strategy, which enables startups to externalize complexity to third party solutions explained within CAT1, *speed-up development*.
- *Requirements refinement and rapid prototyping* as evolutionary development approach to maintain a fast release-cycles during development (CAT2).
- *Incremental development* as main purpose of focusing on *limited number of suitable functionalities* examined in *product quality has low priority* category (CAT3).
- *Great team* as main objective of empowering developers' capabilities described in *team is the catalyst of development*(CAT4).

Brooks' forecast about software development strategies is revealed to be extremely accurate, according to the state-of-practice in modern startups.

6.6.2 Theoretical categories and existing literature

In this section we report the evaluation of our theory in contrast with main relevant contribution of studies retrieved with the mapping study (see results in Section 5.1), discussing the categories of the theoretical model.

To conduct this evaluation, researchers mapped the contextual characteristics of startups, identified by the selected studies (*themes* in Table 5.7), into the categories developed with the GT. Table 6.1 shows how all the 15 *themes* can be wrapped inside the framework's categories.

Category (GT)	Theme (SMS)
Speed-up development (CAT1)	T.10 One product T.15 Rapidly evolving

Table 6.1 – *Continued on next page*

Table 6.1 – *Continued from previous page*

Category (GT)	Theme (SMS)
Evolutionary approach (CAT2)	T.4 Uncertainty T.13 Highly reactive
Product quality has low priority (CAT3)	T.4 Uncertainty
Team is the catalyst of development (CAT4)	T.3 Small team T.9 Flat organization T.11 Innovation
Accumulated technical debt (CAT5)	//
Initial growth hinders performance (CAT6)	//
Severe lack of resource (CAT7)	T.1 Lack of resources T.2 New company T.5 Highly risky T.6 Not self-sustained T.8 Low-experienced team T.12 Time pressure T.14 Third party dependency

Table 6.1: Final comparison - Categories and themes

The capability of the model to accommodate the contextual features of startups, identified by other researchers, gives us more confidence in the validity of our research.

Another kind of comparison has been executed by identifying issues related to the theoretical categories in the retrieved relevant studies. We illustrate how the selected studies contributed to our framework validation as following: for each retrieved article we present the main results, mapping them to categories in accordance to the theoretical framework. Table 6.2 shows the main contributions of the 37 identified studies, where an 'X' is placed in correspondence to the theoretical category they considered. The table has been sorted by computing how many categories of our theoretical frameworks have been considered by the study (column *Count*). Therefore in the highest part of the table there are the studies which somehow discuss the higher number of concepts identified in the early-stage startups framework.

Author (year)	CAT1	CAT2	CAT3	CAT4	CAT5	CAT6	CAT7	**Count**	Ref.
Sutton (2000)	X	X	X	X	X	X	X	7	[13]
Kajko (2008)	X	X	X	X	X	X	X	7	[18]
Crowne (2002)	X	X	X	X	X	X	X	7	[10]
Coleman (2008)	X	X	X	X	X	X	X	7	[29]
Coleman (2008)	X	X	X	X	X	X	X	7	[27]
Coleman (2007)	X	X	X	X	X	X	X	7	[2]
Camel (1994)	X	X	X	X	X	X	X	7	[70]
Yoffie (1999)	X	X	X	X	X	X		6	[81]
Zettel (2001)	X	X	X	X			X	5	[88]
Jansen (2008)	X		X		X	X	X	5	[95]
Heitlager (2007)			X	X	X	X	X	5	[3]
Deias (2002)	X	X		X	X	X		5	[80]
Ambler (2002)	X	X			X	X	X	5	[76]
Wood (2005)	X		X	X			X	4	[160]
Tingling (2007)		X		X	X	X		4	[77]
Taipale (2010)	X	X		X	X			4	[84]
Silva (2005)	X	X		X	X			4	[79]
Mirel (2000)	X		X	X	X			4	[92]
Midler (2008)	X	X		X			X	4	[86]

Table 6.2 – *Continued on next page*

Table 6.2 – *Continued from previous page*

Author (year)	CAT1	CAT2	CAT3	CAT4	CAT5	CAT6	CAT7	**Count**	Ref.
Tanabian (2005)	X			X			X	3	[33]
Stanfill (2007)	X				X		X	3	[159]
Mater (2000)	X	X				X		3	[91]
Kuvinka (2011)	X	X		X				3	[85]
Deakins (2005)	X	X		X				3	[89]
Yogendra (2002)	X	X						2	[87]
Wall (2001)	X						X	2	[96]
Su-Chuang (2007)	X			X				2	[162]
Steenhuis (2008)				X			X	2	[161]
Sau-ling Lai (2010)		X					X	2	[163]
Kakati (2003)		X		X				2	[35]
Himola (2003)	X						X	2	[68]
Häsel (2010)	X			X				2	[158]
Hanna (2010)	X						X	2	[94]
Bean (2005)	X				X			2	[97]
Kim (2005)		X						1	[93]
Fayad (1997)				X				1	[90]
Chorev (2006)				X				1	[34]
Count	29	22	13	26	18	14	20		

Table 6.2: Mapping literature into categories

According to our framework, only 7 studies, out of 37 (18.91%), discuss aspects related to the overall software development process in early stage startups. The remaining part of the studies is only partially covering the phenomenon described by the framework. Looking at the results, we were able to extrapolate information which confirmed the conceptualizations[33], discussed in Section 6.3.

To emphasize which theoretical categories have been most neglected in the current state of the art, we present the frequency of occurrences of each category among the sample (*Count* row of Table 6.2). To provide the reader with a better visualization of this information, we plugged these values directly into the high-level framework, showing bigger blocks in correspondence of more discussed categories and vice-versa (see Figure 6.4).

[33]Analysis of all the studies were conducted in pair. In case of conflicts researchers collaboratively analyzed the underlined articles more in-depth.

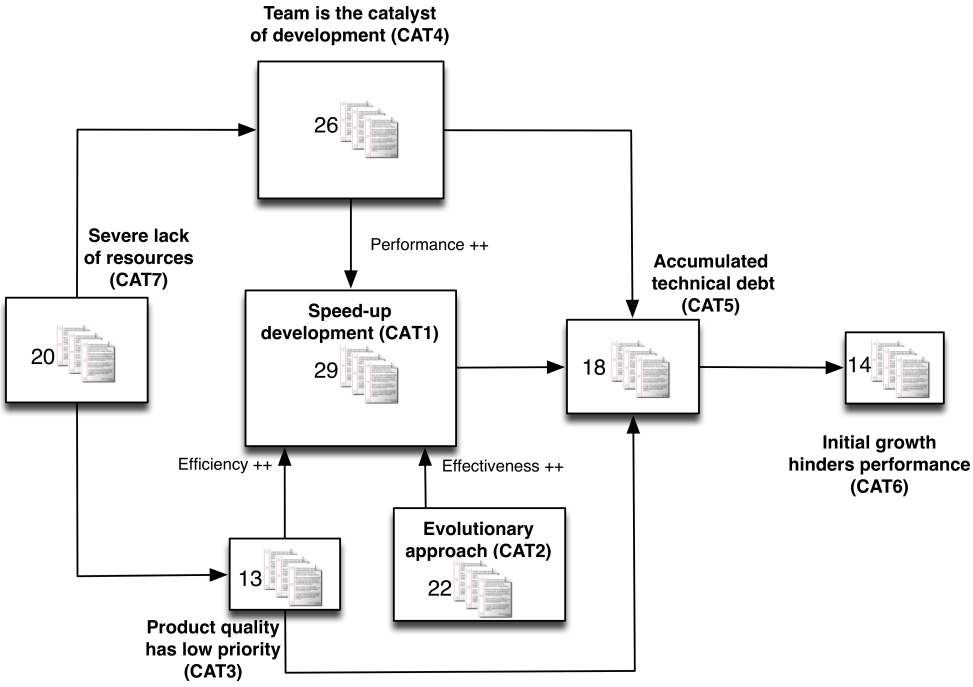


Figure 6.4: Framework validated through literature focus

We can see how the majority of the retrieved studies (29) mention issues related to *speed-up development* which confirms, in fact, the importance of the *core* category. On the other hand we can observe that only a limited number of studies mentions results related to CAT3, CAT5 and CAT6. This suggests the direction of possible future primary studies towards: the technical debt and its future consequences on performance; aspects related to functionality and quality of the first version of the product.

These results confirm the relevance of development teams as widely discussed in the SE literature, here mapping 26 studies to CAT4 (*team catalyst for development*). The importance of people has been widely discussed in other studies in SE advocating for the need of *empowering people*, which is the element with the greatest impact on software development. Furthermore, being humans *non-linear variables*, they are unlikely to follow repeatable prescriptive methodologies. As stated by Highsmith, *"A drawback of each and every methodology is to expect people to behave consistently over time, when is clearly not like that."* [171]. For more detailed discussion about the importance of people the reader is referred to some prominent authors: Cooper [177], DeMarco [178], Coleman [71], Valtanen [50], Adolph [179] and Cockburn [180]. Especially in early-stage startups, where the whole company overlaps with the development team, factors related to people are without any doubt the most important ones (confirmed with the empirical data of our case study, discussed in Subsection 6.6.5).

In addition to the theoretical framework proposed by Coleman previously discussed in Subsection 6.6.1, in the next part of this subsection we present sim-

ilarities and contrasts with other relevant studies selected with the SMS.

Starting from the concept of *severe lack of resources*, described in Subsection 6.3.1, many authors describe startups facing with inescapable constraints defined by engineering/business concerns and constant time-pressure [70, 181] as also shown in Table 5.7. Because of their young and inexperienced working conditions, startups present limited resources in terms of management and strategic alliances [13].

Crowne in [10] identifies four main stages of a startup's lifecycle: startup, stabilization, growth and maturity. Inexperienced practitioners operating in a chaotic environment characterizes the first phase, where there is no strategic plan for developing the product. But, moving towards the *stabilization* phase, the product starts to be unreliable and requirements unmanageable. Consequently, when startups move to the *growth* phase, they initiate structuring and controlling processes, which are incrementally integrated in the development environment. The *maturity* represents the moment when the company is ready to introduce process improvement initiatives. This study profoundly corresponds to different conceptualization of the theoretical framework. Starting from the minimal and low-precision engineering elements, startups start to grow the accumulated technical debt which requires fast integration of more structured and standard processes overtime (discussed in the model presentation, Subsection 6.3.7).

Issues related to low-precision engineering practices during companies' growth are discussed in an experience paper by Ambler [76] where he reports the case studies of two growing internet startups approaching an IPO. Despite the two companies were facing business and team growth, attitudes against the introduction of processes to structure and model the system were evident. The need to re-architect and redevelop the codebase lead the researchers to suggest a tailored version of the RUP[34]. Nevertheless, even with the aid of CASE tools[35], startups failed in practicing engineering activities in view of eccesive lack of flexibility. In fact, developers fast moved to simpler tools such as the whiteboard, where they were able to provide input and insights, question assumptions, share past experiences, and even talk about potential implementation strategies. In general the development remained highly iterative, where project teams started to have a little modeling and testing only when necessary. They didn't take the serial approach of completely developing and accepting the requirements model, analysis model and so on. This let them react swiftly and adapt rapidly according to changes into their highly competitive markets. The researchers report *"modeling*

[34]Iterative software development framework based on sophisticated manage requirements, component-based architecture, modeling, continuous quality verification and control changes (more details available at `http://www.ibm.com/developerworks/rational/library/1826.html#N100E4`).

[35]CASE tools are a class of software that automates many of the activities involved in various engineering activities. See `http://www.unl.csi.cuny.edu/faqs/software-enginering/tools.html` .

was streamlined", that is modeling what was needed and go straightforward to coding, avoiding analysis paralysis.

The use of whiteboards in working environment is in support of the *use of well-integrated and simple tools*. In both the startups studied by Ambler, documentation was minimal, finding the "sweet spot" to have just barely enough tools and artifacts to meet their needs and no more. Generally, the use of *keep a simple and informal workflow*, described in Subsection 6.3.5, endorses the use of low-precision engineering practices which lead to speed-up development initially, but with negative effects on the accumulated technical debt of a minimal project management (see Subsection 6.3.6) and consequently on the final company growth (see Subsection 6.3.7).

Kajko-Mattsson et al. investigated a Swedish software startup company involved in mobile applications [18], reporting the following issues: a lack of requirements gathering process; release cycles and their length were not defined; the releases and their scope were not planned; lack of control over the change requests; lack of process control in terms of absence of any documentation to track the status and progress of the process; defective releases in terms of lack of testing; poor communication since the communication inside the company was informal and not documented at all. In the same study they designed a process improvement model, based on three main phases: *Evaluate*, *Plan* and *Change*.

During the evaluation phase Kajko-Mattsson et al. tried to establish the development workflow status within the company, and the authors document a status of chaos where any initiatives of quantitative methodology assessment would have been a waste of time. Therefore, they applied a qualitative methodology approach to define a model of the release management process. In fact, they moved towards the plan phase where the researchers decided to develop a model to manage releases, introducing a process composed by: release scope preparation, release planning, release development, system testing, acceptance testing and release deployment. Despite the effort spent from the researcher to establish a development workflow, practices were not fully conducted even though some marginal benefits have been reported: planned major release from four to six months; planned minor releases from two to four weeks; encompassing urgent corrective releases from one to five days. Regarding the previously mentioned problems, only partial improvements have been achieved since the major obstacle was to motivate employees to change their habits. With perfect hindsight the researchers claim that understanding and adapting solutions by solving one problem at a time would have increased benefits. Then, despite the reported results, start establishing simple and informal workflow aided the organization to move from the stage of knowing nothing to knowing at least something about their development process.

The integration of minimal project management during the development activities can foster the control over development productivity and control the *accumulated technical debt* (see Subsection 6.3.6), but still with particular attention

to the development workflow status within the company, designing specific and tailored solutions.

In fact, prescribed development models such as SPI in startups are basically ignored by startups, as discussed in [88]. To overcome the need of structure and control to accelerate the development process, in his study, Zettel developed a *"LIghtweight Process for E-business software development"* (LIPE). This lightweight development method integrates XP approach with ideas from the areas of software measurement for project control and process improvement as a compromise between ad-hoc and more rigorous approach. LIPE, as argued from the author, provides scalability of the development process with a certain degree of flexibility, omitting parts of prescribed practices. Nevertheless, we couldn't retrieve any empirical evaluation of the study.

In [89], the author designed a new spiral methodology evolved during fieldwork with a developer team that had limited resources managing innovative product in volatile e-commerce environments. He proposes fast evolutionary approach experimentation with flexible processes obtaining customer's feedback as soon as possible for continuous product improvement. Even in this case the model is compliant with the nature of a random, ad-hoc and visionary product development. In fact, evolutionary development approaches are typically more suitable for web applications, as confirmed in [53]. Then the practices are oriented towards rapid and iterative design with complete freedom to the developer team, enabled to operate anywhere within the organization, according also to the category *team is the catalyst of development*, discussed in Subsection 6.3.3.

Carmel reports in [70] the need of flexible and rapid development solutions to shorten time-to-market, but even more important, the need of high-skilled team developers, which heavily impact on *speeding-up development* (see Subsection 6.3.5). In fact, the author suggests that entrepreneurs need to look for a well formed, skilled core development team and not just a set of product ideas and features. Moreover the team must be empowered with full-stack and self-organization settings with the flexibility of minimal bureaucracy. A studied company stated: *"we didn't need weeks or months of detailed modeling and documentation , but rather modeling the architecture a little and then either exploring strategies by providing users with code or simply starting work on the actual software itself"*.

Eventually to focus on simple solutions, two other studies [96, 97] sugges to *"reuse the wheel, instead of reinventing it"*, taking advantage of open source software whenever it is possible, building systems quickly and effectively as a result. The author conclude stating that even though the principles in leading-edge development processes are the same principles of yesteryear, the way in which the fundamentals are applied has changed from prescriptive to agile solutions with less concern to quality aspects, also confirmed in [91] and discussed in subsections 6.3.3 and 6.3.4.

Thus, by summarizing the comparison, we have found that most of the con-

tributions of the studies are compliant with the behavioural model of the phenomenon we provided.

6.6.3 Confounding factors from the literature

The purpose of this subsection is to identify which confounding factors need to be taken into consideration when evaluating the theoretical framework. The confounding factors are those variables which and have not been considered in the model and might interfere with the theoretical framework positively or negatively [103]. Even though there are no systematic procedure to identify confounding factors, we report those that have been explicitly identified by the literature review, that are related to: *creativity, innovation, market type and application type*.

A prominent contribution investigating creativity and innovation in startups, is presented in [3]. The study reports how development approaches oriented to the product in the early-stage life-cycle, called *fluid* (see Figure 6.5), have main impact to innovative solutions initially. After a while the need of structuring and controlling the growing company size (along with *transition* and *specific* phases) arises the need of project management activities and software processes for long-term scalability issues.

Based on Figure 6.5, obtained by the model presented in [3], the horizontal axis indicates the lifecycle stages of a startup (*fluid*, *transition* and *specific*), whilst the vertical axis indicates for each development approach (product-oriented and process-oriented) the level of innovation achieved. The study reports how product-oriented development, in contrast with forced and mature process imposition, gives degree of freedom to the development team, enhancing *creativity* of developers and augmenting the *innovation* capability of the company in the early-stage.

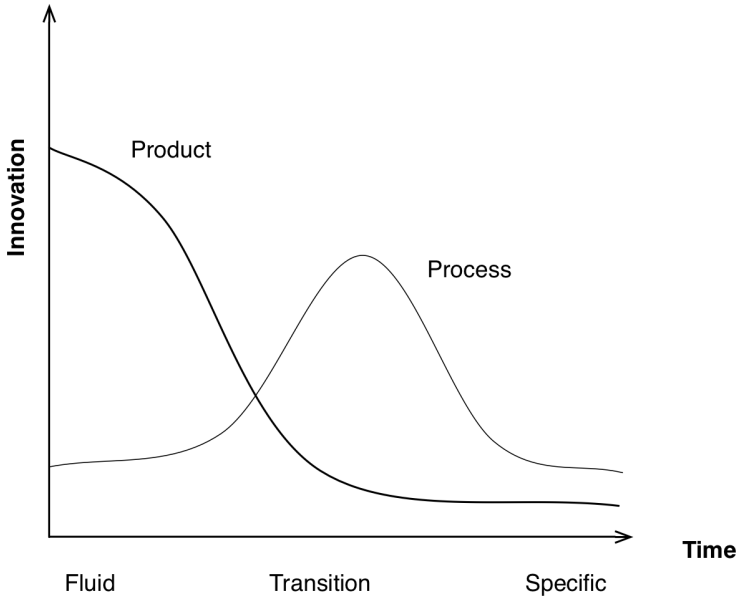


Figure 6.5: Innovation model [3]

The empirical model we presented has been obtained from the transcripts of interviews, where the theme of innovation only seldom emerged. However, if innovative and creative factors were considered in the model, the integration of results would have been straightforward, since the evolution of this model is totally compliant with our assumption of CAT5 and CAT6. From product-oriented development in the *fluid* phase to the process-oriented development in the *specific* phase, the *innovation* confounding factor seems to behave initially the same as the *speed-up development* (see detailed description in Subsection 7.5). Nevertheless, we were unable to ground these evidences in the GT data even though they might present strict correlation with the choice of an early product-development orientation with low-precision engineering elements.

Other two important factors, that are not grounded in our empirical data, are related to: *a.* the requirements expected by a specific market sector (also called *market requirements* in [2]); and *b.* the application type as reported in [13, 27, 29]. The main impact of those confounding factors is related to the adoption of flexible and reactive solutions for the development process. In particular they refer to the necessity of fulfillment of quality concerns that goes beyond scalability and UX defined by the framework. Especially when requirements are rigidly imposed or the application domain is well-known, providing low-quality products to final users might determine the failure of a startup.

For example security in critical applications such as safety-critical systems present quality concerns, which must be granted. But also minor risky application domains such as e-commerce systems have high-impact factors for obtaining the first customer reference, as discussed in [89, 93, 79]. This characterizes those

applications, whose part of functionalities are already successfully implemented in other software systems or whose failure might report serious financing or social damages [182].Yet, limiting our context to early-stage software startups, which typically develop first cutting-edge product in beta versions, partially mitigates the high-quality demands of users.

To understand how to obtain processes in accordance to a set of customer needs, correlating the proper levels of customers' satisfaction, we introduce the Kano model, described in [4] and extensively studied in product development. As shown in the Kano model in Figure 6.6, users preferences can be classified into four categories: *basic* (below the respective curve), *excitement* (above the curve), *performance* (is an unidimensional area represented by the respective segment) and *indifference* (represented by a light-gray square) areas.

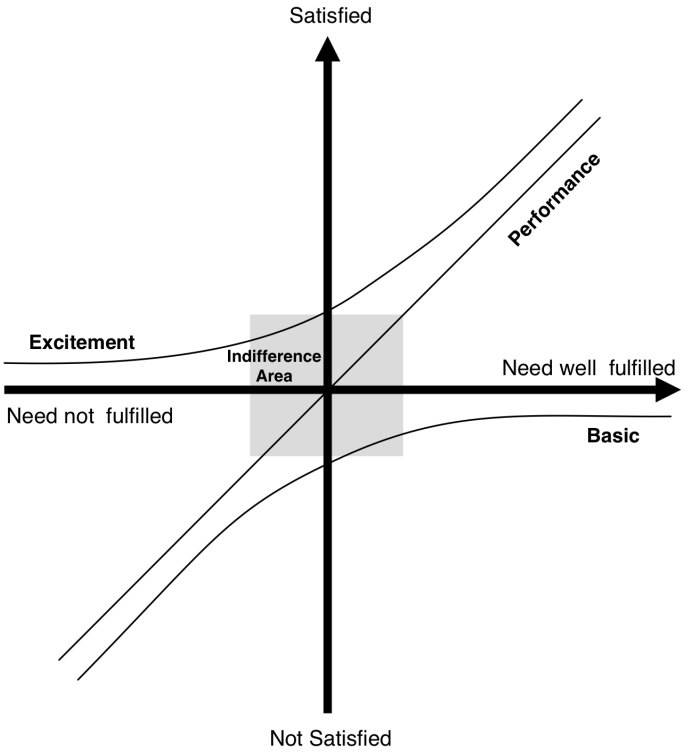


Figure 6.6: Kano model [4]

The *basic* area represents the space where customers are dissatisfied if those *must-be* functionalities are not met. This is the case of similar functionalities already implemented in the current market with well-known solutions. The *performance* area represents the space where customers' satisfaction is proportional to how many of functionalities are implemented. In the *indifference* area the customer does not really care about the functionalities: the satisfaction remains invariant. This represents the case where startups have not (yet) achieved the

right product/market fit. Finally, the *excitement* area represents the space where customers will be more satisfied if functionalities are implemented, but not disappointed if they are not implemented. This is the case of extremely innovative product development.

Well-known domains such as e-commerce applications or other *"quality critical products"* belong to the *basic* area, where if a quality concern such as security in bank transfer services is missing satisfaction of customers will be profoundly affected. But as discussed, in early-stage startups that typically develop first beta products, developers usually deal with cutting-edge technologies. Those innovative solutions, if in the right market, are likely to belong to the *excitement* area, where satisfaction can hardly be negatively affected by faulty products. In fact, in the theoretical framework (see Section 6.3), minimal functionalities and less concern about quality aspects have low-impact on customers' satisfaction, supporting startup's ability to mainly focus on *speed-up development*. Moreover operating in the *excitement area* is fully compliant with an evolutionary approach and minimal functionalities since from the first product released to the market, it is unlikely to lose users because of high-quality demand.

Nevertheless, this aspect profoundly undermines the generalizability of the theoretical model according to the theory presented in Subsection 6.3.4, in particular *CAT3*. Therefore, future studies within wider startups' context are advisable to extrapolate insights from those situations where quality concerns cannot be postponed or externalized.

Finally, also the experience of team-members is considered as an important confounding factor. Since only little evidence has been gathered during the interviews, "lack of experience" was not considered as a sub-category in *severe lack of resources*. Notwithstanding, according to [10], startups at the beginning rely on clever, but inexperienced developers (see Subsection 6.3.2). In fact, having team-members with deep experience is a "double-edge sword", as discussed in [81]. From one hand experience might quickly provide structure and maturity to the development process, from the other it might cause huge challenges in managing human resources in view of the fact that self-confident overachievers will almost inevitably clash. Consequently the team management requires higher control and coordination activities that inevitably hinder the flexibility of the development environment at the beginning, that is essential in early-stage startups as discussed in subsections 6.3.2 and 7.5.2. Accordingly, also Coleman in [2, 29, 27] states that the operating strategies towards *"minimum processes"* in startups is not a matter of poor knowledge and training, but rather it is the necessity of operating with solutions that let the company move faster.

Nevertheless, since the amount of the results did not allow "the lack of experience" to be in the theoretical framework (in view of the fact that not enough

data were identified [36]), this confounding factor requires further investigation.

6.6.4 High-level relations validity

In this subsection we validate the most important causal relations presented in our model by attesting their correctness with a quantification of empirical trends observed in the data[37]. In fact we identified some trends and recurrent patterns which we tried to synthesize in this section by thoroughly analyzing the complete set of data obtained from the case study and the theoretical model (depicted in Figure 6.2). We illustrate our observations aided by simple statistical methods, which provides to researchers an additional tool to validate our model supported by further companies' data.

As previously discussed when presenting the high-level framework (see Subsection 6.2), the most essential feature that startups aim to achieve is high speed. This is reflected in the model (see Figure 6.1), as shown below in the detail of high-level framework (Figure 6.7).

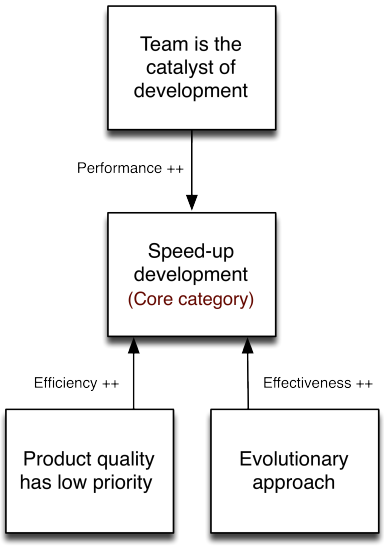


Figure 6.7: High-level framework core category network

The *core category* (*speed-up development*) is influenced by: the degree of quality concerns (contributes to the *efficiency*); the undertaken development approach (provides *effectiveness*); and finally the intrinsic characteristics of the team (guarantee *performance*).

[36] A category is formed if the codes are representative of more than the 70% of the analyzed transcripts.

[37] This is applied according to the *fit* factor discussed in Subsection 4.3.6 within the static assessment process.

Before starting the evaluation of the impact of the three categories on *speed-up development*, we grounded in the interview transcripts the fact that those relations were characterized in terms of performances[38]:

- The team enhanced the overall *performance* of the software development. In fact, recalling that startups deal with severe lack of resources, the team members are the company's main assets. They develop software with no need of formalities and structures, left to the team's own capabilities to efficiently develop the product. Moreover, the effectiveness is granted by continuous advices given by mentors and by a management style oriented towards self-organization, full-stack, and multi-role settings[39] (described in Subsection 6.3.2 and confirmed, among others, by Sutton in [13]).
- The evolutionary approach enhanced the *effectiveness* in implementing the right functionalities. In fact, this approach improves the company's capabilities to adjust the trajectory of product development (see the concept of *pivoting* in Appendix A.2.4).
- Since product quality has low priority, the development was focused only on implementing a limited number of suitable functionalities, enhancing its *efficiency*. In fact, an essential prototype (MVP) does not require to comply with heavy quality constraints, enabling the team to quickly implement functionalities, ready to be validated by final users[40] (see Subsection 2.3.3).

An optimal combination of these three categories enables startups to ship code extremely quickly. However, the price that startups have to pay for achieving such high speed is related to the extent of the *accumulated technical debt*, which will finally contribute to hinder performance in later stages[41]. This is reflected by the right-hand side of the high-level framework, here extracted in Figure 6.8.

Figure 6.8: High-level framework- technical debt network

[38]TIme-to-market was the main concern, as confirmed by the state-of-the-art and state-of-practice (see Subsection 5.1.3 and Section 6.5).

[39]Note that the importance of a skilled team for development performance is expressed in many studies such as [32, 171].

[40]Note that nonfunctional requirements, such as a good level of UX, need to be achieved (see Subsection 6.3.4 for more details).

[41]See subsections 6.3.6 and 6.3.7 for more detailed explanation.

Accordingly, by analysing the data of the 13 companies collected with the empirical study, we observed two distinct trends which further empower the above illustrated relations:

1. *The more* a capable team[42] with few quality constraints uses an evolutionary approach, *the faster* the company releases the product.
2. *The faster* the company releases the product, *the larger* the degree of *accumulated technical debt*[43].

To assess these hypotheses we defined three measures reflecting the theoretical model:

- *Execution speed*: a metric that represents the development speed of the startup during different phases of the first release, computed by means of a weighted average speed for each phase. It was obtained by analysing interview transcripts looking at subcategories of *Speed-up development*.
- *Technical debt*: a metric that represents the extent to which processes are controlled, structured, planned and documented by means of engineering artifacts and practices. It has been computed by means of a weighted average of the debt accumulated in each development phase, observing subcategories of *accumulated technical debt* with consequences on the startups growth.
- *Potential capability*: a metric that represents the degree to which each company reflected the capability of reaction and flexibility to the dynamic environment during the development process, given by the three categories that (theoretically) mostly affect *speed-up development* (see Figure 6.7).

We quantified these measures for each company involved in our empirical study by defining score metrics based on a set of rubrics and evaluating startups accordingly, through an analysis of interview transcripts, codes and questionnaire results. The complete statistical procedure and provided rubrics to quantify the measures are illustrated and discussed in Appendix A.5.

For each dimension the scores have been computed in the range between 1 and 5, where the meaning of boundary values are illustrated[44] in Table 6.3.

Dimension	Boundary values
Execution speed	1 = very poor, 5 = extremely high
Potential capability	1 = very poor, 5 = extremely high
Technical debt	1 = very low, 5 = severely high

Table 6.3: Definition of boundaries for numerical values

[42]The team capability is evaluated according to the characteristics presented in CAT4 (see Subsection 6.3.2).

[43]The debt here is considered regardless of context-specific debt mitigation factors and tactics.

[44]See Appendix A.5 for the complete rubric.

As expected, the 13 startups which participated in this study received general high evaluations for speed, capability and a relatively high amount of accumulated technical debt, as shown in Table 6.4.

Company	Execution speed	Potential capability	Technical debt
C1	4.022556391	3.928571429	3.052631579
C2	3.977443609	3.134920635	3.180451128
C3	3.691729323	2.261904762	2.601503759
C4	4.17699115	4.246031746	3.362831857
C5	4.407079646	3.293650794	3.14159292
C6	3.973451327	2.579365079	3.061946903
C7	3.778761062	2.936507937	3.03539823
C8	4.442477876	4.484126984	3.362831858
C9	3.938053097	3.611111111	2.796460177
C10	3.659574468	3.531746032	2.085106383
C11	4.732290708	5.000000000	3.451701932
C12	4.150442478	2.460317460	3.115044248
C13	3.422812193	2.063492063	2.973451327

Table 6.4: Quantification results of *execution speed,technical debt* and *potential capability*

We conducted statistical tests using the analysis of variance to assess the existence of relations between the dimensions. First we defined two null hypotheses (H0): $H0_1$ = *Startups do not release the product faster when a capable team adopt a more evolutionary approach AND with less quality constraints*; $H0_2$ = *The execution speed does not increase the amount of accumulated technical debt.* Then we tested $H0_1$ and $H0_2$ with an one-tailed test using Pearson's product moment correlation coefficient[45], with positive association analysis, fixing the level of confidence to 95% which means we reject H0 in case the p-value is lower than 0.05.

