Towards a Framework to Guide the Creation of Development Practices for Software Startups

Jorge Melegati(✉)

Faculty of Computer Science, Free University of Bozen-Bolzano, Bolzano, Italy
jorge@jmelegati.com

Abstract. The research on software startups has increased lately, focusing on describing how these companies' unique context influences development practices. The next step for research is the creation of specific practices for these companies grounded in scientific results. An obstacle in this path is which dependent variable these novel practices should improve. A natural answer is these companies' success. This position paper reviews the literature on new ventures and startups' success to show that telling if a startup is successful or not is a complex issue. As a solution to this problem, this paper proposes a conceptual framework, suggesting that novel practices should improve success determinants or reduce inhibitors rather than focusing on the startups' success. Three examples illustrate the framework's use: hypotheses engineering, microservices, and BizDev. The identification of contributors and inhibitors for success of software startups could enrich the framework and indicate possible avenues for the creation of development practices specific tailored for these companies.

Keywords: Software startups · Startup success · Software success · Software engineering practices

1 Introduction

Recently, the research in software startups has increased in number and rigor [4], focusing on describing the consequences of this unique context to software engineering [15]. For instance, Giardino et al. [11] proposed the Greenfield Startup model to describe software development in startups as a sped up process leading to the consequence of accumulated technical debt. However, the creation of specific development practices tailored to the unique context these companies face is still in its infancy [15]. A natural question towards this goal is what the dependent variable is, i.e., which aspects these novel practices should improve to be considered useful. Besides guiding the creation of novel practices, a better definition of what is a successful startup allows the investigation of what makes some startups achieve better results allowing others teams to replicate practices employed.

Software startups are companies searching for a scalable business model for a new software-intensive product [19]. Hence, a logical answer to the dependent variable question is success. However, this answer leads to at least two other questions: what is success for a software startup and to which extent it could be influenced by software development.

This paper explores possible answers to these two questions using results from the literature and examples of failed startups. Based on a review of the literature about success of new ventures and startups, and its determinants, I argue the complexity to define and to assess what a successful startup is, but acknowledge the existence of several contributors and inhibitors to success. Thus, I propose a conceptual framework to guide the creation and evaluation of specific software development practices to startups. According to the framework, instead of focusing on success, studies should aim to improve a contributor or reduce an inhibitor without negatively contributing to other aspects. To illustrate the framework use, I employ it to propose the evaluation of practices that, although not specifically proposed for software startups, could be employed in this context: hypotheses engineering, microservices, and BizDev.

2 Success of Software Projects

A logical approach to guide the creation of development practices for software startups is to distinguish the software project, or "technical," success from the business outcomes. However, this dichotomy is contrary to the literature on software success. The discussion of software project success and failure is a complex argument and to present it in detail would not fit this paper. Nevertheless, an overview, presented below, is essential to the flow of argument.

Ralph and Kelly [17] investigate the success concept in the software engineering context. First, they observe how the understanding of this concept evolved in project management literature. They acknowledge that, for a long time, a model used to describe such a phenomenon was the Iron, or Project, Triangle [3]. This metaphor postulates that the quality of a product (inner of the triangle) is constrained by three vertices: scope, budget, and schedule [1]. Then, the authors describe a more comprehensive taxonomy proposed by Shenhar et al. [18], which characterizes success through four dimensions: project efficiency, impact on the customer, business success, and preparing for the future. Finally, they interviewed 191 design professionals, where 68 were in the software industry, to investigate the practitioners' perceptions on success. The results indicated that professionals understand software engineering success "as a multidimensional variable comprising project efficiency, artifact quality, market performance, and stakeholder impacts over time."

Therefore, software project success does not concern only the quality, scope, and schedule of the delivered software system. Following Shenhar et al.'s taxonomy, other measures that seem especially important for software startups, are, for instance, fulfilling customer needs, commercial success, and creating a large market share. As Ralph and Kelly [17] observe, practitioners perceive the existence of a client who has problems to solve or needs to be fulfilled. Software startups face an even harder challenge: they must find not only the problems but even the customers. Some types of pivots, strategical changes on a startup's business model or product [2], support such an argument. For instance, a customer segment pivot is a shift from one customer segment to another [2]. Zoom-in pivots occur when a single feature becomes the whole product and zoom-out pivots, when a product becomes only a feature of a bigger solution.

