Software Development in Startup Companies: The Green-field Startup Model

FirstName SecondName, *Fellow, IEEE,* BLA BLA, *Fellow, IEEE,* and BLA BLA, *Life Fellow, IEEE*

Abstract—Software startups are newly created companies with no operating history and extremely fast in producing cutting-edge technologies. Despite their increasing importance in economy there are only few scientific studies attempting to address software engineering (SE) issues, especially for early-stage startups. This research aims to understand the software development strategies conducted by practitioners, in the period of time that goes from idea conception to the first open beta release. A *grounded theory* approach guided the execution of a case study to explore the state-of-practice. As result, the Greenfield startup model (GSM) reveals 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 early collected user feedbacks. Nevertheless, the need of shortening time-to-market, by speeding-up the development through low-precision and product-centric engineering activities, is counterbalanced by the need of restructuring the product before setting off for further growth.

Index Terms—Software Development, Startups, Grounded Theory.

1 Introduction

An impressive number of new software startups are launched worldwide every day as a result of an increase of new markets, accessible technologies, and venture capital [1]. With the term *software startups* we refer to 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.

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 [2]. Operating in a chaotic, rapidly evolving and uncertain domain, software startups face intense time-pressure from the market and are exposed to tough competition [3], [4]. 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 [5].

From a software engineering perspective development in startups is challenging since they develop software in a context where processes can hardly follow a prescriptive methodology [5], [6]. 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 [7], [5]. Research is needed to support their engineering activities [6], guiding practitioners in taking decisions and avoiding choices that could easily lead the whole business to failure [8]. However, despite the impressive size of the startup ecosystem [9], the state-of-the-art presents a gap. Through a *Systematic Mapping Study* (SMS) of the literature, conducted by Paternoster et al. [10], only few publications were found related to engineering activities in startups. Moreover the majority of these studies do not possess the relevance and rigour required to form a consistent body of knowledge in the state-of-the-art and in the state-of-practice.

This research 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.

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, as discussed in [3], [4]. 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.

By means of the Green-field startup model (GSM), we captured the underlying phenomenon of software development in early-stage startups. The framework is based on a systematic procedure which is grounded into empirical concepts obtained by conduction of the case-study. GSM presents the most significant themes of the development strategies that characterise startups' context.

The most urgent priority of software development is to shorten time-to-market. The greater part of engineering activities of startups are focused on the implementation

- *Author info.*
  E-mail: see http://something/contact.html
- *Author info.*

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.

Nevertheless, by speeding-up the development process, startups accumulate technical debt, ignoring processes, relying on informal communication and replacing documentation with low-precision artifacts. 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 to contain the severity of accumulated technical debt.

Practitioners can use the GSM model to apply presented strategies to speed-up the development by taking in consideration the amount of technical debt accumulated during the first early-stage period. In this regard, we identified several commonalities between the issues related to software development in startups and the research focused at studying technical debt [11], [12]. On the other hand, researchers can take advantage from the GSM model to understand how the technical debt influences the future growth of startup companies.

The remain of this paper is structured as follow: we provide relevant information about startups in section 2. Section 3 introduces the research questions and discusses the design and execution of the research process. Results are discussed in section 4 which presents the GSM model. Section 6 compares results of this research to literature and frameworks. Section 7 discusses the validity threats considered during this study. Finally, this paper concludes in section 8.

2 Background

Modern entrepreneurship, which was born more than thirty years ago [13], 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 [14]. 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 [15]. 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 [2].

Looking at the number of new business incubators which appeared in the last three years one can evaluate the importance of startups [16]. 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 [17].

The implementation of methodologies to structure and control the development activities in startups is a major challenge for engineers [18]. Generally, the management of software development is achieved through the introduction of software processes, which defineswhat steps the development organizations should take at each stage of production and provide assistance in making estimates, developing plans and measuring quality [19]. 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 [20], [18], [5].

In this 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 result ineffective [5]. Also Bach refers to startups as *"a bunch of energetic and committed people without defined development processes"* [21]. In fact, startups have very limited resources and typically wish to use them to support product development instead of establishing processes [18], [22]. 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"* [23]. Rejecting the notion of repeatable and controlled processes, startups prominently take advantage of unpredictable, reactive and low-precision [24] engineering practices [5], [25], [26], [27].

Moreover, most startups develop packaged applications rather than software for a specific client [28]. 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* [29]). Among other results, researchers emphasize the importance of time-to-market as a key strategic objective [30], [30], [31] 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" [32], "rarely documented" [33], and can be validated only after the product is released to market [34], [35]. Under these circumstances, failure of product launches are largely due to "products not meeting customer needs" [29].

