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

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

Abstract—Software startups are newly created companies with no operating history and oriented in producing cutting-edge technologies. These companies develop software under highly uncertain conditions, tackling fast growing markets with severe lack of resources. 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 feedback. 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]. Therefore, 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: 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 on a limited time-frame to emphasize 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 themes and iteratively adjusting the direction of the research according to the emerging evidence.

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 a case-study. GSM presents the most significant themes of the development strategies that characterise startups' context.

The results of our study suggest that the most urgent priority of software development in startups is to shorten

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

time-to-market. The greater part of engineering activities are focused on the implementation while only little attention is given to more conventional activities (project management, requirement specifications, analysis, architecture design, automatic testing). The first release of the product includes only a limited set of well suitable functionalities.

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 feedback, using flexible and reactive development approaches and adopting mitigation strategies to contain the severity of accumulated technical debt.

Practitioners can use the GSM 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 to understand how the technical debt influences the future growth of startup companies.

The remainder of this paper is structured as follows: 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. Section 5 compares results of this research to literature and frameworks presented in this field. Section 6 discusses the most relevant implications of the GSM. Section 7 discusses the validity threats considered during this study. The paper concludes in section 8.

2 Background

Modern entrepreneurship has been boosted by the advent of the consumer internet markets in the middle of the 1990s and culminated with the notorious *dot-com bubble burst* of 2000 [13], [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 to as the *startup bubble*. The easy access to vast potential markets and the low cost of services distribution are appealing conditions for modern entrepreneurs [15]. Inspired by success stories, 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 decade one can estimate 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 define what 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 *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], [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 General lack of research in startups

Despite some studies tried to address the above mentioned issues, a SMS [10] revealed only a few SE works in this specific area, as confirmed by other studies [18], [20], [36], [5]. Moreover, the studies appear to be highly fragmented and spread across different areas rather than constituting a consistent *body of knowledge*.

Sutton [5] confirmed in 2000 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 by Coleman et al. [18], [20], [36] eight years later.

2.2 Software development in startups

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 a 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.

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 by Kajko-Mattsson [8], investigating a Swedish software startup, reported a heavy lack of requirements gathering processes, 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. [40] reported 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. In this regard, Crowne [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 by Sutton [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 towards prototyping, proof-of-concepts, mock-ups and demos, to test basic functionalities, have been the primary priority in startups [37]. Only when they grow, formal methodologies arise, followed by performing quality assurance and long-term planning processes [41].

The maturity of a company affects the extent to which processes are adopted. Tingling [42] 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. Da Silva and Kon [44] were 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].

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 trial and error 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 by Yogendra [52] as an 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].

2.3 Software process improvement in startups

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 [53]. The process consists of a set of activities and artifacts to identify responsibilities and tools to utilize. But, despite the promising benefits reported by Zettel, we were not able to identify any evaluation in real-world settings. Another attempt of SPI in startups has been conducted by Deakins et al. [54], introducing broad guidelines for a rapid high-quality development process, which underwent however only limited testing in academic settings.

The problem of *one-size-fits-all*, related to the SPI representations for startups, is described by Fayad [55]. The author discusses the problem in actuating the same best-practices criteria for established companies in 10-person software startups. Sutton [5] remarks that problems of SPI in software startups arise due to: 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, and 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 been neglected because it is seen as an obstacle to the development team's creativity and flexibility, and to the need of a quick delivering product process environment [20]. Product quality is left aside in favor of minimal and suitable functionalities to shorten the time-to-market. As reported in two studies by Mater and Mirel [56], [57], quality aspects mostly taken in consideration in internet startups, are oriented towards usability and scalability, even though the market and application type heavily influences the quality-demand [18], [58].

To maintain the development activities, oriented to limited but suitable functionality, studies suggest to externalize the complexity of parts of the project to third party solutions by means of outsourcing activities [59], software reuse [60] and open-source strategies [61], [62].

2.4 Technical debt

Notwithstanding, a new area of SE research, trying to tackle the problem of *technical debt*, brings and encompasses different implications in studying development in software startups [63]. The metaphoric neologism was originally introduced by Cunningham in 1992 [64] and has recently attracted the attention of SE researchers[1]. The concept of technical debt is well illustrated in [66]: *"The idea is that developers sometimes accept compromises in 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).

Since *"all decisions related to product development are trade-off situations"* [51], generally startups optimize workflows to the dynamic context they are involved into. 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 [67]. 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].

2.5 Terminology

In order to set a common ground and to prevent ambiguity, we use the following terminology throughout the paper:

- 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 [68].
- Growth: an increase in the company size with 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/or 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: 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*[2].

Initially, the boundaries of the research domain were set by means of a 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 [70], 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 on here?"* and defined by its creators [71] as *"a set of well-developed categories (e.g. themes, concepts) that are systematically interrelated through statements of relations to form*

1. To attest that *technical debt* is gaining traction among researchers, we mention two important contributions which characterize the *"debt landscape"*: [11], [12] and a dedicated workshop [65] organized by the Software Engineering Institute and ICSE.

2. The above mentioned goal is structured according to the GQM paradigm [69].

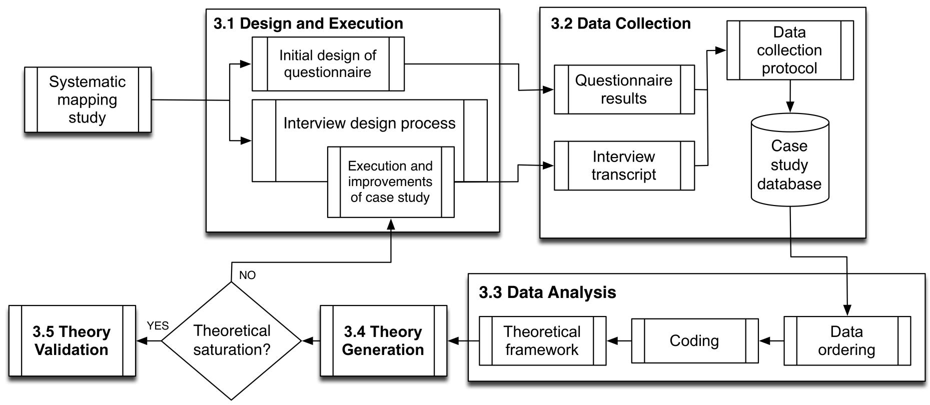

Fig. 1. Grounded theory study process overview

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 [72], and in the last decade we assisted to an increasing number of publications that used a GT-like approach in SE [20], [36], [73].

