Software Engineering Anti-patterns in Start-ups

E. Klotins, M. Unterkalmsteiner, T. Gorschek
SERL, Blekinge Institute of Technology, Karlskrona, Sweden

Software start-up failures are often explained with a poor business model, market issues, insufficient funding, or simply a bad product idea. However, inadequacies in software engineering are relatively unexplored and could be a significant contributing factor to the high start-up failure rate.

In this paper we present the analysis of 88 start-up experience reports, revealing three anti-patterns associated with start-up progression phases. The anti-patterns address challenges of releasing the first version of the product, attracting customers, and expanding the product into new markets.

The anti-patterns show that challenges and failure scenarios that appear to be business or market related are, at least partially, rooted in engineering inadequacies.

Keywords:
D Software/Software Engineering,
D.2.1 Requirements/Specifications < D.2 Software Engineering < D Software/Software Engineering,
D.2.19 Software Quality/SQA < D.2 Software Engineering < D Software/Software Engineering,
D.2.9.i Programming teams < D.2.9 Management < D.2 Software Engineering < D Software/Software Engineering

Key findings, "tweets"

1. A time consuming and expensive MVP is a sign of poor technology choices and overscoping, the first start-up anti-pattern.

2. Lack of customer interest in the product could be rooted in a failure to establish a feedback loop earlier, the second start-up anti-pattern.
3. Difficulties to scale product into new markets could stem from lack of organizational support, the third start-up anti-pattern.
4. To save time, resources and greatly improve chances of successful project launch in start-ups, the focus should be on better engineering, not more of it.

Software start-ups are important suppliers of innovation and software-intensive products and services. There has been a steady growth in capital invested with 2017 setting a record of 16 billion EUR invested in European start-ups [1]. However, only a few percent of start-ups manage to deliver any value. Thus a significant capital is wasted on building unsuccessful products [2]. On the surface, the low success rates can be explained by market challenges, the difficulty in attracting customer interest, and resource shortages among others. However, digging deeper, we found indications that the capability to build software efficiently with minimal resources and limited knowledge about emerging target markets is the foremost challenge in software start-ups – and this is to a large extent closely related to engineering practices [3]. If even only a small part of start-up failure can be attributed to failures in the actual engineering practices and principles applied it translates into a significant loss of investments. Thus this paper focuses on the engineering applied in start-ups.

Very little is known about software engineering in start-ups and to what extent the lack of customer interest and resource overruns are related to deficiencies in engineering [4]. Earlier studies suggest that scoping and building a minimum viable product is a substantial challenge and precedes any market or business related challenges [3], [5]. Thus, failure in engineering could hinder

any subsequent attempts to market the product and to build a sustainable business around it.

To explore how software engineering is applied in start-ups and how inadequacies herein could be linked to start-up failures we examine 88 start-up experience reports. We apply qualitative analysis methods to identify recurring failure scenarios and their root causes. We present our results in the form of three anti-patterns: (1) not getting the first product release out, (2) not attracting customers to the product, and (3) challenges of scaling the product for new markets.

The differentiation between symptoms and root causes, as presented in this paper, of potentially dangerous scenarios enable practitioners to assess their situation better and apply corrective practices to the root causes. Moreover, the analysis pinpoints potentially interesting research directions to further understand software engineering practices and principles used in start-ups.

Research methodology

To maintain transparency of our results we present four steps of our research methodology.

Table 1, Overview of research methodology

#	Step	Description
1	Data collection	We used a repository of start-up experience reports [6]. Reports in the repository are written by start-up practitioners and describe lessons-learned and reflections after critical events such as product launch, buyout, or closure of the company. Although these reports were not compiled for this study, they are a primary data source providing an original insight of how do start-ups operate. The reports are not limited to software engineering; they also cover business, marketing, teamwork and personal issues relevant to start-ups.
2	Screening	The original dataset consisted of 93 reports. We screened contents of the reports and removed 5 reports from companies not developing software, and established companies. Most of the reports are between 1000 and 2000 words long.
3	Coding	We analysed the reports by following qualitative data analysis methods and used descriptive (to summarize), process (to capture ongoing action) and evaluation (to assess the situation) coding jointly to capture analysis points in the experience reports. Through analysis of the described situation, we aimed to differentiate between reported symptoms (e.g., running out of resources) and actual causes (e.g., poor resource planning due to lack of experience). The resulting codes briefly summarize key situations in a company along with contributing factors to the situation.
4	Development of anti-patterns	We grouped similar codes and chained them to create cause-effect diagrams. We use the practitioners' reflections and related work to suggest countermeasures to each anti-pattern.