Accordingly, product-oriented practices help startups having flexible teams, with workflows that leave them the ability to quickly change the direction according to the targeted market [22], [5]. In this regard, many startups focus on team productivity, asserting more control to the employees instead of providing them rigid guidelines [25], [26], [27].

2.1 Lack of research in startups

Despite some studies tried to address the above mentioned issues, [10] found only a few SE works in this

specific area, as confirmed by other studies [18], [20], [36], [5]. 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*.

A research published in 2000 by Sutton [5] 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. [18], [20], [36] eight years later.

2.2 Definition of startup

The word *startup* appeared in the SE literature for the first time in 1994 in an article written by Carmel [37] where he studied the *time-to-completion in young package firm*. 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.

2.3 Evolution of the process

First insights reveal how software startups are product-oriented in the first period of their development phase [22]. Despite good achievements at the beginning, software development and organizational management increase in complexity [38], [39] causing deterioration of performance over time. Briefly, the necessity of establishing initial repeatable and scalable processes cannot be postponed forever.

A study of Kajko-Mattsson [8], 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 [40]. In this regard, Crowne, in [2], 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.

But when startups have no time for training and orienting activities, as discussed in [5], their main focus remains on team capabilities instead of prescriptive processes hiring people who can *"hit the ground running"* [41]. 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 [37]. Only when they grow, formal methodologies arise, followed by a conduction of quality assurance and long-term planning processes [41].

As studied in [42], the maturity of a company affects the extent to which processes are adopted. The author reports how introducing Extreme Programming (XP) principles [43] in the development process was challenging because of the need of trained team-members for fully implementing the methodology. In fact, [44] 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 [45].

As partially discussed above, contributions to flexibility and reactiveness of the development process has been conducted prominently by means of *Lean* [46] and *Agile* [47] methodologies (also reported in [48], [49]), 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 [50], [51], [5]. Customer involvement in software development has also been discussed in [52] as important factor to encourage an early alignment of business concerns to the technology strategies, because both are salient considerations to be successful [42].

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 [5]. In this regard, many startups have been focused on team productivity, providing more control to the employees instead of providing them rigid guidelines [25], [26], [27].

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 [53]. 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* [54]. 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 [55]. The author reveals the problem in actuating the same *best-practices* criteria for established companies in 10-person software startups. Thoroughly remarked in [5], 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 [20] 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 [56], [57], 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 [18], [58].

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 [59], [60], [61], [62].

2.4 Technical debt

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 [63] and has recently attracted the attention of SE researchers[1]. The concept of *technical debt* is well illustrated in [65] : *"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, ...).

In conclusion, since *"all decisions related to product development are trade-off situations"* [51], 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 [66]. Additionally, as remarked by Coleman, *"many managers just decide to apply what they know, as their experience tells them it is merely common sense"* [18].

In this paper we make use of context-specific terminologies, which will be used to facilitate the readability and clarify any ambiguities throughout the whole paper in view of the weak state-of-the-art presented in [10] :

- 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. Traditional engineering activities are, among others, requirement engineering, design, architecture, implementation, testing.
- Engineering elements: any practice, tool and artifacts 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 [67].
- 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.
- Software product: any software product and software service.

3 Research Methodology

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 [68].

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

- RQ-1 : What is the current state-of-practice related to software development strategy in early-stage startups?
  - RQ-1.1 : How do startups structure and execute the main engineering activities?
  - RQ-1.2 : How are product quality attributes considered by startups?

We investigated the software development approach undertaken by practitioners of startups. Following a *Grounded Theory* (GT) methodology [69], we executed 13 semi-structured interviews integrated with follow-up questionnaires. From this, we elaborated and extracted the Green-field startup model (GSM) 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*

1. To attest the fact that *technical debt* is gaining traction among researchers, we mention two important contributions which characterize the *"debt landscape"* : [11], [12] and a dedicated workshop [64] organized by the Software Engineering Institute and ICSE.