We conclude that, in our sample:

1. Higher values of *Execution speed* are strongly associated[46] with *higher* values for *Technical debt.*
2. Higher values of *Execution speed* are strongly associated[47] with *higher values*

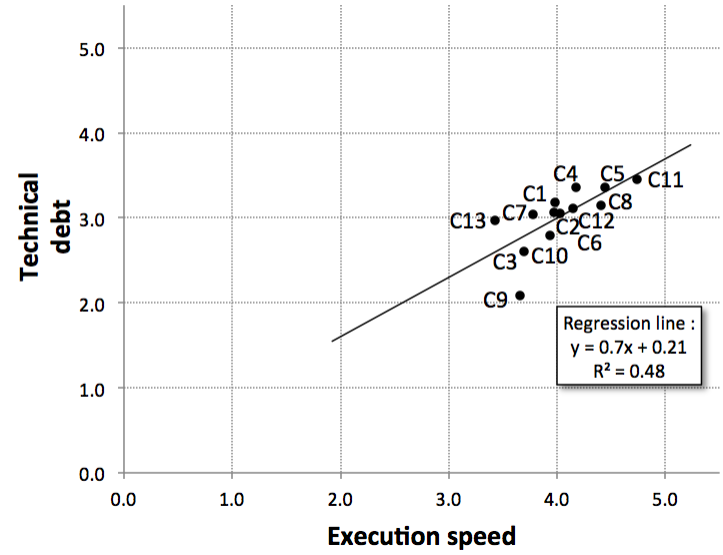
[45]To perform the Pearson's correlation we verified two assumptions: a) data is normally distributed; b) data is on interval scale (see Appendix A.5.3 for more details).

[46]Clear statistical significance, p-value: 0.002073.

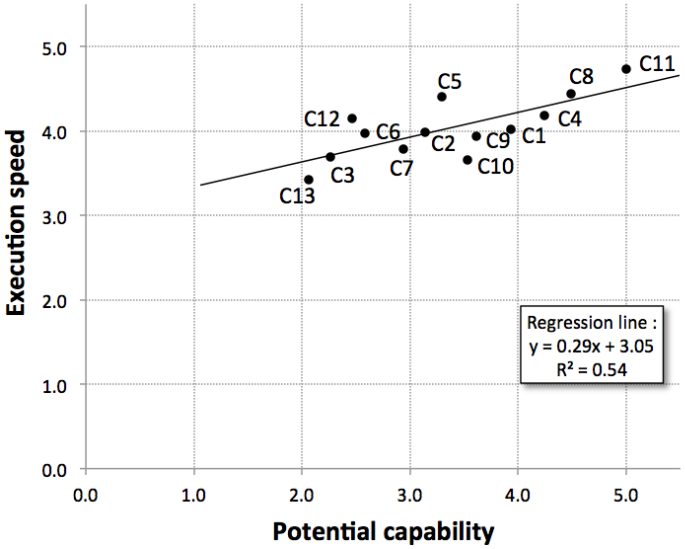
[47]Clear statistical significance, p-value: 0.004549.

for *Potential capability*.

The variables are plotted below together with a regression line in (see Figure 6.9).



(a) Execution speed and Technical debt



(b) Potential capability and Execution speed

Figure 6.9: Measures - Linear regression

Since the numerical scores have been derived by the categories and subcategories of the theoretical model, the most important implication of these statistical results is a confirmation that, for the 13 companies we inquired in the study, the relations between theoretical categories of the high-level framework (see Figure 6.1) are valid. Additionally, the adherence of empirical results to the theoretical framework contributes to attest that the grounded theory study has been correctly conducted, i.e. the theory actually emerged from the underlying data.

Observe that we are not able to extend the above numerical results to other startups out-of-sample. However, throughout the entire document - especially in Appendix A.5 - we provided enough detailed processes, evaluation tables and raw data to enable other researchers to reproduce both *theory generation* and *theory validation* with new data-points.

The validation has been performed on the most critical causal relationships of the model. According to the paradigm model (see Subsection 4.3.5) those are the causal condition which are linked to the core-category. There are four high-level relationships[48] which have not been validated using numerical methods. However their correctness has been verified with a cross-analysis of the extant literature and by attesting the causal relations among codes and categories, through axial and selective coding processes described in Subsection 4.3.4.

6.6.5 Engineering elements and categories

Finally, to provide a concrete example of how the model can be used, we show that the engineering elements mentioned by respondents in the survey (see Subsection 5.2.3) can be mapped into the theoretical categories of the framework. Based on the score assigned with the follow-up questionnaires by practitioners to engineering elements, we create a ranking of the *theoretical categories* which most contribute to the core category *speed-up development*. To produce such ranking we followed the process explained below.

First we mapped the *engineering elements*, presented in Table 5.15, to a specific abstract category of the framework.

To compute the score of each category, we executed the following steps:

- Extracted engineering elements rated as *very* and *extremely useful* from questionnaires' repertory grid results (see Table 5.15).
- Extracted the list of engineering elements rated as most useful with a perfect hindsight from questionnaires' results.
- Assigned arbitrary weights to highlight the contribute to the score given to

[48] CAT7 → CAT4; CAT7 → CAT3; CAT4 → CAT5; CAT3 → CAT5. These relations, according to the paradigm model, represents *intervening conditions* and *action/interaction strategies*.

elements[49] marked as *very useful* ($w_v = 0.2$), *extremely useful* ($w_e = 0.4$), and with *perfect hindsight* ($w_h = 1$).

- Counted the frequency of occurrence in the result for each element and multiplied it by the weight associated with the element type.
- Mapped each engineering element from the questionnaire to a specific *category* of the *theoretical framework* (see Table A.24).
- Finally computed the score of each category by summing up the contribution of single elements which belong to it.

The table that shows the results of this process is presented in Appendix A.4, TableA.25. This procedure allowed the researchers to further refine the categorization and translate low-level concepts in high-level abstract categories, which have been used as starting point to perform the *ranking of categories*. Thus, this process contributes to attest the validity of relations between categories and, at the same time, provides an idea of which categories are perceived as most significant to shorten time-to-market.

All the engineering elements which have been rated in the questionnaire, contributed to a certain degree to the software development process since they have been explicitly mentioned during interviews. However, in this analysis we focused on those elements rated as *very* or *extremely useful* in support of speeding up development. Figure 6.10 shows the final outcome of the *ranking of categories* process described in Subsection 4.3.6, displaying the name and the score of the top categories[50].

[49] It could be very interesting to analyze and understand elements which are perceived as time-wasting and the rationale behind it. However, due to time-effort constraints, those elements have not been considered since we aim to understand which elements are perceived as strongly positive among practitioners.

[50] The minimum threshold on the score, for a category to be displayed, has been arbitrarily set at 1 to avoid including meaningless long lists.

Figure 6.10: Ranking engineering elements

According to practitioners, the most highly rated categories, which contribute to *speed-up development*, are activities and practices incorporated under the *find the product/market fit quickly* category of our framework (see Subsection 6.3.3), followed by the *use of well-integrated and simple tools*, *skilled developers* and *multi-role and full-stack engineers*. It is important to note, among others, how elements and categories related to the team capabilities are regarded as very important by practitioners. This confirms the primary role of the team (CAT4) in contributing to the primary objective of startups - shortening time-to-market.

With this last step we further evaluated the framework's explanatory power by incorporating low-level concepts extracted from the questionnaire responses into abstract categories of the model.

6.6.6 Summary of validation

The causal relations - represented with directional arrows in the framework of Figure 6.1 - emerged from the GT process of selective coding in interview transcripts. In particular, at high level, we were able to identify nine causal relations among categories. To validate the theory which emerged from the framework, in the sections above we used different techniques. As suggested by creators of GT [128] and remarked in Coleman's studies [27, 151], we focused on the validation of the most important relations for the model. In particular, in Subsection 6.6.4 we attested that the combination of CAT2, CAT3 and CAT4 - the categories which are converging to the core category - contribute to *speed up development* (see Appendix A.5.1).

Furthermore we validated two more non-obvious causal relations which are at the foundation of the theory: the relation between *speed-up development* and *accumulated technical debt* (CAT1 → CAT5); and between *accumulated technical debt* and *initial grow hinders performance* (CAT5 → CAT6)[51]. Finally we left open to other researchers the possibility of performing a further validation of the causal relations with additional companies.

Summarizing the overall results of the model validation, the most important achievements are listed below:

- We proposed a theoretical model describing software development in early-stage startups, which is strongly grounded in the underlying empirical data (provided in Appendix A.4.3).
- We generated a formal theory and discussed its implication, answering to RQ-2.
- We discussed how the model is compliant with the GT framework presented by Coleman, and with Brooks 1986's forecasts about software development.
- We observed that the contextual characteristics of startups identified in the literature (*themes*) can be easily accommodated within the theoretical categories of our framework.
- We analyzed which categories received most attention in the literature, and suggested possible new area of interest for future studies.
- We executed an in-depth comparison between the results of other authors and the implication of our model, discussing commonalities and identifying discrepancies.
- We attested the correctness of the theory generation procedure by testing the model against the data which generated it attesting that the causal relationships were correctly extrapolated from the empirical data.
- We verified causal relations between subcategories of the detailed framework by checking their correctness in the interview transcripts.
- Finally, by trying to fit engineering elements independently reported in the follow-up questionnaire by practitioners into our theoretical model, the theory underwent a further validation, demonstrating to be capable of handling low-level concepts expressed in the follow-up into high-level abstracted conceptualizations.

6.7 Generalizability of the theory

The conceptualization of complex systems in a theoretical framework is an approximate representation of the actual underlying reality [2]. The case study, from which our conclusions are generated, has been performed in a cross-sectional analysis of newly created web startups in the time frame that goes from the idea

[51] See Appendix A.5.2 for detailed explanation.

conception to the first public release. Furthermore, in all the companies inquired, the most urgent priority was releasing the product as soon as possible in a context characterized by high uncertainty[52]. Despite GT forces researchers to abstract from specific details, the above-mentioned elements could have influenced the emerging theory. However, the validation of the theory (see Section 6.6), performed within the same context, provided evidence that in these specific environment the results are likely to be adherent to facts.

Particularly, since a number of companies inquired in the study have a mobile part of the application, the majority of results can be extended with a good degree of confidence to startups operating in the mobile application sector too. Furthermore, we conducted an interview with a company (C13) which main product was a desktop application. Even in this case, with some low-level differences related to the release policies, the framework proved to be correct.

Moreover, although the questions that characterize our investigation, are focused on the development activities before the first open release of the product, at the time of the study most of companies had already released it. This allowed us to obtain hindsight contributions from CTOs/CEOs, who started facing the drawbacks of the initial lack of structures trying to absorb the technical debt.

Finally, we believe that by analyzing web and mobile companies (which represent the majority of today's startups [25, 22, 26]), important aspects, captured in the behavioral model, can be extended to a large portion of early software startup businesses. In particular we infer that the high-level framework is sufficiently abstracted to capture theoretical patterns common to most startups, whilst the more detailed framework might not be perfectly suitable for all the domains.

The generalization of the theory has been supported by validation of the theoretical framework with Coleman's studies (see Subsection 6.6.1), who has investigated startups with extended operating history. Moreover we considered studies in small companies in view of some similarities (see Appendix A.2.2). As a result, more mature startups and small companies reveal an absence of general control and structures with heavy lack of processes and basic documentation. Indeed, severe lack of resources seems to cause conditions of uncertain development environment, which could be a starting point for a broader generalization of the model to these kind of companies.

[52]Refer to subcategory *uncertain conditions* (CAT2) in Subsection 6.3.3

Chapter 7

Dynamics and evolution of startups

7.1 Overview

This chapter describes the complexity of operational dynamics and the evolution of startups after the first product release. The explanatory analysis is based on empirical data collected during the case study, which has been evaluated using well-known models. In this chapter we aim to address RQ-3, i.e. *What development strategies can be adopted by startups with the aim of facilitating future growth?*

In Section 7.2 we draw a theoretical baseline by comparing traditional software development methodologies in contrast to the approach undertaken by early-stage startups. In Section 7.4 we propose a life-cycle model for early-stage startups, while in Section 7.3 we discuss how complexity of the startups' domain influence the *modus operandi* in software development. Finally we discuss an evolutionary model analyzing the consequences of the accumulated technical debt on the post-release performances and we propose a structured mitigation strategy for the performance drop-down (see Section 7.5). Findings are summarized in Section 7.6.

7.2 Early-stage startups and methodologies

In this section we compare traditional development methodologies with the software development approach undertaken by early stage startups. We aim to set the theoretical basis to understand which methodology could be adopted to control the initial chaos when the company is more mature. This will draw a conceptual baseline which is used throughout the chapter to address the RQ-3.

So far we referred to startup organizations as *reactive* and *flexible* in nature since they are able to change market and business orientation with a fast execution. But from an operative perspective what makes startups be *reactive* and *flexible* is the easy adaptation of the development processes overtime (*flexible* activities easy to change) and the *partiality* of their approach (lack of complete adoption of methodologies), as discussed in Section 6.3. This enables practitioners to operate with a large degree of freedom in order to prioritize only essential

project management activities, as discussed in the Section 6.5. In fact, as presented in the results of mapping study and grounded theory most of the startups operate with development approaches converging to the *Lean Startup methodology*, whilst still presenting chaotic and unpredictable workflows.

Figure 7.1 shows a bi-dimensional plot which considers the dimensions of *flexibility* and *partiality* of well known development methodologies compared with early-stage startups.

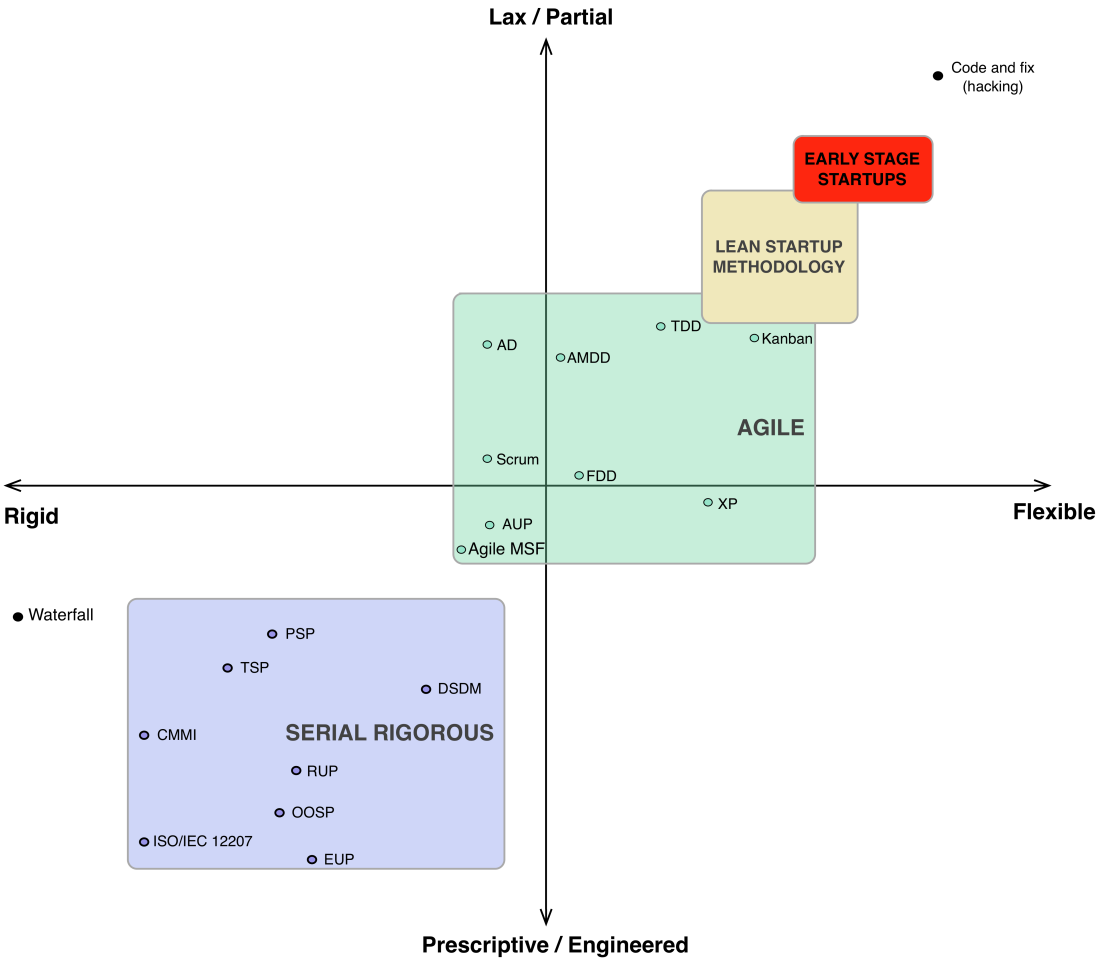


Figure 7.1: Partiality and flexibility, inspired by [5]

Flexibility represents the degree to which development workflows and practices can be tailored whilst *partiality* represents the extent to which development workflows and practices cover all the engineering activities, with a complete description of their implementation. The figure has been adapted from a study performed on agile methodologies [5], adjusting it in face of our empirical data. The figure helps the reader in understanding how early-stage startups belong to the region of flexibility and partiality between *lean startup methodology* (with a

little area of overlapping) and the informal *code and fix approach*[1].

With Figure 7.1 we don't aim to explain all the detailed relations among well-known development strategies, but rather to provide a means to perform visual comparison with existing methodologies in order to motivate the need of a more in-depth analysis of adopted operational strategies in early-stage software startups. Among the different approaches, the most prescriptive and rigid practices are presented by the *Serial rigorous* area, which refers to those detailed methodologies which provide practices to follow in all the engineering activities within well-specified conventions (CMMI, RUP, PSP, DSDM, ...).

By contrast, what emerged from previous studies is that more mature startups are keen towards the implementation of some *Agile methodologies* which are generally more in line with the dynamic contexts (discussed in Chapter 3). However in the early stage of the development (before the beta) we observed that almost none of the Agile practices are actually adopted by practitioners, such as: limiting the work in progress; visualization of the workflow with monitoring and measuring. However some authors suggest that, if the company eventually grows, it will adopt some ad-hoc version of Agile methodologies [2, 84, 85].

In term of flexibility and partiality, the closest area to *early-stage startups* is the Lean Startup methodology [7], partially discussed in *Background* (Section 2.3.3) and Appendix A.2.5. In fact, as reported in Subsection 6.6.5, practitioners rated elements belonging to the category *find the product/market fit quickly* - which subcategories express concepts similar to the principles of *Lean Startup methodology* - as the most useful engineering items for speeding up software development. In particular they referred to fast release cycle based on minimal functionalities, associated with the concept of minimum viable product (MVP).

7.3 Complexity and chaos in startups

Moving faster and handling of disruption[2] characterize the daily operating environment in startups. To understand the reasons of such scenario and obtain insights of adopted solutions we need to introduce[3] the *Cynefin framework* [6].

A good model, to conceptualize the reasons behind the adoption of low-precision development strategies by startups, is the *Cynefin framework* described by its author, Dave Snowden, as a *"sense-making model"*. It is a model used to support decision-making by understanding the domain in which domain organizations are operating. Thus, an *area* of the framework delineates the strategies

[1] Code and fix is the reckless implementation of features and fixing of emerging problems to have the product working as soon as possible.

[2] Disruptive technology is intended as an innovative product which eventually create new markets or value networks better than large bureaucratic companies, as discussed in [24]. This characteristic makes startups an unique world of complex dynamics.

[3] This was only a brief introduction necessary to understand the remaining subsections.

to adopt in decision-making in order to obtain desirable results.

Figure 7.2 represents the *Cynefin* framework. It has four main domains: *known, knowable, complex and chaos*[4]. At the right-hand side of Figure 7.2 there are represented those domains defined as *order area*, and the left-hand side domains defined as *unorder area*.

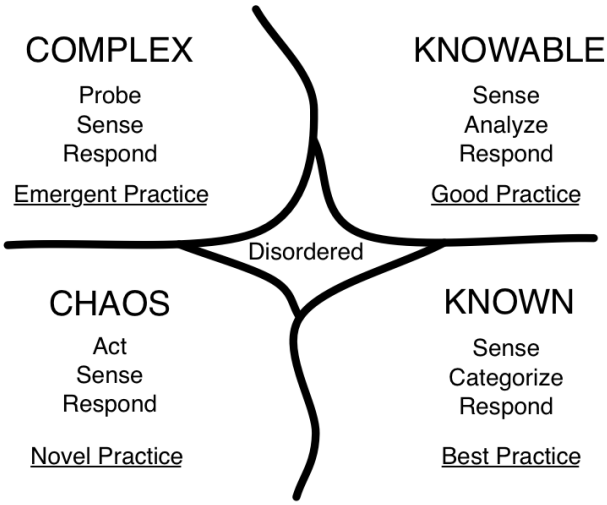


Figure 7.2: Cynefin framework [6].

The *order domain* is composed by *known* and *knowable* areas. The *known* area represents the world of self-evident cause-effect relationships. This is the domain of process reengineering, in which knowledge is captured and embedded in structured processes to ensure consistency (an example might be the development of a basic text-editor, whose technology solutions are easily retrievable and well-documented). As shown in Figure 7.2, in this domain practitioners need to *sense* what are the needs, *categorize* through understanding possible solutions and *respond* by providing the best solution. The *knowable* area represents the world of complicated cause-effect relationship, where expertise are involved to analyze data and then take decisions in accordance with interpretation of that analysis (an example might be the development of a team management tool, in this case an expertise of team management is required to understand the functionalities needed). As shown in Figure 7.2, in this domain practitioners need to *sense* and *analyze* by breaking complex topics in smaller parts and gaining a better understanding of them to finally *respond*.

The *unordered domain* is composed by the *complex* and *chaotic* areas. The *complex* area involves cause-effect relations defined by emergent patterns, which

[4]A fifth domain is presented as the domain of disorder, which is explained later in this subsection.

can be perceived but not predicted; this phenomenon is named *retrospective coherence*. As shown in Figure 7.2, in this domain practitioners need to *probe* first in order to explore the area, obtain *sense* and finally *respond*. Within this space, structured methods would try to seize upon such emergent patterns codifying them into defined procedures, facing with new and different patterns and thus managing them with an improper strategy. By contrary, relying on experts opinions, which are based on historically stable patterns of meaning, will sufficiently prepare researchers to recognize and act upon unexpected patterns. The only decision model viable for this *complex* region is *probing* in order to make potential patterns more visible before taking any decision (an example might be the development of a transportation management system in the harbor of New York city, where solutions can be retrieved only by analysis of previous measured executions of the system within that geographical area).

While in the previous *known*, *knowable* and *complex* domains, relationships between cause and effect were visible, the *chaotic* space involves no such perceivable relations instead. The turbulence of this area makes any attempt of analysis or waiting for patterns to emerge a waste of time. The only possibility to make decision in this space is to *act* as quickly and decisively as possible in order to sense immediately the reaction to that intervention and respond to it accordingly. Unlike the other domains the chaotic space is only temporary since structures and patterns naturally emerge thanks to an evolving operating history. Solutions in this domain are mainly oriented to *code and fix* approach or partial and flexible management practices (as discussed in Section 7.2). As shown in Figure 7.2, in this domain practitioners need to *act* first by means of efficient practices that can create a first attempt, obtain *sense* and finally *respond*.

The domain of disorder is where decision makers have conflicts, looking at the same situation from different points of view. When needed a consensus, the disordered domain has to be reduced in size in order to understand the nature of the situation and the most appropriate response.

7.3.1 Cynefin dynamics in startups

In this subsection we present in which domain startups operate, answering the RQ-3.1. Starting from the ordered systems, we will move towards to the unordered systems characterizing the domains in face of the software startups development environment. Ordered systems assume that by means of best and good practices, we can derive or discover general rules or hypotheses, which can be empirically verified and by which one can create a body of reliable knowledge, able to be developed and expanded. But as revealed from our findings, this assumption does not hold in the arena of early-stage software startups.

The companies we investigated and primary decision-makers in their startups know this: no matter how much they might like things to be ordered, they know that complex factors are always dominant (see Section 6.5). Indeed, according

to Snowden [6], the best option is to recognize and appreciate the freedom of the *unordered* domain, stopping applying methods designed for order and instead focus on legitimate methods that work well in unordered situations.

Startups managers who try to have rigorous control on the development strategies might be reconducted to an example reported by Tom Stewart in [183], describing *"[. . .] a group of graduates were asked to manage a playtime of a kindergarten. They planned, and rationally identified objectives, determining backup and response plans. Trying to order children's play based on rational design principles, they achieved chaos instead. Meanwhile experienced teachers allowed a degree of freedom at the start of the session, and only then, they intervene to stabilize desirable patterns and destabilize undesirable ones"*.

Comparing this simple example to development in startups it is clear that in a dynamic and constantly changing environment, it is possible to provide high-guidelines in the *unordered* domain but not to assume order with strict process definition. It is not possible to consider reductionist approaches to problem solving since every intervening factor (e.g. a developer falls ill) changes the nature of the system. As a result, engineers need to allow a degree of flexibility in order to obtain incremental improvements and stabilization of emergent patterns.

Methods, tools, and techniques of the *known* and *knowable* domains, defined as rigid and prescriptive from software engineering, do not work in software startups. Instead, flexible and reactive methods, designed to stimulate patterns to emerge, increase the number of perspectives and solutions available to a decision maker. Nevertheless, it is not by chance that software startups are more disruptive than established and bureaucratic companies. Moving from complex to chaotic spaces, software startups open up new possibilities of creation, generating the condition for innovation. This dynamic is named *Divergence-Convergence* [6], and it is one of the many movements described by the *Cynefin* framework.

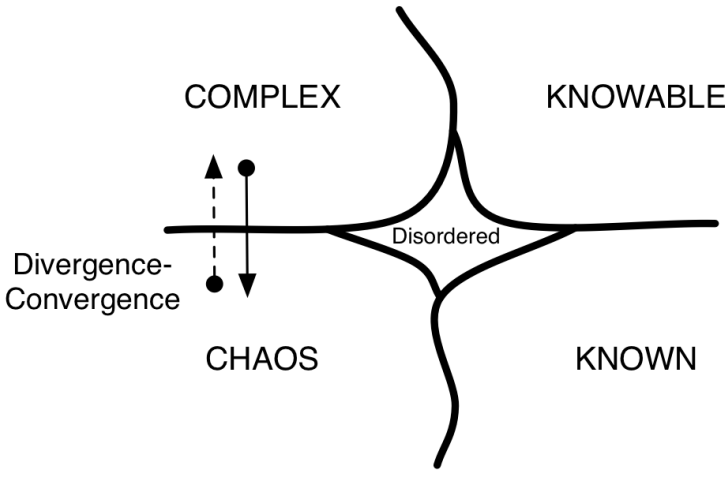


Figure 7.3: Cynefin dynamics [6].

The boundary between chaos and complex spaces is a gradient, where there is always room for interpretation, since there is not an easy way to control when or where one crosses it[5].

Disruption and instability in such area is a natural approach, which can be managed more or less carefully. In this regard, software engineers in startups cannot impose ways of fitting reality into their existing models within unordered domain because those models would be already outdated. Therefore, startups live in between the *chaotic* and *complex* regions. An interviewee claimed: *"We used the scrum board, but when opportunities came out we didn't apply any process anymore, and go straightforward to implementation to take advantage of the hype."* Despite Agile practices appear to be suitable to embrace changes, when the organization crosses from the *complex* area to the *chaotic* border, those practices are often ignored to accommodate the need to release the product to market as quickly as possible. In conclusion, given the Cynefin model, we were able to explore the part of *chaos* and *complexity* where startups operate. As claimed by Snowden, the *chaos* zone is a fertile territory for innovation, where startups take advantage everyday from disruptive ideas [6]. Despite the need to face the accumulated technical debt when the organization start growing (see Subsection 6.3.6), by crossing between the areas of *chaos* and *complexity*, they can start adopting some high-level guidelines to limit the damage in future growth. The effects of operating in this domain and possible solutions will be explained in the next subsections.

7.4 Early-stage startup lifecycle

In order to target the most suitable engineering strategies in startups at different stages of their evolution, it is worth distinguish between different activities carried out in different phases. However, despite many attempts to describe the lifecycle of startups can be found in the literature (see Appendix A.2.6), we were not able to identify any peer-reviewed model fine grained around the early-stage from a software development perspective. For this reason we extracted an early-stage startup lifecycle from our case study data.

A thorough analysis of the case study results revealed a number of product-related *events* which typically occur chronologically to startups in the early-stages[6]:

- *First idea conception (E1)* - the starting point where a first product idea is

[5] *"As a shallow river, anyone at any place can go through the river boundaries. But still it is easy to tell when one has crossed it because one's feet get wet"* [6]. The presence of "shallow rivers" encourages startups to cross over the boundary as much as possible, but still retaining the capacity to monitor and intervene.

[6] We limit the life-cycle to the time-frame where we have enough empirical data.

conceived[7].

- *Start working on product idea (E2)* - the founding team starts to work on the idea realization.
- *First draft of prototype (E3)* - the first prototype of the product is available for internal proof-of-concept.
- *Private alfa-release to closed group (E4)* - a basic version of the product is released to a restricted group of individuals[8].
- *First public open release (E5)* - the first version of the product is made publicly available to potential real customers.
- *Foreign element (E6)* - an external element triggers the growth of new features request, number of users or company size. This event typically causes a loss of initial performances when the increased complexity faces the accumulated technical debt. This event can be mapped to the theoretical category *Triggers for team and user growth*, which belongs to CAT6 in the model.
- *Begin performance recovery* (E7) - when the company takes action to control the initial chaos, the performance starts to be recovered.

Afterwards we mapped the occurrence of the above-mentioned events to the status of the companies which participated to our case study, providing an initial idea of the stage of each startup. Table 7.1 shows a X mark where, at the time of the interview, the specific event E occurred, according to the perception of the respondent.

	E1	E2	E3	E4	E5	E6	E7
C1	X	X	X	X	X	X	-
C2	X	X	X	X	-	-	-
C3	X	X	X	X	X	X	-
C4	X	X	X	X	X	X	X
C5	X	X	X	X	X	X	-
C6	X	X	X	X	X	-	-
C7	X	X	X	X	X	X	-
C8	X	X	X	X	X	X	X
C9	X	X	X	X	X	-	-
C10	X	X	X	X	X	X	-
C11	X	X	X	X	X	X	X
C12	X	X	X	X	X	X	-
C13	X	X	X	X	X	X	-

Table 7.1: Companies and lifecycle events

It can be observed that most of the startups[9], experienced all the *events* up to

[7]Typically the idea conception does not happen at a precise instant of time but is rather over a period of time, more or less extended.

[8]The private alfa can be made available online to private group of users or simply distributed to close group of friends and early adopters.

[9]C2 planned to release the product to the open beta of the product some weeks after the interview.

E5 (*first public open release*). Ten startups experienced a loss of performance triggered by E6 (*Foreign element*), and three of them overcame the initial difficulties and were improving performance after E7 passed (*begin performance recovery*).

The collected information were used to depict a timeline representing the early-stage lifecycle of startups (see Figure 7.4), mapping the status of the companies interviewed according to Table 7.1.