Following the GT principles, we captured 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. 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 GT, researchers should indicate which *"implementation"* of the theory is being used [36]. After the GT methodology was initially introduced, its original authors had 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"* [74], 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 [75].

In this research we address 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 [76], [77], 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 semi-structured interviews. The process depicted in Figure 1 is evolutionary and affects the design at each new iteration. In *Data Collection* the empirical results are integrated and 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 *Theoretical Saturation*[3] of categories has been achieved, we finally proceeded to *Theory Validation*.

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.

The process details are described in the following subsections, structured according to the five macro phases depicted in bold in Figure 1.

3.1 Design and Execution

This case study started based on a SMS of the literature regarding software development in startups [10], refining our research questions. The research questions have

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

been narrowed down to outline the domain of our study, even though still broad enough to allow new theories to emerge and to adjust the scope of the research [74].

The sampling of candidate companies took place in two distinct phases. At first we executed an initial convenience sampling [78], which led to the identification of eight companies. Then we included five additional startups during the theory formation process (theoretical sampling) iteratively improving the sample according to the emerging theory. The characteristics of the sampled companies are reported in Table 1.

All companies, except C10, were founded within the last three years (2009-2012), by an average of 3 starting members, which were in majority developers. Moreover, the number of current employees shows how, to different degrees, companies expanded the initial teams. All companies, except C5, brought their first product to market within less than 6 months from the idea conception. The products consist of web-applications, launched in different nations and market sectors. The growing team size 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. The interview subjects were the companies' CEOs or CTOs. We followed a step-by-step workflow, consisting of the actual interview, collection of additional material, preparation of the customized follow-up questionnaire and the iterative adjustment of the interview package artifacts[4].

The follow-up questionnaires were designed to capture additional data, gather missing information and confirm interview results by triangulation. The questionnaires have been tailored to each startup, partially taking advantage of the repertory grid principles [81]. To obtain better chances of quickly obtaining responses, questionnaires have been sent to respondents immediately after transcribing the interview results. Most of the follow-up questionnaires have been collected within one day after the conclusion of the last interview.

3.2 Data collection

The data collection was designed to allow triangulation, which integrates multiple data sources converging on the same phenomenon. After transcribing the interview, we extracted conceptual relations, integrating 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 by Yin [82]. 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]. 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 [71].
- 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 levels of abstraction and adjusting concepts, aided by a simple knowledge management tool.
- Consistency among categories was validated by exploring and analyzing links among subcategories by means of 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 gives more expressive power to the data"* [71] combined with the more descriptive procedure of open coding. Following the example of other grounded theories, developed in related areas such as Information Systems [83] and Software Process Improvement [84], 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-a-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

4. The interview package and follow-up questionnaire has been released under MIT License [79] and it is available for download on Github [80]. We encourage anyone interested 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

from the analytical process [74], 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 that are linked together according to cause-effect relationships [74]. During the formation of the theoretical framework the researchers operated by means of a bottom-up approach. From the empirical data and coding process, the framework developed into two different levels: a detailed level representing the network of subcategories (identified mainly by the axial coding process), and a high-level representation of the main categories network (identified mainly by the selective coding process).

3.4 Theory generation

As mentioned in Section 3.1, emergent theories have been tested by integrating additional companies into the sample, selected following the principle of theoretical sampling [82].

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 [74]. 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 conditions from 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 underlying phenomenon.

3.5 Theory Validation

Presenting GT is challenging for a researcher, who must pay attention to structure the included level of detail, and to the way data is portrayed to display evidence of the emergent categories. In order to assess our research and determine if the GT is sufficiently grounded, we used a systematic technique to assess the theory.

Strauss and Corbin provided a list of questions to assist in determining how well the findings are grounded [75]:

1) Are concepts generated, and are the concepts systematically related?
2) Are there many conceptual linkages and are the categories well developed?
3) Is variation[6] built into the theory and are the conditions under which variation can be found built into the study and explained?
4) Are the conditions under which variation can be found built into the study and explained?

6. Variation refers to the variety of contexts to which the theory can be applied.

5) Has the process been taken into account, and does the theory stand the test of time?
6) Do the theoretical findings seem significant, and to what extent?

Answering the above questions, the concepts were generated according to the coding process. They were systematically related through the use of a network diagram by which we established the relations between the concepts. At each iteration of the grounded theory process, we considered and examined a concept within different conditions and dimensions, trying to incorporate data from a broader range of practitioners.

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

The process has been designed by the researches in different steps, explaining the purpose of each of them. Thus, the same processes enable other researches to replicate the study within similar contexts. Moreover, a comparison with the state-of-the-art has been performed in order to validate the theory and to strengthen its applicability within a wider time-frame. By this comparison we highlight the areas which have been neglected by existing studies, providing possible directions for future studies (subsections 5.1 and 5.2). Furthermore, we studied the confounding factors which could interfere with the application of the GSM (subsection 5.3).

At last, we performed an evaluation with empirical data (subsection 5.4), quantifying scores for each company involved in our empirical study by defining metrics based on a set of rubrics and evaluating startups accordingly, through an analysis of interview transcripts and codes. The complete statistical procedure and used rubrics to quantify the measures are provided in the Appendix [ADD URL].

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[7], 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 raw codes constitute the empirical foundation of the theoretical categories, which has been used to form the Green-field startup model (GSM).

The GSM captures the underlying phenomenon of software development in early-stage startups. 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 the GSM we provide explanations of the development strategies and engineering activities undertaken by startups.

4.1 Model overview

The main concepts representing the underlying phenomenon 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, 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 forthcoming explanation of the GSM 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"* [75]. The meaning of the labels on the three arrows that reach the core category (efficiency, effectiveness and performance) is discussed in detail in section 5.4.