Threats to validity

We identify two main sources of validity threats, the first stemming from the used data source, and the second, from our interpretation of the reports.

The analysed reports were not created for this study, thus may lack important details about the software engineering process, including precise details about the engineering context, such as team size, roles, and skills. Moreover, the reports are essentially subjective self-evaluations of practitioners. The practitioners could rationalize their shortcomings with external circumstances. However, these threats are alleviated due to a relatively large sample, and diverse population.

Another possible threat stems from single researcher bias in the coding process. To address this threat, we applied researcher triangulation. At multiple points in the coding process, selected reports where independently analysed by all three authors, the results reviewed and discussed.

Studied sample

The studied reports reflect on events between 2001 – 2015 and describe experiences from a diverse set of companies in developed products, geographical location, founders' backgrounds, and different development scenarios. In Figure 1 we illustrate our sample of studied start-ups.

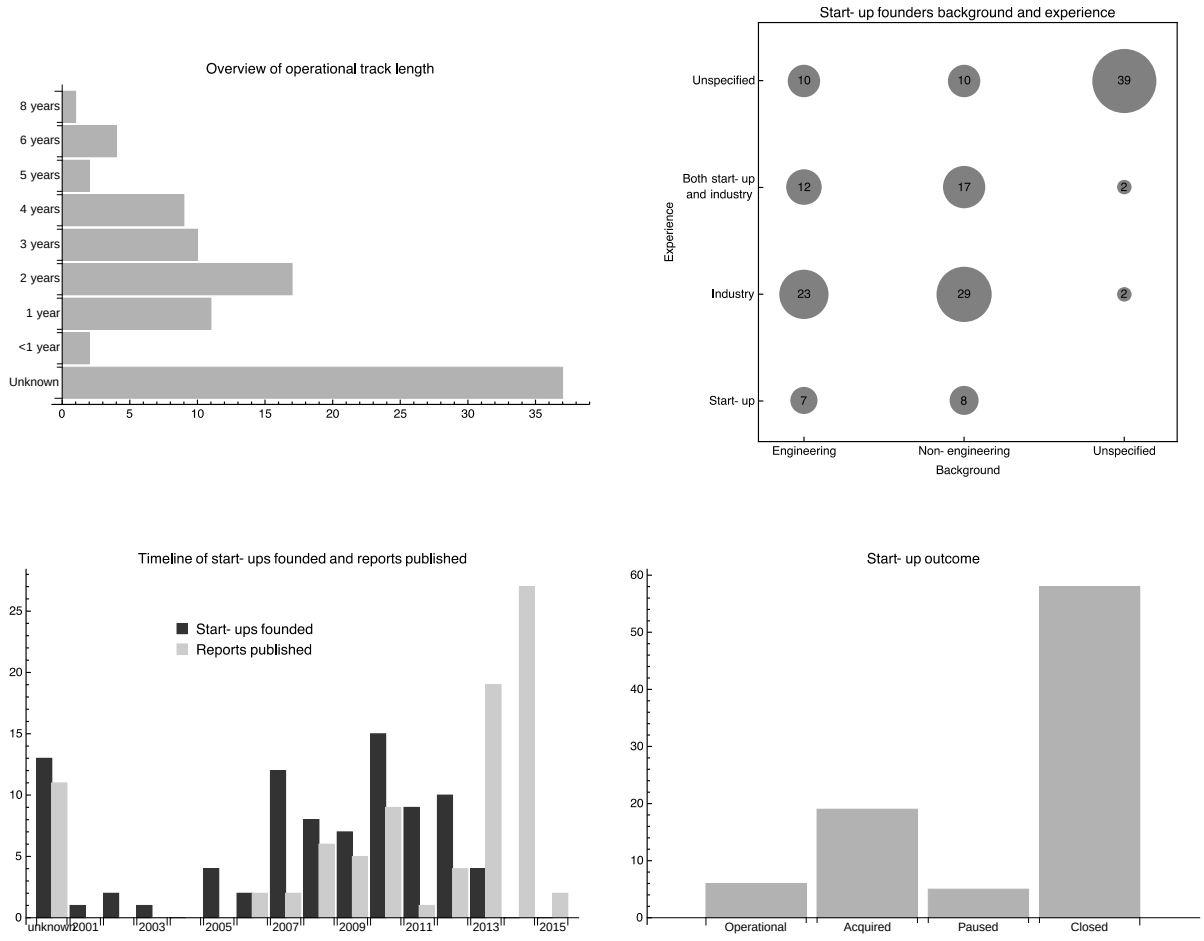

Figure 1, demographics of the sample

To compile the demographical information, we rely on information in the reports and publicly available information about the companies.