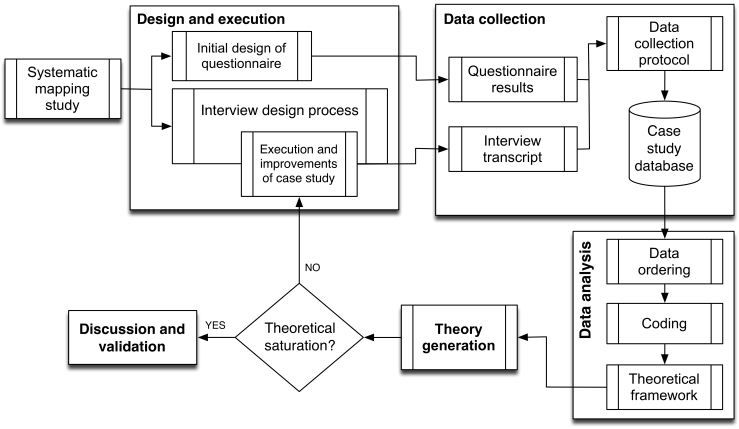

Fig. 1. Grounded theory study process overview

on here?" and defined by its creators [70] 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, it has been increasingly used by *Information System* researchers and has been almost ignored by the SE research [36]. In 1997 Bertelsen advocated for a need of more qualitative research in SE [71], and in the last decade we assisted to an increasing number of publications that used a GT-like approach in SE [20], [36], [72].

Following the *grounded theory* principles we attempted 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 [36]. 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"* [73], and encourages them to outline the research problem beforehand. Since the knowledge obtained from parallel execution of the SMS [10] and our direct experience with startups companies provided a good initial level of knowledge, in this study we use the Corbin and Strauss approach [74].

In this research 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*, 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 [75], [76], in our case study, we inquired web companies with little or no operating history.

A complete overview of the overall case study methodology and execution is shown in Figure 1, which presents how we tailored the general GT methodology to our specific needs.

The initial design of the case study is supported by the results of the *systematic mapping study* (SMS) [10] which contributed to define the initial questions for both questionnaires and semistructured interviews. The process depicted in Figure 1 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*[2] of categories has been achieved, we finally proceeded *theory validation* or performed another complete cycle.

The whole procedure has been executed simultaneously in pair on the same screen by the first two authors, handling conflicts by reviewing the rationale of decision with the third and fourth authors. When necessary we performed an in-depth review of the design of the research and data collected during the execution process.

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

The detail process is thoroughly described in the following subsections which are structured according to the five macro phases depicted in bold in Figure 1.

3.1 Design and execution

By means of the SMS [10] we refined our research questions. The research questions have 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 [73].

The sampling strategy took place in two distinct phases. At first we executed an initial *convenience sampling* [77], 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. Additional information about the companies are reported in Table 1.

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[3] (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.

We executed the case study online, supported by tools for video conferencing recording each complete session. Each interview has been executed only once per company with the interviewee's role of CEO or CTO. A step-by-step workflow have been followed, which allowed us to perform the interviews, collect additional material, prepare the customized follow-up questionnaire and iteratively adjust the *interview package* artifacts[4].

The follow-up questionnaires were designed to capture additional data, gather missing information and confirm interview results with triangulation. Questionnaires have been tailored to each startup, partially taking advantage of the repertory grid principles [80]. To obtain better chances of quickly obtaining responses, questionnaires 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.

3.2 Data collection

The approach undertaken to collect the data, known as *data triangulation*, integrates multiple data sources converging on the same phenomenon. After transcribing the interview, we extracted conceptual relations, and then we integrated them with the questionnaire'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 [81]. The database has been stored and constructed using the qualitative data analysis software package AtlasTI[5] - one of the most suitable tools for GT [36].

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.

3.3 Data analysis

Before starting the analysis, 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 one 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* [70].
- 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.
- Categories were refined several times in the attempt to create different level of abstractions and adjusting concepts, aided by a simple knowledge management tool.
- 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*

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

4. The *interview package* and follow-up questionnaire has been released under MIT License [78] and it is available for download on Github [79]. 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.

5. Available online at http://www.atlasti.com/ .

TABLE 1
Companies overview

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

more expressive power to the data" [70] combined with the more descriptive procedure of *open coding*. Following the example of other grounded theories, developed in related areas such as Information Systems [82] and Software Process Improvement [83], we performed the high-level conceptualization during creation of categories, in the process of refining axial and selective coding.

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, 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.

Given the large number of codes, categories, properties, propositions and related questions that evolved from the analytical process [73], 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 [73]. 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).

3.4 Theory generation

As mentioned above, emergent theories have been tested integrating the sample with additional companies selected following the principle of *theoretical sampling* [81].

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 [73].

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.
- *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.

Within the limits of the critical bounding assumptions, the role of the generated theory is to explain, predict and understand the underling phenomenon.