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 startup 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* with respect to other more urgent needs[8]. 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,

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

8. There are some exceptions where the quality aspects actually matter and such cases will be discussed in section 5.3 .

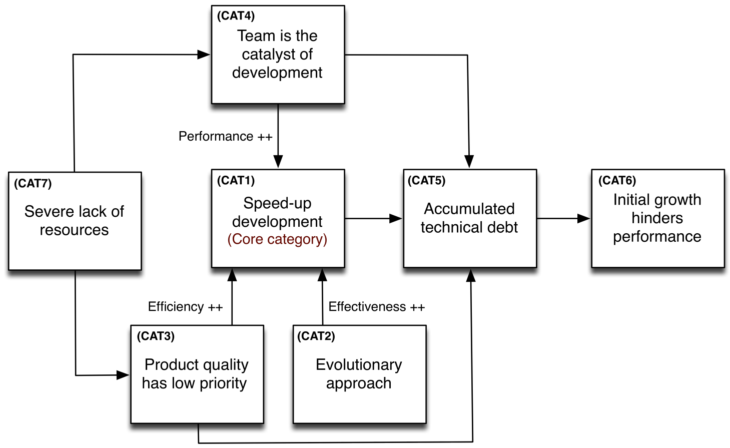

Fig. 2. Green-field startup model

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 cycles. 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 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 following subsections we explain the categories presented in Figure 2, and conclude in subsection 4.9 with the final theory. In the explanations we 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 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, cause 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 startups are typically *multi- role and full-stack engineers*: they are full-stack for being able to tackle different problems at different levels 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 the 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."*

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"* [85]. 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. 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. 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 ill) can have a 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 for defining the product features 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. 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 companies focus on building a small set of functionalities to include in the first version, and *progressively roll-out to a larger number of people* with very *small iterations* (confirmed again by C4: *"we deploy from 5 to 20 times a day"*).

The objective of this evolutionary approach is to avoid wasting time on "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 functionalities developed with the least possible effort that can validate critical features, enabling the startup's survival in the short term (recalling Ries' definition of *Minimum Viable Product* (MVP) [67]).

Supported by *direct contact and observation of users*, *automated feedback collection* and *analysis of product metrics*, 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 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 towards the user experience[9], 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*

9. 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.

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 [86].

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. The level of efficiency can be optimized after attesting the effectiveness of the minimal set of functionalities, obtained according to CAT2 and to the concept of MVP.
- The *product should be reasonably ready-to-scale* to be able to accommodate a potential growth of the user-base[10]. 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[11].

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

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, and development tools and libraries); taking advantage of external services; and deploying the product on external infrastructures, for the sake of delivering 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 code-base, but for also other purposes such as task assignment, responsibility tracing, configuration and issue management, automatic deployment, 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 so-

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

11. A discussion of the impact of innovative products on the user satisfaction is presented in subsection 5.3.

lutions 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 it can lead to poorly maintainable code and developer burnout 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, 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 ticket-based tools to manage low-precision lists of features to implement, written in form of self-explanatory user stories [87]. 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 [88] 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. 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 (characterized by co-location, self-organization and very small team size).

Finally, one of the categories, which most contributes in growing the *accumulated technical debt*, is the substantial use of informal and verbal communication channels on a 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 over time, the number of users 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 a 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 the 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 increasing product demands.

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 servers 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. 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 the 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 above presented GSM:

Theory. *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.*

As discussed in section 5 , we formed the theory by applying the paradigm model, introduced by Corbin [74]. The theory is composed by different elements reflecting the paradigm model:

- *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.
- *Consequences* lead to a temporary performance drop-down.

5 THEORY VALIDATION

In this section we discuss the validity of the GSM by means of cross-methodological observations, as discussed in section 3.5.

5.1 Comparison with other frameworks

To validate the generalization of the theory, we describe conceptualizations defined from the GSM that are in support of previous models developed by Coleman [36], [20], [18], Baskerville [89] and Brooks [90].

We refer to Coleman's work since only he has, to the best of our knowledge, conducted similar research in startups' context, even though with a different focus. Coleman investigated factors in software development that hinder initiatives of software process improvement (SPI) in a later stage, representing also companies in the expansion phase with more than 100 employees.

In this subsection we illustrate the most important aspects considered by Coleman, examining similar and contrasting factors with respect to the GSM. We 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 kinds of processes: *essentials* and *non-essentials*. The *essential* processes are the most closely linked to 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"* [18].

The GSM 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 framework, the GSM doesn't present how *market requirements* can affect the conduction of processes. In this regard, Coleman describes how the more requirements definitions are predictable the more well-defined workflows are established. This particular aspect is considered as a confounding factor[12] 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*.

Coleman reports the 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 our generated theory, which bases the reasons of adopting evolutionary and low-precision engineering elements in the flexibility and reactivity attributes of the development process in startups.

In 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 describe 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*

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

since they have direct relation to the subcategory of *keep a simple and informal workflow*, as discussed in CAT1.

As also reported in the GSM, 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 difficulty in penetrating small software organizations [92]. When startups began establishing any SPI process, they experienced *process erosions* [18] , which resolved to a barely sufficient workflow to satisfy the organizational business needs.

Then, software startups favor 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 formal communication strategies , even though the accumulated tacit knowledge is hard to manage and to transfer to new hires as also discussed in CAT6.

Coleman describes how the *"management approach"* is oriented towards *"embrace and empower"* solutions in contrast with *"command and control"* style, where there is an evidence for trust in the development staff to carry out tasks with less direct supervision, greater delegation of responsibilities and a more generally consensual environment [18]. Nevertheless, software development managers and founders have still impact on management style and indirectly on the 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 on the software development process, the GSM similarly identifies that 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 GSM with Coleman's framework, we have found that despite Coleman's study contributes to define which factors hinder the introduction of SPI in startups his results are coherent with the theory generated within our research, focusing at explaining the overall strategies within software development in startups. 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]. Moving towards flexible and variable processes (easy to adapt to the uncertain conditions), increase the odds of "natural selection" only when some "restrictive conditions" are met. In other words, even though ignoring SPI practices is advisable, under circumstances of high uncertainty and severe lack of resources, not following SPI does not predict major adaptiveness of startups development towards the customer's needs. Nevertheless, to imply a major evolutionary product/market fit, certain conditions explained in GSM need to be considered.