Start-up engineering anti-patterns

Our analysis shows that start-ups experience different challenges depending on how far into product development they are. From the reports, we identified three phases in start-up progression characterized by specific goals, and challenges. Phase I is concerned with building and releasing the first release of a product. Phase II is concerned with attracting first customers to the product beyond

early adopters and beta testers. Phase III is concerned with scaling the product further into new markets.

As illustrated in Figure 2, the reports suggest three distinct progression phases, each with a specific aim and symptoms for the anti-patterns. The plotted line represents the expected growth of a company. The forks denote alternative scenarios, that is, anti-patterns that could hinder the progression of the start-up. Along with the causes at the bottom of the figure, we denote how many reports were the basis for identifying each cause.

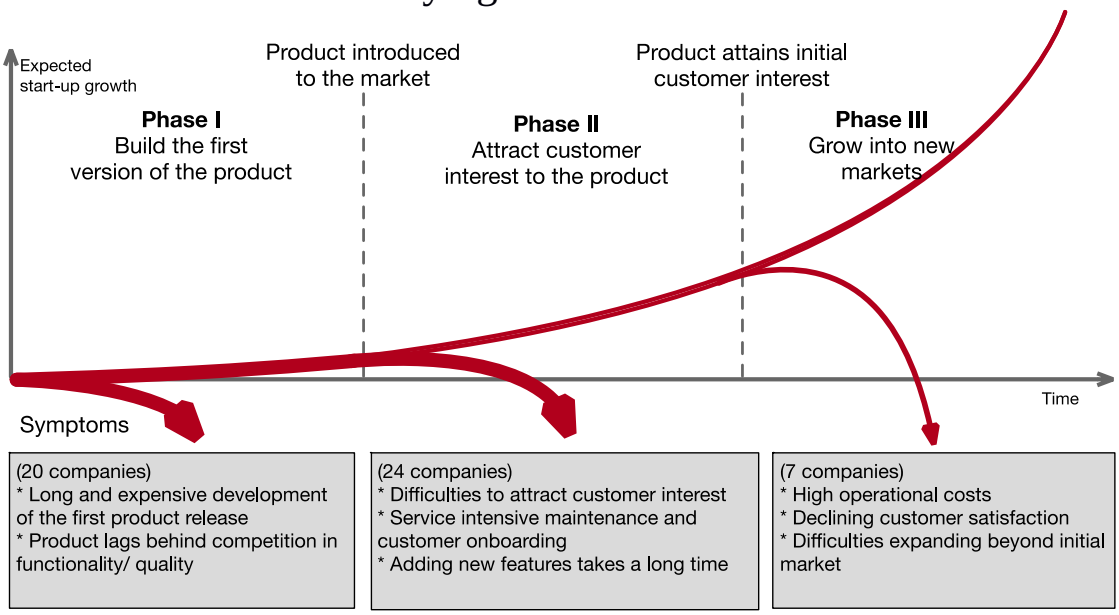

Figure 2, start-up milestones and symptoms for anti-patterns

Anti-pattern I: (not) releasing a market-worthy product

The reports suggest that one of the initial engineering challenges is to build and release the first version of a product: a minimum viable product, a bare-bones version of the product, good enough to be used for its main purpose and to test if there is enough customer interest to justify further investments in product development.