4 Results: Green-field startup model

By analyzing the interview transcripts with the GT process we extracted the information necessary to form

the theoretical model. 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.

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 and constitutes the empirical foundation of the theoretical categories, which has been used to form the theoretical framework (Green-field startup model).

In this section we present the GSM model to capture the underlying phenomenon of software development in early-stage startups. The GSM model is based on a systematic procedure which is grounded into empirical concepts obtained by conduction of the case-study. Starting from 630 raw *codes* extrapolated from the interview transcripts, 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.

4.1 Framework overview

Main concepts representing underlying phenomena have been grouped together to form high-level categories. Figure 2 shows the network of causal relationships (represented by arrows) between categories (represented by blocks). 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.

In the explanation of the GSM model we emphasize labels of theoretical categories using the *italic* font style. For the main categories shown in Figure 2, when necessary we also make use of identifiers (i.e. CATx) to facilitate the reading of this section. 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"* [74]. The meaning of the labels on the three arrows that reach the core category (*efficiency*, *effectiveness* and *performance*) is discussed in detail in section 7.

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 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* respect to other more urgent needs[6]. 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 performance in software development. To support 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*. 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*, 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*.

In the subsequent subsections we are going to explain in detail all the categories presented in Figure 2, to conclude in subsection 4.9 with the final theory. During the following explanations we are going to use identifiers of the companies presented in Table 1 (i.e. Cx) to highlight statements recorded from the interviewees.

4.2 Severe lack of resources

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*.

Since startups want to bring the product to market as quickly as possible, the resource they are the most deprived of is time. Startups operate under a constant

6. There are some exceptions where the quality aspects actually matter and such cases will be discussed later in 7.

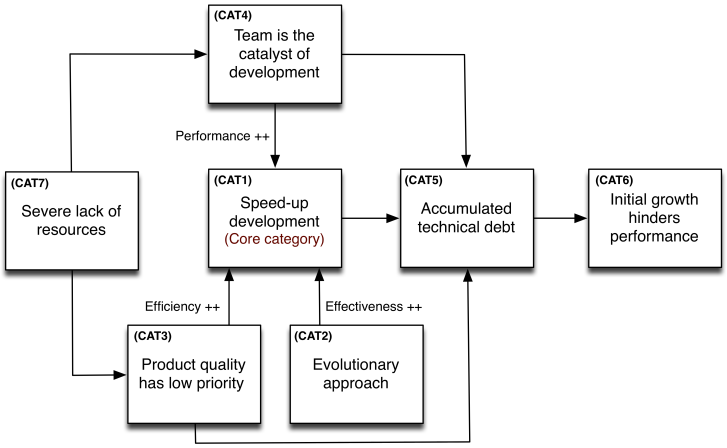

Fig. 2. Green-field startup model

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 [51]).

4.3 Team is the catalyst of development

Among the different aspects fostering the speed of the development process, the startups' main focus is on the characteristics of the initial team.

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.

4.4 Evolutionary approach

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"* [84]. 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 feedback to developers.

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 not documented, 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, characterised by an evolution that cannot be foreseen in the long run. However, a special role in daily decisions is covered by user feedback and requests, which represent the main drivers to define the product features to provide in the short-term.

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* (MVP) [66]).

Thanks to a *direct contact and observation of users*, *automated feedback collection* and *analysis of product metrics*. 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. Startups attempt to *find what is valuable for customers* with activities in agreement with Blank's *Customer validation* stage [7] .

4.5 Product quality has low priority

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.

Exploring quality aspects considered during the development process, the only important concerns are expressed in favour of the user experience[7], 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"*.

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 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 begin to have a central role in determining the level of UX achieved. Product metrics are typically web-based *statistical hypothesis testing*, such as A/B testing.

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

7. 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.

user-base[8]. 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. [9].

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

4.6 Speed-up development

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

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 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.

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 is represented by advanced version control systems, which are not only used to manage the codebase, 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 term since can lead to poorly maintainable code and developers burnouts in the long run.

4.7 Accumulated technical debt

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,

8. Note that startups tackle fast growing markets which are particularly subject to sudden user growths

9. Detailed discussion of the impact of innovative products on the user satisfaction is presented in Appendix according to the Kano model.

they will become increasingly more important for the productivity in the long-term, as we illustrate in CAT6.

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 [85]. 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 [86] diagrams and documentation, and provide a minimum level of maintainability costs. 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 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 *tacit knowledge* and severe lack of any kind of documentation. 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*.