Also Baskerville [89] refers to SPI as a development approach typically effective only in large-scale, long-term development efforts with stable and disciplined processes. 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, by applying a grounded theory analysis in 10 companies, 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 , Baskerville revealed similar causal factors as the GSM.

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 a wider research focus, Brooks [90] discussed what challenges are involved in constructing software products. Brooks 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.

Brooks 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 mitigation strategies presented by Brooks in 1986 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 revealed to be extremely accurate, according to the state-of-practice in modern startups.

5.2 Theoretical categories and existing literature

In this subsection we report the evaluation of our theory with respect to the main contributions of the studies retrieved and discussed within the SMS [10]. For each retrieved article we present the main results, mapping them to categories in accordance to the GSM. Table 2 shows the main contributions of the 37 identified studies, where an 'X' is placed in correspondence to the theoretical category they considered. The table is sorted by the number of GSM categories covered by the studies.

Only 7 studies out of 37, discuss aspects related to the overall software development process in early stage startups. The remaining part of the studies is only partially covering the phenomenon described by the GSM. Looking at the papers' main contribution, we were able to extrapolate information which confirmed the GSM categories.

We can see how the majority of the retrieved studies (29) mention issues related to *speed-up development* (CAT1) which confirms the importance of the core category. On the other hand, we can observe that only a limited number of studies mentions results related to *product quality has low priority* (CAT3), *accumulated technical debt* (CAT5), and *initial grow hinders performance* (CAT6), suggesting directions of possible future primary studies.

These results confirm the relevance of development teams as widely discussed in the SE literature, here mapping 26 studies to *team is the catalyst of development* (CAT4). The importance of people has been widely discussed in other studies in SE (Cooper [101], DeMarco [102], Coleman [103], Valtanen [104], Adolph [105] and Cockburn [106]), advocating for the need of empowering people, which is the element with the greatest impact on software development. Humans, being non-linear variables [107] , are unlikely to follow repeatable prescriptive methodologies. As stated by Highsmith [108], *"A drawback of each and every methodology is to expect people to behave consistently over time, when is clearly not like that"*. Especially in early-stage startups, where the whole company overlaps with the development team, the GSM confirms the importance of factors related to people.

5.2.1 Limited resources

Starting from the category *severe lack of resources* (CAT7), many authors describe startups facing with inescapable constraints defined by engineering/business concerns and constant time-pressure [37], [109]. Because of their young and inexperienced working conditions, startups present limited resources in terms of management and strategic alliances [5].

Crowne [2] identifies four main stages of a startup's lifecycle: startup, stabilization, growth and maturity. Inexperienced practitioners operating in a chaotic environment characterizes the first phase, where there is no strategic plan for developing the product. But, moving towards the stabilization phase, the product starts to be unreliable and requirements unmanageable. Consequently, when startups move to the growth phase, they initiate structuring and controlling processes, which are incrementally integrated in the development environment. The maturity phase represents the moment when the company is ready to introduce process improvement initiatives. Crowne's study profoundly corresponds to different conceptualizations in the GSM. Starting from the minimal and low-precision engineering elements, startups start to grow the accumulated technical debt which requires fast integration of more structured and standard processes overtime. Issues related to low-precision engineering practices during companies' growth are discussed in an experience paper by Ambler [40]. He reports on two case studies of growing Internet startups approaching an initial public offering (IPO). Despite the two companies were facing business

TABLE 2
SMS [10] and GSM

Author (year)	CAT1	CAT2	CAT3	CAT4	CAT5	CAT6	CAT7	Count	Ref.
Sutton (2000)	X	X	X	X	X	X	X	7	[5]
Kajko-Mattson (2008)	X	X	X	X	X	X	X	7	[8]
Crowne (2002)	X	X	X	X	X	X	X	7	[2]
Coleman (2008)	X	X	X	X	X	X	X	7	[20]
Coleman (2008)	X	X	X	X	X	X	X	7	[18]
Coleman (2007)	X	X	X	X	X	X	X	7	[36]
Carmel (1994)	X	X	X	X	X	X	X	7	[37]
Yoffie (1999)	X	X	X	X	X	X		6	[41]
Zettel (2001)	X	X	X	X			X	5	[53]
Jansen (2008)	X		X		X	X	X	5	[60]
Heitlager (2007)			X	X	X	X	X	5	[22]
Deias (2002)	X	X		X	X	X		5	[45]
Ambler (2002)	X	X			X	X	X	5	[40]
Wood (2005)	X		X	X			X	4	[95]
Tingling (2007)		X		X	X	X		4	[42]
Taipale (2010)	X	X		X	X			4	[48]
da Silva (2005)	X	X		X	X			4	[44]
Mirel (2000)	X		X	X	X			4	[57]
Midler (2008)	X	X		X			X	4	[50]
Tanabian (2005)	X			X			X	3	[25]
Stanfill (2007)	X				X		X	3	[96]
Mater (2000)	X	X				X		3	[56]
Kuvinka (2011)	X	X		X				3	[49]
Deakins (2005)	X	X		X				3	[54]
Yogendra (2002)	X	X						2	[52]
Wall (2001)	X						X	2	[61]
Su-Chuang (2007)	X			X				2	[97]
Steenhuis (2008)				X			X	2	[98]
Sau-ling Lai (2010)		X					X	2	[99]
Kakati (2003)		X		X				2	[27]
Himola (2003)	X						X	2	[51]
Häsel (2010)	X			X				2	[100]
Hanna (2010)	X						X	2	[59]
Bean (2005)	X				X			2	[62]
Kim (2005)		X						1	[58]
Fayad (1997)				X				1	[55]
Chorev (2006)				X				1	[26]
Count	29	22	13	26	18	14	20		

and team growth, attitudes against the introduction of processes to structure and model the system were evident. The need to re-engineer the codebase lead the researchers to suggest a tailored version of the Rational Unified Process (RUP) development framework. Nevertheless, even with the aid of CASE tools[13], startups failed in practicing engineering activities in view of excessive lack of flexibility. Developers fast moved to simpler tools such as using a white-board, where they were able to provide input and insights, question assumptions, share past experiences, and even talk about potential implementation strategies.