With the advantage of hindsight, the start-ups emphasize the importance of small and fast to develop first releases of their products. However, some start-ups reflect that it took them an overly long time to build the first version, stretched resources, and drained

motivation, which cost them a market opportunity. Symptoms and outcomes of anti-pattern I are illustrated in Figure 3.

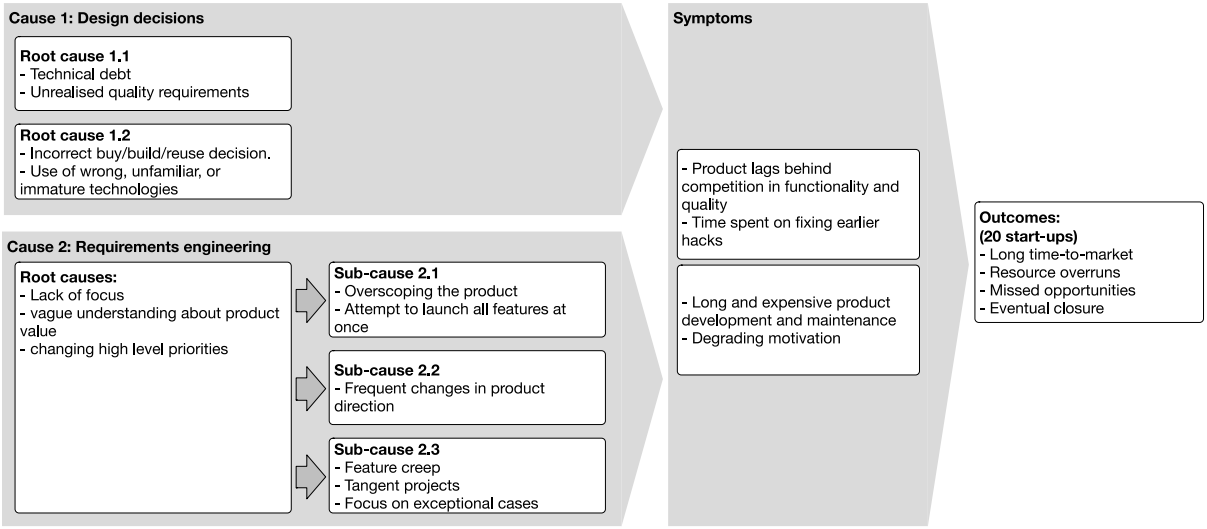

Figure 3, the breakdown of anti-pattern I

Reflections in the reports suggest two main causes for this anti-pattern. Cause 1, design decisions (see Figure 3), is concerned with selecting technologies and components that constitute the product. The choice lies between developing features in-house, utilizing open-source components, or buying certain functionality from a third party supplier [7]. In their reports, start-ups reflect that building commodity features that can be obtained by other means was a waste and significantly prolonged product development. Furthermore, the quality of in-house developed features was often significantly lower than of alternatives regarding functionality, performance, and maintainability.

A potential root cause for inadequate product quality is unrealized quality requirements. Quality requirements drive architecture decisions, including the selection of components and technologies which determine the level of external quality. Incorrectly assessing the required level of quality could lead to rework delaying the product release to market.

Cause 2 in Figure 3 is concerned with selecting features for the first product release. We identify three sub-causes all stemming from the same root cause – inadequacies in requirements engineering.

Overscoping is a consequence of poor requirements engineering and a potential cause for long and expensive development of the product, see sub-cause 2.1. Attempting to launch more than minimum set of features, attempting to implement all possible product use-cases beyond what is necessary for a product to be usable, generates potential waste and inflates the product scope beyond feasible. As in the case of Saaspire:

"The custom platform created a massive overhead on our development work. We ended up over-engineering our systems so they could support both today's and tomorrow's products."

The second sub-cause is general of product direction. With a vague understanding about customer needs, the solution is based on invented requirements and unclear priorities. As there is no real association between product features and needs of specific customers, requirements change whenever a new idea about an interesting feature comes to mind. Thus, the product direction changes frequently prolonging the software development process indefinitely. Furthermore, systematically abandoning already built features creates waste, damages teams' morale and drains motivation.