4.8 Initial growth hinders performance

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* in terms of number of new user stories implemented in a certain amount of time, which brings new suitable functionalities to the users.

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 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*. Part of the effort, which was initially almost entirely dedicated to product development, is required to move to business activities. 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*. 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 .

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 initial decisions of the product design 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"*.

4.9 Theory

In order to explain and understand the development strategies in early-stage software startups we construct the theory generated and supported by the GSM model presented above. As discussed in section 7, we applied the *paradigm model* to form the final theory, introduced by Corbin [73], 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 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*. 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*.
- *Phenomenon* is represented by the core category *speed-up development*.
- *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.
- *Action and interaction strategies* are represented by the accumulation of technical debt (see CAT5).
- *Consequences* lead to a temporary performance drop-down (see CAT6).

5 THEORY IMPLICATIONS

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 GSM model, answering to our research question (*What is the current state-of-practice related to software development strategy 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. From an accurate and in-depth investigation into daily 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 prototype and guided by customer feedbacks is a central aspect of the *Lean startup methodology* pioneered by Ries [66]. 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 [7]. 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* (CAT6), which is partially mitigated by different elements supporting the development. 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 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, the infrastructural complexity of deployment is typically left to third party cloud applications. 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[10]: 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[11]. 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, 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.

- 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*, 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 [90]. 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

10. 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 [87] - because of the little return of investment they would gain, as discussed by Christensen in [88]. Instead, established companies have more interests in acquiring a startup and all the knowledge it has gained during the *customer development* process while focusing effort on acquiring larger shares of a "mass market" sector.

11. 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 [89].

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 [65], which appears to be the most important determinant for the future of software development in the company.

6 Discussion

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.
- 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.
- Finally we demonstrate how engineering elements independently reported in the follow-up questionnaire by practitioners, can be mapped to theoretical categories in the model.

Additional validation procedures are presented in [10][12].

6.1 Comparison with other frameworks

To validate the generalization of the theory, in this section we describe conceptualizations defined from the framework that are in support of a previous model developed by Coleman in [36], [20], [18]. 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.

Coleman's framework 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 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 CAT4. 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[13] in our study since we were not able to ground this concept in the data.

But what characterizes Coleman's network is the *cost of process* (core category) and all the factors that in management contributed to the lack of software process improvements (SPI).

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* and *Flexibility* of the development team. This is in accordance with the generated theory, which bases the reasons of adopting evolutionary and low-precision engineering

12. The author performs a validation of the theoretical framework by triangulating the results of different methodologies with the existing literature. He highlights the areas which have been neglected by existing studies, providing possible directions for future studies. Finally he identifies and discusses confounding factors which could interfere with the theoretical model.

13. The identification of confounding depends on the context in which the study is conducted and on the background knowledge of the researchers [91].

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."* [36].

Verbal *communication*, lack of heavy *documentation* and *bureaucracy* can easily be associated to the *accumulated technical debt* category (see CAT5). According to Coleman's framework, they 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 CAT1.

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 [92]. 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 [18], 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 software development managers as CEOs/CTOs and technical practitioners at the same time. As Coleman identified the influence of the founders' and managers' background over the software development process, also GSM identifies how the CEOs and CTOs background shapes the high-level strategies adopted in developing the initial product. Notwithstanding, team members remain self-organized, able to intervene in all the aspects of the development process without any direct supervision, as discussed in CAT4.

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. In this regard, 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 [93], startups seem to follow the *Fischer's Fundamental Theorem of natural selection* [94]. 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. 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 GSM need to be considered.

Also Baskerville in [95] refers to SPI as a development approach typically effective only in large-scale, long-term development efforts with stable and disciplined processes. By applying a grounded theory analysis in 10 companies he found that major causal factors that influence development are:

1) A desperate rush-to-market.
2) A new and unique software market environment.
3) A lack of experience developing software under the conditions this environment imposed.

As discussed in CAT7, 1) can be mapped to *time shortage* and 3) to *limited access to expertise*, whilst 2) can refer to uncertain conditions make long-term planning not viable explained in CAT2. Even though with different research focus and context of study - understand how and why Internet-speed software development (i.e. an agile method oriented using daily builds aimed at developing a product with high speed) differs from traditional software development - Baskerville revealed similar causal factors.

