The Developers Dilemma: Perfect Product Development or Fast Business Validation?

Henri Terho(✉), Sampo Suonsyrjä, and Kari Systä

Tampere University of Technology, P.O. Box 553, 33101 Tampere, Finland
{henri.terho,sampo.suonsyrja,kari.systa}@tut.fi

Abstract. To find a fast-track to profitability, a startup needs to streamline and speed up two vital processes – developing novel products and finding new markets for their products. These two goals are typically opposed to each other, business development requiring quick iteration and product development requiring focus on quality. This difference in mindsets, where the focus should be on the balance of quality to the business experimentation causes a conflicting environment for the developers to develop products. This problem is aggravated in a startup environment, where the reasons for product failure are not clear, increasing the frustrations felt by the developers. Clear ways to communicate the product goals and even successes between management and developers is needed to create an environment for success. This balancing act between quality and speed to achieve fast product iteration is the developers dilemma.

Keywords: Startups · Software development · Business development · Lean startup · Prototyping · Agile

1 Introduction

The success of a startup, or a potential company looking for a repeatable and scalable business model, is often related to the time it takes for the startup to develop their business model. [4,9] Consequently, the importance of fast iteration cycles is intensified, as the entire business model can be unclear or at least it remains under constant development [3,13].

Mastering this requires optimized techniques and methods for product and customer management [3]. An emerging choice for such a management method is the Lean Startup framework [4,9]. Products are tested through hypothesis driven iterations, where the success of the product is measured by actionable metrics. Moreover, when a product is deemed failed, a pivot is encouraged. As each iteration brings new knowledge, iteration speed is vital – Failing faster means also finding success faster. To meet the requirements of rapid course corrections in the business, i.e. pivoting, the product development process has to adapt to fast iterations [13].

In a startup environment the developers should be constantly aware that the software might become waste and therefore typical quality thinking can become difficult. This creates a situation where a software developer is pulled in two directions: should I follow the values of professional software development or work fast to support the constantly changing business directions?

This delicate balancing act between writing good quality software and spending as little resources as possible on a product that could be scrapped creates the *Developers Dilemma* that we explore further in this paper.

2 Background

The Lean Startup method is a popularized collection of best practices from multiple previous entrepreneurship theories such as Creation theory [1] and Bricolage [2]. With it, business development is seen as an iterative process of confirming business hypotheses with minimum viable products (MVPs) [9]. The method consists of iterative cycles of building, measuring, and learning.

Each cycle is typically linked with its own MVP, which is used to test the hypothesis of the current cycle. Based on these tests, the company either stays on the same path, building additional MVPs on top of the data gained from the first, or pivots their business plan to a new trajectory. Startups could even be said to be defined by their pivot making capabilities [12].

An MVP is a version of the product that enables a full turn of the build-measure-learn loop. It should contain the features that realize the unique value proposition of the software solution and little else. The idea is to cut out all non-essential features and leave just the core features of your application and the tools to enable learning [8].

As a software development method, producing MVPs is somewhat similar to prototyping. Prototyping approaches have been developed for situations where the work steps of a project cannot be clearly detailed before execution [10]. Prototyping incorporates many styles, such as iterative, rapid, evolutionary, throwaway incremental and mock up prototyping [7]. Stephen and Bates [11] define the prototype through two common characteristics:

1. The prototype enables a high degree of user evaluation which substantially affects requirements, specifications, or design.
2. The prototype initiates a learning process for users and developers of the system.

The first definition matches the MVP's aspect of user evaluation. The second definition matches the MVP cycle, where the MVP is designed to enable one cycle of the experimentation and produce learning with minimal development effort.

The prototypes can be split into throwaway and evolutionary prototypes. These two types are classified by their intended life cycle. Development based on evolutionary prototypes goes through sequences of re-design, re-implementation and re-evaluation without knowing the complete set of requirements beforehand [7]. Although the exact requirements for further development might be unclear, the implementation choice still matters as large parts of the code will be reused. On the contrary, throwaway prototypes will not be reused.