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

5.2.2 Minimal project management

The use of whiteboards in working environments supports *use of well-integrated and simple tools*. In both the startups studied by Ambler, documentation was minimal, finding the "sweet spot" to have just barely enough tools and artifacts to meet their needs. Generally, the use of *keep a simple and informal workflow* endorses the use of low-precision engineering practices which lead to speed-up development initially, but with negative effects on the accumulated technical debt of a minimal project management and consequently on the final company growth.

Kajko-Mattsson et al. [8] investigated a Swedish software startup company involved in mobile applications, reporting the following issues: a lack of requirements gathering process; release cycles and their length were not defined; the releases and their scope were not planned; lack of control over the change requests; lack of process control in terms of absence of any documentation to track the status and progress of the process; defective releases in terms of lack of testing; poor communication since the communication inside the company was informal and not documented at all. In the same study they designed a process improvement model, based on three main phases: Evaluate, Plan and Change.

In the evaluation phase, the status of the development workflow within the company is analyzed. The researchers documented a state of chaos where any initiatives of quantitative methodology assessment would have been a waste of time. Therefore, they applied a qualitative methodology to define a model of the release management process. Despite the effort spent by the researchers to establish a development workflow, practices were not fully conducted even though some marginal benefits have been reported: planned major release from four to six months; planned minor releases from two to four weeks; encompassing urgent corrective releases from one to five days. Regarding the previously mentioned problems, only partial improvements have been achieved since the major obstacle was to motivate employees to change their habits. With perfect hindsight the researchers claim that understanding and adapting solutions by solving one problem at a time would have increased the benefits. Then, despite the reported results, starting the establishment of a simple and informal workflow aided the organization to move from the stage of knowing nothing to knowing at least something about their development process.

The integration of minimal project management during the development activities can foster the control over development productivity and control the *accumulated technical debt* (CAT5), but still with particular attention to the development workflow status within the company, designing specific and tailored solutions.

Eventually to externalize complexity of the developing product , two other studies [61], [62] suggest to *"reuse the wheel, instead of reinventing it"*, taking advantage of open source software whenever it is possible, building systems quickly and effectively as a result. Even though the principles in leading-edge development processes are the same principles of yesteryear, the way in which the fundamentals are applied has changed from prescriptive to agile solutions, with less concern on quality aspects, also confirmed by Mater [56].

5.2.3 Simple workflows

Prescribed development models, oriented towards SPI, are basically ignored by startups, as discussed in [53]. To overcome the need of structure and to accelerate the development process, Zettel developed a *"LIghtweight Process for E-business software development"* (LIPE). This lightweight development method integrates the XP approach with ideas from areas of software measurement for project control and process improvement as a compromise between an ad-hoc and a more rigorous approach. Zettel argues that LIPE provides scalability of the development process with a certain degree of flexibility, omitting parts of prescribed practices. Nevertheless, we couldn't identify any empirical evaluation of LIPE that supports its suitability in a startup context.

Deakins [54] designed a new spiral methodology, constructed during a fieldwork with a developer team that had limited resources managing an innovative product in volatile e-commerce environment. This methodology represents a fast evolutionary approach with flexible processes. It aims to obtain customer's feedback as soon as possible for continuous product improvement. Even in this case, the spiral methodology is compliant with the nature of a random, ad-hoc and *evolutionary development approach* (CAT2). Evolutionary development approaches are typically more suitable for web applications, as confirmed in [110]. Moreover, they provide practices oriented towards rapid and iterative design with complete freedom to the developer team, enabled to operate anywhere within the organization.

5.2.4 Full-stack development team

Carmel [37] reports on the need of flexible and rapid development solutions to shorten time-to-market, but even more important, the need of high-skilled team developers, which has an impact on *speed-up development* (CAT1). He suggests that entrepreneurs need to look for a well formed, skilled core development team and not just a set of product ideas and features. Moreover the team must be empowered with full-stack and self-organization settings, paired with the flexibility of minimal bureaucracy. A studied company stated: *"we didn't need weeks or months of detailed modeling and documentation, but rather modeling the architecture a little and then either exploring strategies by providing users with code or simply starting work on the actual software itself"*.

5.2.5 Summary

Summarizing the comparison, we have found that most of the contributions of the studies are compliant with the GSM. Startups focusing on simple workflows and full-stack engineers face the uncertainty and immaturity of the company. Nevertheless few considerations are presented towards the consequences of a minimal project management and the use of low engineering activities as presented in the GSM. However, the capability of the model to accommodate the results identified by previous research provides confidence in the generalizability of the GSM.

5.3 Confounding factors

The purpose of this subsection is to identify which confounding factors need to be taken into consideration when evaluating the GSM. Confounding factors are those variables which and have not been considered in the model and might interfere with the theoretical framework positively or negatively [71]. We report those factors that have been explicitly identified by the SMS [10] , related to: creativity and innovation, market requirements and application type, and developer experience.

5.3.1 Creativity and innovation

Heitlager et al. [22] contributed to the investigation of creativity and innovation in startups. The proposed model identifies three lifecycle stages of a startup: *fluid*, *transition* and *specific*. At each stage, the model presents different development approaches (product-oriented and process-oriented), according to the level of innovation achieved. The study reports how product-oriented development, in contrast with forced and mature process imposition, provides a degree of freedom to the development team, that enhances the creativity of developers and augments the innovation capability of the company in the early-stage.

In the interviews we conducted, and on which the GSM is grounded, the theme of innovation only seldom emerged. However, if innovative and creative factors were considered in this model, the integration of results would have been straightforward, since it is compliant with the description of how the *accumulated technical debt* (CAT5) causes the need of establishing a process in the long run (see how the *initial growth hinders performance* (CAT6)). From the *fluid* phase to the *specific* phase, the innovation confounding factor seems to behave initially the same as the *speed-up development*, both highly focused towards a product-oriented development. Nevertheless, we were unable to ground these evidences in the GT data even though those are factors that might be positively correlated to the choice of an early product-development orientation and the need of low-precision engineering elements.

5.3.2 Market requirements and application type

Market requirements [36] and application type [5], [18], [20] are two further potential confounding factors to the GSM. Their main impact is related to the adoption of flexible and reactive solutions for the development process. In particular, they refer to the necessity of fulfillment of quality concerns that goes beyond scalability and UX defined by the GSM. Especially when requirements are rigidly imposed or the application domain is well-known, providing low-quality products to final users might determine the failure of a startup.