Baskerville argues that the dawn of the Internet era has intensified software development problems by emphasizing shorter cycle times, stating to:

- Speed-up development by releasing more often the software and "implanting" customers in the development environment (similar to the *find the product/market fit* concept presented in CAT2).
- Make heavy use of simple tools and existing components (similar to the concepts expressed in CAT1)
- Invest time in facilitating development of scalable systems by the use of simple but stable architectural solutions (see subcategory *using modularization and frameworks from day 1* discussed in CAT5).
- Tailor the development process daily according to the intense demands for speed. In this regard, he argues that the trend is to skip phases or tasks that might impede the ability to deliver software quickly even though producing lower quality software.

With wider research focus, another study, developed in 1986 by Brooks [96], 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.

Here we present the basic attacks (or mitigation strategies) presented in 1986 by Brooks on the conceptual essence ascertained within the GSM model description:

- 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. It 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- avoid deciding precisely what to build but rather iteratively extract and refine the product requirements with customers and users. It represents the evolutionary development approach to maintain a fast release-cycles during development (CAT2).
- Incremental development- starting from simple solutions allows organizations to early prototype and control complexity overtime. It represents the main purpose of focusing on *limited number of suitable functionalities* examined in *product quality has low priority* category (CAT3).
- Great team- people are the center of a software project and it is profoundly important to empower and liberate their creative mind. It is the 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.

6.2 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. 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 GSM model. 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 GSM model, the most essential feature that startups aim to achieve is high speed.

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*).

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[14]:

- 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[15] (described in CAT4 and confirmed, among others, by Sutton in [5]).
- 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 (in agreement to the concept of *pivoting* discussed in [66]).
- 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.

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

14. Time-to-market was the main concern, as confirmed in [10].

15. Note that the importance of a skilled team for development performance is expressed in many studies such as [24], [90].

performance in later stages.

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[16] 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*[17].

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*.

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.

The validation has been performed on the most critical causal relationships of the model. According to the paradigm model those are the causal condition which are linked to the core-category. There are four high-level relationships 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.

6.3 Engineering elements and categories

Finally, to provide a concrete example of how the model can be used, we show that the engineering elements (reported in Appendix) mentioned by respondents during the interviews can be mapped into the theoretical categories of the framework.

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 the GSM model, 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 (i.e. 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.

7 VALIDITY THREATS

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. 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 [20]. 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 [18] 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 attacks* to the challenges of developing software systems and other extant literature retrieved by [10][18]. In conclusion, regarding the validation of the subcategories, a continuous re-evaluation of the interview transcripts has been performed.

16. The team capability is evaluated according to the characteristics presented in CAT4.

17. The debt here is considered regardless of context-specific debt mitigation factors and tactics.

18. Note that the generalizability of results is applied only to those startups that operate in software-intensive activities.

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.

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. With the final model validation we highlighted and examined similarities, contrasts and explanations as discussed in [97]. 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 GSM model, both in the high level and low level frameworks. Furthermore, by means of comparison between the GSM model and extant studies in the broader fields, we revealed that the theory resembles in wider startup development context. Nevertheless, we discovered important confounding factors which might mine the conclusion and generalizability of the study related to innovation, creativity and market requirements.

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[19]. Moreover, 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.

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. 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[20]. 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. 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.

Grounded theory has been already applied by other researches in similar contexts to attest relationships among conceptualizations of an examined phenomenon (see [36], [98], [20]). 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*.

According to the *theoretical sampling* concept, we adjusted our research design and the emergent theory until only marginal results were generated. 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.

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. 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."* [99]. To obtain major certainty of the obtained results, what we applied was the final validation.

Despite we couldn't apply any power analysis procedures, in grounded theory the sample size depended on

19. 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.

20. Available at https://github.com/adv0r/BTH-Interview-Package .

when data saturation occurred. Ramer in [100], 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.

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.

7.1 Future work

The GSM model enables practitioners in startup companies to speed-up development empowering the role of the team, focusing on minimal functionalities and adopting evolutionary development approaches. Researchers can use the GSM model to reproduce the study with other software startups, and focus on mitigation strategies to minimize the impact of the accumulated technical debt over the need of shortening time-to-market. Moreover, the mutual collaborative dialog between the research in the development strategies in software startups and technical debt can benefits in understanding how both influence the growth of a company after the release of the first product to the market.

8 Conclusion

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 provided an initial explanation of the underlying phenomenon by means of a grounded theory case study, focusing on 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 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. The systematic mapping of the literature [10], 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. 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.

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 transform every decision related to the development 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 controlling 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. In fact, as confirmed by analyzing other framework in SE, 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.

In this research 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 [101], 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 suc-*

cessfully developing software systems in startup companies.

ACKNOWLEDGMENTS

The authors would like to thank...