Comparing these two with MVPs, MVPs cover both aspects and possibilities of prototypes. In MVP development, the key idea is to validate the business case as fast as possible with a minimum set of features. If the experiment fails, the MVP should be thrown away, but in the case of a success it will be used again. However, it is not typically known beforehand if the MVP results in a throwaway prototype or in an evolutionary prototype. Therefore, the developers encounter a dilemma of writing code suitable for either throw-away or evolutionary prototyping.

3 Developers Dilemma

3.1 Environment

Modern software development, especially in a startup business where direction is changed rapidly, challenges the professional mind-set of software developers. These contradicting goals summarize the Developer's dilemma:

– As any professionals, software developers want to create artifacts. However, experimentation and pivoting that are implemented in many startups often lead to abandoning of software that did not receive positive feedback from the users or could not create an attractive business model fast enough.
– One of the main ways of showing your skills as a software developers is to write elegant code. However, when aiming at the minimum viability, developers should not refine their work in terms of quality and functionality to a level that they can be proud of.
– Developers can be used to creating prototypes that are thrown away, but this is not the case if they look at an MVP more as the first version of a final product, i.e. an evolutionary prototype. When such an MVP fails, this can cause a sense of loss for the developers who have poured their talent into the creation of the MVP.
– In the sense that an MVP is actually closer to being a tool for market research than an actual product, developers might spend too much effort on developing features that are ultimately not needed. The stress about features which are not essential is unneeded.

This environment of creating software, where the passion of the software developers might work against the goals of the company is typical in product development and startup environments. It creates a difficult environment to manage and develop software in.

To further elaborate, developers typically want to distinguish prototypes from a real products, but if the operation of the company is based on business experimentation, the choice is not known in advance.

3.2 Organizational View

The developers dilemma is a problem and a strategic question for the whole organization. The disjoint between the quality expectations of the software and the learning that the organization wants to achieve can cause problems.

This problem is illustrated in Fig. 1. The quality of the software and the size of the learning goals are placed on different axes. Typically the more complex the learning objectives, the larger product has to be built. For example, you are assessing solutions to complex networked problems or totally novel technology.

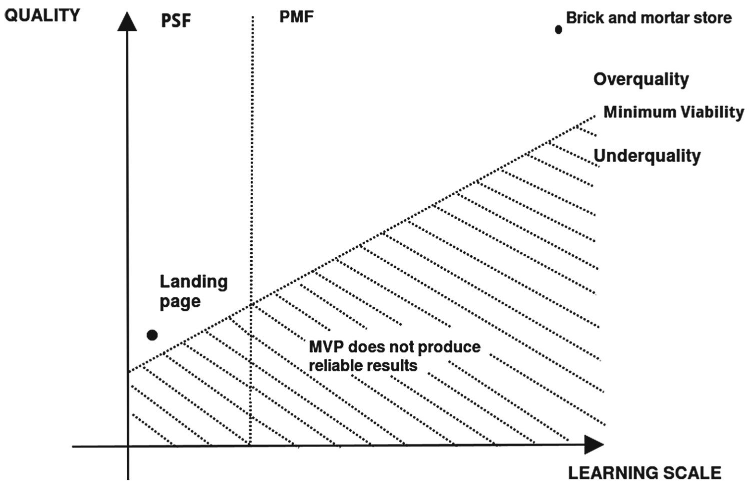

Fig. 1. Quality vs learning scale

The quality of this product has to be above a viable level to produce reliable results. This is illustrated by the Minimum Viability line on the graph. The quality of an MVP above the line is such that it does not interfere in assessing an business hypothesis. To further elaborate, the reason why the customers will not use or buy the MVP is that the they actually do not see it worthy – not because for example that the UI is horrible. Consequently, MVPs that are under the Minimum Viability line do not produce reliable results to the learning goals.

On the other hand, when the learning goals are still small and abstract, such as when an organization is just looking for the initial problem solution fit (PSF), MVPs can be smaller and of worse quality than when looking for product market fit (PMF). This is illustrated by the dotted vertical line.

In this case, we define quality to include not only typical quality attributes, but also the effort level it takes to develop an MVP such as additional features and keeping track of technical debt. So the challenge is to optimize quality as close to the Minimum Viability line as possible to reduce the amount of wasted quality and work. Also the two steps to product validation, PSF and PMF are marked to the chart to show the different phases of validation [4].