The third sub-cause is related to focusing efforts on key features. Some start-ups report that they always felt that the product lacks a feature thus kept adding more features. Furthermore, some companies report starting side projects along the primary product thus increasing the scope of the work substantially. As described by Disruptive Media:

"We kept building more features since we always felt that the service needs X because Flickr has it too or he/she said he needs that feature".

Analysis of the causes and remedies

Releasing the first version of their product is a test whether a start-up team can coordinate their work, and has enough skills to produce a meaningful output. Looking at how many start-ups encountered this anti-pattern (20, see Figure 2), getting the first product out could be a substantial challenge for many.

The practitioners argue that product time-to-market can be substantially reduced by combining existing open source or third-party components. Such components typically offer a set of related functionalities out-of-the-box, thus counterbalancing the extreme focus on the key features only [8]. However, a decision what to build, buy or what components to use depends on desired product qualities. Therefore, a clear understanding of what quality aspects are important and what is the expected quality level is a prerequisite [9].

A strategy to minimize overscoping and to shorten time-to-market is to invest in understanding the customer needs and to strip any non-critical features from the product [10]. In hindsight, the start-ups suggest scoping the first release to solve one simple problem for a customer. Earlier research suggests focussing on how a customer will be using the product rather than generating lists of feature ideas [11]

Aiming for fewer features will save development time and resources. Minimizing effort of developing the first release will also minimize the effort of rework in case some of the features turn out to be unsuccessful. That said, minimizing effort by scoping the product should not be confused with reducing effort by lowering the engineering quality. The goal should be to build fewer features at good quality.

Anti-pattern II: (not) attracting customers

The reports from start-up that have launched their products suggest that the next challenge is to bring the product to market. While coordinated marketing and sales activities are important to attract customers, the companies reflect that characteristics of the product,

stemming from earlier engineering decisions, hindered their marketing activities.

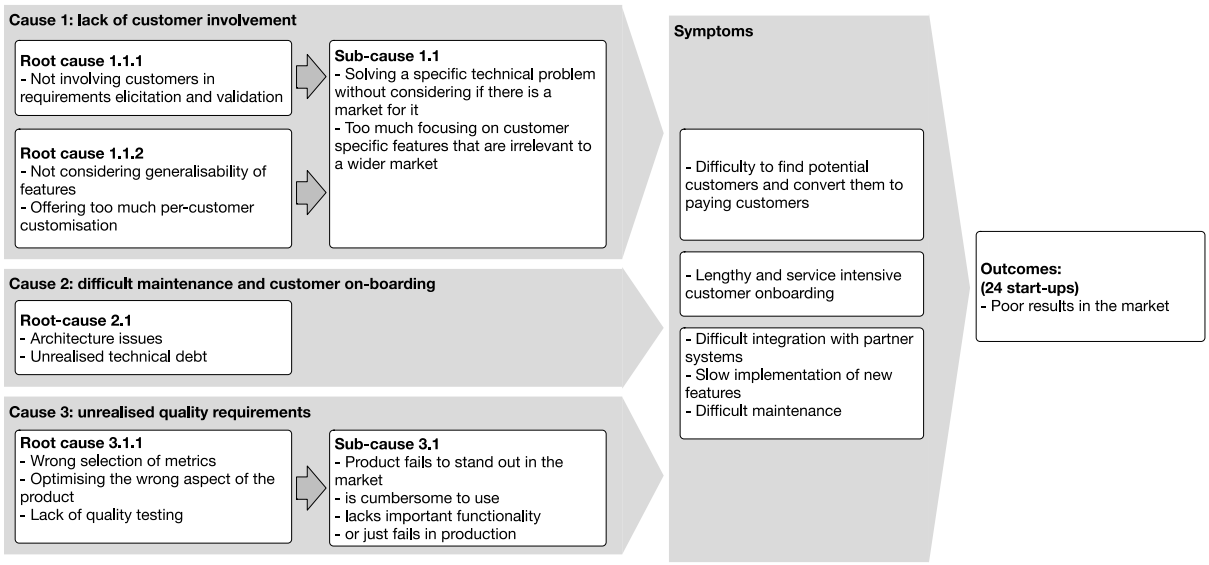

Figure 4, the breakdown of anti-pattern II

The analysis of the reports shows 3 potential causes, see Figure 4, that could hinder efforts to market the product. The most common symptom is the difficulty to find potential customers and convert them into paying customers. Reflections in the reports suggest that lack of customer involvement at the right level in developing the product (Cause I) largely contributes to marketing issues later. We identified two related scenarios. The first is that customers are not involved in requirements elicitation and validation, and the product the product turns out to be irrelevant to market, as described by one practitioner:

"We rarely had meaningful conversations with our target end-users. We huddled together to decide on ideas that sounded nice, built prototypes, put on our salesman hats, and didn't understand why we weren't closing deals".