In summary, the success of a software system goes beyond developing a set of features in a determined period of time using a prescribed budget. It also means fulfilling customer needs and reaching commercial success. Therefore, in the context of software startups is natural to expect the company success. The next section presents a review on this aspect.

3 Success of Startups

Some studies in the literature focused on the success of startups, either to learn from them, e.g. [21], or to predict them, e.g., [7]. To do so, researchers had to define what a successful startup is. Zaheer et al. [21] considered companies that had achieved one or more of the following: continued survival, a high sales volume, a stock exchange listing, or acquisition. For Dellermann et al. [7], it was sufficient to have received a series A funding, that is, "a venture capital backed funding that allows angel investors to exit the startup." A quick search on specialized media shows that these criteria for success are spread among practitioners.

However, these aspects are problematic in two ways. First, it is hard to precisely assess them either by the time needed or by the levels distinguishing different outcomes. For instance, how long should a startup run to be considered successful? Or what is a high volume of sales? Second, and most important, there are several companies that featured these criteria but were considered failures at the moment of assessment or later. A criterion based on stock exchange listing does not work, for instance, when analyzing startups that failed during the dot-com bubble. Another issue with adopting this rule is a recent trend to stay private rather than becoming public [9]. Startups can decide to not run an Initial Public Offering (IPO) to, among other reasons, wait for a better moment or to avoid the compliance required as a public company. Regarding the acquisition criterion, a counterexample is the concept of acquihire, defined by the Cambridge Dictionary[1] as: "to acquire (= buy) a company in order to use its employees' skills or knowledge, rather than for its products or services." Thus, a startup whose product failed might still be acquired by a larger company as a way to hire all team members at once. With respect to investments received, a piece of evidence against the criterion comes from a study by Cantamessa et al. [5], that investigated the failure of 214 startups based on postmortems. Among these companies, 14% failed after more than five years operating and, in this cohort, startups closed an average of 2.16 round of investments, receiving, in average, 16.39 million dollars.

[1] https://dictionary.cambridge.org/dictionary/english/acquihire.

It is beyond the scope of this paper to thoroughly define what a successful startup is. However, this brief discussion showed the complexity of the success concept for startups. A reliable assessment of this aspect requires a thorough, holistic inspection of the company, combining several aspects of the business and the product, to overcome the deficiency of employing a single criterion [12]. Besides the difficulty to reliably judge which startups are successful, there is an even major concern to software engineering research in this context: to which extent development techniques could influence the success of these companies.

Some studies focused on identifying determinants of success in high-tech new ventures. Based on a survey with 27 venture capitalists, Kakati [12] concludes that the critical determinants are entrepreneur quality, resource-based capability, and competitive strategy. Besides that, an interesting result is the distinction between winning and qualifying criteria. Winning aspects are those "which directly and significantly contribute to wining business," while qualifying criteria may not be "major determinants of success," but "any reduction [...] will be particularly serious if it drops below the critical level." The authors also performed a principal component analysis to identify factors that could explain the variance from success and failure, finding nine factors. The first factor, "incapability risk," is related to the capabilities needed to succeed. The second, "inexperience risk," captures the lack of track record and market knowledge. The third, "product risk," is the first that could be associated to software development. In the case of novel ideas, as the case of software startups, the authors acknowledge that "there is a risk of whether the product can be produced and commercialized" and even "technically elegant products may fail to exploit the untapped market."

A similar result comes from the investigation of startups' failures. These studies focus on failures rather than success since, by making these issues evident, their mitigation is easier, and more startups could be successful. In their investigation of startup failures, Cantamessa et al. [5] observed a "typical failure pattern related to the Business Development process." The authors analyzed 214 postmortems and identified one or more reasons to failure. The first two aspects were wrong business model and the lack of business development, occurring in 35% and 28% of the cases, respectively. Behind these issues, the authors observe "a high focus on the product or service by the management and founders, but an insufficient attention to commercial development." After the third position being running out of cash, happening in 21% of the cases, the fourth aspect is lack of product/market fit, in 18% of the cases, another issue related to the business development.