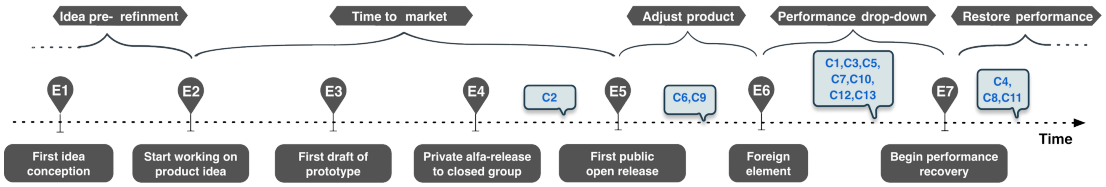


Figure 7.4: Lifecycle model for early-stage startups

A label associated with relevant time-intervals is shown on the upper part of the model. For instance, the period of time between E2 (*start working on product idea*) and E5 (*first public open release*) is marked with the label *time to market*. Following, in the time between E5 and E6, startups typically iterate and improve the product to achieve a better product/market fit (hence *adjust product*), and between E6 and E7, where most of the startups we interviewed belong, they usually experience a *performance drop-down*. For each of the phases, depicted in Figure 7.4, companies have different priorities.

7.5 Evolutionary model

Effects of operating continuously in a *chaotic* and *complex* domain with uncontrolled and unstructured development strategies, lead organizations to experience a temporary drop-down in performance which can be explained by making use of the *Satir Change Model* [184].

The *Satir change model*[10] describes performance behaviour through the definition of five stages (see Figure 7.5): *status quo*, *resistance and chaos*, *integration and practice* and *new status quo*.

[10]The Satir change model is a standard model used to deal with changes in the state of complexity and it consists of five pre-selected and sequenced stages.

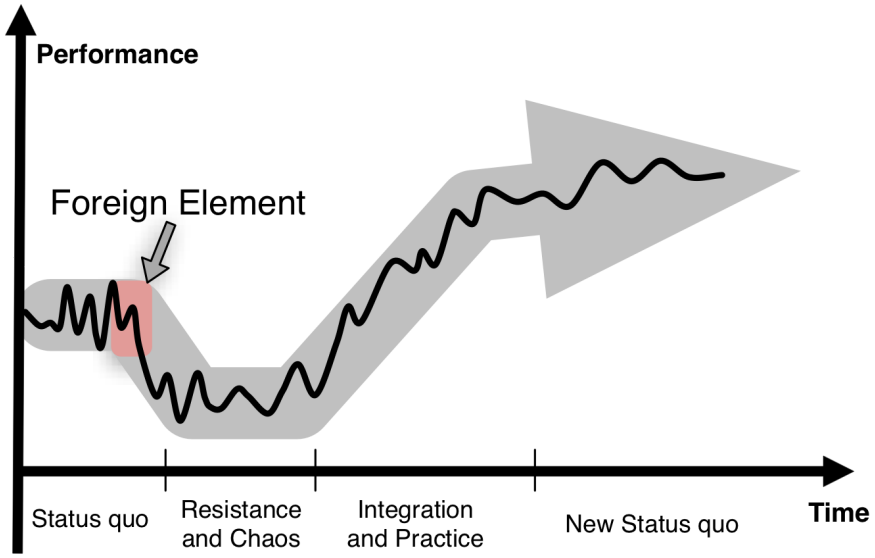


Figure 7.5: Satir model

Observe that the first time period, here referred with the label *status quo*, corresponds to the entire time-span up to E7 of the early-stage lifecycle model previously presented in Figure 7.1. What follows the *event* E7 is here divided in three stages (*resistance and chaos*, *integration and practice*, and *new status quo*), corresponding to *restore performance* in the lifecycle of Figure 7.1.

For the sake of brevity, moving towards the different stages, the model is approximated with its general trend, even though in reality the curve might be more similar to articulated oscillation as presented in Figure 7.5. In the rest of this section we describe each stage separately in relation to the early-stage startups' context. Within the context of early-stage software startups, the first phase - named the *"status quo"* in [184] - represents the time-window where startups are product oriented. They lead their effort and performance in the implementation of one or more products, neglecting any established process, as reported also in [3] and confirmed by our empirical results.

Despite the initial positive performance level, as discussed in the theoretical model (Chapter 6), *speed-up development* generates *technical debt*. When the product start scaling, the weakness and vulnerability of structures and workflows arise. In fact, at the time the *foreign element* occurs (see Figure 7.5), the drop-down in performance is inevitable, leading startups to the *"resistance and chaos"* space. If the amount of previously accumulated technical debt is significant, the *foreign element* (E6 in the early-stage lifecycle model, Figure 7.1) creates a critical mass of discomfort: the organization enters a state where expectations are not fulfilled, and the system becomes disarranged and out-of-control.

The reaction to the generated chaos is an attempt to control it with a gradual integration of high-level guidelines of workflows and processes. Starting by engag-

ing rapidly changing solutions, control and structures of patterns start to emerge, entering into the *integration and practice* stage (at this stage performance start to increase as shown by Figure 7.5). At the same time, systems entering in this stage, are not yet mature to be stable enough and need time to learn and grow into the new state. This time-window is called the *"integration and practice"* stage since it represents the time the team-members are gradually integrating the use of new processes and tools within a new structure of working.

On the other hand, as described in [184], there might also be a rejection to change and return to chaos. This might be caused by the negative reactions of managers who expect to see immediate results, or by time and schedule pressure, which might inhibit the learning process. However, if the integration phase is properly managed, the benefits of new methods become evident and might be experienced as useful, gradually forming a *new status quo*, where initial performance is not only restored, but even improved over time. From here on, the performance growth rate decreases in view of marginal improvements. In fact, focusing on lightweight and agile methodologies, startups might obtain additional achievements within the project development and management, as discussed in [2, 88, 89][11].

This model has correspondence also with the lifecycle model described by Crowne in [10], who synthesized the startup lifecycle in four stages: startup, stabilization, growth and maturity. Starting from the startup stage, it represents the phase in which startups create and refine the initial idea, up to the first release. This time frame is characterized most from the need to assemble a small executive team with the necessary skills to start to build the product. The stabilization phase begins after the first release and it lasts until the product is stable enough for accepting a new customer without causing any overhead on the product development (i.e. being able to treat maintenance and scalability in such a way that the development team maintains the same performance). The growth phase begins with a stable product development process and lasts until when the market size, share and growth rate have been established: all the business processes necessary to support product development and sales. Finally, the startup evolves to a mature organization, where the product development becomes robust and predictable with proven processes for new product inventions. Only then, incremental process improvements can be further adopted. The reader is referred to Appendix A.2.6 for a review of existing startups-related lifecycles.

As described, the lifecycle of Crowne corresponds with the *Satir change model*, only with different names of stages. Indeed, starting from the left side, we can assume that the *startup phase* in [10] represents the initial *status quo* of the *Satir model*, where performance is stable enough to have refinement of the initial

[11]Note that those marginal improvements are reported by the referenced studies. We didn't conduct any evaluation in this concern since out of the scope of our research focus and context area, as discussed in Subsection 7.5.4.

idea up to the first release. From the first release, the foreign element triggers scalability and maintainability issues, which causes the drop-down performance that is in correspondence of the *stabilization* phase (named *Resistence and Chaos* in the first model). Finally the *integration and practice* allows startups to grow, up to the *new status quo* that represents the *maturity* phase in the second model.

The *Satir change model* has also been described in [185] (renamed *"stabilization curve"*) to define the start-up process of a new part of an organization from the perspective of people, process and technology development. Although it represents a different context, we found similar objectives and thus adapted their experiences within our study. The stabilization process, described in [185], presents many similarities with the evolution in which the software startups evolve. Both are focused on addressing stabilization challenges as quickly and effectively as possible, minimizing interruption of day-to-day activities. Therefore, we tried to adapt the stabilization objectives in face of our theoretical framework, and identified patterns. The *stabilization curve* is presented in Figure 7.6.

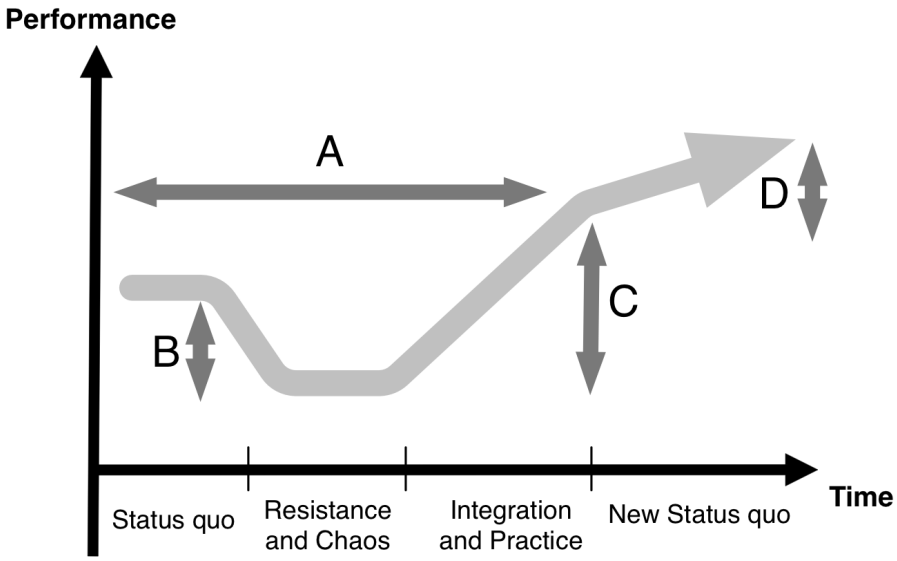


Figure 7.6: Satir model measures

The main scope of the curve, beside contributing to answer to RQ-3.2 and RQ-3.3, is trying to explain how to maximize the overall performance evolution by leveraging the flexibility of methods affecting four measures:

- Minimizing the length of time required to integrate long-term scalable solutions (time-span represented by A).
- Minimizing the depth of performance drop-off during the *resistance and chaos period* (measure represented by B).

- Maximizing the performance level after the initial chaos has been overcome (measure represented by C).
- Maximizing long-term performance and continuous improvement after reached a stage of stabilized processes (measure represented by D). This objective resulted out of the scope of our research since it was retrieved only in studies of more mature startups and not in the early-stage period.

In the following part we explain each of the defined measures in Figure 7.6 to understand their importance in startup companies.

7.5.1 Integrating scalable solutions

As described in the theoretical model (see Chapter 6), the use of an evolutionary development approach with fast iterations and minimal set of functionalities lead startups to maintain effective planning and realistic expectations. Analysis of GT case study (see Subsection 6.3.6) suggests to decrease the time needed to integrate scalable solutions and maintain the long-term balance between technical debt and development speed.

Main contributions within the model describe the advantage of using standard/known technologies aided by: past experience in the domain, support of community and availability of documentation (see Subsection 6.3.5). Additionally, the externalization of complexity to third party solutions has been reported to provide high advantages on maintaining simple workflows, easy to integrate within the existing development environment. Especially the use of standard code guidelines as well as known development frameworks let startups minimize the time to re-engineer the product and obtain higher scalability.

Moreover, practitioners confirmed that entering early in the market and adapting quickly to customers' needs increased chances of stabilization processes and smoothly crossing-over the performance drop-down chasm. In fact, startups preferred building a functioning prototyping with minimal set of functionalities and iterate on it within a fast evolutionary approach, avoiding feasibility study and formal architectural design. Indeed, keeping only the product with minimal functionalities has led startups to integrate engineering activities faster and start moving from a "product-oriented" development focus to integrated "process-oriented" approach, when it has been necessary (see Subsection 6.3.7).

7.5.2 Performance drop-down

In order to minimize the performance drop-down, startups focused on workflows with characteristics of high flexibility and partiality (see Section 7.2), giving to the development team higher freedom and responsibility concerns. In fact all practitioners were responsible of the overall engineering and management activities (see Subsection 6.3.2).

Empowering team-members with self-organized and full-stack capabilities is vital for a software startup. In fact, the most effective management approach, which startups adopt, is the *"embrace and empower"* style (see Section 6.2) (also highlighted by Coleman in [27]). In contrast with the *"command and control"*, *"embrace and empower"* style enhances trust into the development team and carries out tasks with less direct supervision, greater delegation of responsibilities, and a more consensual environment.

Going back to the *Cynefin* framework (see Section 7.3), Snowden also introduced the best configuration of team management approaches within different domains. Figure 7.6 represents the configuration of teams with symbols of dots and connective lines. Full dots and empty dots respectively represent the role of supervision and normal executive team-member. Straight and dashed lines respectively represent strong and weak supervision, coordination and control between defined roles.

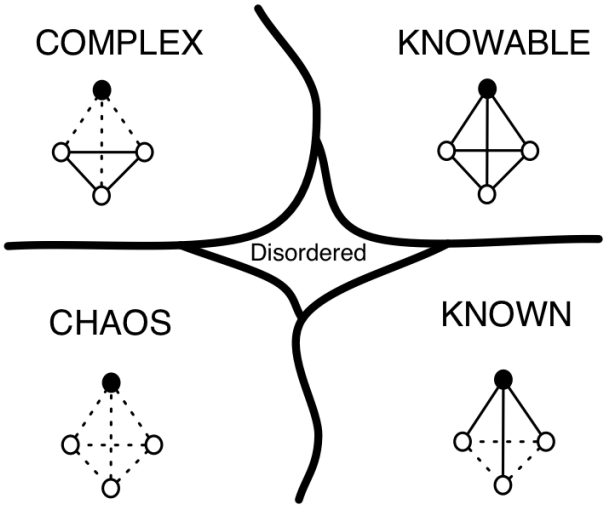


Figure 7.7: Cynefin team management [6]

By mapping startups in the *unordered* domain, we can observe how the defined team management configuration anticipates weak central rules of supervision, control and coordination. As discussed in the previous subsection, structures that attempt to restrict behaviors in the *complex* and *chaotic* space are harmful for the overall performance because of the lack of visibility and feasibility analysis. Indeed, giving power to the team in the *complex* space and acting quickly to understand emerging patterns in the *chaotic* space are the most suitable strategies for software startups.

Yet, the more researchers know which space organizations belong to, the better companies can choose a reasonable strategy to capitalize on the obtained stability. Within the *order* domain, where solutions are well-known or prescriptively

analyzable, strong connection exists with supervising activities in such a way that processes are defined and controlled. On the other hand, in an unordered domain, where taking actions and learning process are the activities to generate solutions, only the freedom and renewal afforded by weak connections can foster performance.

Hence, development teams that present full-stack and self-organizing settings, can handle the fast changing development activities and unexpected wide range of problems that might arise better than conducting a direct supervision with a software development manager. Independent developers with cross-domain knowledge can minimize the performance dropdown in the execution of daily tasks. By contrast a direct supervision would severely hinder the flexibility of the process, as seen in the Cynefin framework.

7.5.3 Improve desirable workflow patterns

Moving ahead into a more mature stage of a startup lifecycle, development starts to be less chaotic and controlled by gradually integrating practices and workflows. In fact, moving from the chaotic towards the complex area, the importance of having knowledge of the process has the main objective of observing which practices were useful during the first release. In this new phase, considering benefits of desirable practices, collecting metrics of project progress and team management can aid organizations to take advantages of emergent patterns , and sustain continuous improvements (see Subsection 6.3.6).

In fact, an interviewee claimed: *"To mitigate this (lack of frameworks) I had to make a schema for other developers when we hired them. We had to do a big refactoring of the codebase, moving it from custom php to Django, normalizing the model and making it stick with the business strategy, and not the other way around. I had the code in different php server communicating via JSON, some engineering horror. Now that we are fixing it, it's really painful. We had to trash some code. However I don't regret that I didn't make this choice sooner, it was the only way."*

Control over the engineering activities and elements that are identified as bottlenecks can sustain the development speed in the long-run. Basic measurements facilitate rational decision-making such as revealing when and if additional developers are needed, or certain activities need to be organized more effectively, or when gaps exist in either a process or tool[12].

Keeping simple and informal workflows in the previous stage helped startups to let useful development strategies emerge naturally. In fact, initiating a minimal project management with informal schedules, naive metrics to trace project progress and task assignment mechanisms helped a gradually integration of practices to control the chaos and the structure of the workflow.

[12]Gaps might be related to: missing functionalities; lack of customizable solutions, ...

In this regard, startups facing with growing number of users and team-members started to replace informal communication with traceable systems and to introduce basic metrics for measuring project and team progress (e.g. Kanban wall, issue tracking systems, automatic testing, ...).

7.5.4 Long-term performance

Maximize long-term performance and continuous improvement after reached a stage of stabilized processes represents the last objective. Briefly, evidences in retrieved studies report that, when startups mature, adopting lightweight methodologies might provide additional benefits. Natural trends seem to tend towards Agile and Lean methodologies to accommodate fast changing environments as discussed in Section 7.2, giving them a flexible starting point to establish repeatable processes. As previously mentioned, this part of startups' evolution is out of the scope of our research. For more details the reader is referred to other studies which consider more mature startups [13, 2, 88, 89, 90].

7.6 Dynamics and evolution summary (RQ-3)

In this subsection we summarize the most important implications discussed in this chapter:

- The development strategy in early-stage startups presents characteristics which are different from existing methodologies (see Section 7.2). The orientation towards reactiveness and flexibility are dictated by uncertain conditions and time-to-market necessities. Nevertheless, the minimal project management activities seem to approach towards the *Lean Startups methodology*, which present several commonalities with the early approach: especially the evolutionary prototyping adjusted according to user feedbacks can be mapped to the MVP.
- Answering RQ-3.1, the nature of such reactiveness and flexibility is grounded in the essence of complex systems, here introduced with the *Cynefin* framework (see Section 7.3). We observed how startups operate across the border of the *complex* and *chaotic* regions, which foster innovation and creativity but at the same time make the imposition of prescriptive and rigid structures ineffective on software development (see Subsection 7.3.1).
- By analysing the empirical data obtained with the case study we proposed an early-stage life cycle model, identifying the main steps involved in the development of the first product. We mapped the 13 companies to the different phases, obtaining an overview of the distribution of sample and internally validating the model (see Section 7.4).
- Answering RQ-3.2, we elaborated a more sophisticated evolutionary curve

based on the *Satir change model*. We characterized the performance drop-down caused by the necessity of returning the accumulated technical debt while expanding the company's operation and structuring mitigation strategies with four software development objectives: minimizing the time required to integrate scalable solutions; minimizing the depth of performance drop-off; maximizing the performance level after the initial chaos has been overcome; and maximizing the long-term performance (see Section 7.5).

- Answering RQ-3.3 (see subsections from 7.5.1 to 7.5.4), the extrapolated development strategies, which helps in reaching long-term objectives, are: integrating scalable solutions with fast iterations and minimal set of functionalities (this allows startups to maintain effective planning and realistic expectations); empowering team members enabling them to operate horizontally in all the activities of the development environment simultaneously; improve desirable workflow patterns through the initiation of a minimal project management over time, as a natural result of emerging activities of tracing project progress and task assignment mechanisms; and finally, only when the chaos has been initially managed, long-term performance can be achieved implementing Agile and Lean practices.

Chapter 8

Summary

In this section we summarize our findings by: addressing the research questions (RQs) and some lessons learned (Section 8.4); discussing the validity of results (Section 8.5); and finally suggesting possible directions for future studies (Section 8.6). The first part of this chapter is structured according to the three RQs initially defined in Subsection 4.1.2:

- RQ-1: What is the state-of-the-art in the SE literature pertaining to engineering activities in startups?
- RQ-2: What is the current state-of-practice related to software development strategies in early-stage startups?
- RQ-3: What development strategies can be adopted by startups with the aim of facilitating future growth?

8.1 RQ 1 - State of the art

To understand the current state-of-the-art pertaining to engineering activities in startups we executed a *systematic mapping study* and evaluated the rigor and relevance of the retrieved studies (see Section 4.2). This approach allowed us to quickly obtain an overview of the research field, identifying prominent contributions and evaluate the quality of the studies. The results of this process, which are discussed in Section 5.1, led to the selection of 37 studies from an initial sample of 943 items. The content of the articles has been discussed in *Related Work* (Chapter 3). The analysis performed on the configuration and the quality of the selected studies, provided evidence that the existing literature is inadequate to understand the underlying phenomenon of software development in startups:

- Only 13 studies are entirely dedicated to the research area of software development in startups while the remaining articles are only partially or marginally mentioning contributions relevant to the field.
- The studies are poorly interrelated by mutual citations and hardly constitute a consistent body of knowledge.

- By visualizing the distribution of articles in the four dimensions of the *systematic map* we were able to provide detailed insights and simultaneously analyze multiple facets (see Subsection 5.1.1).
- Considering that the conducted studies are generally non-rigorous and mediocrely relevant for the industry (see Subsection 5.1.2), it is very unlikely that the findings can have a real impact in real settings.
- We identified only four publications bringing prominent contributions to the field (three of which refer to the same data set), according to a ranking functions based on different features (publication year, venue, research type, contribution type, focus, pertinence, rigor and relevance as detailed in Subsection 5.1.4).
- Although software startups have some common characteristics with other types of companies, from an engineering perspective the combination of different elements poses a set of new challenges for SE, as confirmed by different authors.
- The creation of a coherent body of knowledge about software startups, is restrained by the fact that different authors use the word *'startup'* in different contexts (see discussion in Subsection 5.1.3). Given the ambiguities in the use of the word *startup*, it should be responsibility of the researcher to mention explicitly the particular context affected by the study (mention which has been neglected in most of selected studies).
- Since most of the retrieved studies focus on *mature* startups, we couldn't identify any relevant empirical evidences discussing engineering activities in the very early-stage of the startup creation.

The importance of development challenges is further attested by the proliferation of non-peer reviewed books dedicated to startups, reviewed and discussed in Appendix A.2.5. The literature gap represents a clear signal which empowered our motivation in pursuing scientific research in this field.

8.2 RQ 2 - State of practice

The case study has been conducted in the field by combining different research methodologies (see Section 4.3). It provided a wide set of evidences fostering the understanding of the underlying phenomenon of software development from idea conception to the first open beta release in startups. Thanks to a systematic procedure, the low-level concepts extracted from interviews' transcripts (630 GT *codes*) contributed to the formation of a theoretical model presented and validated in Chapter 6.

The GT process has led to the creation of a formal theory, grounded in the data, describing software development in early-stage startups: *focusing on limited number of suitable functionalities, and adopting partial and rapid evolutionary de-*

velopment approaches, early-stage software startups operate at high development speed, aided by skilled and highly co-located developers. Through these development strategies, early-stage software startups aim to find early a product/market fit within extremely uncertain conditions and severe lack of resources. Nevertheless, by speeding-up the development process, they accumulate technical debt, causing an initial and temporary drop-down in performance before setting off for further growth.

The implications derived from the extensive model created using practitioners' perception are discussed in Section 6.5 and can be summarized as following:

- The most urgent priority of software development is to shorten time-to-market.
- Startups do not apply any standard development methodology: the closest development approach undertaken by early-stage startups tends towards the *Lean startup methodology*.
- The greater part of engineering activities of startups are focused on the implementation while only little attention is given to more conventional activities (project management, requirement specifications, analysis, architecture design, automatic testing, ...).
- The first release of the product includes only a limited set of well suitable functionalities focused on user experience.
- Engineering activities are supported by low-precision artifacts and collaborative tools integrated with the development environment.
- Characteristics of the founding team are the most important determinants of speed.
- The initial lack of structures and processes negatively affect the performances when the company starts growing.
- Startups bring the first product to market in a very short-time and practitioners are satisfied with the adopted software development strategies.

By speeding-up the development process, startups accumulate technical debt, ignoring processes, relying on informal communication and replacing documentation with low-precision artifacts (see Section 6.3). As discussed in Chapter 7, it takes more than just implementing features to run a sustainable project[1]. For startups it is essential to look ahead within short-term deadlines and determine the right product/market fit driven by user feedbacks, using flexible and reactive development approaches and adopting mitigation strategies (see Chapter 6) to contain the severity of accumulated technical debt.

[1]It refers to those development strategies that don't cause a critical accumulation of technical debt.

8.3 RQ 3 - Dynamics and evolution in startups

In Chapter 7 we discussed the operational dynamics of startups analyzing the components of the context and identifying elements which characterize the evolution of software development after the first release. The most relevant findings and contributions are summarized as follows:

- The product-centric structure and *modus operandi* of early-stage startups require the introduction of minimal structures and control when the company starts to grow. By making a comparison with the *flexibility* and *partiality* of other methodologies (see Section 7.2), and analyzing the empirical data, we found that after the first chaotic stages startups are likely to adopt some form of Agile/Lean development methodologies to structure and control the development.
- By using the Cynefin framework to understand complexity of interactions, we have found that startups live and operate in the borderline between the chaotic domain and the complex domain, crossing the border several times. This temporary condition provide benefit for creativity and disruption, and at the same time makes ineffective the attempt to impose any prescriptive and rigid structure over the development activities (see Section 7.3).
- We presented a lifecycle model of early-stage startups which, from a software development point of view, contributes to build a common vocabulary for researchers and provide a framework for grounding the findings (see Section 7.4).
- By looking at the trend of performances after product release, we characterized the performance dropdown provoked by the necessity of returning the accumulated technical debt and simultaneously by expanding the company's size and number of users (see Section 7.5). We elaborated a structured mitigation solution based on the *satir change model* with four objectives for software development (minimizing the time required to integrate scalable solutions; minimizing the depth of performance drop-off; maximizing the performance level after the initial chaos has been overcome; and maximizing the long-term performance).

8.4 Lessons learned

During the intense period of research work we learned several other lessons which do not address any particular research question, but are worth mentioning:

- Online surveys are delicate instruments that must be used with additional care when involving busy practitioners, especially in startups (see discussion in Subsection 5.2.4).

- Semi-structured interviews need to be quickly adjusted to each interview context, adapting the flow to the gathered answers: original questions must serve as a guideline. We observed that, when investigating engineering activities, the most interesting concepts emerged after the researchers delved into each answers by replying with a *why?* to the interviewee's statements.
- The different activities involved in the validation of the theory (see Section 6.6) allowed us to iteratively enhance the quality of the model through a continuous comparison with both the empirical data and the existing literature.
- Scheduling an interview with practitioners of startups, especially in the early stage of product creation, is challenging. The hype, which surrounds successful startups, makes their CEOs and CTOs likely to receive many marketing research surveys and request for interviews which they rarely fill out. Moreover due the high-pressure for releasing the product, they usually work overtime to meet deadlines. However we have found that the best approach to get an interview is by attracting technical people and engage them in informal discussions prior to the formal interview.
- Software development is a *complex* phenomenon and to be fully understood requires the researchers to take into account the complex dynamics with several non-linear elements playing a first-order role. Tailoring an adequate research methodology to a specific complex research problem represents a challenge which shouldn't be underestimated. However, especially in this field, the research skills alone are not enough to perform an effective research if not balanced with field expertise.
- Probably due to space restrictions in journal publications, it's very difficult to find complete examples of grounded theory studies which show details behind the coding process. If not fully documented, the explanatory power of grounded theory faces the risk of being considered not transparent and thus unreliable. To foster the use of grounded theory in SE, researches should use modern tools for knowledge management and make their databases available for an external reviewer to verify that the coding process followed the procedure specified by the research methodology.

Summarizing the findings, we recall the *research goal* that we defined in Subsection 4.1.1, i.e. to understand how software development strategies are engineered by practitioners in startup companies in terms of level of: structure, planning and control of software projects, in the period of time that goes from idea conception to the first open beta release of the software product.

Throughout the execution of the research we inquired practitioners exploring their perception of a perfect hindsight early-stage development strategy. We have found that the key elements that characterize the development strategies are:

- Flexible *structures* of activities, tools and artifacts, which cannot be pre-

scriptively defined. Rather, an evolutionary approach (with a minimal set of suitable functionalities) is the only methodology which allows startups to find early the right product/market fit. Moreover, the partial integration of long-term scalable solutions, as described in Subsection 7.5.1, allows startups to accommodate the fast changing environment and business growth.

- Just-in-time *planning* perspective, keeping simple and minimal project management to improve desirable workflow patterns which occur in the short-term. Infact, only by acting quickly and looking back to results obtained from the market, startups can learn from their mistakes and plan for improving those activities that had positive achievements (see Subsection 7.5.3).
- Distributed *control* settings, empowering team-members with self-organized and full-stack capabilities to enhance greater delegation of responsibilities and more consensual environment, as discussed in Subsection 7.5.2.

In this study we provided a theoretical baseline for other researchers to tackle challenging issues, which characterize this research domain. The next two sections will discuss respectively the generalizability of results and possible future works.

8.5 Validity threats

In this section we discuss the validity of the overall results. Methodology-related threats to validity, i.e. *systematic mapping study* and *grounded theory case study*, were previously discussed respectively in subsections 4.2.8 and 4.3.8.

In the remaining part of this section we structure the validity in four parts, accordingly with Wholin taxonomy [129]:

- External validity - which requires establishing the domain to which the study's findings can be generalized (partially discussed above).
- Internal validity - which is enhanced by establishing causal relationships whereby we attest that certain conditions lead to other conditions, as distinguished from spurious relationships[2].
- Construct validity - which is enforced by clearly specifing operational procedures, and providing an adequate description of constructs[3] that are used to generate the theory.
- Conclusion validity - which represent the degree to which conclusions are reasonable, according to the defined relationships in the data. It involves ensuring adequate sampling procedures, appropriate analytical methods and reliable measurement procedures. Reliability requires to demonstrate that

[2] A *spurious* relationship is a relationship in which two events have no direct causal connection, but rather due to either a coincidence or the presence of confounding factors.

[3] Constructs are complex ideas that we as humans form in order to summarize observations about things that we cannot see directly. For example a construct is *"job satisfaction"*, which to be defined, requires a higher level of abstraction than a concrete object(e.g. a book).

the operations of a study - such as data collection procedures - can be repeated by other researchers, obtaining the same results.

The prominent contribution of this research has been obtained by means of a systematic mapping study, on-line questionnaires and grounded theory study. As suggested by [125], we did our best to maximize internal validity, external validity and ease of replication by integrating complementary empirical research methods. In fact, Dickson claims: *"One experimental study on a topic, no matter how good, cannot, in isolation, demonstrate much of anything conclusively"* [186];

Despite the validity and reliability advantages which derive from the triangulation of the three methodologies [149], we had to consider many validity threats during their design and analysis.

8.5.1 External validity

In this subsection we provide further explanation about the generalizability of results, previously discussed in Section 6.7.

The systematic mapping study helped us to sharpen and delimitate the boundaries of our research area by defining the initial research questions, eliminating irrelevant variations of the context and narrowing the research focus. The flexible design, provided by the GT approach, allowed us to further iterate and adjust the context of the study during its execution.

The first threat related to external validity might be caused by a possible wrong selection of subjects interviewed for the study. This threat profoundly affects GT, since it is a qualitative research method, which makes the use of semi-structured interviews, in fact, centres the research on respondent's opinions. In order to mitigate this threat we selected interviewees which covered positions of CTOs' and CEOs'. Their perspectives on what it is taking place within the organization, were the only meaningful data taken in consideration in the study. In many cases there was no supporting evidence to verify the opinion expressed. As reported in Subsection 5.2.4 questionnaire dedicated to "data triangulation" was not of primary support because of the little time practitioners were able to grant to it. However, researchers have to accept the veracity of what respondents reported during the interviews [29]. Notwithstanding the issues surrounding semi-structured interviews are vital. Within our study, although the reality might be different from what has been described, the reported data are CTOs' and CEOs' perception, which is used to base their decisions.

Finally the comparison with similar frameworks also helped in establishing the domain to which the study's findings can be generalized. In particular the previous framework developed by Coleman in [27] has allowed researchers to find similarities, differences and broader reasoning related to factors that hinder mature processes to be established in startups. Moreover a comparison has been conducted also with the notorious Brook's framework describing the *basic at-*

tacks to the challenges of developing software systems (see Subsection 6.6.1) and other extant literature retrieved during the conduction of SMS[4] (see Subsection 6.6.2). In conclusion, regarding the validation of the subcategories, a continuous re-evaluation of the interview transcripts has been performed (see Subsection 4.3.5). Additional validation instruments have been focused on testing the conditions which define the core category and its characteristics, presented in the high-level framework description (see Subsection 4.3.5 and Section 6.4).