REFERENCES

[1] D. Smagalla, "The truth about software startups," *MIT Sloan Manage. Rev. (USA)*, vol. 45, no. 2, p. 7, Winter 2004.

[2] M. Crowne, "Why software product startups fail and what to do about it," in *Proceedings of 2002 IEEE International Engineering*, 2002, pp. 338–343.

[3] A. Maccormack, "How Internet Companies Build Software," *MIT Sloan Management Review*, pp. 75–84, 2001.

[4] 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.

[5] S. M. Sutton, "The role of process in software start-up," *IEEE Software*, vol. 17, no. 4, pp. 33–39, Aug. 2000.

[6] 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.

[7] S. Blank, *The four steps to the epiphany*. Cafepress. com, 2005.

[8] 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.

[9] 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.

[10] N. Paternoster, "Evolutionary analysis of software development in startup companies," Master's thesis, Free University of Bolzano, 2012.

[11] 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.

[12] 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.

[13] D. Storey, *Entrepreneurship and the New Firm*. Croom Helm, 1982.

[14] 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.

[15] 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.

[16] 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.

[17] C. M. Christensen, *The Innovator's Dilemma*. Harvard Business School Press, 1997.

[18] 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.

[19] G. Chroust, "What is a software process?" *Journal of Systems Architecture*, vol. 42, no. 8, pp. 591–600, Dec. 1996.

[20] 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.

[21] J. Bach, "Microdynamics of process evolution," *Computer*, vol. 31, pp. 111–113, 1998.

[22] 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.

[23] 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.

[24] A. Cockburn, *Surviving Object-Oriented Projects*. Addison-Wesley Professional, 1998.

[25] M. Tanabian, "Building high-performance team through effective job design for an early stage software start-up," in *Management Conference*, 2005, pp. 789–792.

[26] 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.

[27] M. Kakati, "Success criteria in high-tech new ventures," *Technovation*, vol. 23, no. 5, pp. 447–457, May 2003.

[28] M. Marmer, B. L. Herrmann, E. Dogrultan, R. Berman, C. Eesley, and S. Blank, "The startup ecosystem report 2012," Startup Genome, Tech. Rep., 2012.

[29] C. Alves, S. Pereira, and J. Castro, "A Study in Market-Driven Requirements Engineering," Universidade Federal de Pernambuco, Tech. Rep., 2006.

[30] 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.

[31] P. Sawyer, I. Sommerville, and G. Kotonya, "Improving market-driven re processes," in *International Conference on Product-Focused Software Process Improvement (Profes '99)*, 1999.

[32] 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.

[33] 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.

[34] D. A., "Study of current practices in marketdriven requirements engineering," in *Third Conference for the Promotion of Research in IT*, University Colleges Sweden, 2003.

[35] M. Keil and E. Carmel, "Customer-developer links in software development," *Commun. ACM*, vol. 38, no. 5, pp. 33–44, May 1995.

[36] 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.

[37] E. Carmel, "Time-to-completion in software package startups," *Proceedings of the System Sciences, 1994.*, pp. 498–507, 1994.

[38] M. Lehman, "Programs, life cycles, and laws of software evolution," in *Proceedings of the IEEE*, vol. 68, no. 9, Sep. 1980, pp. 1060 – 1076.

[39] R. Banker and G. Davis, "Software development practices, software complexity, and software maintenance performance: A field study," *Management Science*, 1998.

[40] S. Ambler, "Lessons in agility from Internet-based development," *IEEE Software*, no. April, pp. 66–73, 2002.

[41] D. Yoffie, "Building a company on Internet time: Lessons from netscape," *California Management Review*, vol. 4, no. 3, 1999.

[42] P. Tingling, "Extreme programming in action: a longitudinal case study," *Proceedings of the 12th international conference*, pp. 242–251, 2007.

[43] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[44] 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.

[45] R. Deias and G. Mugheddu, "Introducing XP in a start-up," *European Internet Services Company*, 2002.

[46] 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.

[47] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile software development methods," *Relatório Técnico, Finlândia*, 2002.

[48] 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.

[49] K. Kuvinka, "Scrum and the Single Writer," in *Proceedings of Technical Communication Summit*, no. May, 2011, pp. 18–19.