Research into the motivations of programmers show that competence, not experiments and financial incentives motivate them. [5] For the developers the way to show talent is to write quality code, so the natural tendency for quality in this is to go up. When such overquality prototypes are thrown away, the efforts of the developer are wasted and it results in frustration for the developers. The goal of the organization would be to make sure that the developers understand the goals of the prototype/MVP. The management has to message to the developers the point in the x-axis for the current product. The developers are then responsible for the positioning in the y-axis.

Some typical first products are positioned on the scale. Landing pages are small products which enable learning on a small scale, but do not require much quality. Typically the goal of the landing page MVP is to just assess the initial PSF. Brick and mortar stores as the first product on the other hand require a huge investment of resources to start and they may result in an immediate failure, if the clientele is not interested in the products of the store. The idea being that initially starting a brick-and-mortar store to asses your initial business hypothesis produces a high quality "product", but is a huge risk.

This positioning on the axes is at the heart of the developers dilemma and how the wanted position in these axes is messaged to the developers. The positioning on both axes is important to the success of the company, but the management typically forgets to take into account the quality tendencies and its psychological effects when deciding the goals.

3.3 Example

The issues that result from the developers dilemma in development can be better shown through an example:

A small startup is investigating their new hypothesis that people want to buy blood pressure measurement services. An MVP to test the validity of the idea is created. This MVP is a small website which enables customers to contact a person to come and measure their blood pressure. For the software development the user story in the backlog would be "As a user I want to fill in my contact details so the company can contact me to measure my blood pressure". In this first iteration, the experiment is to see if people want to even buy these services. The enabling MVP is the website that collects and stores contact information.

After the success of MVP, the company decides to experiment if they can expand the business by changing the contact form to a time reservation system. This way they can allocate personnel more efficiently. To facilitate this, a time reservation website is built as the second MVP.

At the third stage, the company has seen that there is demand for blood pressure measurements and decides to experiment if the people are willing to measure their blood pressure in predesignated locations. To test this, they develop a mobile application with GPS location to show the closest free blood measurement point. The new user story is: "As a user I want to be able to find out the nearest point where I can measure my blood pressure and reserve it." Here the experiment has expanded from a web form to a mobile application.

Again from the angle of the software developer, the development of the mobile application is a new production of a new stand alone application.

On the business development side, the continuous development of the theme progresses and builds upon the results from the previous application, expanding the scope of the project to one direction at a time. This however is not reflected even in the backlog items created. The actual goals of the company are not reflected by the user stories that the development is based upon. To improve upon this, the startup should have a backlog item that shows the goals of the current experiment, e.g. “We as a startup want to know if the customers want to come to our premises for measurement.”

This communication problem causes the developers to develop three different projects. First a web application contact form is developed. Next a reservation system is developed and third a separate GPS location application is built. These three separate projects do not have large amounts of overlap and also the two previous versions of the product have now been throwaway prototypes. On top of this, the developer has seen two of his products thrown out as waste.

Even though the company has achieved its goals, this feeling might not have been transferred to the developers. The feeling with the developers might even be that most of their work was wasted, even though it clearly contributed to the company goals. Similarly, if the developers have used huge amount of resources in refining for example the first contact form MVP and its back-end scalability, the organization can leave the developers without proper recognition as their efforts have done in vain in the eyes of the organization.

3.4 Lean Startup Difficulties

One of the most difficult things in the execution of Lean Startup’s Build-Measure-Learn cycle is understanding why an MVP actually fails. This assessment of product failures is a critical part of the Developer’s Dilemma. The Lean Startup tries to avoid the dilemma, however, by focusing on learning goals, not on development goals.

By developing different types of MVPs, a startup can split their learning goals into appropriately sized fragments. The different types of MVPs have been described e.g. in [9,13]. By sharing the vision of using many of them along the life cycle within the whole organization, we think that the risk of Developer’s Dilemma can be mitigated.

However, if the targets of developing an MVP are not made clear for the whole organization (including developers) Lean Startup will not help in resolving the Developer’s Dilemma. Therefore, the underlying communications problems and the way developers valuate their own work form the crux of the Developer’s Dilemma.