8.5.2 Internal validity

To enhance the internal validity, we created a three-dimensional research framework to perform a triangulation validity procedure. By means of grounded theory approach formed by a *systematic mapping study*, *interviews* and *follow-up questionnaires* we searched for convergence among different sources of information to confirm our theories.

During data collection we strengthen the grounding process of the theory by triangulating qualitative with quantitative collection methods (see Subsection 4.3.3). However, in view of the poor reliability of the questionnaire results, we used only partial integration of quantitative data that were confirmed during the interview process (for detailed explanation refer to Section 5.2) in support of the framework validation. Then, questionnaires constituted only a side-element of this research and their results have never been used alone to draw conclusions, which are instead based on systematic processes of literature review and interview analysis.

By improving and validating the theory definition, we conducted a final comparison of the emergent theory with: extant literature, previously developed models and evaluation of empirical data (see subsections 6.6.1, 6.6.2 and 6.6.4 respectively). With the final model validation we highlighted and examined similarities, contrasts and explanations as discussed in [187]. In this regard Eisenhardt stated: "Tying the emergent theory to existing literature enhances the internal validity, generalizability, and theoretical level of the theory building from case study research [...] because the findings often rest on a very limited number of cases".

Little ambiguities about direction of causal influences have been reduced by introducing arrows to indicate the direction of the causal relation in the theoretical model, both in the high level and low level frameworks. Furthermore, by means of comparison between our theoretical model and extant studies in the broader fields, we revealed that the theory resembles in wider startup development context, as discussed in Section 6.4. Nevertheless, we discovered important confounding factors which might mine the conclusion and generalizability of the study related

[4]Note that the generalizability of results is applied only to those startups that operate in software-intensive activities as described by the theory generation in Subsection 6.4.

to innovation, creativity and market requirements, as discussed in Subsection 6.6.3.

8.5.3 Construct validity

The first important threat to the construct validity of the study is a possible inadequate description of constructs. In order to dismiss this risk, the entire study constructs have been adapted to the terminologies utilized by practitioners and defined at an adequate level for each theoretical conceptualization[5]. Moreover, as described in Subsection 4.3.4, during the coding of interview transcripts the researchers adopted wide and explanatory descriptive labels for theoretical categories, to capture the underlying phenomenon without losing relevant details.

The second important threat is caused by the fact that interviewees might already be aware of the possible emergent theories analyzed by researchers. To reduce the risk that some hypothesis could have been involved in the design of the research, we let mainly participants drive the direction of the case study, as discussed in Section 4.3.

Another threat associated with the construct validity is the evaluation apprehension, which deals with people's rejection attitude towards the assessment of their behaviour. To overcome this risk we didn't attained to first answers from practitioners, but rather went towards details of engineering elements and activities (see Subsection 4.3.2). Moreover, we developed a rigorous data collection protocol to create case study databases and employ integration of multiple data collection methods. Reporting bias was mitigated by packaging all the needed material for conducting new researches in other contexts in order to criticize our findings (interview package with instructions has been made available online[6]. Moreover, two academic supervisors with expertise in the area have conducted a peer-review analysis of framework's constructs.

To control distortion during analysis we made extensive use of memos. To demonstrate that we were aware of personal biases, memos provided us another important function in controlling the quality of data analysis (see Subsection 4.3.4). Through the use of memos and comparative analysis we were able to check if data fitted into the emerging theory and also countering subjectivity that ultimately enhanced the likelihood of producing accurate research findings.

Moreover, the GT categories of the theoretical model are based on the results of interviews which have been conducted by asking a number of questions defined in the interview package and adapted to the context. Given that different questions lead to different answers, the creation of specific categories have been influenced by the particular choice of questions asked during the interview. To

[5]For instance defining *Time shortage*, the researchers explained it in terms of *Investor pressure, CEO/business pressure, Demo presentations at events and internal final deadline* used by most of the interviewees during the study.

[6]Available at `https://github.com/adv0r/BTH-Interview-Package` .

reduce the bias introduced on the final model by choosing certain questions, we: made use of open broad questions, which give to the respondents enough freedom to move across different concepts; used the script only as a high-level guideline to conduct the interview, letting the respondent drive the direction of next questions; iteratively adapted the interview package at each new interview, according to the partial results we were obtaining from the emerging theory.

Finally, throughout the all thesis document we provided the detailed procedures that have been followed in executing the study, to reduce the risk of introducing personal biases. The same approach has been followed when constructing the model, by showing the complete list of raw codes, involved in the generation of categories (see Appendix A.4.3).

8.5.4 Conclusion validity

Grounded theory has been already applied by other researches in similar contexts to attest relationships among conceptualizations of an examined phenomenon (see [2, 188, 29]). Those relationships between sampling and analysis should be verified in such a way that emerging findings remain consistent as further data are collected. To attest this, we explored and tested:

- Each category and the strength of relations between them.
- Hypotheses, derived from and related to the emergent theory.
- Deviant cases to ensure robustness and general applicability.

In particular we were prepared to modify generated categories so that the new data could be adapted into the emerging theory according to the concepts of: *theoretical sampling* and *theoretical saturation* (see Subsection 4.3.4).

According to the *theoretical sampling* concept, we adjusted our research design and the emergent theory until only marginal results were generated (see Subsection 4.3.5). Moreover to enhance reliability of the outcome conceptualizations and relations, the process of coding interviews has been systematically conducted following a detailed process. Then, we asked to other practitioners a confirmation to further validate the generated framework and enhance the reliability of the relations among the identified categories (see ection 4.3.2).

An important issue is related to the fact that the limited number of interviews might not represent the complete scenarios in our context of study. Nevertheless, this issue has been partially mitigated as result of the theoretical saturation concept (see Subsection 4.3.8). This experience is attested by Martin's statement: *"By the time three or four sets of data have been analysed, the majority of useful concepts will have been discovered"* [189]. To obtain major certainty of the obtained results, what we applied was the final validation, discussed in Section 6.6.

Despite we couldn't apply any power analysis procedures[7], in grounded theory the sample size depended on when data saturation occurs. Ramer in [190], comparing quantitative to qualitative researches, states: *"reaching data saturation, which involves obtaining data until no new information emerges, is critical for obtaining applicability in qualitative research"*. To avoid interrupting the research prematurely, after attesting that no more relevant information could be gathered from executing additional interviews, we iterated grounded theory study one more time, verifying that the explanatory power of the core category was fulfilled (see Subsection 4.3.5).

The approach of generating theory allowed us to check emerging categories and their properties by gathering new evidence within an iterative and evolutionary approach, in view of the wide context of the research area that was under investigation.

8.6 Future work

We believe that one of the most urgent priority to start fulfilling the gap in this field is the creation of a common vocabulary to base future researches. In fact, without a consistent body of knowledge, it is difficult to have a strong consensus on the terminology (beginning with the term *startup* itself).

Furthermore, the behavioral model presented in this thesis can be used as a starting point for further studies, where additional researchers can focus on specific issues which appear in the model (low-precision artifacts, integrated online collaborative tools, minimal workflows and informal testing and other issues discussed in Chapter 6) and their impact on the development process.

Other researchers can use the material presented throughout the thesis to reproduce the study with other software startups, adjusting the model presented in this thesis or creating a different one. The interview package[8], presented in Section 4.3.2, can be used for conducting the semi-structured interviews and improved during the execution.

Moreover, with the use of rubrics and theoretical framework (see Section A.5), researchers and practitioners can evaluate the measures of *potential capacity*, *execution speed* and *technical debt* in new studies.

Furthermore the early-stage life cycle model we provided (see Section 7.4) can be further extended and adapted. Additionally, researchers can use the systematic map of existing studies to spot weak areas and topics to tackle. We identified several commonalities between the issues related to software development in startups and the thematic faced by the emerging research area dedicated to *technical*

[7]Procedures used prior to the study to determine how large the sample size is required to be in order to achieve statistical significance.

[8]The interview package can be found online at `https://github.com/adv0r/BTH-Interview-Package` .

debt (a recent article depicts the scenario of research on technical debt [46]). We strongly believe that both emerging disciplines can benefit by a mutual collaborative dialog, especially in understanding how the technical debt influences the growth of the company. In particular, researchers can focus on mitigation strategies that can be adopted in the early-stage to limit the technical debt, with the least possible impact on the time-to-market.

Chapter 9

Conclusions

Startups are able to produce cutting-edge software products with a wide impact on the market, significantly contributing to the global economy. Software development, especially in the early-stage, is at the core of the company's daily activities. Despite their severely high failure-rate, we have found that the quick proliferation of startups is not supported by an adequate and scientific body of knowledge. To be able to intervene on the software development strategy with scientific and engineering approaches, the first step is to understand startups' behavior. Hence, in this research work, we attempted to provide an initial explanation of the underlying phenomenon by means of a systematic mapping of the literature combined with a grounded theory case study, focusing on the early engineering activities from the idea conception to the first open beta release of the product.

Through an intense exploratory research conducted with a systematic approach, we produced a theoretical model grounded into the hindsight knowledge collected among startup practitioners (see Chapter 6) with the aim of explaining how development strategies are engineered by practitioners. The explanatory capability and correctness of the model have been validated by means of systematic comparisons with the state-of-the-art results and empirical data (see Section 6.6). The systematic mapping of the literature, with 37 studies extracted from an initial sample of 943 items, revealed a multi-faceted state-of-the-art inadequate to support software development activities in startup companies (see Subsection 5.1.4). On the other hand, the case study conducted in 13 early-stage startups, provided a broad set of empirical evidences obtained by combining different research methodologies in a grounded theory approach (see Section 4.3).

The overall results of our research confirm that startups possess unique characteristics of uncertainty, lack of resources and time-pressure. These factors influence the software development to an extent that transforms every decision related to its strategies into a difficult trade-off for the company. Moreover, although startups share different characteristics with other similar SE domains (e.g. market-driven development, small companies and web engineering), the unique combination of coexisting factors poses a whole new set of challenges which need to be addressed by primary studies. Especially when bringing the first product

to market, startups' most urgent priority is releasing the product as quickly as possible to verify the product/market fit and to adjust the business and product trajectory according to early feedback and collected metrics. In this stage startups often discard any formal project management, documentation, analysis, planning, testing and other traditional process activities. In fact, practitioners take advantage of an evolutionary prototyping approach, using well-integrated tools and externalizing complexity to third party solutions.

However, the initial gain obtained in terms of flexibility and speed is counterbalanced by the need of restructuring the product and control the engineering activities when the company starts to grow. In fact, if successful, the startup will face a growth in terms of number of customers, employees and product functionalities that leads to the necessity to control the initially chaotic software development environment. In this context the most significant challenge for early-stage startups is finding a sweet spot between being fast enough to enter the market early while controlling the amount of accumulated technical debt. The product-centric development approach, adopted in the early stage, fosters characteristics of *flexibility* and *partiality* in comparison to other traditional SE methodologies. In fact, as confirmed by analyzing the Cynefin framework, the use of prescriptive and rigid structures over the development activities would hinder performance since startups operate across the border between chaotic and complex domain. Therefore, from a software development perspective, working with low-precision engineering activities represents the only viable strategy that startups can follow in the early stage of their lifecycle.

Nevertheless, looking at the growing phase after the first open beta release, a performance dropdown is provoked by the necessity of returning the accumulated technical debt under delicate circumstances, where an increased market demand meets a poorly engineered product and development process. Indeed, to mitigate the lack of established engineering strategies, startups need to integrate scalable solutions from the beginning, and to empower the development team - the primary resource to shorten time-to-market - by an *embrace and empower* management style. Notwithstanding, when the initial approach is no longer sustainable to support the increased software development complexity, startups start adopting workflows that *naturally emerged* during the first phases. Eventually they increase long-term performance by gradually moving from a product-centric approach towards the adoption of self-tailored development practices.

In this thesis we discussed a number of novel challenges for both practitioners and researchers, while presenting a first set of concepts, terms and activities which set the software engineering scene for the rapidly increasing startup phenomenon. By making a comparison with Berry's definition of SE [191], we would like to see the rise of a new discipline - *startup engineering* - which can be defined as *the use of scientific, engineering, managerial and systematic approaches with the aim of successfully developing software systems in startup companies.*

References

[1] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *12th International Conference on Evaluation and Assessment in Software Engineering*, vol. 17, no. 1, 2007, pp. 1–10.

[2] G. Coleman and R. O'Connor, "Using grounded theory to understand software process improvement: A study of Irish software product companies," *Information and Software Technology*, vol. 49, no. 6, pp. 654–667, 2007.

[3] I. Heitlager, R. Helms, and S. Brinkkemper, "A tentative technique for the study and planning of co-evolution in product," in *Software Evolvability, 2007 Third International IEEE Workshop on*, oct. 2007, pp. 42 –47.

[4] A.M.M. Sharif Ullah and J. Tamaki, "Analysis of kano-model-based customer needs for product development," *Journal of System Engineering*, vol. 14, no. 2, pp. 154–172, Jun. 2011.

[5] Scott W. Ambler. Choose the Right Software Method for the Job. [Online]. Available: http://www.agiledata.org/essays/differentStrategies.html (Accessed : Aug. 25, 2012).

[6] C.F. Kurtz and D.J. Snowden, "The new dynamics of strategy: Sense-making in a complex and complicated world," *IBM Systems Journal*, vol. 42, no. 3, pp. 462 –483, 2003.

[7] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[8] Y. Mashiko and V.R. Basili, "Using the gqm paradigm to investigate influential factors for software process improvement," *J. Syst. Softw.*, vol. 36, no. 1, pp. 17–32, Jan. 1997.

[9] D. Smagalla, "The truth about software startups," *MIT Sloan Manage. Rev. (USA)*, vol. 45, no. 2, p. 7, Winter 2004.

[10] M. Crowne, "Why software product startups fail and what to do about it," in *Proceedings of 2002 IEEE International Engineering*, 2002, pp. 338–343.

[11] A. Maccormack, "How Internet Companies Build Software," *MIT Sloan Management Review*, pp. 75–84, 2001.

[12] K.M. Eisenhardt and S.L. Brown, "Time pacing: competing in markets that won't stand still." *Harvard Business Review*, vol. 76, no. 2, pp. 59–69, 1998.

[13] S.M. Sutton, "The role of process in software start-up," *IEEE Software*, vol. 17, no. 4, pp. 33–39, Aug. 2000.

[14] M. Andreessen. Why software is eating the world. [Online]. Available: http://goo.gl/6CEVN (Accessed : Aug. 25, 2012).

[15] T. Kane, "The Importance of Startups in Job Creation and Job Destruction," Kauffman Foundation, Tech. Rep., July 2010.

[16] G. Coleman, "An empirical study of software process in practice," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, vol. 00, 2005, p. 315c.

[17] S. Blank, *The four steps to the epiphany*. Cafepress. com, 2005.

[18] M. Kajko-Mattsson and N. Nikitina, "From Knowing Nothing to Knowing a Little: Experiences Gained from Process Improvement in a Start-Up Company," in *2008 International Conference on Computer Science and Software Engineering*. IEEE, 2008, pp. 617–621.

[19] T.W. Archibald, L.C. Thomas, and E. Possani, "Keep or return? Managing ordering and return policies in start-up companies," *European Journal of Operational Research*, vol. 179, no. 1, pp. 97–113, May 2007.

[20] D. Storey, *Entrepreneurship and the New Firm*. Croom Helm, 1982.

[21] A.B. Perkins and M.C. Perkins, *The Internet Bubble: Inside the Overvalued World of High-Tech Stocks – And What You Need to Know to Avoid the Coming Catastrophe*. HarperInformation, 1999.

[22] M. Marmer, B.L. Herrmann, E. Dogrultan, R. Berman, C. Eesley, and S. Blank, "Startup Genome Report Extra: Premature Scaling," Startup Genome, Tech. Rep., 2011.

[23] R. Grimaldi and A. Grandi, "Business incubators and new venture creation: an assessment of incubating models," *Technovation*, vol. 25, no. 2, pp. 111–121, Feb. 2005.

[24] C.M. Christensen, *The Innovator's Dilemma*. Harvard Business School Press, 1997.

[25] Startup types distribution. [Online]. Available: http://techcrunch.com/2012/02/17/crunchbase/ (Accessed : Aug. 25, 2012).

[26] P. Allen, M. Ramachandran, and H. Abushama, "PRISMS: an approach to software process improvement for small to medium enterprises," in *Third International Conference on Quality Software*. IEEE, 2003, pp. 211–214.

[27] G. Coleman and R.V. O'Connor, "An investigation into software development process formation in software start-ups," *Journal of Enterprise Information Management*, vol. 21, no. 6, pp. 633–648, 2008.

[28] G. Chroust, "What is a software process?" *Journal of Systems Architecture*, vol. 42, no. 8, pp. 591–600, Dec. 1996.

[29] G. Coleman and R. Oconnor, "Investigating software process in practice: A grounded theory perspective," *Journal of Systems and Software*, vol. 81, no. 5, pp. 772–784, May 2008.

[30] J. Bach, "Microdynamics of process evolution," *Computer*, vol. 31, pp. 111–113, 1998.

[31] K. Martin and B. Hoffman, "An open source approach to developing software in a small organization," *Software, IEEE*, vol. 24, no. 1, pp. 46 –53, Jan. 2007.

[32] A. Cockburn, *Surviving Object-Oriented Projects*. Addison-Wesley Professional, 1998.

[33] M. Tanabian, "Building high-performance team through effective job design for an early stage software start-up," in *Management Conference*, 2005, pp. 789–792.

[34] S. Chorev and A.R. Anderson, "Success in Israeli high-tech start-ups; Critical factors and process," *Technovation*, vol. 26, no. 2, pp. 162–174, Feb. 2006.

[35] M. Kakati, "Success criteria in high-tech new ventures," *Technovation*, vol. 23, no. 5, pp. 447–457, May 2003.

[36] M. Marmer, B.L. Herrmann, E. Dogrultan, R. Berman, C. Eesley, and S. Blank, "The startup ecosystem report 2012," Startup Genome, Tech. Rep., 2012.

[37] C. Alves, S. Pereira, and J. Castro, "A Study in Market-Driven Requirements Engineering," Universidade Federal de Pernambuco, Tech. Rep., 2006.

[38] O.D. Johan Natt, "Elicitation and management of user requirements in market-driven software development," Ph.D. dissertation, Department of Communication Systems Lund Institute of Technology, 2002.

[39] P. Sawyer, I. Sommerville, and G. Kotonya, "Improving market-driven re processes," in *International Conference on Product-Focused Software Process Improvement (Profes '99)*, 1999.

[40] C. Potts, "Invented requirements and imagined customers: requirements engineering for off-the-shelf software," in *Requirements Engineering, 1995., Proceedings of the Second IEEE International Symposium on*, mar 1995, pp. 128 – 130.

[41] L. Karlsson, Å.G. Dahlstedt, J.N.O. Dag, B. Regnell, and A. Persson, "Challenges in market-driven requirements engineering - an industrial interview study," in *Proceedings of the Eighth International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'02*, 2002.

[42] D. A., "Study of current practices in marketdriven requirements engineering," in *Third Conference for the Promotion of Research in IT*, University Colleges Sweden, 2003.

[43] M. Keil and E. Carmel, "Customer-developer links in software development," *Commun. ACM*, vol. 38, no. 5, pp. 33–44, May 1995.

[44] W. Cunningham. The WyCash Portfolio Management System. [Online]. Available: http://c2.com/doc/oopsla92.html (Accessed : Sep. 15, 2012).

[45] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, ser. MTD '11. New York, NY, USA: ACM, 2011, pp. 1–8.

[46] C. Izurieta, A. Vetrò, and N. Zazworka, "Organizing the technical debt landscape," in *Workshop on Managing Technical Debt (MTD), 2012 Third International*, 2012, pp. 23–26.

[47] Third international workshop on Managing Technical Debt. [Online]. Available: http://www.sei.cmu.edu/community/td2012/ (Accessed : Sep. 15, 2012).

[48] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, New York, NY, USA, 2010, pp. 47–52.

[49] I. Richardson and C. Gresse von Wangenheim, "Why Are Small Software Organizations Different?" *IEEE Software*, pp. 18–22, 2007.

[50] A. Valtanen, "Big Improvements with Small Changes : Improving the Processes of a Small Software Company," *Product-Focused Software Process Improvement*, no. 2006, pp. 258–272, 2008.

[51] F. Mc Caffery, P. Taylor, and G. Coleman, "Adept: a unified assessment method for small software companies," *Software, IEEE*, vol. 24, no. 1, pp. 24–31, 2007.

[52] K. Dangle, P. Larsen, M. Shaw, and M. Zelkowitz, "Software Process Improvement in Small Organizations: A Case Study," *IEEE Software*, vol. 22, no. 6, pp. 68–75, 2005.

[53] Y. Deshpande and S. Hansen, "Web engineering: creating a discipline among disciplines," *IEEE Multimedia*, vol. 8, no. 2, pp. 82–87, 2001.

[54] E. Mendes and M. Nile, *Web Engineering*. Springer, Jan. 2005, vol. 4, no. 3.

[55] R. Baskerville, L. Levine, J. Pries-Heje, and S. Slaughter, "How internet software companies negotiate quality," *Computer*, vol. 34, no. 5, pp. 51–57, May 2001.

[56] H. Ran, W. Zhuo, and X. Jianfeng, "Web Quality of Agile Web Development," *2009 IITA International Conference on Services Science, Management and Engineering*, pp. 426–429, Jul. 2009.

[57] D. Reifer, "Web Development: Estimating Quick-to-Market Software," *Software, IEEE*, pp. 57–64, 2000.

[58] M. Sulayman, C. Urquhart, E. Mendes, and S. Seidel, "Software process improvement success factors for small and medium Web companies: A qualitative study," *Information and Software Technology*, vol. 54, no. 5, pp. 479–500, May 2012.

[59] S. Blank and B. Dorf, *The Startup Owner's Manual: The Step-by-Step Guide for Building a Great Company*. K&S Ranch Publishing LLC, 2012.

[60] J. Grudin, "Interactive systems: bridging the gaps between developers and users," *Computer*, vol. 24, no. 4, pp. 59 –69, april 1991.

[61] V. Basili and A. Turner, "Iterative Enhancement: A Practical Technique for Software Development," *IEEE Transactions on Software Engineering*, vol. 1, no. 4, pp. 390–396, 1975.

[62] E. Carmel, "A Process Model for Packaged Software Development," *Engineering Management, IEEE*, vol. 42, no. 1, pp. 50–61, 1995.

[63] H.L. Schachter, "The role played by Frederick Taylor in the rise of the academic management fields," *Journal of Management History*, vol. 16, no. 4, pp. 437–448, 2010.

[64] N. Ali and H. Edison, "Towards innovation measurement in software industry," Master's thesis, Blekinge Institute of Technology, May 2010.

[65] P. Banerjee, "Some indicators of dynamic technological competencies: understanding of Indian software managers," *Technovation*, vol. 23, no. 7, pp. 593–602, Jul. 2003.

[66] P. Brudlo, "Entrepreneurship in internet business," in *1st International Conference on Information Technology*, May 2008, pp. 1 –4.

[67] T. Burger-Helmchen, "Capabilities in small high-tech firms: a case of plural-entrepreneurship," *Journal of Small Business and Enterprise Development*, vol. 16, no. 3, pp. 391 – 405, 2009.

[68] O.P. Hilmola, P. Helo, and L. Ojala, "The value of product development lead time in software startup," *System Dynamics Review*, vol. 19, no. 1, pp. 75–82, 2003.

[69] O.P. Hilmola, "Question of software start-up finance: system dynamics simulation analysis," *World Rev. Sci. Technol. Sustain. Dev. (Switzerland)*, vol. 6, no. 2-4, pp. 204 – 16, 2009.

[70] E. Carmel, "Time-to-completion in software package startups," *Proceedings of the System Sciences, 1994.*, pp. 498–507, 1994.

[71] G. Coleman, "eXtreme Programming (XP) as a "Minimum" Software Process : A grounded theory," in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, pp. 4–5.

[72] G. Coleman, "Practice Not Process : Improving the Capability of Software Startups," 2004, proposal at Department of Computing and Mathematics, Duldalk Institute of Technology.

[73] M. Lehman, "Programs, life cycles, and laws of software evolution," in *Proceedings of the IEEE*, vol. 68, no. 9, Sep. 1980, pp. 1060 – 1076.

[74] R. Banker and G. Davis, "Software development practices, software complexity, and software maintenance performance: A field study," *Management Science*, 1998.

[75] P. Thiel. Notes on CS183. [Online]. Available: http://blakemasters.tumblr.com/post/21742864570/peter-thiels-cs183-startup-class-6-notes-essay (Accessed : Aug. 25, 2012).

[76] S. Ambler, "Lessons in agility from Internet-based development," *IEEE Software*, no. April, pp. 66–73, 2002.

[77] P. Tingling, "Extreme programming in action: a longitudinal case study," *Proceedings of the 12th international conference*, pp. 242–251, 2007.

[78] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[79] A. da Silva and F. Kon, "Xp south of the equator: An experience implementing xp in brazil," *Extreme Programming and Agile Processes*, pp. 10–18, 2005.

[80] R. Deias and G. Mugheddu, "Introducing XP in a start-up," *European Internet Services Company*, 2002.

[81] D. Yoffie, "Building a company on Internet time: Lessons from netscape," *California Management Review*, vol. 4, no. 3, 1999.

[82] N. Gautam and N. Singh, "Lean product development: Maximizing the customer perceived value through design change (redesign)," *International Journal of Production Economics*, vol. 114, no. 1, pp. 313–332, Jul. 2008.

[83] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods," *Relatório Técnico, Finlândia*, 2002.

[84] M. Taipale, "Huitale - A story of a Finnish lean startup," in *Proceedings of the First International Conference*, vol. 65 LNBIP, Helsinki, Finland, 2010, pp. 111–114.

[85] K. Kuvinka, "Scrum and the Single Writer," in *Proceedings of Technical Communication Summit*, no. May, 2011, pp. 18–19.

[86] C. Midler and P. Silberzahn, "Managing robust development process for high-tech startups through multi-project learning: The case of two European start-ups," *International Journal of Project Management*, vol. 26, no. 5, pp. 479–486, Jul. 2008.

[87] S. Yogendra, "Aligning business and technology strategies: a comparison of established and start-up business contexts," in *Management Conference*, 2002.

[88] J. Zettel, F. Maurer, J. Münch, and L. Wong, "LIPE: a lightweight process for e-business startup companies based on extreme programming," *Product Focused Software Process Improvement*, pp. 255–270, 2001.

[89] E. Deakins and S. Dillon, "A helical model for managing innovative product and service initiatives in volatile commercial environments," *International Journal of Project Management*, vol. 23, no. 1, pp. 65–74, Jan. 2005.

[90] M. Fayad, "Process assessment considered wasteful," *Communications of the ACM*, vol. 40, no. 11, pp. 125–128, 1997.

[91] J. Mater and B. Subramanian, "Solving the software quality management problem in Internet startups," in *Proceedings of the 18th annual pacific northwest software quality conference*, 2000, pp. 297–306.

[92] B. Mirel, "Product, process, and profit: the politics of usability in a software venture," *ACM Journal of Computer Documentation (JCD)*, vol. 24, no. 4, pp. 185–203, 2000.

[93] E. Kim and S. Tadisina, "Factors impacting customers' initial trust in e-businesses: an empirical study," in *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, vol. 07, 2005, pp. 1–10.

[94] R. Hanna and T.U. Daim, "Information technology acquisition in the service sector," *International Journal of Services Sciences*, vol. 3, no. 1, p. 21, 2010.

[95] S. Jansen, S. Brinkkemper, and I. Hunink, "Pragmatic and Opportunistic Reuse in Innovative Start-up Companies," *Software, IEEE*, pp. 42–49, 2008.

[96] D. Wall, "Using open source for a profitable startup," *Computer*, vol. 34, no. 12, pp. 158 –160, dec 2001.

[97] L. Bean and D.D. Hott, "Wiki: A speedy new tool to manage projects," *Journal of Corporate Accounting & Finance*, vol. 16, no. 5, pp. 3–8, Jul. 2005.

[98] Rob Walling, *Start Small, Stay Small: A developer's Guide to Launching a Startup.* The Numa Group, 2010.

[99] M. Marmer, B.L. Herrmann, E. Dogrultan, R. Berman, C. Eesley, and S. Blank, "Startup Genome Report Extra: Premature Scaling," Startup Genome, Tech. Rep., 2011.

[100] Microsoft, *Microsoft Application Architecture Guide.* Microsoft Press, 2009.

[101] V.R. Basili, "Software modeling and measurement: the goal/question/metric paradigm," Institute for Advanced Computer Studies - University of Maryland, College Park, MD, USA, Tech. Rep., 1992.

[102] Engineering Village. [Online]. Available: http://www.engineeringvillage2.org/ (Accessed : Aug. 25, 2012).

[103] Colin Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers.* John Wiley and Sons, 2009.

[104] M. Kasunic, "Designing an Effective Survey," Carnegie Mellon Software Engineering Institute, Tech. Rep., 2005. [Online]. Available: www.sei.cmu.edu/pub/documents/05.reports/pdf/05hb004.pdf

[105] B.G. Glaser, *Theoretical sensitivity : advances in the methodology of grounded theory.* Sociology Press, 1978, vol. 2.

[106] T. Dyba, B. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *Software, IEEE*, vol. 22, no. 1, pp. 58 – 65, Feb 2005.

[107] B. Kitchenham, T. Dyba, and M. Jorgensen, "Evidence-based software engineering," in *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, May 2004, pp. 273 – 281.

[108] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, 2007.

[109] T. Dybå, V.B. Kampenes, and D.I. Sjø berg, "A systematic review of statistical power in software engineering experiments," *Information and Software Technology*, vol. 48, no. 8, pp. 745–755, Aug. 2006.

[110] D. Budgen, M. Turner, P. Brereton, and B. Kitchenham, "Using Mapping Studies in Software Engineering," in *Proceedings of PPIG 2008*. Lancaster University, 2008, pp. 195–204.

[111] S. Rumsey, *How to find information : a guide for researchers*. McGraw-Hill, 2008.

[112] ACM Search. [Online]. Available: http://dl.acm.org/advsearch.cfm (Accessed : Aug. 25, 2012).

[113] IEEExplore. [Online]. Available: http://ieeexplore.ieee.org/ (Accessed : Aug. 25, 2012).

[114] Web of Knowledge. [Online]. Available: http://wokinfo.com/ (Accessed : Aug. 25, 2012).

[115] Scopus. [Online]. Available: http://www.scopus.com/ (Accessed : Aug. 25, 2012).

[116] Google Scholar. [Online]. Available: http://scholar.google.com/ (Accessed : Aug. 25, 2012).

[117] BibDesk. [Online]. Available: http://bibdesk.sourceforge.net/ (Accessed : Aug. 25, 2012).

[118] T. Dybåand T. Dingsø yr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*, vol. 50, no. 9-10, pp. 833–859, Aug. 2008.

[119] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *Software Engineering, IEEE Transactions on*, vol. 33, no. 1, pp. 33 –53, Jan. 2007.

[120] A. Sayers, "Tips and tricks in performing a systematic review," *Br J Gen Practice*, vol. 1, no. 57, pp. 542–759, 2007.

[121] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, vol. 16, no. 3, pp. 365–395, Oct. 2010.

[122] M. Ivarsson and T. Gorschek, "Technology transfer decision support in requirements engineering research: a systematic review of rej," *Requir. Eng.*, vol. 14, no. 3, pp. 155–175, Jun. 2009.

[123] M. Unterkalmsteiner, T. Gorschek, A. Islam, C.K. Cheng, R. Permadi, and R. Feldt, "Evaluation and measurement of software process improvement – a systematic literature review," *Software Engineering, IEEE Transactions on*, vol. 38, no. 2, pp. 398 –424, March 2012.