A natural reaction to this conclusion is that founders should focus on improving the business development process and the software engineering team should care after the business elements are set. However, experimentation, the basis of the business development usually advocated for startups, where product assumptions are taken as hypotheses and tested, represents a unique problem for software engineering, since currently available methods for software development revolve around requirements obtained with varying degrees of customer contact frequency [15]. This uniqueness means several consequences to software development. First, since experiments are based on hypotheses and not in requirements as conventional practices, requirements engineering practices might not be the most suitable to the this context. Second, the heavy use of experiments and the probable consequence of the implemented features influence the software system's architecture and design. Bad choices could contribute to technical debt accumulation and compromise the system capacity to handle larger loads or the team ability to maintain the code or add new features during scaling the product. Third, startups are characterized by small teams where members often perform multiple and diverse tasks. A software developer in a startup has a higher probability to be involved in other tasks than programming.

In summary, defining whether a startup is successful or not is a complex endeavor and represents a challenge to studies aiming to either learn from these companies or to create novel techniques to improve these companies' chance of success. A more viable option is to focus on improving (or detecting the existence of) determinants of success and reducing aspects that increase the risk of failure. To summarize this idea, the next section proposes a conceptual framework to guide the research on practices specific for software startups.

4 Conceptual Framework

Based on the review above, it is complex and, often, dubious to assess if a startup is successful or not. This section presents a conceptual framework to support researchers, or practitioners, interested in developing, or evaluating, practices to software startups. It is essential, though, to highlight that this framework does not aim to answer what success is for these companies.

Research has identified many contributors and inhibitors to success, that is, aspects influencing the success of a startup either positively (contributors) or negatively (inhibitors). Similarly, several other aspects, including software engineering practices, could influence positively (or negatively) these antecedents of success. Clearly, not all of them could be influenced by software development practices. For instance, changes on legal or economical frameworks might derail a startup's business model. To simplify the framework, I did not explicitly model aspects that could act as a contributor or inhibitor depending on the level. These aspects could be modeled in a way to fit this framework without losing generality.

Figure 1 depicts the framework using arrows to represent the influence of several aspects, represented by boxes, including software development practices to guide the creation of practices specific to software startups. The labels associated with the arrows tells if the influence is positive (represented by the '+' arrow), negatively ('−' arrow), or not ('0' arrow).

To use the framework, a researcher, aiming to create a novel technique, should identify which aspects associated, either positively or negatively, to startup success that activity could influence. Then, the researcher should demonstrate that the new technique improves contributors, reduce inhibitors, or both. The proposed technique should also lead to no, or limited, negative consequences or the

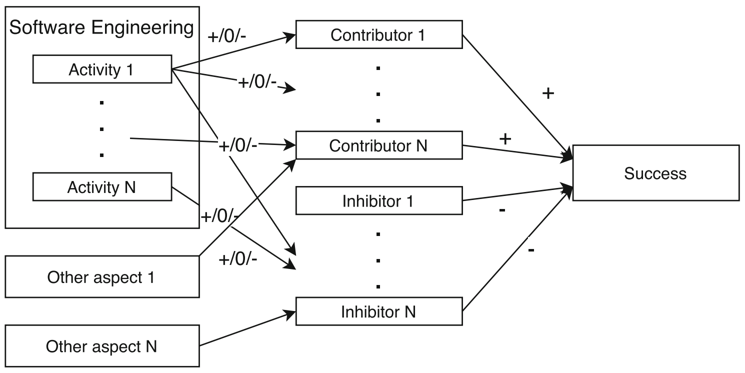

Fig. 1. A conceptual framework to guide the creation of practices for software startups.

researcher should, at least, acknowledge this issue and point out this problem as an issue to be solved. In this initial version of the framework, I do not list possible contributors and inhibitors to success but, in future work, this addition could be achieved by reviewing the literature on new ventures and inspecting failure cases. This list creation process should also consider the different stages a startup faces [13]: inception, stabilization, growth, and maturity. Some contributors and inhibitors could be associated with goals of specific stages. For instance, in the stabilization stage, the startup should prepare for scaling in the next stage. If a technique is proposed to be used in this stage, it should not have negative consequences to this goal.

5 Some Examples of the Framework Use

To exemplify the framework use, this section describes how researchers could evaluate if three proposed practices from different areas of software engineering are useful for software startups based on their influence on a contributor and on an inhibitor to success. Based on the review presented in Sect. 3, the business development speed is a contributor, an aspect positively associated to startup success. On the other hand, the accumulated technical debt could act as an inhibitor, that is, negatively associated, to success, especially when a startup reaches the scaling stage.