[50] 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.
[51] 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.
[52] S. Yogendra, "Aligning business and technology strategies: a comparison of established and start-up business contexts," in *Management Conference*, 2002.
[53] 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.
[54] 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.
[55] M. Fayad, "Process assessment considered wasteful," *Communications of the ACM*, vol. 40, no. 11, pp. 125–128, 1997.
[56] 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.
[57] 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.
[58] 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.
[59] 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.
[60] S. Jansen, S. Brinkkemper, and I. Hunink, "Pragmatic and Opportunistic Reuse in Innovative Start-up Companies," *Software, IEEE*, pp. 42–49, 2008.
[61] D. Wall, "Using open source for a profitable startup," *Computer*, vol. 34, no. 12, pp. 158 –160, dec 2001.
[62] 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.
[63] W. Cunningham. The WyCash Portfolio Management System. [Online]. Available: http://c2.com/doc/oopsla92.html (Accessed : Sep. 15, 2012).
[64] Third international workshop on Managing Technical Debt. [Online]. Available: http://www.sei.cmu.edu/community/td2012/ (Accessed : Sep. 15, 2012).
[65] 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.
[66] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.
[67] Microsoft, *Microsoft Application Architecture Guide*. Microsoft Press, 2009.
[68] 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.
[69] B. G. Glaser, *Theoretical sensitivity : advances in the methodology of grounded theory*. Sociology Press, 1978, vol. 2.
[70] Colin Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. John Wiley and Sons, 2009.
[71] O. W. Bertelsen, "Toward A Unified Field Of SE Research And Practice," *IEEE Software*, vol. 14, no. 6, pp. 87–88, 1997.
[72] 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.
[73] J. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, no. 1, pp. 3–21, 1990.
[74] 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.
[75] Startup types distribution. [Online]. Available: http://techcrunch.com/2012/02/17/crunchbase/ (Accessed : Aug. 25, 2012).
[76] 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.
[77] C. W. Dawson, *Projects in Computing and Information Systems A Student ' s Guide*. Pearson Prentice Hall, 2009.
[78] MIT License. [Online]. Available: http://www.opensource.org/licenses/mit-license.php/ (Accessed : Aug. 25, 2012).
[79] C. Giardino and N. Paternoster. Interview Package. [Online]. Available: https://github.com/adv0r/BTH-Interview-Package, [Aug. 25, 2012).
[80] 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.
[81] R. K. Yin, *Case study research: design and methods*, R. K. Yin, Ed. Sage Publications, Inc, 1994, vol. 1, no. 3.
[82] 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.
[83] G. Coleman, *Software process in practice: A Grounded Theory of the irish software industry*. Springer, 2006.
[84] A. Davis, "Operational prototyping: a new development approach," *Software, IEEE*, vol. 9, no. 5, pp. 70 –78, sept. 1992.
[85] S. Zhong, C. Liping, and C. Tian-en, "Agile planning and development methods," in *Computer Research and Development (ICCRD), 2011 3rd International Conference on*, vol. 1, march 2011, pp. 488 –491.
[86] R. Pooley and P. King, "The unified modelling language and performance engineering," *Software, IEE Proceedings -*, vol. 146, no. 1, pp. 2 –10, feb 1999.
[87] E. M. Rogers, *Diffusion of Innovations*. Free Press, 1995.
[88] C. M. Christensen and M. E. Raynor, *The Innovator's Solution*. Harvard Business School Press, 2003.
[89] J. Offutt, "Quality attributes of web software applications," *IEEE Softw.*, vol. 19, no. 2, pp. 25–32, Mar. 2002.
[90] J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House, 2000, vol. 12.
[91] 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.
[92] 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.
[93] 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).
[94] F. Steven and S. Montgomery, "Fisher's fundamental theorem of natural selection," *TREE*, vol. 7, no. 3, pp. 92–95, 1992.
[95] R. Baskerville, B. Ramesh, L. Levine, J. Pries-Heje, and S. Slaughter, "Is "internet-speed" software development different?" *Software, IEEE*, vol. 20, no. 6, pp. 70–77, nov.-dec. 2003.
[96] F. Brooks Jr, "No Silver Bullet — Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.
[97] 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.
[98] 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.
[99] P. Y. Martin, "Grounded theory and organizational research," *The Journal of Applied Behavioral Science*, vol. 22, no. 2, pp. 141–157, Apr. 1986.
[100] L. Ramer, "Quantitative versus qualitative research?" *Journal of obstetric, gynecologic, and neonatal nursing : Jognn / Naacog*, vol. 18, no. 1, pp. 7–8, 1989.
[101] D. Berry, "Academic Legitimacy of the Software Engineering Discipline," Software Engineering Institute, Tech. Rep. November, 1992.

John Doe Biography text here.

Jane Doe Biography text here.