[124] T. Saracevic, "Evaluation of evaluation in information retrieval," in *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, ser. SIGIR '95. New York, NY, USA: ACM, 1995, pp. 138–146.

[125] M. Wood, J. Daly, J. Miller, and M. Roper, "Multi-method research: an empirical investigation of object-oriented technology," *Journal of Systems and Software*, vol. 48, no. 1, pp. 13–26, Aug. 1999.

[126] O.W. Bertelsen, "Toward A Unified Field Of SE Research And Practice," *IEEE Software*, vol. 14, no. 6, pp. 87–88, 1997.

[127] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[128] A.L. Strauss and J.M. Corbin, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, P. Labella, Ed. Sage Publications, 1998, vol. 2nd.

[129] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[130] C.W. Dawson, *Projects in Computing and Information Systems A Student ' s Guide*. Pearson Prentice Hall, 2009.

[131] Rewards for Startups. [Online]. Available: http://rewards.softwarestartups.org/ (Accessed : Aug. 25, 2012).

[132] Mashape. [Online]. Available: http://mashape.com (Accessed : Aug. 25, 2012).

[133] Blomming. [Online]. Available: http://blomming.com (Accessed : Aug. 25, 2012).

[134] Searcheeze. [Online]. Available: http://www.searcheeze.com (Accessed : Aug. 25, 2012).

[135] Proliker. [Online]. Available: http://proliker.com (Accessed : Aug. 25, 2012).

[136] Wedding Snaps. [Online]. Available: http://weddingsnap.com (Accessed : Aug. 25, 2012).

[137] Podium. [Online]. Available: http://podium.younoodle.com (Accessed : Aug. 25, 2012).

[138] Mangatar. [Online]. Available: http://www.mangatar.net (Accessed : Aug. 25, 2012).

[139] Circleme. [Online]. Available: http://circleme.com (Accessed : Aug. 25, 2012).

[140] Timedoctor. [Online]. Available: http://www.timedoctor.com (Accessed : Aug. 25, 2012).

[141] The Beta Family. [Online]. Available: http://thebetafamily.com (Accessed : Aug. 25, 2012).

[142] Next. [Online]. Available: http://www.nextopenspace.it (Accessed : Aug. 25, 2012).

[143] Amen. [Online]. Available: https://getamen.com. (Accessed : Aug. 25, 2012).

[144] Jonh W. Creswell and Dana Miller, "Determining validity in qualitative inquiry," *Theory into practice*, vol. 39, no. 3, pp. 124–130, 2000.

[145] MIT License. [Online]. Available: http://www.opensource.org/licenses/mit-license.php/ (Accessed : Aug. 25, 2012).

[146] C. Giardino and N. Paternoster. Interview Package. [Online]. Available: https://github.com/adv0r/BTH-Interview-Package, [Aug. 25, 2012).

[147] H.M. Edwards, S. McDonald, and S. Michelle Young, "The repertory grid technique: Its place in empirical software engineering research," *Information and Software Technology*, vol. 51, no. 4, pp. 785–798, Apr. 2009.

[148] G.A. Kelly, *A Theory of Personality: The Psychology of Personal Constructs*. Norton, 1963, vol. 1.

[149] R.K. Yin, *Case study research: design and methods*, R.K. Yin, Ed. Sage Publications, Inc, 1994, vol. 1, no. 3.

[150] W.J. Orlikowski, "CASE Tools as Organizational Change: Investigating Incremental and Radical Changes in Systems Development," *MIS Quarterly*, vol. 17, no. 3, p. 309, Sep. 1993.

[151] G. Coleman, *Software process in practice: A Grounded Theory of the irish software industry*. Springer, 2006.

[152] J. Hietala, J. Kontio, J.P. Jokinen, and J. Pyysiainen, "Challenges of software product companies: results of a national survey in finland," in *Software Metrics, 2004. Proceedings. 10th International Symposium on*, sept. 2004, pp. 232 – 243.

[153] A. Bryant and K. Charmaz, *The SAGE handbook of grounded theory*, A. Bryant and K. Charmaz, Eds. SAGE Publications, 2007.

[154] List of cognitive biases (Wikipedia). [Online]. Available: http://en.wikipedia.org/wiki/List_of_cognitive_biases (Accessed : Aug. 25, 2012).

[155] J.G. Brodman and D.L. Johnson, "What small business and small organizations say about the cmm: experience report," in *Proceedings of the 16th international conference on Software engineering*, ser. ICSE. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 331–340.

[156] R. Wieringa, N. Maiden, N. Mead, and C. Rolland, "Requirements engineering paper classification and evaluation criteria: a proposal and a discussion," *Requir. Eng.*, vol. 11, no. 1, pp. 102–107, Dec. 2005.

[157] M. Shaw, "Writing good software engineering research papers," in *International Conference on Software Engineering, Proceedings. 25th*, May 2003, pp. 726 – 736.

[158] M. Häsel, N. Breugst, and T. Kollmann, "IT Competence in Internet Founder Teams An Analysis of Preferences and Product Innovativity," *Business & Information System Engineering*, vol. 52, no. 4, pp. 210–217, 2010.

[159] R. Stanfill and T. Astleford, "Improving Entrepreneurship Team Performance through Market Feasibility Analysis, Early Identification of Technical Requirements, and Intellectual Property Support," in *American Society for Engineering Education Annual Conference*, 2007.

[160] D. Wood, "Open Source Software Strategies for Venture-Funded Startups," MIND Laboratory, University of Maryland, Tech. Rep. TR-MS1287, 2005.

[161] H.J. Steenhuis and E. de Bruijn, "Innovation and technology based economic development: Are there short-cuts?" in *Management of Innovation and Technology, 2008. ICMIT 2008. 4th IEEE International Conference on*, Sept. 2008, pp. 837 –841.

[162] S.C. Li, "The role of value proposition and value co-production in new internet startups: How new venture e-businesses achieve competitive advantage," in *Management of Engineering and Technology, Portland International Center for*, Aug. 2007, pp. 1126 –1132.

[163] S.l. Lai, "Chinese Entrepreneurship in the Internet Age : Lessons from Alibaba.com," *World Academy of Science, Engineering and Technology*, vol. 72, pp. 405–411, 2010.

[164] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: a systematic review," *Empirical Software Engineering*, vol. 15, no. 1, pp. 91–118, Dec. 2009.

[165] C. von Wangenheim, A. Anacleto, and C. Salviano, "Helping small companies assess software processes," *Software, IEEE*, vol. 23, no. 1, pp. 91 –98, Feb. 2006.

[166] Crunchbase. [Online]. Available: http://www.crunchbase.com/ (Accessed : Aug. 25, 2012).

[167] M.L.S. Thomas, B. Jabine, "Cognitive aspects of survey methodology: Building a bridge between disciplines," National Research Council, Tech. Rep., 1984.

[168] E.M. Rogers, *Diffusion of Innovations*. Free Press, 1995.

[169] C.M. Christensen and M.E. Raynor, *The Innovator's Solution*. Harvard Business School Press, 2003.

[170] J. Offutt, "Quality attributes of web software applications," *IEEE Softw.*, vol. 19, no. 2, pp. 25–32, Mar. 2002.

[171] J.A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House, 2000, vol. 12.

[172] J. Pearl, "Why there is no statistical test for confounding, why many think there is, and why they are almost right," UCLA Cognitive Systems Laboratory, Tech. Rep., 1998.

[173] M. Staples, M. Niazi, R. Jeffery, A. Abrahams, P. Byatt, and R. Murphy, "An exploratory study of why organizations do not adopt CMMI," *Journal of Systems and Software*, vol. 80, no. 6, pp. 883–895, Jun. 2007.

[174] The Inventors of Six Sigma. [Online]. Available: http://web.archive.org/web/20051106025733/http://www.motorola.com/content/0,,3079,00.html (Accessed : Aug. 25, 2012).

[175] F. Steven and S. Montgomery, "Fisher's fundamental theorem of natural selection," *TREE*, vol. 7, no. 3, pp. 92–95, 1992.

[176] F. Brooks Jr, "No Silver Bullet — Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[177] R. Cooper, "An Investigation into the New Product Process : Steps, Deficiencies, and Impact," *Journal of product innovation management*, vol. 3, no. 2, pp. 71–85, 1986.

[178] T. DeMarco, *Peopleware: Productive Projects and Teams (Second Edition)*. Dorset House; 2nd edition, 1999.

[179] S. Adolph and P. Kruchten, "Reconciling Perspectives: How People Manage the Process of Software Development," in *AGILE Conference*. IEEE, Aug. 2011, pp. 48–56.

[180] A. Cockburn, "Characterizing people as non-linear, first-order components in software development." Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, Tech. Rep., 1999.

[181] G. Cugola and C. Ghezzi, "Software processes: a retrospective and a path to the future," *Software Process: Improvement and Practice*, vol. 4, no. 3, pp. 101–123, 1998.

[182] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[183] B.T.A. Stewart and N. Issue, "How to Think With Your Gut Business 2 . 0," *How to Think With Your Gut*, pp. 1–6, 2002.

[184] G. Cheung and C. Chan, "The satir model and cultural sensitivity: A hong kong reflection," *Contemporary Family Therapy*, vol. 24, no. 1, pp. 199–215, 2002.

[185] S. of Accenture, "Shared Services Start-Up: From Launch to Stabilization," Accenture, Tech. Rep., 2007.

[186] G. Dickson, "A programmatic approach to information system research : an experimentalist's view," *The Information Systems Research Challenge : Experimental Research Methods*, pp. 147–170, 1989.

[187] K.M. Eisenhardt and K.M. Eisenhardt, "Building Theories from Case Study Research," *The Academy of Management Review*, vol. 14, no. 4, pp. 532–550, 2007.

[188] S. Basri and R.V.O. Connor, "Towards an Understanding of Software Development Process Knowledge in Very Small Companies," *Informatics Engineering and Information Science*, vol. 253, pp. 62–71, 2011.

[189] P.Y. Martin, "Grounded theory and organizational research," *The Journal of Applied Behavioral Science*, vol. 22, no. 2, pp. 141–157, Apr. 1986.

[190] L. Ramer, "Quantitative versus qualitative research?" *Journal of obstetric, gynecologic, and neonatal nursing : Jognn / Naacog*, vol. 18, no. 1, pp. 7–8, 1989.

[191] D. Berry, "Academic Legitimacy of the Software Engineering Discipline," Software Engineering Institute, Tech. Rep. November, 1992.

[192] D. Martens, C. Vanhoutte, S. De Winne, B. Baesens, L. Sels, and C. Mues, "Identifying financially successful start-up profiles with data mining," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5794 – 5800, 2011.

[193] J. Ruokolainen, "Gear-up your software start-up company by the first reference customer—nomothetic research study in the Thai software industry," *Technovation*, vol. 25, no. 2, pp. 135–144, Feb. 2005.

[194] J. Ruokolainen and B. Igel, "The factors of making the first successful customer reference to leverage the business of start-up software company — multiple case study in Thai software industry," *Technovation*, vol. 24, no. 9, pp. 673–681, Sep. 2004.

[195] J. Livingston, *Founders at Work: Stories of Startups' Early Days*. Apress, 2008.

[196] J. Azar and R. Smith, "Value-oriented requirements prioritization in a small development organization," *IEEE software*, vol. 24, no. 1, pp. 32–37, Jan. 2007.

[197] M.E. Fayad, M. Laitinen, and R.P. Ward, "Software Engineering in the Small," *Communications of the ACM*, vol. 43, no. 3, pp. 115–118, 2000.

[198] A. Ginige and S. Murugesan, "Web engineering: an introduction," *IEEE Multimedia*, vol. 8, no. 1, pp. 14–18, 2001.

[199] M.A. Cusumano and D.B. Yoffie, "Software development on internet time," *Computer*, vol. 32, no. 10, pp. 60–69, Oct. 1999.

[200] K. Wiegers, "Software process improvement in web time," *IEEE Softw.*, vol. 16, no. 4, pp. 78–86, Jul. 1999.

[201] Principles behind the Agile Manifesto. [Online]. Available: http://www.agilemanifesto.org/principles.html (Accessed : Aug. 25, 2012).

[202] L. Rising, "The Scrum Software Development Process for Small Teams," *Software, IEEE*, no. August, 2000.

[203] M. Zayko, "A Systematic View of Lean Principles : Lean Enterprise Management," in *Lean Enterprise Institute*, 2006, pp. 1–16.

[204] L.J. Krajewski, B.E. King, L.P. Ritzman, and D.S. Wong, "Kanban, MRP, and Shaping the Manufacturing Environment," *Management Science*, vol. 33, no. 1, pp. 39–57, 1987.

[205] D.J. Anderson, *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, 2010.

[206] N. Nikitina, M. Kajko-Mattsson, and M. Strale, "From scrum to scrumban: A case study of a process transition," in *Software and System Process (ICSSP), 2012 International Conference on*, June 2012, pp. 140 –149.

[207] N. Wasserman, *The Founder's Dilemmas: Anticipating and Avoiding the Pitfalls That Can Sink a Startup*. Kauffman Foundation, 2012.

[208] T. Hoff. Startups Are Creating A New System Of The World For IT. [Online]. Available: http://highscalability.com/blog/2012/5/7/startups-are-creating-a-new-system-of-the-world-for-it.html (Accessed : Nov. 06, 2012).

[209] G. Kawasaki, *The Art of the Start*. Penguin Books Ltd, 2004.

[210] R.G. McGrath and I.C. MacMillan, *The Entrepreneurial Mindset: Strategies for Continuously Creating Opportunity in an Age of Uncertainty*. Harvard Business School Press, 2000, vol. 26, no. 3.

[211] G.A. Moore, *Crossing the Chasm: Marketing and Selling Disruptive Products to Mainstream Customers*. Harper Paperbacks, 2002, vol. 24, no. April 2002.

[212] M. Marmer, B.L. Herrmann, E. Dogrultan, R. Berman, C. Eesley, and S. Blank, "Startup Genome Report Extra: Premature Scaling," Startup Genome, Tech. Rep., 2011.

[213] Murphy lifecycle. [Online]. Available: http://www.skmurphy.com/startup-stages/ (Accessed : Aug. 25, 2012).

[214] F. Destin. Startup lifecycle. [Online]. Available: http://prezi.com/nfvyxijdmj7v/startup-lifecycle-by-fred-destin/ (Accessed : Aug. 25, 2012).

[215] Corporate lifecycle. [Online]. Available: http://www.adizes.com/corporate_lifecycle.html (Accessed : Aug. 25, 2012).

[216] Angel investing. [Online]. Available: http://gust.com/angel-investing/startup-blogs/2011/11/22/the-right-investors-for-the-right-stage/ (Accessed : Aug. 25, 2012).

[217] Startup lifecycle. [Online]. Available: http://www.shirlawscoaching.co.uk/shirlawsresources/2011/8/25/article-from-paul-grahams-trough-of-sorrow-to-infinity-and-b.html (Accessed : Aug. 25, 2012).

[218] J.L. Nesheim, *High Tech Start Up, Revised And Updated*. Free Press, 2000.

[219] I. Heitlager, S. Jansen, R. Helms, and S. Brinkkemper, "Understanding the dynamics of product software development using the concept of coevolution," in *Proceedings of the Second International IEEE Workshop on Software Evolvability*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 16–22.

[220] Market curve lifecycle. [Online]. Available: http://techcrunch.com/2012/04/01/the-market-curve-the-life-cycle/ (Accessed : Aug. 25, 2012).

[221] GT Categories. [Online]. Available: https://workflowy.com/shared/3f8945a0-bd36-68bf-9276-355244c75cd4/ (Accessed : Aug. 25, 2012).

[222] S. Jamieson, "Likert scales: How to (ab)use them," *Medical Education*, vol. 38, no. 12, pp. 1217–1218, 2004.

[223] G. Normann, "Likert scales, levels of measurement and the "laws" of statistics," *Advances in Health Science Education*, vol. 15, no. 5, pp. 625–632, 2010.

[224] E.L.C. Law, V. Roto, M. Hassenzahl, A.P. Vermeeren, and J. Kort, "Understanding, scoping and defining user experience: a survey approach," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '09, New York, NY, USA, 2009, pp. 719–728.

Appendix A

Appendix

A.1 Conventions

Figure A.1 describes guideline symbols that are used in the representation of figures within the document. Note that all the figures are read from the left to right.

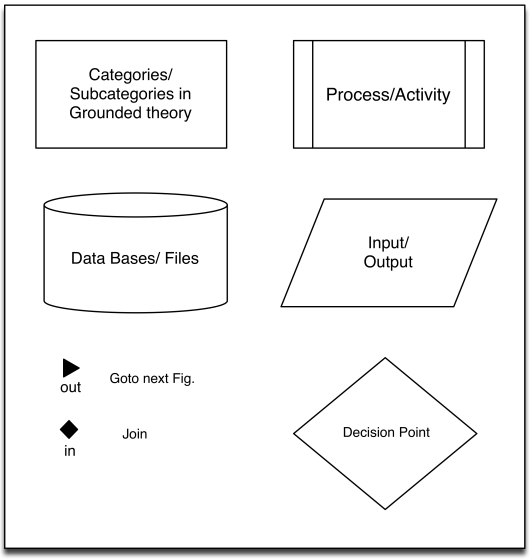


Figure A.1: Conventions

A.2 Related areas review

This appendix contains the results of multiple non-systematic reviews that have been executed to explore sibling areas. Doing this work helped us in getting a better understanding of the ecosystem in which a startup lives and operates and gives us the possibility of identifying specific areas of interest. The subsections included in this appendix are independent from one another, namely:

- *Managing software startups* (see Subsection A.2.1).
- *Software engineering in the small* (see Subsection A.2.2).
- *Web engineering* (see Subsection A.2.3).
- *Lean/Agile methodologies* (see Subsection A.2.4).
- *Grey literature review* (see Subsection A.2.5).

A.2.1 Managing software startups

During the *systematic mapping* of the SE literature, a clear subset of articles with a managerial background emerged. Despite these articles didn't fit in our selection criteria, in the *data screening* process we saved them into a separate file (see Subsection 4.2.3). This chapter contains a high-level overview of the important findings regarding management of software startups.

What makes startups fail can be determined by diverse factors. Despite the controversial findings of researchers in the last years, one finding that stakes in the ground is reported by practitioners and academics: the believe the success rate of startups has the potential to dramatically increase economic growth on global scale [22].

Among many resources, the knowledge management has been highlighted as essential resource to keep track in today's fast-paced economy [192]. Among others, *customer development* has become a widely used process [17].. According to Ruokalainen's research [193, 194] the focus of success moves to the ability of finding the first customer and to the ability of expanding the business abroad, after saturating the limited size of the local market. From the marketing viewpoint, the credibility of a startup is crucial, and it must be found with the help of the first customer, trying to develop the business concept and not by focusing only on technology solutions.

The dispute between those advocating for *"product first"* and those sustaining *"market first"* seems to be blown away by the unanimous acceptance that for startup success the most important factor is the *product/market fit*, i.e. , the right product for the right market with the right timing. The recent book *"The Lean Startup"* by Eric Ries [7], reviewed in Section 7.2, addresses this specific problem.

A.2.2 Software Engineering in the small

In terms of the number of early stage employees, startups are small companies. On the other hand it is not true that a small company is a startup [13]: some software companies with less than ten employees have been operating under stable conditions for more than a decade whilst most startups live in a chaotic domain, failing, getting acquired or evolving in established companies in a very short timeframe [195]. Furthermore researchers consider *small* software companies with less

than 50 employees [49], whilst the majority of startups founding teams are formed by less than five people.

But despite the essential differences between small companies and startups, when it comes to software engineering both deal with a limited number of human resources. Hence, it is worth giving a small overview of most important results achieved by researchers and practitioners in such a closely related field. Despite several SPI models for small companies have been proposed in the last years (CMM/CMMI, ISO 9001, ISO/IEC 15504), small companies still find difficult applying them since what they need is a customized assessment method within reasonable cost and time [165].

Introducing proper process improvement activities in small companies (for instance requirements prioritization) can determine not only the success or failure of the project but also the survivability of the entire company [196]. Indeed, the need of efficient and effective software engineering solutions brought engineers to be flexible and more reactive to context issue instead of prescribing activities to follow [49]. Hence, the existing practices applied in large companies are hard to apply in small contexts. In this regard, industry has recognized that integrated environments that you can't adapt to a company-specific process don't enhance productivity and quality [197].

A.2.3 Web engineering

In 1998, Web engineering was established as a new discipline in view of substantial differences revealed from the traditional software engineering. The rapid changes in web systems, evolving in their functionalities, scope, content and use are more frequent than traditional software, information and engineering systems [198, 57].

In web applications, the main reason why small companies always strive in conducting SPI models relies on the regular trade-off among *"feature slip"*, *"time-to-market"* and *"software quality"*. The same trade-offs go through the level of process definition and adopted formality. Especially for new projects, companies focus more on people and product rather than processes. In fact, managers believe that if people are mature and talented, there is less need for definition of processes [55]. A good experience paper written by a practitioner in the field shows other small differences between web and traditional software companies, such as: lack of quality assurance, short building time, coding freezing periods.

Nevertheless, web-based systems become more and more large and complex, impacting negatively on product quality aspects. Since most of *web-based* systems development must be completed in the short term, as described in [56], web engineering activities generally lack of integrated and formal testing methodologies. Moreover estimating process and product metrics, such as robustness and maintainability, can be really challenging since formal requirements, analysis and design are almost neglected [57].

Web and mobile companies introduce products faster than companies with

more stable technologies such as desktop or embedded software. In order to foster fast-track production, they must use more flexible development techniques with short product life-cycle [199]. Being flexible and fast, as well as creative, are important characteristics for success in the Internet era. The ability to rapidly adapt to changes and to be able to introduce more disciplined process, are two necessary capabilities, which help dealing with unpredictable markets. In fact, as advocated in [200], even companies that works with cutting-edge technologies, rapid project development and heavy business pressures can improve their methods for managing and implementing software projects by means of minimal project management activities.

A.2.4 Lean/Agile development

In this section we provide a brief explanation of Lean and Agile software development methodologies, which appear to be in line with the entrepreneurs demands especially in Internet and mobile industries [54, 71].

As stated by Abhrahamsson in [83], *"what makes a development method agile is the case when it is incremental (small software releases, with rapid cycles), cooperative (customer and developers working constantly together with close communication), straightforward (the method itself is easy to learn and to modify, well documented), and adaptive (able to make last moment changes)."*

According to the original Manifesto [201], the principles are expression of reaction against heavyweight methodologies in terms of:

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

The most common implementations of Agile methodologies, among many others [5, 118], are *eXtreme Programming* [78] (development-oriented) and *Scrum* [202] (more project managment-oriented) [83].

On a higher level of abstraction Agile methodologies rely on the principles derived from *Lean manifacturing*, which promote the elimination of anything that doesn't contribute to he creation of *value* for the final users [203]. Specifically, eliminating waste means reducing useless meetings, tasks and documentation. It also means eliminating time spent building features which are not necessary in the short term since the environment is constantly changing and consequently the needs of customers change [82].

A widely used *Lean* practice is known as *Kanban*, which was developed for the first time in the Toyota production system [204]. Kanban is based on six *core principles* [205], namely:

- Visualize workflow.
- Limit work in progress (WIP).
- Manage flow.
- Make policies explicit.
- Implement feedback loops.
- Improve collaboratively, evolve Experimentally (using models and the scientific method).

Differently from Scrum, the implementation of Kanban does not require a sprint-based schedule, and it allows *user stories* to change status with a continuous flow based on the Kanban board. In the literature it is possible to find hybrid methodologies (e.g. *Scrumban* [206, 85] which combines Agile and Lean principles).

A.2.5 Grey literature Review

Taking a quick look at the non peer-reviewed world, defined by Kitchenham as *grey literature* [108], it is clear that today's startup ecosystem is extremely vast, and includes a wide set of personalities: hackers, entrepreneurs, investors, managers, and developers, to name a few. What this community is lacking is the presence of a strong technical support from empirical research. In this subsection we review some of the most important books, which are relevant for this study.

Even if it is not strictly addressed to startups, one of the most important book that many authors refer to, when talking about technological startups, is *The Innovator's Dilemma* [24], written in 1997 by Clayton Christensen from the *Harvard Business School*. He discusses how outstanding companies, which had their business oriented to customers' needs and which aggressively invested in new technologies, still loose their market leadership when confronted to *disruptive technologies*[1]. The author explains how established companies are inclined to develop *sustainable technologies*, which support incremental development of existing solution in accordance to their customers' needs. By means of lessons learned from successes and failures of leading companies, this book demonstrates how new entrants to the market that introduce a disruptive innovation can attack small part of businesses, where established companies decide to don't compete, in view of an insufficient profit perspective and afraid of corroding their main business. Along with *The Innovator's Dilemma* sequel, *The Innovator's Solution* [169] introduces specific suggestions for established companies to create autonomous

[1]Disruptive technology can be represented by a new product, a new technology to produce a product, a new way to distribute a product or a new way to provide services.

division to pursue start-up's like innovation. Similar themes are discussed and analyzed in 2012's book *"The Founder's Dilemmas"* [207] of Harvard professor N. Wasserman, who analyzes *"the common pitfalls founders face and how to avoid them"*. Yet, it lacks technicalities and is rather oriented to the business perspective. Nevertheless, the book is based on a large survey conducted in the last 10 years with *"more than 6,000"* companies he confirms the difficulties in obtaining empirical data about startups: *"Data about startups and private companies in general is not publicly available and is very hard to retrieve"*.

One of the modern pioneer of software startups' research is Steven Blank, who has confirmed the profound diversity between startups and smaller versions of large companies from a entrepreneurial management's perspective. He, as practitioner and academic, developed the *Customer Development process*, extensively described in *The Four Steps to the Epiphany* [17] (2005) and *The Startup Owner's Manual* [59] (2012). In the course of attracting and keeping customers, Blank suggests a process to place aside to the product development, which aims to discover and validate the right market for an idea, by: building the right product features that solve customers' needs; testing the correct model and tactics for acquiring and converting customers; and deploying the right organization and resources to scale the business.

"The best student" that Blank has ever had was Eric Ries, now successful entrepreneur and engineer, who is recognized for pioneering the *Lean Startup Movement*, which combines the Japanese concept of Lean production with Blank's Customer development process, to establish a new sort of discipline. In his best-seller book *The Lean Startup* [7], Ries presents how entrepreneurs in every settings make the same mistakes: they build elaborate products before daring to test them with final users, making decisions on wrong information. He introduced the concept of *"minimal viable product"* (MVP), which is a strategy used for fast and quantitative market testing of a product that has just those essential features that allow the product to be released. From a technical perspective he developed the Lean Startup model, which core is represented by the *Build-Measure-Learn* feedback loop.

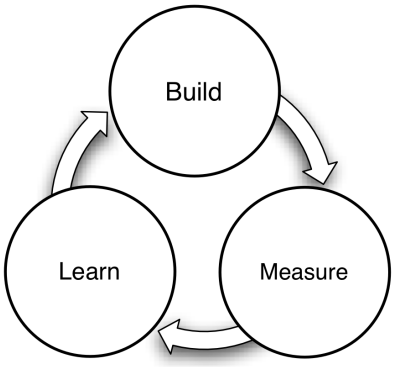


Figure A.2: Lean Startup cycle [7]

Through this model, he explains how it is important to test the riskiest elements of a startup's plan, the parts on which everything depends.

To process these hypotheses (*leap-of-faith* assumptions) the startup must enter the *Build* phase as quickly as possible with a MVP. As previously discussed the MVP lacks many features that may prove essentials later on, but enables a first deployment with a minimum amount of effort and the least amount of development time. Following the *Build* phase, the *Measure* phase determines whether the product development efforts are leading to real progress. The aim is to understand if the product is what the customers really want. Finally he presents the Pivot concept, which is described in the *Learn* phase as *"a change in strategy without changing vision"*. If the startup discovers that one of its essential hypothesis is false, then it is time to make a major change to a new strategic hypothesis. From the new hypothesis the *Build-Measure-Learn* feedback loop restarts again. Even though many insights are not apparently applicable for the context of early stage startups, unlike many authors in this field, Ries writes clinically and intelligently how a startup could innovate with less waste.

According to the Lean Startup methodology, building something people want is by providing a great first experience, improving *Activation* and *Retention* rates. *Activation* refers to those actions that typically start with sign-up process and end with key activities of the product. *Retention* refers to those actions which maintain the users involved or engaged in the use of the product. Eventually, there is a third metric which involves the endorsements of the product to new users, called *Referral*. Since it is typically provided by affiliate programs and marketing processes, it is not a main concern from a software engineering perspective.

He defines a start-up as a *"human institution designed to create a new product or service under conditions of extreme uncertainty"*. The words *human* and *uncertainty* are essential since he argues that a successful start-up doesn't just rely on a brilliant idea, but also requires managing people through all the challenges of innovation and growth. According to Ries, startups that apply the *lean method*

achieve dramatically lower development costs, faster time to market, and higher quality products in the years to come.

An extremely interesting group of researchers and entrepreneurs have recently founded a startup (*The Startup Genome*) that is trying to disclose the patterns behind startups' failures. They conducted a large-scale survey among startups' founders and they published the results, in summer 2012, in a technical report [22]. Another recent technical report on *HighScalability.com* [208] draw interesting conclusion on how software startup are shaping the new scenario of distributes software architectures in the cloud.

Other major contributions worth mentioning, but mainly focused on business aspects and plans generation are: *The Art Of The Start* [209], *Start Small, Stay Small: A Developer's Guide to Launching a Startup* [98], *Founders at Work: Stories of Startups' Early Days* [195], *The Entrepreneurial Mindset* [210] and *Crossing the Chasm* [211].

A.2.6 Lifecycle models

From idea conception to the maturity level, in the literature, different lifecycle models have been identified and reported in this section.

A prominent contribution from technical perspective is the model presented by Crowne [10], who synthesized the startup lifecycle in four stages: *startup*, *stabilization*, *growth* and *maturity*. Starting from the *startup* stage, the author describes it as the phase in which startups create and refine the idea conception, up the first sale. This time frame is characterized most from the need to assemble a small executive team with the necessary skills to start to build the product. The *stabilization* phase begins from the first sale, and it lasts when the product is stable enough to be commissioned to a new customer without causing any overhead on product development: being able to treat maintenance in such a way that the development team preserve the same productivity. The *growth* phase begins with a stable product development process and lasts when the market size, share and growth rate have been established: all the business processes necessary to support product development and sales. Finally, the startup evolves to a *mature* organization, where the product development becomes robust and predictable with proven processes for new product inventions. Only then, SPI can be further adopted for incremental process improvements.

In both peer-reviewed and grey literature we retrieved many other startups life-cycle models, defining the phases from different *perspectives*. The models are not explained in detail, but only summarized and referenced in Table A.1.

Author	Lifecycle stages	Perspective
S. Blank [17]	customer discovery; validation; creation; company building.	Market
M. Marmer [212]	discovery; validation;efficiency; scale; sustain & conservation.	Market
S. Murphy [213]	ideas and team formation; open for business; early customers; finding your niche;scaling up.	Business
F. Destin [214]	start; lunch; build; chasm; scale; mature.	Business
I. Adizes [215]	courtship; infant, go-go; adolescence; prime; stable;aristocracy;early bureaucracy; bureaucracy; death.	Business
M. Zwilling [216]	idea stage; early or embryonic stage; funding or rollout stage; growth stage; exit stage.	Business
P. Graham [217]	initiation; wearing off of novelty; trough of sorrow; release of improvement; crash of Ineptitude, wiggles of false hope, promised land, acquisition of liquidity, upside of buyer.	Social
J. L. Neisheim [218]	the idea; the demo; the prototype; the first product; the launch (growth).	Product
I. Heitlager [219]	fluid, transition time, specific.	Innovation
D. Pepper [220]	hype cycle;facing reality; liftoff to ultimate marketing opportunity; sustainable market.	Market

Table A.1: Startup lifecycle models

Many of the models presented in Table A.1 are obtained from non-peer reviewed sources, and additionally we were not able to find a fine-grained model describing the phases of an early-stage startup from a *product* perspective. Therefore we proposed our own model in Section 7.3.