Hypotheses Engineering [16] is a proposal to support experiment-driven software development, as a parallel to Requirements Engineering in the conventional, requirements-driven software development. Rather than describing features to be implemented, hypotheses define uncertain questions about the business model or other software product aspects that the team could assess with an experiment. Since Requirements Engineering's goal is to precisely define the problem the software should solve [6], software startups represent a unique challenge to development. A customer on whom to rely to elicit requirements, a feature present in traditional and agile methodologies alike [15], lacks in the context of software startups. Therefore, Hypotheses Engineering could be a valuable practices for software startups.

Following the framework, an evaluation of the Hypotheses Engineering for software startups could assess, for instance, the speed with which startups adjust their plans based on information about the customer and market rather than the companies' success.

However, although the use of experiments can reduce technical debt by not building unnecessary features, it could also increase the technical debt [20]. For instance, one type of experiment is a prototype to deliver features as soon as possible to get customers' feedback. However, these quick solutions probably lead to high levels of technical debt. Therefore, an evaluation of Hypotheses Engineering for software startups should also assess the influence of these practices to technical debt.

Microservices is "an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms" [14]. It could be useful for a fast changing product as a way to isolate features, remove them if not useful, facilitate the software modularization, and scale. Using the framework, rather than assessing success, the employment of microservices should facilitate business experimentation while keeping technical debt low.

BizDev is a role proposed by Fitzgerald and Stol [10], as a closer integration between business and software development functions, a parallel to the well-known DevOps phenomenon. DevOps is "an organizational shift in which [...] cross-functional teams work on continuous operational feature delivery" by employing "automated development, deployment, and infrastructure monitoring" [8]. That is, instead of separated teams, development and operations work together in the delivery of features. Similarly, BizDev proposes collaboration between development and business strategy [10]. This new role could be an answer for the biggest issue for professionals in software startups: the mindset shift from strictly developing a feature defined by customers or other stakeholders to an active participant in the business development. For instance, several pivots mean partial or total modifications to software [2]. To evaluate the effectiveness of this new role, a researcher could evaluate if it indeed increases the speed on which the startup performs business development.

Figure 2 summarizes how the framework could be used to evaluate these three proposals when applied in software startups. Rather than measuring the companies' success, using or not these techniques, researchers could test, for instance, if they improve the speed of business development without increasing the technical debt.

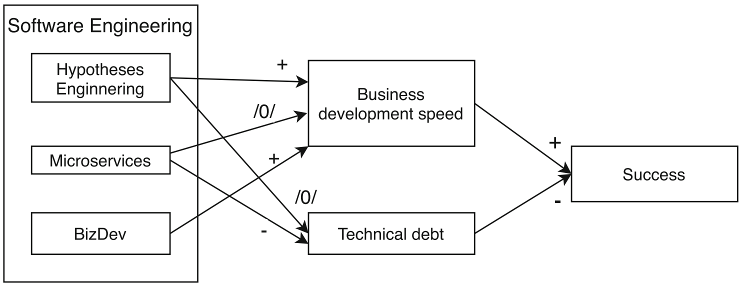

Fig. 2. Three examples applying the framework.

6 Conclusions

Researchers focused on improving software engineering practices for startups depend on clearly defining which dependent variable to improve. Although success is a natural answer, to tell if a software startup is successful or not is a complex task, requiring an holistic analysis of the company. This issue also hinders research aiming to learning from successful cases. To support these researchers, this paper reviews the literature to argue that success is complex and studies should focus on improving determinants and reducing inhibitors of success. To operationalize this argument, I propose a conceptual framework and, to illustrate its use, I gave three examples of how it could help the evaluation of novel practices from diverse areas of software engineering in startups.

The argument presented in this paper is not essentially new and a similar one has been implicitly employed in software engineering research. That is, researchers evaluate proposed techniques against diverse aspects rather than software project success. However, the uniqueness of software startups induces the use of success as a final goal, or a reference, to the development of new practices. This paper argued that this choice is not practical. Besides that, it showed that focusing on diverse aspects than those generally used in software engineering research might be useful as the case of business development. This paper is an initial proposal and, in future work, we will develop the framework further identifying contributing factors from the literature and inhibitors from failed startup cases which are abundantly available in Internet.