The second is that some customers are involved, and the product is tailored to their specific needs and lacks generalizability to a wider customer base, see root-cause 1.1.2.

The companies reflect that after launching a minimum viable product, they rush to add more utility features to make the product more usable for customers. For example, by adding integrations with other systems, data import and export functionalities, support for other software and hardware platforms. The ability to add new features quickly demonstrates to potential customers that the product is evolving fast and any functional or quality shortcomings can be removed swiftly. However, the speed of adding new features could be reduced by product architecture issues and unrealized technical debt.

Another potential cause for poor results in the market is that the product does not stand out among the competition, is cumbersome to use, lacks important functionality, or is unreliable. As in a case of Flowtab,

"A big-bang product launch involving many partners and customers become a catastrophe when the app failed".

The root cause for these scenarios lies in unrealized quality requirements, lack of quality testing, or using the wrong metrics to optimize the wrong aspect of the product. Multiple start-ups reflect that their attempts to improve results in the market by tweaking user interfaces turned out ineffective because their product functionality lacked relevance. As described by Dinnr:

"The hypothesis of making the product visually more appealing and people will come back more often because you address their emotions didn't work out. You can't design your way out of a fundamental flaw."

Start-ups that have nailed features and quality of their minimum viable product report that slow customer onboarding is a significant hindrance to attracting customers at a desired rate. As experienced by Treehouse Logic:

"If a client confirms they want to work with us, they must hire an agency to handle the integration work. The reality is that the client expects full service and involving another party was too much for most of them."

Analysis of causes and remedies

Speed is one of the most praised start-up advantages over larger companies. However, a start-up can build itself into a corner by accumulating technical debt. Technical debt slows down development of new features, introduces hard to address quality issues, and degrades teams' morale [12]. Unwanted technical debt seeps in due to poor engineering decisions, sloppy individual attitudes, and insufficient coordination of engineering work. Unwanted technical debt can be prevented and removed by planned refactoring of the product and considering the impact on technical debt in all product related decisions.

An earlier study of customer complaints about mobile apps shows that reports functional errors, sluggishness, and unexpected app crashing are the most common among customer complaints [13]. Such results suggest, that engineers should pay extra attention of removing any such issues from the software before releasing it to the public.

Optimally, there is a pool of interested customers even before the product is launched. As suggested by several reports, such customers are a valuable source of requirements and assure that there is a market need for the product. The reports also suggest that this is not always the case, and the product launch is often the first time when a start-up attempts to reach out to customers. Therefore, to minimize the risk of market failure, start-ups should reach out to potential customers and involve them in developing the product.

At this phase, the response from markets could trigger a pivot in products' direction regarding features, used technologies, and targeted market segments. As shown by earlier research, flexibility to abandon ideas that do not work out and be prepared to explore new opportunities is crucial [14]. Thus, a lightweight product engineering process, such as Scrum, can support quick testing of feature ideas with little upfront work [15].

Anti-pattern III: A good problem to have

Start-ups that had attracted a substantial number of customers reflect on challenges concerning growth beyond their initial markets. At this stage, start-ups report declining customer satisfaction, difficulties expanding into new markets, and high operational costs hindering sustainability of the company. The outcomes of this anti-pattern are difficulties establishing a sustainable business model and growth issues.