A.3 Systematic mapping study details

A.3.1 Search Strings

The search strings, utilized to retrieve relevant studies (see Subsection 4.2.2), have been adapted to the underlying search technology, as shown in Table A.2.

String A - Compendex/Inspec [102]
('early-stage firm' OR 'early-stage company' OR 'high-tech venture' OR 'high-tech ventures' OR 'high-tech start-up' OR 'high-tech start-ups' OR 'high-tech startups' OR 'high-tech startup' OR 'start-up company' OR 'start-up companies' OR 'startup company' OR 'startup companies' OR 'software startup' OR 'lean startup' OR 'lean start-up' OR 'lean startups' OR 'software startups' OR 'software package startups' OR 'software package start-up' OR 'software package start-ups' OR 'software package startup' OR 'IT start-ups' OR 'IT start-up' OR 'IT startup' OR 'IT startups' OR 'software start-up' OR 'software start-ups' OR 'software product startup' OR 'software product startups' OR 'software start up' OR 'web startup' OR 'web start-up' OR 'web startups' OR 'web start-ups' OR'internet startup' OR 'internet start-up' OR 'internet startups' OR 'internet start-ups' OR 'mobile startup' OR 'mobile start-up' OR 'mobile startups' OR 'mobile start-ups')AND (develop* OR engineer* OR model* OR construct* OR implement* OR cod* OR creat* OR build*) AND (software OR product* OR service* OR process* OR methodolog* OR tool* OR method* OR practice* OR artifact* OR artefact* OR qualit* OR 'non-functional requirement' OR 'non-functional requirements' OR ilit* OR strateg*)
String B - IEEE xplore [113]

Table A.2 – *Continued on next page*

Table A.2 – *Continued from previous page*

('early-stage firm' OR 'early-stage company' OR 'high-tech venture' OR 'high-tech ventures' OR 'start-up company' OR 'start-up companies' OR 'startup company' OR 'high-tech start-up' OR 'high-tech start-ups' OR 'high-tech startups' OR 'high-tech startup' OR 'startup companies' OR 'software startup' OR 'lean startup' OR 'lean start-up' OR 'lean startups' OR 'software startups' OR 'software package startups' OR 'software package start-up' OR 'software package start-ups' OR 'software package startup' OR 'IT start-ups' OR 'IT start-up' OR 'IT startup' OR 'IT startups' OR 'software start-up' OR 'software start-ups' OR 'software product startup' OR 'software product startups' OR 'software start up' OR 'web startup' OR 'web start-up' OR 'web startups' OR 'web start-ups' OR'internet startup' OR 'internet start-up' OR 'internet startups' OR 'internet start-ups' OR 'mobile startup' OR 'mobile start-up' OR 'mobile startups' OR 'mobile start-ups') AND (development OR developing OR engineer OR engineering OR model OR construct* OR implement* OR cod* OR creat* OR build*) AND (software OR product OR products OR service OR services OR process OR processes OR artifact* OR artefact* OR quality OR qualities OR 'non-functional requirement' OR 'non-functional requirements' OR ilities OR methodology OR methodologies OR tool OR tools OR method OR methods OR practice OR practices OR strategy OR strategies)

String C - Scopus [115]
ABS(('early-stage firm' OR 'early-stage company' OR 'high-tech venture' OR 'high-tech ventures' OR 'start-up company' OR 'start-up companies' OR 'high-tech start-up' OR 'high-tech start-ups' OR 'high-tech startups' OR 'high-tech startup' OR 'startup company' OR 'startup companies' OR 'software startup' OR 'lean startup' OR 'lean start-up' OR 'lean startups' OR 'software startups' OR 'software package startups' OR 'software package start-up' OR 'software package start-ups' OR 'software package startup' OR 'IT start-ups' OR 'IT start-up' OR 'IT startup' OR 'IT startups' OR 'software start-up' OR 'software start-ups' OR 'software product startup' OR 'software product startups' OR 'software start up' OR 'web startup' OR 'web start-up' OR 'web startups' OR 'web start-ups' OR'internet startup' OR 'internet start-up' OR 'internet startups' OR 'internet start-ups' OR 'mobile startup' OR 'mobile start-up' OR 'mobile startups' OR 'mobile start-ups')AND (develop* OR engineer* OR model* OR construct* OR implement* OR cod* OR creat* OR build*) AND (software OR product* OR service* OR process* OR artifact* OR artefact* OR qualit* OR 'non-functional requirement' OR 'non-functional requirements' OR ilit* OR methodolog* OR tool* OR method* OR practice* OR strateg*))

Table A.2 – *Continued on next page*

Table A.2 – *Continued from previous page*

String D - ISI Web of Knowledge [114]
TS=((’early-stage firm’ OR ’early-stage company’ OR ’high-tech venture’ OR ’high-tech ventures’ OR ’start-up company’ OR ’start-up companies’ OR ’high-tech start-up’ OR ’high-tech start-ups’ OR ’high-tech startups’ OR ’high-tech startup’ OR ’startup company’ OR ’startup companies’ OR ’software startup’ OR ’lean startup’ OR ’lean start-up’ OR ’lean startups’ OR ’software startups’ OR ’software package startups’ OR ’software package start-up’ OR ’software package start-ups’ OR ’software package startup’ OR ’IT start-ups’ OR ’IT start-up’ OR ’IT startup’ OR ’IT startups’ OR ’software start-up’ OR ’software start-ups’ OR ’software product startup’ OR ’software product startups’ OR ’software start up’ OR ’web startup’ OR ’web start-up’ OR ’web startups’ OR ’web start-ups’ OR’internet startup’ OR ’internet start-up’ OR ’internet startups’ OR ’internet start-ups’ OR ’mobile startup’ OR ’mobile start-up’ OR ’mobile startups’ OR ’mobile start-ups’)AND (develop* OR engineer* OR model* OR construct* OR implement* OR cod* OR creat* OR build*) AND (software OR product* OR service* OR process* OR artifact* OR artefact* OR qualit* OR ’non-functional requirement’ OR ’non-functional requirements’ OR ilit* OR methodolog* OR tool* OR method* OR practice* OR strateg*))
String E - ACM [112]
((Abstract:(’early-stage firm’ OR ’early-stage company’ OR ’high-tech venture’ OR ’high-tech ventures’ OR ’start-up company’ OR ’start-up companies’ OR ’startup company’ OR ’startup companies’ OR ’software startup’ OR ’lean startup’ OR ’lean start-up’ OR ’high-tech start-up’ OR ’high-tech start-ups’ OR ’high-tech startups’ OR ’high-tech startup’ OR ’lean startups’ OR ’software startups’ OR ’software package startups’ OR ’software package start-up’ OR ’software package start-ups’ OR ’software package startup’ OR ’IT start-ups’ OR ’IT start-up’ OR ’IT startup’ OR ’IT startups’ OR ’software start-up’ OR ’software start-ups’ OR ’software product startup’ OR ’software product startups’ OR ’software start up’ OR ’web startup’ OR ’web start-up’ OR ’web startups’ OR ’web start-ups’ OR’internet startup’ OR ’internet start-up’ OR ’internet startups’ OR ’internet start-ups’ OR ’mobile startup’ OR ’mobile start-up’ OR ’mobile startups’ OR ’mobile start-ups’)) AND (Abstract:(develop* OR engineer* OR model* OR construct* OR implement* OR cod* OR creat* OR build*)) AND (Abstract:(’software’ OR product* OR service* OR process* OR methodolog* OR tool* OR method* OR practice* OR artifact* OR artefact* OR qualit* OR ’non-functional requirement’ OR ’non-functional requirements’ OR ilit* OR strateg*)))
String F - Google Scholar [116]
(’software startup’ OR ’software startups’)AND (develop* OR engineer* OR cod* OR creat*) AND (software OR product* OR process* OR methodolog* OR tool* OR method* OR practice* OR artifact* OR qualit* OR ’non-functional requirements’ OR strateg*)

Table A.2: Search strings

A.3.2 Selected studies overview

We extracted a brief *one-line* sentence which summarize the content of each study and can be used by the reader to grasp the idea behind the articles without reading the full text. A set of keywords have been assigned to each article during the initial stages of creation of the classification schema (see analysis of results in Section 5.1). The result of this process is presented in Table A.3.

Ref.	Author (year)	One-line contribution	Keywords
[27]	Coleman (2008)	' [...] the previous experience of the person tasked with managing the development work is the prime influencer on the process a company initially uses. Other influencers include the market sector in which the company is operating, the style of management used and the size and scale of the company operations.'	Software Process, Process formation, Resources, Founder, XP, RUP, Agile Methodologies.
[2]	Coleman (2007)	'Background of Software Development Manager was central to the initial process that a software company used.'	Software Process, Software Process Improvement, Grounded Theory, Factors Influencing Process.
[29]	Coleman (2008)	'Our research found that SPI programmes are implemented reactively and many software managers are reluctant to implement SPI best practice models because of the associated costs.'	Software Process, Software Process Improvement, Grounded Theory, Factors Influencing Process.
[18]	Kajko-Mattsson (2008)	By applying their process in a start-up they obtained good results: manage requirements, define development release, and control releases, improve quality.	Process improvement, Software Process, Release Management, Communication, Maintenance, Founder, early-stage, Improve Quality.
[158]	Hásel (2010)	'[...] the competence profiles preferred by founders with innovative products differ from those preferred by founders with less innovative products.'	Founder Background, Founder Teams, IT competence, Team, Know-how.
[94]	Hanna (2010)	The model presented in this paper offers a powerful tool for understanding the challenges of offshoring	Outsourcing, Offshore, Know-how, Management, decision-making.
[89]	Deakins(2005)	'The dotcom development environment is highly volatile and requirements can change rapidly in response to competitor offerings and customer needs; customers are unreliable predictors of their future needs. Need of multi-skilled teams, Adaptiveness over efficiency, Rapid development, early-stage, Experimentation, Improvement based on customer'	Innovation, Management, Software Process, Volatile environment, Time-to-market, Adaptiveness, Rapid Development,Web-development.
[70]	Camel (1994)	'The presence of several other time-to-completion accelerators appears to be weak in software startups: they did not fully use development methodologies, they made little for increasing use of software tools to increase productivity, weak risk analysis and project control'	Time-to-market, Package Software, Founder, Software Process, Tools.
[79]	Silva (2005)	'We have successfully used all of XP practices, adopted most of them and even came up with some unique practices of our own.'	XP, Software Process, Agile methodologies, Practices, Adaption, Rapid Changes.
[86]	Midler (2008)	'While exploitation provides vital short-term resources,exploration enhances the adaptation of the organization to a changing environment because it increases the variance of organizational activities'	Management, Multi-project, Software Process, Maturity, Learning, Internet, Founder, Uncertainty.
[84]	Taipale (2010)	'Our workflow is predictable within acceptable variance and we can change direction of the business at any given time'	XP, Lean, Agile Methodologies, Software Process, Pivoting.
[34]	Chorev (2006)	'Marketing is very important for success and is underestimated by product startups [...] core team competences is crucial as well'	Management, Software Process, Influencing Factors, Success, Marketing, early-stage, Team Competences.

Table A.3 – *Continued on next page*

Table A.3 – *Continued from previous page*

Ref.	Author (year)	One-line contribution	Keywords
[88]	Zettel (2001)	'This paper proposes a lightweight software process for a specific application domain (i.e., database-and user-interface-oriented off-the-shelf e-business applications).'	Process Improvement, XP, Agile Methodologies, Lightweight process, IT, Project Management.
[95]	Jansen (2008)	'Here, we describe two start-ups that have opportunistically and pragmatically developed their products, reusing functionality from others that they could never have built independently.'	Reuse, Product-line, Unstructured, Method, third-party, Founder, IT, COTS.
[13]	Sutton (2000)	'Startups represent a software industry segment that has been mostly neglected in process studies, and it is possible that lessons drawn from start-ups also apply to other develop-ment organizations.'	Software Process, Process formation, early-stage,Time-to-market, Resources.
[3]	Heitlager (2007)	'These companies start very ad-hoc, trying to overcome the uncertainties of market, team and platform. The biggest struggle for these companies is to survive with only scarce resources.' - This paper provides a matrix to analyze the dynamics of the maturity of product development.	Software Process, early-stage, Innovation, Internet, Process Improvement, Product Development.
[77]	Tingling (2007)	'Small releases, on-site customer, continuous integration and refactoring were most vigorously advanced by management and adopted by developers. Paired programming on the other hand was culturally avoided.'	Rapid Development, XP, IT, Software Process, Internet, Practices, Agile methodologies.
[80]	Deias 2002)	'We are enthusiastic about XP, and it is difficult for us to imagine a software project where we should not try to use XP, at least in the domain of Internet development.'	Software Process, Agile Methodologies, XP, Web Development, Risk Management, Project Management, Internet, Quality Assurance, Business, Founder,Customer relation.
[159]	Stanfill (2007)	'To improve the chances of successfully adopting a new technological innovation and boosting entrepreneurial team performance, we propose an improved way to select suitable technologies, better timing for delivering market-driven requirements to product designers, and enhanced understanding of the implications of business and technical decisions with regards to impact on intellectual property.'	Team Performance, Market Feasibility, Management, Market-driven Requirements, early-stage, Founder,Technology-Driven decisions, IT.
[160]	Wood (2005)	'Taken together, these strategies provide guidance to entrepreneurs, board members and business and engineering managers of startups for the effective use of Open Source Software.'	Open Source, Release, Software development, early-stage,Strategy, Internet, Cost-reduction,License.
[161]	Steenhuis (2008)	'It is therefore unlikely that follower regions or nations are able to catch-up with the leading regions or nations unless the leading regions or nations enter the high portion of the S-curve, i.e. their economic growth slows down.'	Innovation, Technological Development, Critical Mass,Business,Economic Growth Internet, Business.
[87]	Yogendra (2002)	'With the role of technology varying from 'enabler' to 'driver' of the business strategy, business and technology strategies need be in close alignment'	Management, Technology Driven Decisions, Planning, Monitoring, Quality.
[76]	Ambler (2002)	'I wondered whether the rules of software development had also changed. Were we witnessing a paradigm shift in the way we develop software?'	Agile Methodologies, Web Development, RUP, Comparison, Paradigm-shift.

Table A.3 – *Continued on next page*

Table A.3 – *Continued from previous page*

Ref.	Author (year)	One-line contribution	Keywords
[10]	Crowne (2002)	'A model for the evolution of product development from startup to maturity is provided, consisting of three phases:phases: Startup ,Stabilization , Growth [...] Successful development of new software products is a key value driver for many startup companies.'	Software Process, Founder, Life-Cycle, Internet, Maturity.
[91]	Mater (2000)	'Short-time-to market, fast growth, changing requirement are Entrepreneurs dream and Engineer nightmare'	Management, Business,Web Development, Quality, UX, Time-to-Market.
[35]	Kakati (2003)	'Product uniqueness was shown not to be a significant factor in determining initial success, despite the tendency of high-tech firm to emphasize RnD and technological excellence'	Management, Success Criteria, Risk.
[85]	Kuvinka (2011)	'Scrum involves many meetings, much planning overhead, and time-consuming team collaboration. Is it possible for a single writer to keep up?'	Management, Internet, Business, Agile Methodologies, early-stage, Scrum, Kanban.
[162]	Su-Chuang (2007)	' [...] a properly constructed value proposition is essential to the value creation process in e-business, and value is essential to the value creation process in e-business, and value co-production is the building blocks for value protection mechanism in network economy'	Value Proposition, Web Development, E-Business, Internet, Business, Value,Co-production.
[163]	Sau-ling Lai (2010)	'Technology is not Alibaba's core competency (non tech-founder) [...] Customer First, employee next [...] Small is beautiful'	Web Development, Founder Background, Know-how, Value Proposition, Finance, Customer Relation, IT, Internet, Product Design , Business.
[92]	Mirel (2000)	' [...] usability improvements depend on more than innovative and user-centered technical designs and implementations. Equally important for creating useful and usable software are the social and political forces that shape the development context.'	Organizational Factors, Usability, Political Support, Sociology, IT, Conflicts, Internet, Innovation.
[68]	Himola (2003)	'On the basis of the results of this article, it is suggested that the improvement of product development lead time is one of the most important parameters in the software startup environment [...] all decisions related to product development are tradeoff situations.'	Time-to-market, Improvement,Business, Management, Finance.
[93]	Kim (2005)	'Initial trust is regarded as a critical factor for many e-businesses to succeed in the business-to-customer e-markets, especially startups, because it creates initial relationships with customers.'	Trust, Customer Relation, Initial Trust, b2c, E-commerce, Internet, early-stage, Quality.
[96]	Wall (2001)	'When money are scarce, OSS can help your business launch without breaking your budget'	Open Source, Tool, Java, Software development, License, Distributed Development, Cost-reduction.
[81]	Yoffie (1999)	'That youthfulness also helps to explain why most start-ups fail: exuberance can only get you so far. Jim Clark and Marc Andreessen made a conscious choice to scale the company with a different type of person. They targeted maturity as well as technical expertise.'	Management, Internet, Web Development, Team Formation.

Table A.3 – *Continued on next page*

Table A.3 – *Continued from previous page*

Ref.	Author (year)	One-line contribution	Keywords
[97]	Bean (2005)	'Smaller firms like Aperture Technologies Inc. are using wikis to brainstorm, track projects, write and edit documentation, and coordinate marketing. Software startups like Stata Lab-oratories Inc. are using wikis to lower teleconferencing costs for outsourced engineering to India!'	Open Source, Method, Tool, Internet, Communication, IT, Knowledge Management, Management.
[33]	Tanabian (2005)	'Because of the small size of the firm, the amount of uncertainty of its business, and lack of financial strength, Many practices in place may appear to be in contradiction with guidelines for a productive and healthy job.'	Management, Founder, Job Design, Business, Team Performance, IT, Features, Workload, Internet.
[90]	Fayad (1997)	'The process should be treated differently from startups to established companies [...] 'startup effect' in which new initiatives get much more highly qualified and motivated people than standard projects, and the idea of 'heroic efforts' '	Software Process, IT, Process Improvement, Developers Skill, Resources Scarcity, Internet, Motivation.

Table A.3: Mapping study - One line content review

A.3.3 Ranking quantification

In this appendix we refer to the process of ranking the selected studies as discussed in *Research Methodology*, specifically in Subsection 4.2.7. The final score has been computed by summing up contributions from eight dimensions: age; rigor; relevance; venue; pertinence; contribution type; research type; and focus. For each dimension we defined conversion tables to quantify our criteria, i.e. - assigning higher scores to recent rigorous journal articles entirely devoted to the topic and presenting empirical results relevant to practitioners.

The left table (a) shows the scores assigned to each subcategory of the classification schema, while the table on the right side (b) shows the scores associated to the remaining dimensions.

Rigor	
Ri	Score
3	10
2.5	8.33
2	6.67
1.5	5
1	3.33
.5	1.67
0	0

Relevance	
Re	Score
4	10
3	7.5
2	5.
1	2.5
0	0

Age	
Age	Score
[0, 2]	10
[3, 5]	8
[6, 9]	6
[10, 14]	4
[15, 40]	1

Venue	
Venue	Score
Journal Article	10
Conference Proceeding	7
Magazine Article	6

(a) Other dimensions

Research	
Type	Score
Evaluation Research	10
Solution Proposal	6
Philosophical Papers	3
Opinion Papers	3
Experience Papers	3

Focus	
Type	Score
Software development	10
Process management	10
Tools and technology	8
Managerial/organizational	6

Pertinence	
Type	Score
Full	10
Partial	5
Marginal	3

Contribution	
Type	Score
Model	10
Theory	10
Framework/Methods	8
Guidelines	6
Lesson learned	6
Advice/Implications	6
Tool	6

(b) Classification schema

Figure A.3: Conversion table for the scoring function

The results of the final ranking of studies is presented in Subsection 5.1.4, while limitations of this approach are discussed in Subsection 4.2.8.

A.4 Case study details

This appendix presents the *interview package*[2] used during the case study, dividing the description according to the research design process, defined in Subsection 4.3.2. Moreover we present all the collected raw codes and categories as described in Subsection 4.3.4. Finally we present the engineering elements identified by practitioners in support of the categories defined in the theoretical model (see Chapter 6).

A.4.1 Interview package content

In this subsection we briefly present the content of the interview package. As previously discussed in Subsection 4.3.2, the interview package has been incrementally improved and structured to represent the output of all the activities

[2]Published at: `https://github.com/adv0r/BTH-Interview-Package`.

involved in the interview design process. The package, which contains a set of artifacts (38) clustered in 9 directories, is available for download on Github [146] under MIT licence[3]. The directories accompanied by their identifier are listed below:

- TMPL - Templates.
- SUP - Support material.
- TC - Topic cards.
- CLIST - Check lists.
- HLIST - Hand lists.
- TOOL - Tools.
- FUP - Follow-up.
- RC - Recordings.
- TAN - Data Triangulation.

First of all we created general *templates* to conduct and record information of the sampled companies, shown in Table A.4.

ID	File Name	Description
TMPL.0	Companies Overview	Startups' information related to: name, category, website and brief description of their main product or service.
TMPL.1	Cold Call Script	Dynamic Script with verbatims and reaction, to use during phone calls with companies.
TMPL.2	Ack Mail	Request for interview template email for informing candidate startups about our research and our interest.
TMPL.3	Thanks Mail	Mail to send to startups immediately after the interview.
TMPL.4	Reward Mail	Mail to endorse rewarding to startups after they filled the follow up questionnaire.
TMPL.5	Web call-for-interview	Web-page to inform about the interview and allow spontaneous participation.

Table A.4: Interview Package - Templates

The *support materials* aim to help researchers in executing the interviews, providing tools to control the process and structure the interview conduction (see Table A.5).

[3]MIT license is a free software license (information available `http://opensource.org/licenses/MIT`).

ID	File Name	Description
SUP.0	Package Content	List of all available documents to conduct the interview.
SUP.1	Whiteboard Wireframe	Wireframe of the interviews workflow with a graphical representation that can be followed on a whiteboard.
SUP.2	Definition Vocabulary	List of definition of engineering terms used during the interview, constantly updated to ensure consistency.
SUP.3	Company Info	Condensed summary of information about a startup to prepare before the interview containing: respondent and company data; developed product features and qualities.
SUP.4	Package Usage	Model representing the usage timeline of each artifact present in the interview package.

Table A.5: Interview Package - Support Material

Table A.6 lists all the *topic cards* we used during the interview. These artifacts represent the actual script which has been followed in the execution of the interviews.

ID	File Name	Description
TC.0	Kick-off Card	Verbatim to start the interview session introducing the interviewers, clarifying the execution details and a little disclaimer.
TC.1	Opening Questions Card	Script describing the first questions to ask for making the interviewee comfortable.
TC.2	Feature Elicitation Card	Questions to quickly elicitate the main features of the companies product.
TC.3	Non Functional Card	Questions to elicitate the main quality aspects considered during the development process.
TC.4	Process Card	Script to elicitate if standard processes and methodologies has been considered during development and to keep track of topic covered during the interview.
TC.5	Requirements Card	Script to elicitate main methods, tools and measures used during requirements specifications.
TC.6	Analysis Card	Script to elicitate main methods, tools and measures used during analysis of critical parts of the project and feasibility assessment.
TC.7	Design Card	Questions about the high-level architecture decision and low level design choices made during the development process.
TC.8	Implementation Card	Questions to elicitate the implementation methods, tools and measures.
TC.9	Testing Card	Questions to elicitate the methods, tools and measures for validating and verifying the developed product.
TC.10	Deployment Card	Questions to elicitate methods, tools and measures during the deployment of the product.
TC.11	Cool-down Card	Script for thanking the respondent for his participation and recalling to answer the follow-up questionnaire.

Table A.6: Interview Package - Topic Cards

In order to have control over the engineering elements we created a checklist of well known practices, tools and methodologies which, the company might have used (see Table A.7).

ID	File Name	Description
CLIST.0	Practices List	List of best practices in software engineering.
CLIST.1	Tools List	List of the most used tools by practitioners.
CLIST.2	Methodologies List	List of the most recent software development methodologies described in software engineering.

Table A.7: Interview Package - Check List

Moreover we packed lists to provide to practitioners possible engineering elements they might have used but that were not mentioned during the interviews (see Table A.8).

ID	File Name	Description
HLIST.0	Qualities List	List of qualities attributes as listed in ISO 9126.
HLIST.1	SE Artifacts List	List of well-known engineering artifacts most used from software engineers.

Table A.8: Interview Package - Hand List

We created a tool for managing the interview conduction (*Slider.app*), and we made use of *Prezi* to visualize the workflow of the entire interview (see Table A.9).

ID	File Name	Description
TOOL.0	Slider.app	Web-application for conducting the follow-up questionnaire online and collecting data.
TOOL.1	Prezi.com	Web-tool for conducting the interview remotely and visualizing the workflow and lists.

Table A.9: Interview Package - Tools

Table A.10 presents the artifact which were used to capture results using an online questionnaire.

ID	File Name	Description
FUP.0	Questionnaire	Follow-up questionnaire with both free text form and likert scale question to rate items mentioned during the interview.
FUP.1	Result Summary	Short report of results obtained from the interview.
FUP.2	Repertory Grids	Grid of methods and engineering artifacts retrieved from the interview as elements for scaling..

Table A.10: Interview Package - Follow-up

Table A.11 shows the structure of all the recordings that were stored during the interviews.

ID	File Name	Description
RC.0	Audio	Audio records of the interview.
RC.1	Notes	Written notes obtained during the interview
RC.2	Whiteboard	Workflow of main topics (product, quality, process) discussed during the interview.
RC.3	Others	Other obtained records not mentioned before.

Table A.11: Interview Package - Recordings

In some circumstances we were able to triangulate data with provided artifacts after the interviews (see Table A.12).

ID	File Name	Description
TAN.0	Provided Artifacts	Artifacts discussed during the interview, then delivered by the startups to the interviewers.

Table A.12: Interview Package - Triangulation

The use and advantages of such triangulation are described in Subsection 4.3.3.

We encourage anyone who has interests in pushing this work forward, to fork the repository and contribute to it. If used in a research context, please inform us, in order to be able to track where and how the interview package has been used.

A.4.2 Interview questions

During the interview design process we prepared scripts to use as guidelines in conducting the interviews. We embedded them in the previously described *interview package*, and in Table A.13we present the list of questions grouped by *topic card*.

1. Opening questions	
ID	Question
Q.0.0	When was (company name) founded?
Q.0.1	How was the initial team composed? And now?
Q.0.2	What was your role initially?
Q.0.3	How long did it take to release (product-name) to the public the first version* of your product?
2. Features elicitation	
ID	Question
Q.1.0	Is (product-name) (company name) first product? (yes/no)
Q.1.1	Does (product-name) still represent good part of your current core business? (yes/no)
Q.1.2	Can you briefly describe it? What does it do?
Q.1.3	Could you help me to write a list of the main features of this product, briefly, on this whiteboard? Let's try to write essential functionalities (around 3-5) of your system.
3. Non-functional attribute	
ID	Question
Q.2.0	You have a (product-type) in place ... I guess that in this case, is important to (some-important-non-functional-aspect) . Am I right?
Q.2.1	Why was this important/unimportant?
Q.2.2	How did you realize it? (ask for each quality he mention as important)

Table A.13 – *Continued on next page*

Table A.13 – *Continued from previous page*

4. Process	
ID	Question
Q.3.1	Did you use any specific development methodology?
Q.3.2	Did you use any project management process? How do you schedule the progress of your project?
Q.3.2	If any,who was the manager? (critical decsion)
Q.3.3	What were your tipical working hours?
Q.3.4	Did you had any pressure for deliverying the product fast?
5. Requirements engineering	
ID	Question
Q.4.1	Where does the idea behind (product name) come from?
Q.4.3	Did you discuss the idea with other founders/team-members? Any other stakeholders involved in the discussion? Did you document it?
Q.4.3	Have you formilized the behaviour/requirements you wanted to implement in the first release? How?
Q.4.4	Did you structure/organize the requirements?
Q.4.5	What happened when requirements were changed, added or deleted? How did you manage them? Any tools?
Q.4.6	Did you trace functionalities/requirements during subsequent activities?
6. Analysis	
ID	Question
Q.5.0	Did you analyzed what were the main challenges of the project from the development perspective and how to mitigate them? Did you document it?
Q.5.1	Did you consider the skills and time needed for realizing the project? Did you document it?
Q.5.2	Have you applied any measure to assess feasibility? And potential risks? "What if" analysis?
Q.5.3	Any particular considerations for critical requirements?
7. Design	
ID	Question
Q.6.0	Have you considered any design architecture before start implementing the actual code? If no: have you structured your system in different parts? How do they interact?
Q.6.1	Have you created models or diagrams for documenting those decisions?
Q.6.2	Have you considered measuring your architectural decisions? Such as maintanability effort required to change a component of your system.
Q.6.3	Have you considered well-know standards to adopt (such as design or architectural patterns)? Did you document them?
Q.6.4	Have you utilized any particular tools for designing your system?
8. Implementation	
ID	Question
Q.7.0	How did you approach coding the first days? You had the idea, you had the requirements and the design... and then?
Q.7.1	Have you considered any workflow guidelines before or during the coding phase?
Q.7.2	How did you divide the work between team-members?
Q.7.3	How did you manage the code base?
Q.7.4	What documentation have you produced during coding?
Q.7.5	Have you considered any configuration process for the development environment? Did you document it?
Q.7.6	How did you manage issues and bugs?
Q.7.7	Did you monitor aspects of your development such as team productivity, size/complexity ... ?
Q.7.8	Which are the tools that most have helped you producing code?
Q.7.9	What programming language did you use?
9. Testing	
ID	Question
Q.8.0	Did you perform any kind of tests for the implemented code? Such as acceptance, unit, integration and system tests?
Q.8.1	When did you write the tests? Are they documented?
Q.8.2	Quality assurance was an important concern? (to deduct also from the discussed qualities)
Q.8.3	Have you conducted any verification and validation process? Did anyone tried your product before the first release? Any reports and analysis of results?
Q.8.4	Have you conducted any measurement for assessing the validation and verification results?
Q.8.5	Have you utilized any specific tools for performing testing?
10. Deployment	

Table A.13 – *Continued on next page*

Table A.13 – *Continued from previous page*

ID	Question
Q.9.0	How did you deploy your project? Are you about scalability?
Q.9.1	Have you utilized any specific tools?
11. Closing questions	
ID	Question
Q.10.0	Have you considered improving the development process (in terms of efficiency and effectiveness)?
Q.10.1	Have you experienced a drop-down performance during the development, if any? What could be the reason and when did it happen?
Q.10.2	What is the most valuable improvements you would apply with perfect hindsight?

Table A.13: Grounded Theory - Interviews guiding questions

A.4.3 Interviews - Open coding

In this section we present all the codes conceptualized during the *open coding process*[4] (the process is described in Subsection 4.3.4 and results presented in Subsection 5.2.2). The following tables shows the 630 which contributed in the formation of the theoretical model, divided by thematic area.

Code stats - Opening Questions

Code
Small team
Recently Founded
Technical Respondent
Technical Founders
Short Product Building Time
Web and Mobile Product
Web Application
Tech/Biz Founders
Very short product building time
Web and desktop

Table A.14: Opening questions

Code stats - Product Priorities

Code
Growing Team
Minimum and essential set of functionalities is important
Limited budget
Automatization of deployment
Tradeoff: Quality and Time-to-Market
UX
Assess usability with user feedbacks is important
Development speed is the most important

Table A.15 – *Continued on next page*

[4]During the process we used tags such as *SUGG* and *ERR* to differentiate the conceptualization between what was the current state-of-art and gathered hindsight suggestions for the future or mistakes occurred in the past.