References

1. Atkinson, R.: Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. Int. J. Project Manag. **17**(6), 337–342 (1999)
2. Bajwa, S.S., Wang, X., Nguyen Duc, A., Abrahamsson, P.: "Failures" to be celebrated: an analysis of major pivots of software startups. Empirical Softw. Eng. **22**(5), 2373–2408 (2017)

3. Bano, M., Zowghi, D., da Rimini, F.: User satisfaction and system success: an empirical exploration of user involvement in software development. Empirical Softw. Eng. **22**(5), 2339–2372 (2017)
4. Berg, V., Birkeland, J., Nguyen-Duc, A., Pappas, I.O., Jaccheri, L.: Software startup engineering: a systematic mapping study. J. Syst. Softw. **144**, 255–274 (2018)
5. Cantamessa, M., Gatteschi, V., Perboli, G., Rosano, M.: Startups' roads to failure. Sustainability **10**(7), 2346 (2018)
6. Cheng, B.H.C., Atlee, J.M., Joanne, M.: Research directions in requirements engineering. In: Proceedings of the 2007 Future of Software Engineering, FOSE 2007, pp. 285–303 (2007)
7. Dellermann, D., Ebel, P., Lipusch, N., Popp, K.M., Leimeister, J.M.: Finding the unicorn: predicting early stage startup success through a hybrid intelligence method. In: ICIS 2017: Transforming Society with Digital Innovation (2018)
8. Ebert, C., Gallardo, G., Hernantes, J., Serrano, N.: DevOps. IEEE Softw. **33**(3), 94–100 (2016). https://doi.org/10.1109/MS.2016.68
9. Ewens, M., Farre-Mensa, J.: The Evolution of the Private Equity Market and the Decline in IPOs (2017)
10. Fitzgerald, B., Stol, K.J.: Continuous software engineering: a roadmap and agenda. J. Syst. Softw. **123**, 176–189 (2017)
11. Giardino, C., Paternoster, N., Unterkalmsteiner, M., Gorschek, T., Abrahamsson, P.: Software development in startup companies: the greenfield startup model. IEEE Trans. Softw. Eng. **42**(6), 585–604 (2016)
12. Kakati, M.: Success criteria in high-tech new ventures. Technovation **23**(5), 447–457 (2003)
13. Klotins, E., et al.: A progression model of software engineering goals, challenges, and practices in start-ups. IEEE Trans. Softw. Eng. **47**(3), 498–521 (2021)
14. Lewis, J., Fowler, M.: Microservices (2014). https://martinfowler.com/articles/microservices.html. Accessed 30 June 2021
15. Melegati, J., Chanin, R., Sales, A., Prikladnicki, R.: Towards specific software engineering practices for early-stage startups. In: Paasivaara, M., Kruchten, P. (eds.) XP 2020. LNBIP, vol. 396, pp. 18–22. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58858-8_2
16. Melegati, J., Wang, X., Abrahamsson, P.: Hypotheses engineering: first essential steps of experiment-driven software development. In: IEEE/ACM Joint 4th International Workshop on Rapid Continuous Software Engineering and 1st International Workshop on Data-Driven Decisions, Experimentation and Evolution (RCoSE/DDrEE), pp. 16–19 (2019)
17. Ralph, P., Kelly, P.: The dimensions of software engineering success. In: Proceedings of the 36th International Conference on Software Engineering - ICSE 2014, no. 1, pp. 24–35. ACM Press, New York (2014)
18. Shenhar, A.J., et al.: Project success: a multidimensional strategic concept. Long Range Plan. **34**(6), 699–725 (2001)
19. Unterkalmsteiner, M., et al.: Software startups - a research agenda. e-Inform. Softw. Eng. J. **10**(1), 1–28 (2016)
20. Yli-Huumo, J., et al.: The Relationship Between Business Model Experimentation and Technical Debt, vol. 210, pp. 17–29 (2015)
21. Zaheer, H., Breyer, Y., Dumay, J., Enjeti, M.: Straight from the horse's mouth: founders' perspectives on achieving 'traction' in digital start-ups. Comput. Hum. Behav. **95**, 262–274 (2019)