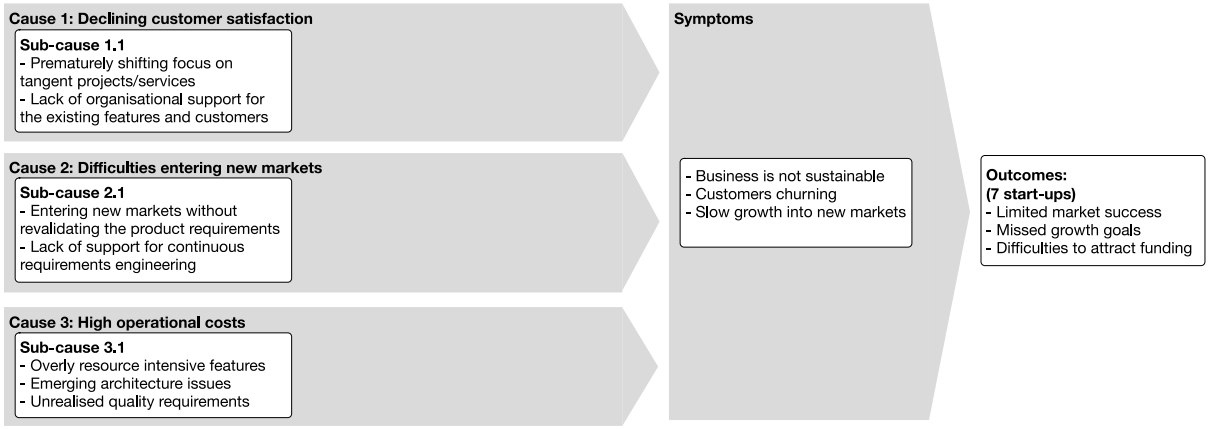

Figure 5, the breakdown of anti-pattern III

The reports discuss declining customer satisfaction, cause I, in association of start-ups undertaking tangent projects, and focusing efforts to enter new markets. Such activities could take attention and resources off existing customers, thus hindering their satisfaction. The root cause for such a scenario is the lack of an organizational framework supporting growing number of customers, continue to develop the product and to expand.

According to the reports, another common challenge is to expand into new markets. Part of the difficulty is to conduct sales and marketing activities over a geographical distance. However, another part of the challenge, cause II, is to identify what are the differences in customer requirements between markets and to tweak the product for the new market. As experienced first-hand by Dinnr:

"I thought that because businesses [like ours] have sprung up everywhere around Scandinavia, it will be a breeze to start something similar in London. [..] The lesson is: No, it doesn't have to work in country #2 only because it works in country #1."

Cause III, hindering start-ups at this stage is cope with high product operational and maintenance costs. The increasing costs are associated with extra workforce and computing power required to serve an increasing number of users. When costs grow larger than the actual value provided by the product, the business model of the company cannot be sustainable. As in case of Serendip, a start-up that used machine learning and big-data analysis to create customized playlists:

"The high costs of processing millions of posts every day, and serving relevant and engaging playlists to our users are really bigger than we can handle."

Analysis of causes and remedies

Experiencing challenges associated with growth into new markets is a good place to be for a start-up. However, as described in several reports, these challenges can bring the company down if not addressed timely.

High operational costs could be associated with inadequate requirements analysis and product design earlier. Specifically, not considering how much it would cost to run a feature and how much of customer value the feature creates. For instance, Everipix provided a freemium service to organize photos. However, costs of hosting pictures on Amazon Web Services turned to be higher than customers were ready to pay for the service. Thus, the company could not turn to profit and eventually went bankrupt. A potential remedy could have been to focus on features organizing the photos and to minimize the need of hosting the pictures.

When expanding into new market segments start-ups should make sure that the product is relevant in these new segments and to make necessary adjustments. This can be done by repeating requirements validation and aiming to discover new, the market

segment specific requirements. For example, user interfaces and user documentation may need translations.

Conclusions

The analysis of the start-up experience reports shows an association between shortcomings in software engineering practices and principles and start-up failures. The association highlights the importance of using good software engineering practices in start-ups. However, improving engineering practices does not necessarily mean spending more time and resources on engineering work. Timely and efficiently addressing root causes of potential problems down the road could save resources.

In essence, the focus should be on better engineering, not more of it. Thus saving time, resources and can greatly improving chances of successful product launch.

We present the anti-patterns along three distinct phases of start-up progression. Thus, putting the anti-patterns into temporal perspective. In combination, this creates a map that can support start-ups in improving their goals and prevent common pitfalls relevant to their specific progression stage.