Table A.15 – *Continued from previous page*

Code
Time pressure
Entrepreneurial attitude
Interoperability
Not paying attention to non-functional aspects makes development faster
Value to the user is the most important
Being fast to learn what brings value to user
Delay choices which could limit technological flexibility
Delicate balance between code maintainability and development speed
Desire to roll out something new as fast as possible
Ease of use
ERR: Not using a framework to avoid learning, consequences with low maintainability
Framework to Improve Maintainability
Metrics for assess Usability
Quality was not a priority in early stage
Short Time-to-Market is the main focus
SUG Build scalable product/infrastructure from data 1
SUG Technology is not enough
Usability
Usability more important than functionalities
Build scalable product/infrastructure from day 1
Compliance with third party components
Ease of use (Product usage without interruption) is important
ERR: Not considering UX from day 1 led to unsatisfied initial users
ERR Technical issues for scalability
Idea conception from personal problem
Internal Deadlines
Lack of scalability
Lack of usability expertise
Mobile application compliance with Apple standard is important
Performance
Pivoting from initial prototype
Poor performance drive users away from the product
Portability from web to mobile
Portability in mind from day 1
Proof of concept by prototype
Prototyping for assessing efficiency
Prototyping for assessing usability
Reliability important for infrastructural product
Ruby on Rails increase Maintainability
Scalability realized soon based on past experience
SUG Enhance scalability
Talk with customers
Tradeoff: Effectiveness more important than UX
Tradeoff: High portability and UX for Mobile Apps
Tradeoff: Quality vs. Budget
Tradeoff: Usability functionality
Analysis for communication of different technologies
Attend startup events to prove idea/concept/prototype validity
Close friends feedbacks for increasing usability
Cultural usability expectation
Customer giving spontaneous feedback
Dedicated person for UX portability to Android
Efficiency
Efficiency to enhance UX
Efficiency will emerge only when using a prototype
Find a mentor in early stage
First prototype to solve personal problem
Focus group to assess usability (with potential users)
Hire more people
Interoperability studies beforehand (with third parties)
Lack of people
Lack of reliance on third party services

Table A.15 – *Continued on next page*

Table A.15 – *Continued from previous page*

Code
Limited hw resources
Maintainability becomes important when complexity and size increases
Maintainability enhance testability
Medium Product Building Time
Minimum level of maintainability (decent) from day 1
Mobile application portability covered mainly with iOs and Android
Mobile MVP hard to get accepted on Apple Store
Mobile portability deals with different OS
Mobile portability is important
Mobile/web portability first version using HTML5 to save time
Not perfect usability is not critical
people are usually more attracted by interfaces
Portability of Web application is Browser Compliance
Product changes quickly
Product is not security-critical
Reliability becomes a problem when users increase
Reliability for web-consumer product is not important
Reliability in early stage is not important
Remote team (non co-located)
Scalability issues solved using Elastic Infrastructures (EC2)
Scalability vs UX
Social network integration boost usability
Sometime startup fail because over-engineer the product before launching
Starting as side project cause low attention to maintainability
Studying competitors to differentiate
SUG: Maintainability from day 1 is easier than huge in late stage
SUG: Scrum helps in clarify product vision
SUG: Scrum helps in visualize project progress with the team
SUG Solid product infrastructure increases confidence in the product
SUG system oriented infrastructure instead of monolithic app
SUG working prototype is essential for fund raising
Technical background hinder usability
Tradeoff: Mobile native apps takes longer but more Effective (UX, Efficiency)
Tradeoff: Working overtime → quality code
Tradeoff: Development speed vs Reliability
Two respondents together
Usability assessment cannot be based only on feedback (users do not realize some design detail)
Usability at first designed with personal experience
Usability essential for games
Usability important when founders background is design
Usability improved by studying cutting-edge examples
Usability outsourcing
Users intensively using a system needs a high UX
UX background
Working overtime decreases productivity in long run
You have to find the balance between UX and functionality
Lack of scalability because of budget
Lack of scalability because of people
SUG working product is essential for fund raising

Table A.15: Product priorities

Code stats - General Process

Code
Respondents claim to use light version of scrum partial principles implemented
Communication facilitated by small team size and co-location
Hacker culture
Started from pre-developed technology / prototype
Automatization of deployment

Table A.16 – *Continued on next page*

Table A.16 – *Continued from previous page*

Code
Collaborative online tools for task management
Co-located Team
Critical decision taken by the CEO/CTO which have global overview of the project
Developers usually work overtime to meet goals
Development speed is the most important
Fast and Informal Development approach
Short and flexible iterations ($\lesssim$ 2 weeks)
Time pressures from investors
Agile methodologies are not effective with one/two person teams
Developers are self-organized in choosing tasks
Entrepreneurial attitude
Evolutionary MVP approach helps bring value to customers
Feedback from users is a priority since day 1
Growing team requires more control / management on initial chaos""
Having a deadline for final releasing is necessary to set a limit to improvements
Idea originated from hole in the market
MVP evolutionary approach to (software) development
New features idea proposed by CEO
Reduce wasting time on specifications and analysis and focus on code
Simple products doesn't require formal process
Virtual kanban board style to manage the user stories
Being fast to learn what brings value to user
Delay choices which could limit technological flexibility
Desire to roll out something new as fast as possible
Feedbacks collected via email
File sharing between team-members using online tools
Founder background as scrum-muster drove his attitude towards software development
Informal internal deadlines
Post-its for task management
Small milestones ($\lesssim$ 2 weeks) help developers' awareness of project progress.
SUG: Limit planning because plans will be subject to changes, no matter what
SUG Technology is not enough
Build scalable product/infrastructure from day 1
Co-located working environment does not require tools for know who was doing what
Collaborative online tools for project management
Developers satisfaction is influenced by how the sprint goal is achieved/not achieved
Development team priorities in contrast with company operations
Discussing ideas informally
ERR: Low experience with project management -¿ No schedule
ERR: A key developer acting as manager is much less effective
Flat organizational structure
Focusing on one task at the time improve productivity
Github for managing the code base
Idea conception from personal problem
Implemented some principle of Lean Startup
Internal communication tools, simple chat, are very important for communication
Internal Deadlines
Knowledge sharing with online dedicated tools
No process
Not able to keep documentation updated during the process (lack of time)
Pivoting from initial prototype
Plan with a rough spreadsheet is sufficient
Post-it
Project management (tools) become necessary with growing team and product complexity
Project management is not needed because of co-located work env
Stand-up meetings are simple and worthwhile
Stand-up meetings to discuss task assignment of the day
SUG: for non-collocated teams more prescriptive guidelines needed (process)
SUG: Outsourcing in Low-wages countries helps saving budget
SUG: Scrum can work in the context of startups given that one person has done it before
SUG: Too many tools negatively affect speed
Talk with customers

Table A.16 – *Continued on next page*

Table A.16 – *Continued from previous page*

Code
Team progress was not measured
Time pressure from media coverage (hype)
Uncertainty
Very short time between idea discussion and feature implementation
Whiteboard for task management
Whiteboard to monitor project progress
A team which worked together in the past develops ad-hoc approaches
Agile helped receiving quickly user feedback
Agile Practices such as TDD and PP can cause overhead
Analysis for communication of different technologies
Attend startup events to prove idea/concept/prototype validity
Automated source control tools when team grows
Avoid formalities to release faster
Back-end skills is important for developers in startups
Build fast (asap) without schedule
Comfortable environment negatively affect productivity
Community manager is a very important figure to manage large number of users
Cross functional teams (full stack developers - front-end backend - mobile and web)
Customer Development helped in identifying the market niche
Customer dictated usage scenarios,and developers team extract functionalities
Customer giving spontaneous feedback
Developer responsible for designing, coding and testing a feature
Didn't executed radical pivot of main features
E-mails for communication on distance
ERR: Bug fixing process not integrated in Scrum
ERR: Lack of documentation can lead to poor understanding of the system
ERR: Little or no software development experience
ERR: Losing precious time engineering the product without releasing early
ERR: Scrum without expedite-lane for dealing with emergencies was problematic
ERR: When all developers can modify the kanban wall (add move remove) confusion arises
Feature Meeting when necessary to discuss new features
Final integration (one-time) when single parts completed
Find a mentor in early stage
First prototype to solve personal problem
Good impression on first reference
Graphical design tool
Growing team led to necessity of more meetings
Growing team require tasks assignment
Having a user manual is important
Highly experienced developers
Highly experienced developers made the process faster
Hire more people
Idea started as a side project
Impediment board helps in solving bottlenecks
In the first stage of a startup team productivity is not essential (other priorities)
Informal Meetings
Instagram founders were working.
Introducing a process decreases productivity
Iteratively adjust product tracking metrics
Measure conversion rate of landing page (desktop application)
Meeting for discussing new feature ideas
Show-and-tell to informally share weekly achievements are very useful to boost motivation
Mobile MVP hard to get accepted on Apple Store
No formal schedule
No pressure (time, budget, investors)
No Project management tool
Overhead of request hard to handle after successful launch
Overhead of request hard to handle when tired
Pair-programming helps with new hires
Pivoting from b2b to b2c
Process is not necessary until you start collecting real users feedback
Process perceived as limitation to speed

Table A.16 – *Continued on next page*

Table A.16 – *Continued from previous page*

Code
Product backlog for monitoring project progress
Product changes quickly
Proximity to release brought fear for credibility which led to more formalities
Release step-by-step to an increasing number of friends/user before open the product
Rigid Macro-Milestones, flexible internal smaller milestones
Rigid Weekly Sprints (Scrum) when pressure higher
Scrum works better with skilled and experienced people
Scrumboard (simplified) with only a few boards
Simple products does not require project management tools
Solo developer
Sometime startup fail because over-engineer the product before launching
Source of pressure because they were using the product for their work
Starting as side project cause low attention to maintainability
Stories / Features did not need any prioritization (all necessaries)
Studying competitors to differentiate
SUG: A mentor in the early stage is really useful
SUG: Be flexible in apply only useful Agile practices
SUG: Make some beforehand analysis to define data structure
SUG: Scrum brings advantages: team-building, responsibilities, better code quality
SUG: Scrum practices for software development, together with Lean Startup methodology
SUG: Shorten iteration coding time (from 5d to 3d) dedicate 2 days / week bug fixing
SUG: Small engineering teams (3 persons) are quick, adaptive and responsive
SUG: The only possible approach for startups is using Lean Startup Methodology
SUG: Use a physical kanban wall when co-located
SUG: Use kanban wall to track team velocity
SUG: Working overtime on the code, makes poor code quality
SUG For non-co-located teams more documentation is needed
SUG Solid product infrastructure increases confidence in the product
SUG system oriented infrastructure instead of monolithic app
SUG working prototype is essential for fund raising
SUG: many small customers bring more value than a single big company
SUG: Setting deadlines increase productivity
Team with no or small working history
Technical founders are aware of development team necessities more than non-technical managers
User documentation
Users feedback not necessary for the simple and specific application type (well-defined)
Workflow driven by user feedbacks, new ideas and necessities without scheduling
Working overtime only in the first phase
Young Employees

Table A.16: General Process Codes

Code stats - Requirement Engineering

Code
Dissemination of the idea with informal discussion and support of tools (whiteboard, paper, views)
Automated tools for collecting TODO lists, Requirements,User Stories
Started from pre-developed technology / prototype
Critical decision taken by the CEO/CTO which have global overview of the project
Estimations based on developer experience
Manual tools (whiteboard / paper) for collecting stories
Feedback from uservoice
Idea originated from hole in the market
New features idea proposed by CEO
Stories/features prioritization using personal experience and user feedback
Work break-down in smaller stories to let developers work independently
Idea refinement through networking
Lack of Requirement Specification Documentation
Trello
Automated tools for managing stories/features are necessary when complexity grows

Table A.17 – *Continued on next page*

Table A.17 – *Continued from previous page*

Code
Developers can manage user stories independently (create and assign)
Prototype (rough) to explain features and share vision among team members
Specification written from user perspective (stories)
Stories / List updated throughout the process via automated tools
Stories/features traceability via version control
Whiteboard not updated throughout the process (lack of time)
A prototype can substitute stories / feature documentation
Ability to write detailed specification from day one, thanks to prev. experience
i've never seen a document...
Clarify product vision through important use cases only
Co-working space limit whiteboard utility
Cross divide technology (HTML5) helped building faster prototype
Customer dictated usage scenarios,and developers team extract functionalities
Dissemination of the idea with informal discussion via email
ERR: Don't ask early customer feedback
Features / stories collected in an informal magazine (the startup product itself)
Feedback from prototype to drive user stories / features
Idea coming from prev. founder working experience (hole in market)
Idea refinement through focus group
Idea started from authority request for application open
Lack of trust in customer's feedback (next)
New features idea through focus group
New stories collected and prioritized during weekly meeting discussion
One developer assigned to one task at the time (generally)
Paper prototype (wireframes) to demonstrate the views
Physical wall for task / todo / user stories
Product based on a Contract with public authority (Requirements fixed)
Product support three family of users
Scrum board to manage stories (physical) with post-it notes
Simple semi-automated todo list software (google spreadsheet) to collect stories/features/feedback
Started with more structure and process that increasingly degenerated in chaos/spaghetti
Stories / feature traceability using skype logs
Stories / featured prioritization with 3 simple labels (urgent, todo,ideas) was enough
Stories / Features did not need any prioritization (all necessaries)
Stories / Lists used initially and not really updated
Stories/feature prioritization on a whiteboard is much better than Electronic tools
Stories/Feature prioritization with the help of tools (categorize etc)
Stories/features captured with top-down approach
Stories/features prioritization analyzing which were useless to pursue the vision
Stories/features prioritization documented side to side with business plan
Stories/features prioritization through informal discussion
SUG: Use Customer Development approach to collect user feedback before implementation
Team formation during events (startup weekend)
User feedback to choose among different UI
User views mockups represented the feature list (product as a flow)
Video on Landing page to explain features to users and share vision among teams
Visualize feature idea through mind-mapping tools
Whiteboard to clarify product vision is excellent

Table A.17: Requirement Engineering Codes

Code stats - Analysis

Code
Analysis not important, partially replaced by informal discussions
Analysis of feasibility not important because past experience/knowledge with similar domains
For non-core functionalities user feedback outperforms a formal analysis
Simple product does not require analysis for expert developers
Small informal analysis to make important technological decisions
Competitor informal analysis to investigate improvements to tackle

Table A.18 – *Continued on next page*

Table A.18 – *Continued from previous page*

Code
Using well-known, traditional, tested technologies foster team performance
Analysis does not work with innovative products never done before
Analysis of interoperability with critical third party components
Analysis replaced by experimenting critical technologies before implementation
Analysis replaced by exploring solutions within developers community (ruby gems)
Analyzing competitor feedback to understand what to improve
Create a document to show customers the difference from competitors
Decisions taken with recorded brainstorming for traceability
Disruptive technology
Evaluated and documented pros and cons of decisions made
Evolutionary Prototyping substitute analysis for innovative product
Facebook is still using mysql...
Specifying the product features precisely beforehand makes development cycle shorter
Let users evaluate between different front-end functionalities alternatives
Third party newsletter for external third party APIs updates
Transposed feature list to use cases (Analysis)
Underlying technology (third party) changing quickly (Interoperability) do not allow Analysis
User feedback to decide application name
Using well-known, traditional, tested technologies makes hiring developers easier

Table A.18: Analysis Codes

Code stats - Design

Code
Hacker culture
Well known architectural framework for web application (MVC)
Document communication among components (high-level)
Maintainability through de-coupled modules (modularization)
Design of the architecture conducted through informal and poorly documented discussions
Design of the architecture not documented at all (based on personal experience)
Designed the data-structure
Framework reduce the need of documentation (well-known and structured)
General high level mock-ups of views was the only documented design
Tradeoff: Engineering the product/process vs. Flexibility in first phases
Delay choices which could limit technological flexibility
There is no time to keep documentation updated out-of-the-code
Clear and standard code minimise the need of a design documentation
ERR: Not having an initial design led to problems in later phases
Initially defined modules and communication to enhance efficiency
Not able to keep documentation updated during the process (lack of time)
Simple product do not require formal design (replaced by naive diagrams)
Technical debt
UML complex diagram replaced by naive diagrams
With extremely small teams architectural choices do not need to be documented
Academic background led to more formal design diagrams
Automated tool for managing the product architecture (MVC)
Automatic tools for documentation
Co-location and the high communication volume makes design documentation un-necessary
Design informally made using whiteboard between engineers helps a lot
Electronic tools for UI design
ERR: Academic background led to standard formal UML-like diagrams useless for early stage startup
ERR: Lack of initial analysis/design of data structure led to overhead later on to fix problems
ERR: Small mistake in initial data structuring led to bad consequences
ERR: Traceability of decision taken was a problem
Growing team lead to necessity of refactoring
Growing team lead to waste and trash code (ERR)
Lack of documentation can sustain a growing team at 1-2 employee at the time through training
Pair programming helps in the first phase when structuring the application
Reflect the code structure in the UI using different colors for different user-category

Table A.19 – *Continued on next page*

Table A.19 – *Continued from previous page*

Code
Simple product do not require architectural design (replaced with evolutionary prototyping)
Specification for critical communication between components
Structure the code-base differentiating by user group
SUG: Document at least communication among components
SUG: The fastest MVP prototype is a piece of paper with the view in front of real users
SUG: When working remotely, even with past experienced teams, documentation is important
SUG: Remote teams, design only the final definitive mock-up, to save communication time
SUG: When working with remote teams design documentation is important
SUG: Wireframes will not reflect the actual outcome (limited utility)
SUG: not sure about choice of framework
Tradeoff: Time pressure leads to lack of documentation
Tradeoff: Trash code when growing instead of Write quick code in the beginning
Using ready open source components for UI elements made development faster
With evolutionary prototyping there is no design phase (in the waterfall sense)

Table A.19: Design/Architecture Codes

Code stats - Implementation

Code
Collaborative online tools for task management
Productivity metrics are ignored
Chat tools for internal communication and traceability
Clear code does not need in-text comments
Comments inside the code when necessary
Critical decision taken by the CEO/CTO which have global overview of the project
Estimations based on developer experience
Git / GitHub as version control system for the code-base
Github for issue / bug management
Developers are self-organized in choosing tasks
Framework reduce the need of documentation (well-known and structured)
Growing team requires more code documentation
Project management tool for issue / bug management
Task are assigned by CEO/CTO on personal experience
Virtual kanban board style to manage the user stories
Work break-down in smaller stories to let developers work independently
Extremely small development team did not require Version Control System
Git for branching/merging useful for the codebase
Refactoring the code only when stricly necessary
SUG: Treat issues/bug and user stories/new features equally (same board)
Trello
Well-known framework for the product (RoR)
Code metrics ignored (complexity, size, etc ...)
Documentation perceived as a waste
Framework (RoR) easy to learn and with big advantages
Growing team requires use of Version Control System
Hero developer helped in meet deadlines
IDE
Lack of experience caused some re-work
Need of documenting the code is bad code smell
No workflow / guidelines for implementation
Php
Refactor as-you-go
Team productivity measured naively looking at closed tickets
Versioning system not necessary when no-overlapping between developers' concern
Begin implementation with internal APIs
C++ (desktop application)
Chose technologies familiar to developers (language and framework)
Code documentation only for long-lasting pieces of code
Code n fix.

Table A.20 – *Continued on next page*

Table A.20 – *Continued from previous page*

Code
Code standards
Developers can decide which bug to work on
Critical bugs fixed immediately (no need of issue tracking system)
CSS3
developed locally, tested locally
Developer environment consistent with production environment
Documentation not necessary with clean code Documentation, even in-code, is waste
ERR: Time wasted for not analysing existing technical solutions (libraries)
Github useful for code-reviews
Github useful to see who is doing what
Growing teams require to trace who is doing what
HTML5
Internal APIs to improve portability
Jira used for issue / bug management
Mercurial for version control system
Minimal set of tools for code implementation (electronic board + chat + version control)
New hires training encourage developers to refactor the code
No need of task system (no even manual) thanks to hero developer
Node.js
Non relational database
Pair programming help communication between developers
Paper for issue / bug management
Process perceived as killer for creativity
RoR helps in managing code
RoR for scalability
SUG: Chose the technology according to the nationality of developer you want to hire
SUG: Github useful to see who is doing what (growing team)
SUG: Given good experience TDD is best way of writing software
SUG: Trello for issue management (and task) is efficient
SUG: Version control system automatic tools are well integrated and do not cause overhead
SUG: When working with remote teams, let one person decide tasks assignments
Tickets (stories) not useful in the very early stage (one big story implement the product")"
Track for bugs in the beginning, then stopped.
Version control system using project management tool integration

Table A.20: Implementation Codes

Code stats - Verification and Validation

Code
Absence of automatic testing (replaced by experience and usage)
In house validation by trying the product
Test only critical parts is enough, Secondary bug found by users
Progressively have the product used and tested by increasing number of persons refining it
Feedback from uservoice After releasing a new feature let a trustworthy set of superusers try it and report bugs works very good
Feedbacks collected via email
For non-core functionalities user feedback outperforms a formal analysis
In house validation for core features identifies critical malfunctioning before releasing it
Automatic tools to asses product usage help adjust flaws
Contacting users personally to identify malfunctioning
If code is well tested, documentation can be replaced by test cases
In web applications bug are usually client side, hard to automatically detect
Landing page helps you having feedbacks before releasing the actual product
New features manually tested every week before the weekly roll-out
Release the product to let the users report bugs
RoR test suite framework are well defined
TDD requires a paradigm shift that is easier for newer generation of developers
Testing absent because lack of knowledge
Testing almost non existing so that process can be faster

Table A.21 – *Continued on next page*

Table A.21 – *Continued from previous page*

Code
Tradeoff: Amount of tests (slow down the process) and reliability (user are fault tolerant)
Tradeoff: Testing require hiring VS. Small teams are more flexible
Tradeoff: Time-to-market more important than testing
Unit testing only when strictly necessary
Agile Practices such as TDD and PP can cause overhead
Continuous Integration testing using automated tool (selenium)
Developer responsible for testing his own code makes process faster
Development team were using the product itself for their development process
Didn't found bug in production: Lucky
Email used for bug reports
ERR: Not testing UX with real users
ERR: in the prev. project we did UX ourselves
For iOs/mobile product Apple offer automatic testing
Framework facilitate testing
Growing (remote) team require a tester
Growing team requires increasing number of tests
Growing teams are facilitated by having TDD already in place
Include a on/off switch in new features to de-activate it if something goes wrong
Inspect logs to find bugs
Integration testing executed manually sometimes
Landing page used for idea validation
Maintainability enhance testability
Scalability through testing over databases
Self explanatory tests don't need test cases (cocumber)
Simple projects does not require much testing
SUG: Customer service dedicated to collect feedback is very important
SUG: Developers are Testers and should test someone else's code
SUG: Given good experience TDD is best way of writing software
SUG: User stories itself suggest acceptance tests
TDD, not religious but only on critical parts, helps a lot
TDD helps keeping the focus on the current task
TDD requires experience
Test to assess performance/efficiency
Testing helps innovation by improving confidence in the code (not being afraid of breaking stuff)
Testing necessary when the product become more complex they still do not have testing)
The interviewee not expert in automatic testing techniques
User retention is the most significant metrics to understand how the product is working in the market
We test mainly the most used features

Table A.21: Verification and Validation Codes

Code stats - Deployment

Code
Deploy on third party infrastructure (cloud)
Deployed on Virtual Private Server (VPS)
Automatic tools for staging and deployment
Direct deploy from development machines to production
SUG: Use simple automatic tools (Chef) for managing staging and production
Deploy new features using Git merging feature branches with master
Manual deploy from development machines to staging and then production
Weekly scheduled deploy
Deploy with the help of automatic tools
Extremely frequent new deployments
Heroku speed up the process avoiding infrastructural complexity
SUG: Increasing number of users require a staging environment before deploy in production
we deploy from 5 to 20 times per day

Table A.22: Deployment Codes

Code stats - Closing Questions

Code
Enthusiasm and motivation keep productivity high
Growing team requires more control / management on initial chaos
Working history between developer facilitated execution
ERR: Create a complex and big project for long time without evaluation
Growing team makes high volume of informal communication is a problem
SUG: Project management tools, if well integrated, boost productivity
Time pressure (Beating competitors)
Find product/market fit is a priority
Product/market fit
Productivity drop-down when growing team
Company is now mature and moving towards process and structure
Customers likes small improvements (not-expected small features)
Developers fear notify ticket status (feel monitored)
Developers in startups have big responsibilities
ERR: Not estimating because lack of experience
Even in the early stage at least two developers are required
Frameworks help growing faster
Front-end developers are over-rated
Good developers willing to work in a startup are hard to find
Good technical founders should hire excellent engineers
Growing is making release time longer (fear of break things) + (Releasing more features in one pack)
Growing requires a release plan
Growing requires efficient system for managing big number of small releases
Growing requires the CEO to slowly moving away from the code
Growing will always produce a productivity drop-down
Growing will cause an initial drop-down in productivity but with following improvement of speed
In startups developers multi skilled (generalists) are more useful than gurus in one technology
It's hard to say anything before the product is released
Not having a data schema and hiring a new developer is not a big deal
Past experience with similar product make development much more effective and fast
Productivity will improve when the teams work together for some time (feeling)
Speed and flexibility is the most important factor in the beginning
SUG: Clear business direction help development to go faster and reduce wasted features
SUG: Customer development in parallel to software development is very important
SUG: Get out of the office soon (to verify business assumption)
SUG: if you can't pitch your idea in 5 seconds, something is wrong
SUG: Introduction tutor/course for new hires
SUG: Is better to have a drop-down in productivity when team grows than lose time before
SUG: Record relevant metrics from day 1 to see what's happening with the product
SUG: Release weekly
SUG: To boost motivation start think day 1 how to make profit
SUG: After initial enthusiasm, you need revenue to boost morale
The more users the more feedback to manage (grow)
Time-pressure for media coverage led to productivity boost
Tradeoff - Drop-down productivity for lack of process (win) VS. Introducing early process (not worth it)
Using a rigid process led to problem with emergencies
Very satisfied on development approach

Table A.23: Closing Questions Codes

A.4.4 Interviews - Axial coding

During the generation of the theoretical framework, categories have been grouped together, organized into a tree-like structure. At the highest level of abstraction we identified 6 macro categories, in addition to the core category, that is *"Speed up development"*, considered as the most urgent priority by the totality of the

respondents. This is extensively described in Subsection 4.3.4. Table A.24 summarizes categories and subcategories identified in the study.

Category	Subcategory
Speed-up development	Working overtime to meet deadlines Use of standard/known technologies Development aided with well-integrated and simple tools Externalize infrastructural complexity on third party solutions Keep simple and informal workflow
Evolutionary approach	Find the product/market fit quickly Uncertain conditions make long-term planning not viable
Product quality has low priority	UX is the only important qualities Suitable and limited functionalities Efficiency emerges after using the product User is fault-tolerant in innovative beta product Cross-browser and cross-device compliance with aid of automatic tools Product should be reasonably ready-to-scale
Team is the catalyst of development	High-impact of CTO/CEO background Very small and co-located development team Developers have big responsibilities (self-organized) Multi-role and full-stack engineers Skilled developers are essential for high speed Team works under constant pressure Limited need of formalities between team-members Access to external expertise (Mentors)
Accumulated technical debt	Minimal Project Management Informal specification of functionalities Rough and quick feasibility study Lack of architectural design Lack of automated testing Tacit Knowledge replaces formal documentation
Growth harms performance	Pay off the accumulated technical debt Need of re-engineer the product Focus shifts to business concerns Company and user size grow
Severe Lack of resources	Time shortage Limited human resources Limited access to expertise

Table A.24: Grounded theory - Categories and sub-categories

In view of the relatively high complexity of the entire categories tree, the lower level sub-groups are not shown in this section, rather they are presented in detail in the theoretical model (see Section 6.3 or visit [221]).

A.4.5 Categories and engineering elements

Finally all the engineering elements identified in the follow-up results were mapped on the categories identified in the theoretical framework (see detailed discussion in Subsection 6.6.5).

Engineering elements	Category	Freq.
Very useful		
Analysis of critical/important use case scenarios	Rough and quick feasibility study	1

Table A.25 – *Continued on next page*

Table A.25 – *Continued from previous page*

Engineering elements	Category	Freq.
Asking user feedbacks for little adjustments only	Find the product/market fit quickly	1
Assembla for managing tickets/tasks	Use of well-integrated and simple tools	1
Basecamp for bugs/issues	Use of well-integrated and simple tools	1
Break-down of big tasks in smaller tasks	Lack of requirement engineering	1
Build APIs to export functionalities to mobile	Limited number of suitable functionalities	1
Create ticket on-the-fly without any analysis and design	Rough and quick feasibility study	1
Customer development and Lean startup methodology	Find the product/market fit quickly	1
Deployment on Amazon infrastructure	Externalize complexity to third party solutions	1
Developer responsible for designing,coding and testing a feature.	Multi-role and full-stack engineers	1
Developing the product without having schemas/diagrams	Lack of architectural design	1
Dropbox for sharing documents (x3)	Use of well-integrated and simple tools	3
Electronic Kanban Wall for managing features (Agile Zen)	Ticket-based tools to manage stories/features	1
Evolutionary prototyping/MVP (X4)	Find the product/market fit quickly	4
Focus Group for assessing graphical aesthetic	Find the product/market fit quickly	1
Focus Group for setting the main functionalities	Find the product/market fit quickly	1
No need of formal analysis (past experience)	Rough and quick feasibility study	1
Having Mentors in early stage	Access to mentors expertise	1
Hip-chat for internal communication	Use of well-integrated and simple tools	1
Lack of detailed documentation	Tacit Knowledge	1
Lack of formal and automatic testing	Lack of automated testing	1
List of features (paper notes)	Use of well-integrated and simple tools	1
Mock-ups of the UI	Lack of architectural design	1
Naive diagrams (disposable) instead of UML communication diagrams	Lack of architectural design	1
Postpone potential choices which could "limit"	Uncertain conditions make long-term planning not viable	1
Structure the app in a self-explanatory way with Rails	Tacit Knowledge	2
Collecting initial feedbacks before coding	Find the product/market fit quickly	1
Setting informal deadlines between co-founders	Minimal Project Management	1
Short release time (weekly deployment)	Light lean startup principles	1
Sketches (wireframe)	Lack of architectural design	1
Skype for assigning bugs	Use of well-integrated and simple tools	1
Starting from a previously developed technology	Use of standard/known technologies	1
SVN for the codebase	Use of well-integrated and simple tools	1
Let user test secondary functionality	Lack of automated testing	1
Treating bugs as user stories	Minimal Project Management	1
Trying the product internally to identify bugs/issues	Lack of automated testing	1
Use Cases (Assembla)	Rough and quick feasibility study	1
User feedbacks (by means of the "super circle" of users)	Find the product/market fit quickly	1
UserVoice for collecting users' feedback	Find the product/market fit quickly	1
Using a Whiteboard	Use of well-integrated and simple tools	1
Using Linode to deploy the application	Use of well-integrated and simple tools	1
Extremely useful		
Amazon EC2	Externalize complexity to third party solutions	1
Analyzing competitors' feedbacks	Rough and quick feasibility study	1
Basecamp's tasklist	Lack of requirement engineering	1
CEO solving conflicts in development decisions	Tacit Knowledge	1
Chef for deployment	Use of well-integrated and simple tools	1
Clean-code	Tacit Knowledge	1
Co-located team members	Very small and co-located development team	1

Table A.25 – *Continued on next page*

Table A.25 – *Continued from previous page*

Engineering elements	Category	Freq.
Collecting feedback from pre-launch through landing page	Light lean startup principles	1
Competitor's analysis	Rough and quick feasibility study	1
Consulting available gems before start implementing	Rough and quick feasibility study	1
Multi-role and full-stack engineers (full stack developers)	Multi-role and full-stack engineers	1
Daily stand-ups	Keep simple and informal workflow	1
Database model	Lack of architectural design	1
Deploy workflow (feature brench-¿ pull req -¿ Capistrano)	Use of well-integrated and simple tools	1
Development experience	Skilled developers are essential for high speed	1
Feature meetings	Naive task assignment mechanism	1
Flat hierarchy of the team	Multi-role and full-stack engineers	1
Get early feedback from customers	Find the product/market fit quickly	1
Git for code-base	Use of well-integrated and simple tools	1
Github for having review of the code	Use of well-integrated and simple tools	1
Having senior developers	Skilled developers are essential for high speed	1
Heroku for deployment	Externalize complexity to third party solutions	1
High-experience developers	Skilled developers are essential for high speed	1
HTML5, CSS3	Use of well-integrated and simple tools	1
Hype of media for increasing moral of developers	High enthusiasm boosts productivity	1
Informal meetings for discussing biggest changes only	Keep simple and informal workflow	1
Initial feature list (whiteboard)	Use of well-integrated and simple tools	1
Initial survey to collect user feedbacks	Light lean startup principles	1
Mind mapping instead of text writing communication	Tacit Knowledge	1
MySQL as DBMs	Use of standard/known technologies	1
Only in-line comments to document the code	Tacit Knowledge	1
Oral communication	Tacit Knowledge	1
Personal experience for story estimation	Rough and quick feasibility study	1
Post-it notes for tracing tasks and bugs	Keep simple and informal workflow	1
Prototype an Hybrid Django/Php	Use of standard/known technologies	1
RESTful API	Use of standard/known technologies	1
Scrum board (by means of whiteboard with post-it notes)	Keep simple and informal workflow	1
Self-imposed informal deadlines (Google spread-sheet)	Minimal Project Management	1
Skype for communication	Use of well-integrated and simple tools	1
Using an MVP approach	Find the product/market fit quickly	1
Using whiteboard for main focus of the product/vision	Use of well-integrated and simple tools	1
Whiteboard for tracing the progress (modules implemented)	Minimal Project Management	1
Hindsights		
Crazy egg for ux testing	Adapt to early feedbacks	1
Ruby On Rails that forced me to use an MVC approach on development	Lack of architectural design	1
Basecamp and TODO-list	Lack of Requirement Engineering	1
Delelop and release fast each time we had a new feature.	Find the product/market fit quickly	1
High skilled team.	Informal specification of functionalities	1
I wanted to focus more attention on designing the interface and the analysis of the needs of end users (the tourist)	Find the product/market fit quickly	1
90% of developers were full stack	Multi-role and full-stack engineers	1
The excellence of the whole technical team	Skilled developers are essential for high speed	1
Continuous deployment	Find the product/market fit quickly	1
Also UX/CPO/UI could code	Multi-role and full-stack engineers	1
Developers and the heterogeneity of knowledge	Skilled developers are essential for high speed	1

Table A.25 – *Continued on next page*

Table A.25 – *Continued from previous page*

Engineering elements	Category	Freq.
Motivation to innovate all played an extremely important role	Multi-role and full-stack engineers	1

Table A.25: Questionnaire results to theoretical framework

A.5 Technical debt, potential capability and speed measurement

This appendix discusses how the numerical results related to *potential capability*, *technical debt* and *execution speed* (presented in Subsection 6.6.4) have been obtained to validate the high level relation of the model.