For example, safety-critical systems possess quality concerns that must be considered. Even in e-commerce systems factors related to quality have high-impact effects on obtaining the first customer reference, as discussed in [54], [58], [44]. Quality attributes especially characterize those applications, whose part of functionalities are already successfully implemented in other software systems or whose failure might inflict serious financial or social damages [111]. Yet, limiting our context to early-stage software startups, which typically develop first cutting-edge products in beta versions, partially mitigates the high quality demands of users.

5.3.3 Developer experience

Finally, also the experience of team-members is considered as an important confounding factor. Since only little evidence has been gathered during the interviews, "lack of experience" was not considered as a sub-category in *severe lack of resources*. Nevertheless, startups rely at the beginning on clever, but inexperienced developers [2]. Having team-members with deep experience is a "double-edge sword" [41]. On one hand, experience might quickly provide structure and maturity to the development process, on the other it might cause challenges in managing self-confident overachievers that almost inevitably clash. Consequently, team management requires higher control and coordination activities that hinder the flexibility of the development environment, that is essential in early-stage startups. Accordingly, also Coleman [36], [20], [18] states that the operating strategies towards *"minimum processes"* in startups is not a matter of poor knowledge and training, but rather it is the necessity of operating with solutions that let the company move faster.

5.4 Static assessment of the GSM

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 interview data.

We argued in section 4.1, that the most essential feature startups aim to achieve is high speed. As illustrated in Figure 2, 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 "effectiveness" and "efficiency" since time-to-market was the main concern (as confirmed in [10]):

- The team enhances the overall *performance* of software development. 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 advice given by mentors and by a management style oriented towards self-organization, full-stack, and multi-role settings. The importance of a skilled team for development performance is expressed in many studies such as [24], [108], described in CAT4 and confirmed by Sutton [5].

- The evolutionary approach enhances the *effectiveness* in implementing the correct functionalities. This approach improves the company's capabilities to adjust the trajectory of product development (in agreement to the concept of pivoting discussed by Ries [67]).
- Since product quality has low priority, the development is focused only on implementing a limited number of suitable functionalities, enhancing its *efficiency*. 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 performance in later stages.

Accordingly, by analyzing the data of the 13 companies, we formulated two hypotheses which further describe the above illustrated relations:

1) The more a capable[14] team, 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[15].

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 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 [ADD URL].

We conducted statistical tests using analysis of variance to assess the existence of relations between the dimensions. First we defined two null hypotheses:

$H0_1$: *Startups do not release the product faster when a capable team adopts a more evolutionary approach AND with less quality constraints.*

$H0_2$: *The execution speed does not increase the amount of accumulated technical debt.*

The formulation of the two hypotheses reflects the representation of the categories related to *speed-up development* presented in the GSM model (see figure 2). We tested $H0_1$ and $H0_2$ with an one-tailed test using Pearson's product moment correlation coefficient[16], with positive association analysis, fixing the level of confidence to 95%. We reject both $H0_1$ and $H0_2$, and conclude that:

1) Higher values of *Execution speed* are strongly associated (p-value: 0.002) with higher values for *Technical debt*.
2) Higher values of *Execution speed* are strongly associated (p-value: 0.005) with higher values for *Potential capability*.

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 (see Figure 2). There are four high-level relationships which have not been validated by quantitative methods. However their correctness has been verified by a cross-analysis with existing literature and by attesting the causal relations among codes and categories, through axial and selective coding processes.

6 Implications of the GSM

In this section we present relevant implications which emerge from the behavior of early-stage startups, formally expressed in the GSM, 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 some patterns that emerged very clearly.

6.1 Shortening time-to-market

The most urgent priority of software development in startups 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,

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

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

16. To perform the Pearson's correlation we verified two assumptions: a) data is normally distributed; b) data is on interval scale.

coming up with very complex solutions is often 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.

Introducing a software development process in this early phases of high pressure can be challenging for startups. However, some development approaches, mostly investigated in mature companies, could help in shortening time-to-market. Among several methodologies that can be found in literature, the rapid application development (RAD) [112] facilitates prototyping and promotes a continuous interactive process that allows early-adopters to understand, modify and approve a working product that meets their needs. A study of Gerbel et. al. states that the use of RAD, which promotes minimal planning and rapid prototyping, allows software to be released faster and makes easier to change requirements [113].

6.2 Light-weight methodology

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*).

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 feedback is a central aspect of the *Lean startup methodology* pioneered by Ries [67]. 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 suggested good practices. Instead, startups in the early stage prefer to apply fast cycles of "build and fix" when necessary, hacking the first functionalities to present to potential customers and investors in order to obtain quick feedback. Moreover, when it comes to user feedback, startups do not explicitly follow the step-by-step process of *customer development* formally defined by Blank [7]. Instead, startups absorb and implement the high-level principles that outlines the customer development methodology, reflected in the GSM by the theoretical category *find the product/market fit quickly*.

While the absence of a basic process might enable startups to focus more on the product, some studies claim that companies can take advantage from the use of some engineering activities even in the early stages [114]. For instance an experience report conducted by Taipale [48] shows how their startup took advantage from tailoring some simple XP practices to their needs.

6.3 Focus on minimal set of functionalities

The first release of the product includes only a limited set of well suitable functionalities focused on user experience. These functionalities are usually set by the founders and their experience in a particular business domain. As soon as possible they provide demos to demonstrate the idea to potential customers.

To ship a product with the right features built in, startups need to prioritize and filter. From an engineering point of view, most startups do not explicitly apply traditional Requirement Engineering (RE) activities to collect and manage requirements. However, some studies have found that by integrating simple techniques, such as Persona and Scenario, companies can improve the effectiveness of the requirements elicitation even whether the final users is mostly unknown [115]. Another study suggest that using a lightweight project-initiation framework such as the Agile Inception Deck can help in preventing premature failure of the software project due to wrong understanding of the project requirements [116]. 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: 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[17]. 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 [118] - because of the little return of investment they would gain, as discussed by Christensen in [119]. 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.