For researchers, the anti-patterns along with the start-up progression phases present a framework that can be further developed by identifying new milestones, improved anti-patterns, and best engineering practices.

References

1. “The State of European tech 2017,” 2017. Online. Available: https://2017.stateofeuropeantech.com/.

2. S. Blank, “Why the Lean Start Up Changes Everything,” Harv. Bus. Rev., vol. 91, no. 5, p. 64, 2013.
3. C. Giardino, S. S. Bajwa, and X. Wang, “Key Challenges in Early-Stage Software Startups,” in Agile Processes, in Software Engineering, and Extreme Programming, 2015, vol. 212, pp. 52–63.
4. C. Giardino, M. Unterkalmsteiner, N. Paternoster, T. Gorschek, and P. Abrahamsson, “What Do We Know about Software Development in Startups?,” IEEE Softw., vol. 31, no. 5, pp. 28–32, Sep. 2014.
5. M. Crowne, “Why software product startups fail and what to do about it,” in Engineering Management Conference, 2002, pp. 338–343.
6. “CB Insights.” Online.. Available: https://www.cbinsights.com/blog/startup-failure-post-mortem/.
7. K. Petersen, D. Badampudi, S. Shah, K. Wnuk, T. Gorschek, E. Papatheocharous, J. Axelsson, S. Sentilles, I. Crnkovic, and A. Cicchetti, “Choosing Component Origins for Software Intensive Systems: In-house, COTS, OSS or Outsourcing? -- A Case Survey,” IEEE Trans. Softw. Eng., vol. 44, no. 2, pp. 237–261, 2017.
8. H. Munir, K. Wnuk, and P. Runeson, “Open innovation in software engineering : a systematic mapping study,” Empir. Softw. Eng., vol. 21, no. 2, pp. 684–723, 2016.
9. B. Regnell, R. B. Svensson, and T. Olsson, “Supporting roadmapping of quality requirements,” IEEE Softw., vol. 25, no. 2, pp. 42–47, 2008.
10. J. Melegati, A. Goldman, and S. Paulo, “Requirements Engineering in Software Startups : a Grounded Theory approach,” in 2nd International Workshop on Software Startups, Trondheim, Norway, 2016.
11. M. H. Cloyd, “Designing user-centered web applications in web time,” IEEE Softw., vol. 18, no. 1, pp. 62–69, 2001.
12. E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” J. Syst. Softw., vol. 86, no. 6, pp. 1498–1516, Jun. 2013.
13. H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, “What do mobile app users complain about?,” IEEE Softw., vol. 32, no. 3, pp. 70–77, 2015.
14. S. S. Bajwa, X. Wang, A. N. Duc, R. M. Chanin, R. Prikladnicki, L. B. Pompermaier, and P. Abrahamsson, “Start-Ups Must Be Ready to Pivot,” IEEE Softw., vol. 34, no. 3, pp. 18–22, 2017.
15. L. Rising and N. S. Janoff, “The Scrum software development process for small teams,” IEEE Softw., vol. 17, no. 4, pp. 26–32, 2000.

About the authors

Eriks Klotins is a PhD student of Software Engineering at Blekinge Institute of Technology (BTH). The focus of his thesis is software engineering practices in start-ups. He has over nine years of experience in managing software development projects ranging from large government IT systems to several start-ups projects.

Contact him by: eriks.klotins@bth.se

Michael Unterkalmsteiner received the BSc degree in applied computer science from Free University of Bozen-Bolzano in 2007, and the MSc and PhD degrees in software engineering from Blekinge Institute of Technology (BTH) in 2009 and 2015, respectively. He is a senior lecturer at BTH. His research interests include software repository mining, software measurement and testing, process improvement, and requirements engineering. He is a member of the IEEE. For more information or contact: www.lmsteiner.com, michael.unterkalmsteiner@bth.se

Tony Gorschek is a professor of Software Engineering at Blekinge Institute of Technology (BTH. He has over ten years' industrial experience as a CTO, senior executive consultant and engineer, but also as chief architect and product manager. In addition, he has built up five start-ups in fields ranging from logistics to internet based services. Contact him by: tony.gorschek@bth.se