In particular we explain the process we utilized to measure three measures:

- *Potential capability*: a metric that represents the degree to which each company reflected the capability of reaction and flexibility to the dynamic environment during the development process, given by the three categories that (theoretically) mostly affect *speed-up development* (see Figure 6.7 and Subsection A.5.1).
- *Execution speed*: a metric that represents the development speed of the startup during different phases of the first release, computed by means of a weighted average speed for each phase, by analyzing interview transcripts looking at subcategories of *Speed-up development* (see Subsection A.5.2).
- *Technical debt*: a metric that represents the extent to which processes are controlled, structured, planned and documented by means of engineering artifacts and practices. It has been computed by means of a weighted average of the debt accumulated in each development phase observing subcategories of *accumulated technical debt*, with consequences on startups' growth (see Subsection A.5.2).

A.5.1 Potential capability

To define the last measure - *potential capability* - we quantified characteristics related to three theoretical categories, as discussed in Subsection 6.6.4: *team is catalyst of development* (CAT4), *product quality has low priority* (CAT3), and *evolutionary approach* (CAT2). According to the framework, these categories contribute respectively to performance, efficiency and effectiveness - and we want to attest the validity of these relations.

Following the example of the SMS, the procedure has been executed simultaneously in pair on the same screen. When necessary we performed an in-depth

review of the transcript[5].

To begin the numerical evaluation, we associated a weight to each category (Table A.26), reflecting the importance according to the empirical data (see Subsection 6.6.5). Factors related to the team have the largest impact on the score (0.5). Factors related to the methodology undertaken are slightly more important (0.3) than quality-related concerns (0.2).

ID	Category	Weight
CAT4	Skilled team is the catalyst of development	0.5
CAT2	Evolutionary approach	0.3
CAT3	Product quality has low priority	0.2

Table A.26: Capability - Weights

Subsequently we evaluated the three categories separately, assigning a score to each company according to relevant subcategories[6]. In the next subsections we present the detailed evaluation performed on the three categories. Each subcategory has been evaluated using a discrete scale 0 to 2 where 0 represent a null contribution, 1 is average, and 2 is above the average.

Team factors

Selected subcategories from *team is the catalyst of development (CAT4)*:

- T1: High-impact of CTO background.
- T2: Very small and co-located development team.
- T3: Developers have big responsibilities (self-organized).
- Multi-role and full-stack engineers:
  - T4: Engineers are responsible for marketing/sales/development (flat structure).
  - T5: Generalists developers instead of specialists (full-stack).
- T6: Skilled developers are essential for high speed.
- T7: Team works under constant pressure.
- Limited need of formalities between team-members:
  - T8: Positive impact of high co-location.
  - T9: Previous working experience.
  - T10: Knowing each other before starting the company.

The results of the evaluation are reported in Table A.27, where the *weighted score* has been computed by summing up the individual scores obtained in subcategory, multiplying the value by the weight previously defined and finally normalize

[5]If the conflicts persisted after an in-depth review of the transcript, we let a third expert person (i.e. our supervisors) take the final decision.

[6]We excluded categories equally impacting all companies since contributing 0.

it on a scale 1 to 5 in order to be able to make a comparison with the *technical debt* and *execution speed*.

Company	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	**Weighted score**
C1	2	1	2	1	1	1	2	1	2	1	**2.78**
C2	0	1	2	1	2	0	2	1	1	1	**2.18**
C3	1	0	1	0	0	1	1	2	1	0	**1.39**
C4	2	1	2	0	1	2	1	1	1	0	**2.18**
C5	2	1	1	1	1	1	1	1	1	0	**1.98**
C6	1	1	1	1	0	1	1	1	0	0	**1.39**
C7	1	0	1	1	0	0	1	2	1	0	**1.39**
C8	2	1	2	1	1	2	1	2	1	1	**2.78**
C9	1	1	2	0	1	1	1	2	2	2	**2.58**
C10	1	1	1	0	1	2	1	2	1	1	**2.18**
C11	2	1	2	1	1	2	2	2	2	2	**3.37**
C12	1	1	1	1	0	1	1	1	2	2	**2.18**
C13	1	0	1	1	1	1	0	1	0	2	**1.59**

Table A.27: Capability - Team

Development approach factors

Selected subcategories from *Evolutionary approach (CAT2)*:

- E1: Flexibility and Reactiveness are main objectives.
- E2: Build a functioning prototype and iterate on it (MVP).
- E3: Progressively roll-out to larger number of people.
- E4: Focus on minimal set of functionalities (suitability).
- E5: Small iterations (release often).
- E6: Find product-market fit as soon as possible.

The results of the evaluation are reported in Table A.28.

Company	E1	E2	E3	E4	E5	E6	**Weighted score**
C1	1	1	1	1	2	1	**0.83**
C2	1	1	0	2	1	1	**0.71**
C3	1	1	1	1	1	1	**0.71**
C4	2	2	2	2	2	2	**1.43**
C5	1	1	1	1	1	2	**0.83**
C6	2	2	0	2	1	1	**0.95**
C7	2	2	1	2	2	2	**1.31**
C8	2	1	2	2	2	2	**1.31**
C9	1	1	1	1	1	1	**0.71**
C10	1	0	2	1	2	2	**0.95**
C11	2	2	2	1	2	2	**1.31**
C12	0	0	0	0	1	0	**0.12**
C13	0	0	1	0	1	0	**0.24**

Table A.28: Capability - Evolutionary

Quality factors

Selected subcategories from *Product quality has low priority (CAT3)*:

- Q1: UX is the only important quality.
- Q2: Limited number of suitable functionalities.
- Q3: Users are fault-tolerant in innovative beta products.
- Q4: Efficiency emerges after using the product.
- Q5: Product should be reasonably ready-to-scale.

The results of the evaluation are reported in Table A.29.

Company	Q1	Q2	Q3	Q4	Q5	**Weighted score**
C1	1	1	1	1	0	**0.32**
C2	1	1	1	0	0	**0.24**
C3	1	0	1	0	0	**0.16**
C4	2	2	1	1	2	**0.63**
C5	1	1	1	1	2	**0.48**
C6	1	1	0	1	0	**0.24**
C7	2	0	0	0	1	**0.24**
C8	0	2	1	1	1	**0.40**
C9	1	1	1	1	0	**0.32**
C10	1	1	1	1	1	**0.40**
C11	0	2	0	1	1	**0.32**
C12	1	0	0	0	1	**0.16**
C13	1	0	1	0	1	**0.24**

Table A.29: Capability - Quality

Finally the three overall scores for *potential capability* have been computed by summing up the weighted contributions of the three categories. Table A.30 show the final scores of *potential capability*.

Company	Potential capability
C1	3.928571429
C2	3.134920635
C3	2.261904762
C4	4.246031746
C5	3.293650794
C6	2.579365079
C7	2.936507937
C8	4.484126984
C9	3.611111111
C10	3.531746032
C11	5.000000000
C12	2.460317460
C13	2.063492063

Table A.30: Potential capability

In conclusion, all the operations executed in this appendix had the only scope

of attesting the correctness of relations between high-level categories of the framework (see Subsection 6.6.4). Although the scores assigned are subject to researchers personal bias, we executed the whole process in pair, and we provided to other researchers the detailed rubrics and instructions necessaries for executing similar evaluations.

A.5.2 Execution speed and Technical debt

Since both *executions speed* and *technical debt* have been computed by summing up weighted contributions phase by phase, they are presented together. First, the phases have been structured outlining the configuration of the interviews[7]:

- S1: idea/vision/objectives dissemination.
- S2: requirements engineering.
- S3: analysis.
- S4: architecture design.
- S5: coding/debugging.
- S6: verification and validation.
- S7: deployment.
- S8: general project management.

Afterwards, for each company we assigned a weight to each phase based on the effort declared by the respondents in the follow-up questionnaire[8] (see Subsection 5.2.3). The weights are shown in Table A.31.

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Sum**
C1	0.02	0.08	0.11	0.04	0.23	0.23	0.08	0.23	1
C2	0.02	0.08	0.08	0.08	0.23	0.15	0.15	0.23	1
C3	0.02	0.11	0.04	0.08	0.45	0.06	0.02	0.23	1
C4	0.03	0.13	0.04	0.09	0.53	0.07	0.02	0.09	1
C5	0.03	0.09	0.09	0.13	0.40	0.16	0.02	0.09	1
C6	0.03	0.09	0.09	0.27	0.27	0.09	0.09	0.09	1
C7	0.03	0.09	0.04	0.04	0.44	0.22	0.04	0.09	1
C8	0.03	0.13	0.04	0.09	0.53	0.07	0.02	0.09	1

Table A.31 – *Continued on next page*

[7]Observe how we added a new "phase" that was not present initially in the structure of the interview, but emerged from respondents, which typically started to answer our questions of requirement engineering talking about *how they transmitted the initial idea to other team members*.

[8]For the four companies, which didn't filled the survey, we used average values, adjusted according to interviews.

Table A.31 – *Continued from previous page*

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Sum**
C9	0.03	0.18	0.09	0.18	0.27	0.09	0.09	0.09	1
C10	0.02	0.14	0.07	0.11	0.25	0.07	0.07	0.27	1
C11	0.02	0.08	0.08	0.12	0.41	0.13	0.07	0.08	1
C12	0.03	0.13	0.13	0.18	0.35	0.07	0.02	0.09	1
C13	0.03	0.12	0.09	0.13	0.34	0.14	0.07	0.09	1

Table A.31: Weights

Figure A.4 is a graphical representation of the weights assigned to each phase presented in Table A.31.

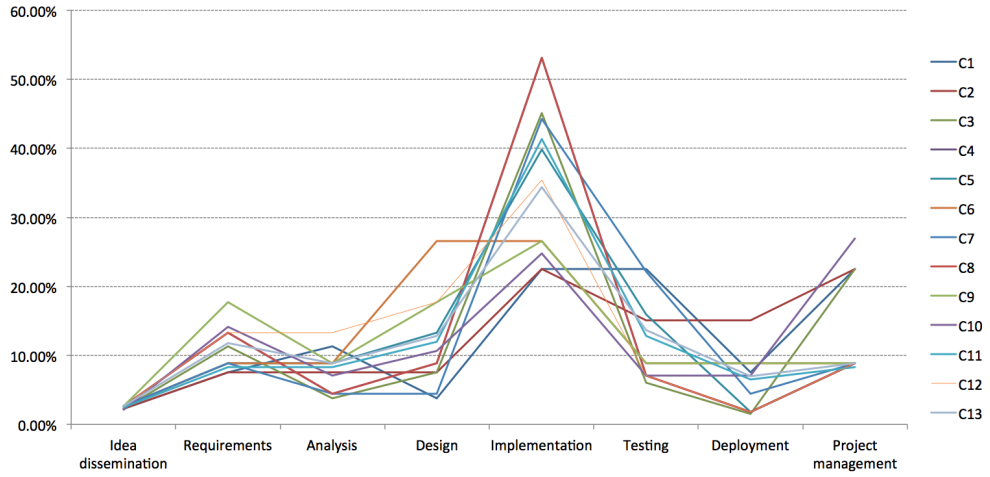


Figure A.4: Execution speed and Technical debt, by phase

As expected, more importance has been assigned to the implementation phase, which occupied most of time and resources of the company. Afterwards we defined a rubric table with extreme values indication phase by phase, used to assign a score from 1 to 5 to each company, for *execution speed* and *technical debt* (see Table A.32).

Idea dissemination (S1)		
Score	Execution speed	Technical debt
1	The vision and objectives formally stated with heavy documentation, which needs to be manually updated.	The vision and objectives are maintained through automatic tools, which promptly inform the team-members.
5	Vision clearly shared among team-members.	Not having any means to share the vision.
Requirements engineering (S2)		
Score	Execution speed	Technical debt
1	The company needs to execute a slow process because of a long list of artifacts and formalized specifications.	Requirements artifacts are complete, up-to-date, accessible, structured, traceable and with shared ownership.

Table A.32 – *Continued on next page*

Table A.32 – *Continued from previous page*

5	Use of highly automated tools and low-precision artifacts to specify the initial list of features/stories.	Features are not documented and shared through oral communication and tacit knowledge

Analysis (S3)		
Score	Execution speed	Technical debt
1	Complete analysis requires formal processes such as risk analysis, feasibility study, critical evaluation of alternative technologies	Pre-defined modus operandi in view of all possible risky situations, fully documented and up-to-date.
5	The only analysis is conducted by logical thinking and reasoning on possible scenarios.	Lack of minimal risk mitigation strategies, and limiting decision taking.

Design (S4)		
Score	Execution speed	Technical debt
1	Formal architecture analysis with detailed diagrams and extended design documentation.	Complete, traceable and up-to-date architectural design artifacts or rigid use of established framework.
5	Little up-front-design supported by well-known framework solutions.	Home-made/No-standard or monolithic architecture design without any support of documentation.

Implementation (S5)		
Score	Execution speed	Technical debt
1	Heavy processes definition with lack of automated/integrated tools and use of out-of-code documentation.	Use of coding standard and clean code accessible by any developer (self-explanatory) with advanced versioning system and task management. Collective ownership and critical decision documented and traceable.
5	Lack of out-of-code documentation to update and minimal workflow guidelines without any burocracy and simple high automated/integrated tools (advanced versioning system, online collaborative ticket-management)	Lack of coding standards and documentation of critical parts in and out of code. Lack of any task management and versioning system

Verification and validation (S6)		
Score	Execution speed	Technical debt
1	Systematic and rigorous quality assurance processes.	High industry standard met for extensive automatic testing systems with documented test cases.
5	Highly automized and quick tests of critical parts of the system only.	Complete lack of both automatic and manual testing systems.

Deployment (S7)		
Score	Execution speed	Technical debt
1	Formal deployment policies with multi-staging system or heavily procedures manually conducted.	Advanced automatic tools or services for multi-staging, easy-to-scale deployment, with a defined documentation for deployment steps.
5	New features directly deployed to production with support of simple and automatic tools.	Manual deployment without any documented procedure to follow.

General project management (S8)		
Score	Execution speed	Technical debt
1	Heavy project management with detailed scheduling, tracing team metrics...	Documented and up-to-date well-defined workflow of activities with support of automatic/online tools.
5	Minimal low-precision tools for task management and internal deadlines for the critical milestones.	Lack of any simple workflow with heavy use of manual tools and oral communications.
Extreme values: 1 = very low; 5 = extremely high		

Table A.32: Rubrics for execution speed and technical debt

We used these guidelines to consistently evaluate in pair the companies, based on the data collected during the case study. The rubrics naturally emerged dur-

ing the process of evaluating the first companies, and were constantly updated throughout the process. In case of disagreement, conflicts were examined in-depth to reach a final consensus, sometimes by consulting interview transcripts.[9] The results of the evaluation of *execution speed* is shown in Table A.33, and for *technical debt* in Table A.34.

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Weighted score**
C1	5	4	4	4	4	4	4	4	**4.022556391**
C2	3	4	4	4	3	4	4	5	**3.977443609**
C3	3	4	4	3	4	4	5	3	**3.691729323**
C4	4	5	5	5	4	3	3	4	**4.17699115**
C5	5	4	5	5	4	5	4	4	**4.407079646**
C6	3	3	5	4	4	4	4	4	**3.973451327**
C7	4	5	4	4	3	4	5	5	**3.778761062**
C8	4	3	3	4	5	4	4	5	**4.442477876**
C9	5	4	5	3	4	4	4	4	**3.938053097**
C10	3	4	3	4	3	4	4	4	**3.659574468**
C11	5	5	5	4	5	5	4	4	**4.732290708**
C12	3	4	4	4	4	5	5	5	**4.150442478**
C13	3	3	3	4	3	4	4	4	**3.422812193**
Average	3.85	4.00	4.15	4.00	3.85	4.15	4.15	4.23	**4.03**

Table A.33: Execution speed

Company	S1	S2	S3	S4	S5	S6	S7	S8	**Weighted score**
C1	2	4	2	4	2	4	4	4	**3.052631579**
C2	1	2	3	3	3	5	3	3	**3.180451128**
C3	2	3	1	4	2	4	3	3	**2.601503759**
C4	3	4	3	4	3	2	4	4	**3.362831857**
C5	3	3	3	3	3	2	3	3	**3.14159292**
C6	2	4	3	3	3	3	3	3	**3.061946903**
C7	1	2	1	3	3	2	4	4	**3.03539823**
C8	2	2	2	3	4	1	3	3	**3.362831858**
C9	2	3	2	2	3	3	3	3	**2.796460177**
C10	1	2	1	3	2	2	2	2	**2.085106383**
C11	2	3	3	3	4	2	3	3	**3.451701932**
C12	2	2	2	3	4	2	3	3	**3.115044248**
C13	2	3	3	3	3	3	3	3	**2.973451327**
Average	1.92	2.85	2.23	3.15	3.00	3.16	2.69	3.15	**3.02**

Table A.34: Technical debt

A.5.3 Statistical tests

Summarizing the results of the dimensions discussed in the previous subsections, we present the results in Table A.35. Following, as discussed in Subsection 6.6.4,

[9] Observe that we tried to define two metrics which can be measured independently from each other and from external factors (team experience, project type, ...). How these factors can influence the amount of accumulated technical debt or the execution speed can vary case by case. However, it is not in the scope of this thesis exploring those relations.

we conducted statistical tests using the analysis of variance to assess the existence of relations between *execution speed*, *potential capability* and *technical debt*.

Company	Execution speed	Potential capability	Technical debt
C1	4.022556391	3.928571429	3.052631579
C2	3.977443609	3.134920635	3.180451128
C3	3.691729323	2.261904762	2.601503759
C4	4.17699115	4.246031746	3.362831857
C5	4.407079646	3.293650794	3.14159292
C6	3.973451327	2.579365079	3.061946903
C7	3.778761062	2.936507937	3.03539823
C8	4.442477876	4.484126984	3.362831858
C9	3.938053097	3.611111111	2.796460177
C10	3.659574468	3.531746032	2.085106383
C11	4.732290708	5.000000000	3.451701932
C12	4.150442478	2.46031746	3.115044248
C13	3.422812193	2.063492063	2.973451327

Table A.35: Quantification results of *execution speed*, *technical debt* and *potential capability*

First we defined two null hypotheses (H0): $H0_1$ = *Startups do not release the product faster when a capable team adopt a more evolutionary approach AND with less quality constraints*; $H0_2$ = *The execution speed does not increase the amount of accumulated technical debt.* Then we tested $H0_1$ and $H0_2$ with an one-tailed test using Pearson's product moment correlation coefficient, with positive association analysis, fixing the level of confidence to 95% which means we reject H0 in case the p-value is lower than 0.05.

We concluded that, in our sample:

1. Higher values of *Execution speed* are strongly associated with *higher* values for *Technical debt* (with a clear statistical significance, p-value: 0.002073).
2. Higher values of *Execution speed* are strongly associated with *higher values* for *Potential capability* (with a clear statistical significance, p-value: 0.004549).

To perform the Pearson's correlation we verified two assumptions: a) data is on interval scale; b) data is normally distributed.

In regard to assumption a) there is a considerable disagreement in the literature whether individual *Likert* items can be considered as interval-level data

[222] [223]. However, we provide a symmetric *Likert scale* with a middle category and clearly defined linguistic qualifiers for each item (with the aid of the theoretical framework and rubrics, presented respectively in subsections A.5.1 and A.5.2). Then, we made our best to present evaluations that are homogeneously (with same interpretation) distributed across the different companies data. Furthermore we improved the approximation of an interval-level measurement by adjusting weights of scores of categories to equally space the 'distance' between the final scores with the use of validated follow-up questionnaire results (see Subsection 5.2.3).

In regard to assumption b) we validated it by conducting the *Kolmogorov-Smirnov* (K-S) test. Each dimension has been tested separately, comparing them with a normal distribution with the same mean and standard deviation. The output gives the output statistic deviation (D) and then a p-value associated with that statistic. Moreover each dimension's distribution is presented by a histogram and respective normal curve.

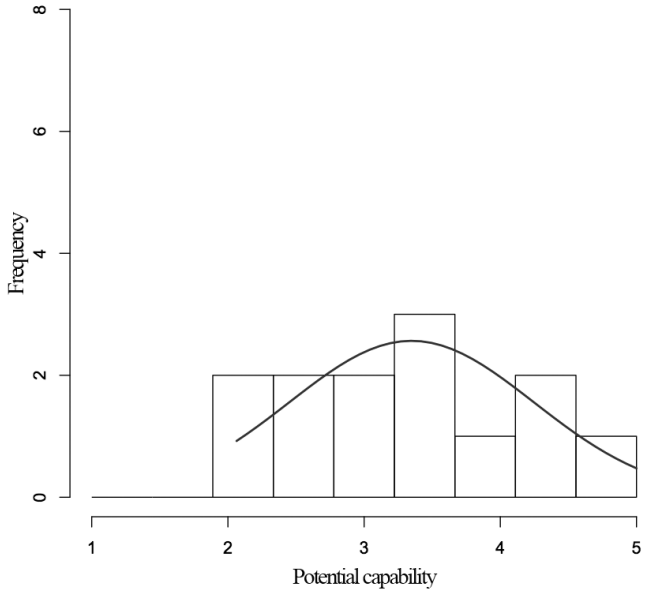


Figure A.5: Distribution of *potential capability*

The output of the *potential capability* K-S test is: D = 0.1117, p-value: 0.9909.

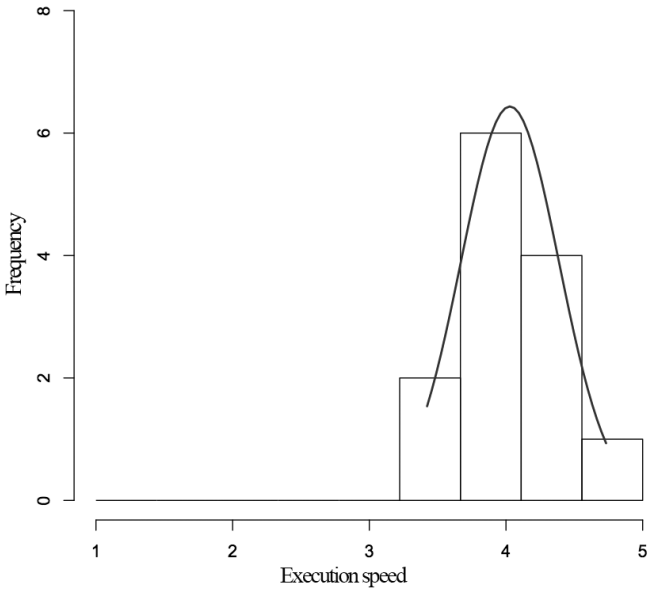


Figure A.6: Distribution of *execution speed*

The output of the *execution speed* K-S test is: D = 0.1223, p-value: 0.9769.

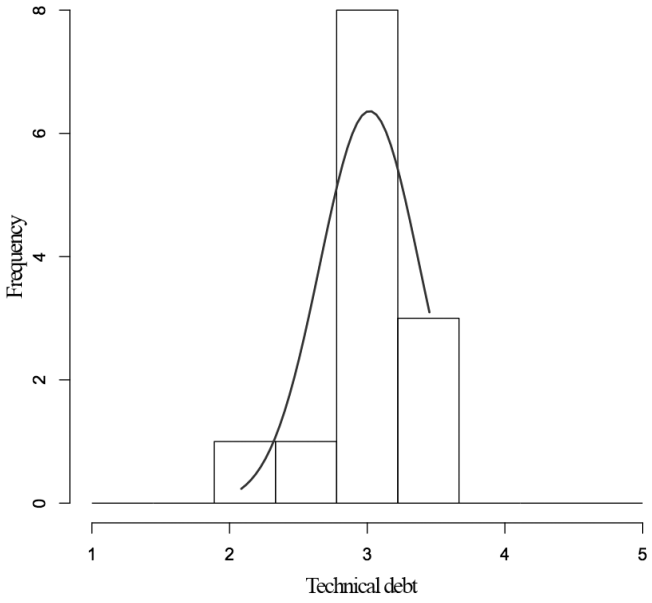


Figure A.7: Distribution of *technical debt*

The output of the *technical debt* K-S test is: D = 0.2214, p-value: 0.4799. As shown above in all the cases the p-values are well above 0.05, so we accept that there is no difference between the observations and a set of random observations drawn from a perfect normal distribution with the same mean and variance.

Glossary

analysis study of feasibility of the project and risks, discovering possible mitigation strategies.

architecture design conceptualization and abstraction of possible solutions by means of modelling processes.

CEO stands for chief executive officer, which represents an administrative position in a company, focusing on total management of an organization. This role has main responsibility of developing and implementing high-level strategies, making major corporate decision, managing the overall operations and resources of a company.

coding/debugging development process involving maintenance activity as debugging and refactoring.

conceptual framework explains, either graphically or in narrative form, the object of study (the key factors, constructs or variables) and the presumed relationships among them. Framework can be rudimentary or elaborate, theory driven or commonsensical, descriptive or causal [153].

control the process of collecting information procedures about the system, process, person, or group of people in order to make necessary decisions about each of them.

cross-sectional study a class of research methods that involve observation of a phenomenon in a specific period of time.

CTO stands for chief technology officer, which represents an executive-level position in a company, focusing on scientific and technological issues. This role has main responsibilities in long and short term technology directions, focusing on research and development of software product.

customer development is a process that might be conducted in parallel with the software development process. It consists of four main steps: customer discovery, customer validation (that might go back to customer discovery iteratively), customer creation and company building [17].

deployment process of making the software system available for use.

engineering activities are those tasks necessary to develop and maintain a product from the idea conceptualization to the first release to the market (requirements engineering, architecture design, verification and validation, ...).

engineering elements are any method/practice/tool/framework/technique documentation/artifact contributing and supporting the engineering activities.

general project management is the discipline of planning, organizing, securing managing, leading and controlling resource to achieve specific goals.

growth an increase in the company size respect to the initial conditions in terms of either employees or users/customers, and product complexity in terms of handling an increasing number of feature requests.

highly scalable markets is a characteristic of startups that describes their capability to cope and perform with an highly expanding number of users.

idea/vision/objectives dissemination specification and dissemination of the idea conception from the startup founders.

Lean startup methodology is a methodology involving three main steps: learning, building measure in cycle. Then validated learning, scientific experimentation, and iterative product releases to shorten product development cycles, measure progress. Important aspect is gaining the valuable customer feedback as soon as possible by means of the minimum viable product [7].

minimum viable product is a version of a new product with the minimal set of features, which allows a company to quickly release the product. This allows a team to collect the maximum amount of knowledge about product/market fit with the least effort. MVP is a strategy and process directed toward making an initial prototype by an iterative process of idea generation, data collection, analysis and learning, minimizing the total time spent on each iteration [7].

operational dynamics the approaches of the company in making decisions.

planning is the process of thinking and organizing the activities required to achieve a desired goal. Planning involves the creation and maintenance of any diagram or list of steps with timing and needed resources.

quality attributes are those overall factors that affect run-time behavior, system design, and user experience. They represent areas of concern that have the potential for applications to impact across layers and tiers. Some of these attributes are related to the overall system design, while others are specific to run time, design time, or user centric issues.

requirements engineering process of discovering, documenting and maintaining requirements specification.

software development strategy is the overall approach adopted by the company to develop the product with the help of engineering activities and elements.

stabilization occurs when an organization reaches an operational steady state. In other words, the processes are executed in a predictable, repeatable and controllable way. Exception processes are minimal, if they exist at all.

structure denotes the definitions of components and relations that actually constitute a particular process adopted during software development.

technical debt is a design or development approach thats expedient in the short term but that increases complexity and is more costly in the long term.

technical practitioners are those practitioners who show the ability to demonstrate an understanding of the engineering principles and knowledge in developing software products. In the case of startups, CEOs typically assume the role of technical practitioner too.

trade-off is a situation of compromising a quality aspect of a certain attribute for gaining advantage of another quality aspect of the same or different attribute.

user experience are the all aspects related to the end-users interaction with the company, its services and its products. The first requirement for an exemplary user experience is to meet the exact needs of the customer without fuss or bother. Next comes simplicity and elegance that produce products that are a joy to own, a joy to use. True user experience goes far beyond giving customers what they say they want, or providing checklist features. The selected definition is UX definition, undertaken by the companys perspective, illustrated in [224].

verification and validation testing processes and validation by means of customers feedbacks (acceptance testing).