6.4 Empowering the team members

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

17. 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 [117].

truly important for the startup, among other factors, is the extremely high-colocation which enables continuous informal and colloquial coordination and discussions. The large amount of communication, spontaneously held in informal places, has been proven to be an extremely effective element of several successful software projects [108]. Hence, the vastly *informal* development environment of startups acts as a catalyst of speed.

6.5 Paying back the technical debt

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. If successful, the company will face a growth in terms of number of customers, employees and product capabilities. Then, the increased complexity requires the team to start controlling the initial 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.

As stated by many interviewee, modern coding platforms (e.g. github[18]) and the use of well-known frameworks can be leveraged to mitigate the technical debt. Coding platforms allow developers to integrate in one place several engineering activities such as requirements lists, issue tracking, code repository, documentation, continuous integration, release and configuration management. Frameworks include support programs, compilers, code libraries and tool sets to enable the initial development of a project with limited overhead.

6.6 Summary

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 implications we presented are reflected in the behavior 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 [66], which appears to be the most important determinant for the future of software development in the company.

Future research can investigate the performance drop-down caused by the necessity of returning the accumulated technical debt while expanding the company's operation and structuring mitigation strategies with four software development objectives: integrating scalable solutions with fast iterations and minimal set of functionalities (this allows startups to maintain effective planning and realistic expectations); empowering team members enabling them to operate horizontally in all the activities of the development environment simultaneously; improve desirable workflow patterns through the initiation of a minimal project management over time, as a natural result of emerging activities of tracing project progress and task assignment mechanisms; and finally, only when the chaos has been initially managed, planning long-term performance by adoption of Agile and Lean development practices.

18. http://www.github.com

7 Threats to Validity

In this section we discuss the validity of the overall research methodology. We structure the discussion according to Wohlin's taxonomy [120].

7.1 External validity

One threat to external validity is the 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, and centers 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 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. The questionnaire dedicated to "data triangulation" was not of primary support because of the limited time practitioners were able to grant to it. However, researchers have to accept the veracity of what respondents reported during the interviews [20].

The comparison of the GSM with similar frameworks helped in establishing the domain to which the study's findings can be generalized. In particular the previous framework developed by Coleman [18] has allowed us to find similarities, differences and broader reasoning related to factors that hinder maturing processes in startups. Moreover, we analyzed Brook's framework describing the *basic attacks* to the challenges of developing software systems and other literature covered by the SMS on startups [10][19].

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

7.2 Internal validity

To enhance the internal validity, we created a three-dimensional research framework. By means of a 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 the generated theory.

To validate the GSM we conducted a comparison of the emergent theory with existing literature and previously developed models. Furthermore we quantitatively evaluated three important relationships in the model. With the theory validation we highlighted and examined similarities, contrasts and explanations as discussed in [121]. 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 a case study research [...] because the findings often rest on a very limited number of cases."

Nevertheless, we identified important confounding factors, related to innovation, market requirements and developer experience, which might impact the internal validity of the GSM.

7.3 Construct validity

One threat to this study is a possible inadequate description of constructs. In order to diminish this risk, the entire study constructs have been adapted to the terminology utilized by practitioners and defined at an adequate level for each theoretical conceptualization. For instance, we defined *Time shortage* in terms of *Investor pressure, CEO/business pressure, Demo presentations at events and internal final deadline* as used by most of the interviewees in the study. Moreover, during the coding of interview transcripts, we adopted 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 didn't disclose any goal and emergent results of our research to the interviewees.

Another threat associated with construct validity is evaluation apprehension, which describes people's rejection attitude towards the assessment of their behavior. To address this risk we didn't attain to the first answers from practitioners, but rather asked for details of artifacts and engineering strategies. 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 research, in other contexts, in order to criticize our findings (interview package with instructions has been made available online[20]. Moreover, two researchers not involved in the execution of the study conducted a peer-review analysis of the model'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.

7.4 Conclusion validity

Grounded theory has been already applied by other researches in similar contexts to attest relationships among conceptualizations of an examined phenomenon (see [36], [122], [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."* [123].

Despite we couldn't apply any power analysis procedures, in grounded theory the sample size depended on when data saturation occurred. Ramer in [124], comparing quantitative to qualitative researches, states: *"reaching data saturation, which involves obtaining data until no new information emerges, is critical for obtaining applicability*

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

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.

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, the SMS by Paternoster et al. [10] 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 exploratory research, conducted by a systematic approach, we produced a theoretical model grounded into the hindsight knowledge collected among practitioners with the aim of explaining how development strategies are engineered in startups. The explanatory capability and correctness of the model has been validated by means of systematic comparisons with the state-of-the-art and empirical data. The systematic mapping study [10], with 37 studies extracted, 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 characteristics of uncertainty, lack of resources and time-pressure. These factors influence the software development to an extent that transforms 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. 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. 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. 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 [125], we would like to see the rise of a new discipline - *startup engineering* - which can be defined as *the use of scientific, engineering, managerial and systematic approaches with the aim of successfully developing software systems in startup companies*.

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, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, 2013.

[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] 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.

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

[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 startups; 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] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” *Journal of Systems and Software*, no. 0, pp. – , 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121213000022

[64] W. Cunningham. The WyCash Portfolio Management System. [Online]. Available: http://c2.com/doc/oopsla92.html (Accessed : Sep. 15, 2012).

[65] Third international workshop on Managing Technical Debt. [Online]. Available: http://www.sei.cmu.edu/community/td2012/ (Accessed : Sep. 15, 2012).

[66] 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.

[67] E. Ries, *The Lean Startup: How Today’s Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business, 2011.

[68] Microsoft, *Microsoft Application Architecture Guide*. Microsoft Press, 2009.

[69] 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.

[70] B. G. Glaser, *Theoretical sensitivity : advances in the methodology of grounded theory*. Sociology Press, 1978, vol. 2.

[71] Colin Robson, *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. John Wiley and Sons, 2009.

[72] O. W. Bertelsen, “Toward A Unified Field Of SE Research And Practice,” *IEEE Software*, vol. 14, no. 6, pp. 87–88, 1997.

[73] 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.

[74] J. Corbin and A. Strauss, “Grounded theory research: Procedures, canons, and evaluative criteria,” *Qualitative Sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[75] 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.

[76] Startup types distribution. [Online]. Available: http://techcrunch.com/2012/02/17/crunchbase/ (Accessed : Aug. 25, 2012).

[77] 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.

[78] C. W. Dawson, *Projects in Computing and Information Systems A Student ’ s Guide*. Pearson Prentice Hall, 2009.

[79] MIT License. [Online]. Available: http://www.opensource.org/licenses/mit-license.php/ (Accessed : Aug. 25, 2012).

[80] C. Giardino and N. Paternoster. Interview Package. [Online]. Available: https://github.com/adv0r/BTH-Interview-Package, [Aug. 25, 2012).

[81] 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.

[82] R. K. Yin, *Case study research: design and methods*, R. K. Yin, Ed. Sage Publications, Inc, 1994, vol. 1, no. 3.

[83] 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.

[84] G. Coleman, *Software process in practice: A Grounded Theory of the irish software industry*. Springer, 2006.

[85] A. Davis, “Operational prototyping: a new development approach,” *Software, IEEE*, vol. 9, no. 5, pp. 70 –78, sept. 1992.

[86] S. Borodovsky and S. Rosset, “A/b testing at sweetim: The importance of proper statistical analysis,” in *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, dec. 2011, pp. 733 –740.

[87] 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.

[88] R. Pooley and P. King, “The unified modelling language and performance engineering,” *Software, IEE Proceedings -*, vol. 146, no. 1, pp. 2 –10, feb 1999.

[89] 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.

[90] F. Brooks Jr, “No Silver Bullet — Essence and Accidents of Software Engineering,” *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[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] D. Wood, “Open Source Software Strategies for Venture-Funded Startups,” MIND Laboratory, University of Maryland, Tech. Rep. TR-MS1287, 2005.

[96] R. Stanfill and T. Astleford, “Improving Entrepreneurship Team Performance through Market Feasibility Analysis, Early Identification of Technical Requirements, and Intellectual Property Support,” in *American Society for Engineering Education Annual Conference*, 2007.

[97] S.-C. Li, “The role of value proposition and value co-production in new internet startups: How new venture e-businesses achieve competitive advantage,” in *Management of Engineering and Technology, Portland International Center for*, Aug. 2007, pp. 1126 –1132.

[98] H.-J. Steenhuis and E. de Bruijn, “Innovation and technology based economic development: Are there short-cuts?” in *Management of Innovation and Technology, 2008. ICMIT 2008. 4th IEEE International Conference on*, Sept. 2008, pp. 837 –841.

[99] S.-l. Lai, “Chinese Entrepreneurship in the Internet Age : Lessons from Alibaba.com,” *World Academy of Science, Engineering and Technology*, vol. 72, pp. 405–411, 2010.

[100] M. Häsel, N. Breugst, and T. Kollmann, “IT Competence in Internet Founder Teams An Analysis of Preferences and Product Innovativity,” *Business & Information System Engineering*, vol. 52, no. 4, pp. 210–217, 2010.

[101] R. Cooper, “An Investigation into the New Product Process : Steps, Deficiencies, and Impact,” *Journal of product innovation management*, vol. 3, no. 2, pp. 71–85, 1986.

[102] T. DeMarco, *Peopleware: Productive Projects and Teams (Second Edition)*. Dorset House; 2nd edition, 1999.

[103] G. Coleman, “eXtreme Programming (XP) as a ”Minimum” Software Process : A grounded theory,” in *Proceedings of the 28th Annual International Computer Software and Applications Conference*, 2004, pp. 4–5.

[104] A. Valtanen, “Big Improvements with Small Changes : Improving the Processes of a Small Software Company,” *Product-Focused Software Process Improvement*, no. 2006, pp. 258–272, 2008.

[105] S. Adolph and P. Kruchten, “Reconciling Perspectives: How People Manage the Process of Software Development,” in *AGILE Conference*. IEEE, Aug. 2011, pp. 48–56.

[106] A. Cockburn, “Characterizing people as non-linear, first-order components in software development.” Multi-Conference on Systems, Cybernetics and Informatics, Orlando, Florida, Tech. Rep., 1999.

[107] R. Huys and V. Jirsa, *Nonlinear Dynamics in Human Behavior*. Springer, 2011.

[108] J. A. Highsmith, *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House, 2000, vol. 12.
[109] G. Cugola and C. Ghezzi, "Software processes: a retrospective and a path to the future," *Software Process: Improvement and Practice*, vol. 4, no. 3, pp. 101–123, 1998.
[110] Y. Deshpande and S. Hansen, "Web engineering: creating a discipline among disciplines," *IEEE Multimedia*, vol. 8, no. 2, pp. 82–87, 2001.
[111] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.
[112] S. McConnell, *Professional software development: shorter schedules, higher quality products, more successful projects, enhanced careers*. ADDISON WESLEY Publishing Company Incorporated, 2004. [Online]. Available: http://books.google.de/books?id=rYKyDS-raBEC
[113] A. Gerber, A. V. D. Merwe, and R. Alberts, "Practical implications of rapid development methodologies," 2007.
[114] F. Maurer and S. Martel, "Extreme programming. rapid development for web-based applications," *Internet Computing, IEEE*, vol. 6, no. 1, pp. 86–90, 2002.
[115] M. Aoyama, "Persona-and-scenario based requirements engineering for software embedded in digital consumer products," in *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*, 2005, pp. 85–94.
[116] J. Rasmusson, "Agile project initiation techniques - the inception deck amp; boot camp," in *Agile Conference, 2006*, 2006, pp. 5 pp.–341.
[117] J. Offutt, "Quality attributes of web software applications," *IEEE Softw.*, vol. 19, no. 2, pp. 25–32, Mar. 2002.
[118] E. M. Rogers, *Diffusion of Innovations*. Free Press, 1995.
[119] C. M. Christensen and M. E. Raynor, *The Innovator's Solution*. Harvard Business School Press, 2003.
[120] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.
[121] 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.
[122] 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.
[123] P. Y. Martin, "Grounded theory and organizational research," *The Journal of Applied Behavioral Science*, vol. 22, no. 2, pp. 141–157, Apr. 1986.
[124] L. Ramer, "Quantitative versus qualitative research?" *Journal of obstetric, gynecologic, and neonatal nursing : Jognn / Naacog*, vol. 18, no. 1, pp. 7–8, 1989.
[125] 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.