We propose that developers are intrinsically drawn to think that it is the lack of refining, missing features and things such as technical debt that make an MVP fail. If this opinion is given the most value, an unsetting can be created in which an MVP is refined forever or until the organization runs out of funds, but no one really understands (or accepts) why no user actually needs the product.

On the other hand, the development organization can be drawn to think that an idea and the related MVP do not have any business value because the MVP does not succeed straight away at a targeted level. In such a case, the organization might perceive the developed MVP as a perfect artifact to find out if a business problem is worth solving. This, however, is quite often not the case. Rather, MVPs usually need some refining to be able to produce reliable results for learning, if a business problem is actually worth solving. Again, this premature rejection of an MVP can create an unfavorable setting. In such, MVPs are thrown away before they are developed to their minimum viability.

We propose that the difficulty of balancing between these two unfavorable settings, can be a root cause for the Developer's Dilemma. On one hand, the technical development of MVPs that can produce reliable results is required and expected from the software developers. Also the developers naturally tend toward better quality because it is a way for them to show their skills. However, developing MVPs above the quality of minimum viability can be considered waste. Thus, developers need to understand thoroughly the level of quality they are expected to produce. On the other hand, the organization needs to have an amazing competence in chopping down the learning goals to appropriate size experiments. If these two premises do not exist in a case, the setup is ready for example for contradicting opinions on why an MVP fails, i.e. one of the result of the Developer's Dilemma.

3.5 Analysis of the Developer's Dilemma

This conflict between business-driven and technical-driven goals is not new. Similar dilemma exists in most Agile processes since the development should focus on the tasks of the biggest business value. This often leads to compromises in the architecture, and the maintenance of the architecture requires special care in Agile development [6]. In one way, Lean approaches partly amplify this since non-productive work is considered *waste* but by including principles like *build quality in* the Lean community has recognized the importance of professional development.

Many aspects of the developer's dilemma are known but the startup approach amplifies the developer dilemma since the process is driven by the business experiments. As in all organizations, the key element is communication and understanding of common goals. Based on these assumptions we propose several aspects that can be used to identify the developers dilemma:

– *Do the same people develop the software and business aspects of the software?* The easiest way to understand both sides of the problem is to work on both development and management roles. This allows for a larger perspective on the whole company. The management should be transparent and allow for opinions from both sides to be intermixed.
– *Is the software development outsourced?* Outsourcing the software development splits the goals of the company in two. The software company is responsible for delivering software and they have their own goals. The original business is left with just the business goals in mind and can optimize to fulfill

  those. For the outsourcing companies, the fulfillment of developer ethos might be easier because the clients bring the problems to them and the company does not have to do validation on those. Even if the product would fail on the business side, the software developers have met their goals.
- *What are the performance indicators that are used to measure the software development?* You get what you measure is an old mantra that still holds true. If the development is only measured in regard to quality or number of tickets completed the development work is separated from the business goals. The measures should be developed in such a way to encompass business and quality indicators.
- *If asked, do management and software development teams have the same goals for the product?* Do they see the same future for it, i.e. still in use after 6 months, profitable. Also what do the developers see as the core targets, optimizing performance, creating products quick? Does management share these goals?
- *How does the company handle the failures of the previous MVP?* Is it taken as a way to learn from failure and do the management and development see it in the same way. Are questions like was the UI good enough handled in the same light as was there a market for the product?

4 Conclusions and Future Work

In this paper, we have analyzed the mismatch and especially the Developers Dilemma between professionalism in software development and the needs of fast business development. Thus this problem is not just limited to lean startup, but also to other environments, where rapid product development is needed.

As as response, we propose a set of questions that can be used to analyze if the developers dilemma is a problem in your company, outlined in Subsect. 3.4. These questions allow for the initial recognition of possible problems between unified goals of the developers and the management. It seems that one core aspect to the developers dilemma might be the usage of different measurements to measure product success on the business development side and the software development side. If these two sides are brought closer together and both sides share a deeper understanding of the project goals the problem can be mitigated.

Although the idea comes from our personal experience and communication with a number of local startups, there is still work to be done in refining and validating the proposed concepts.

As future work, we want to analyze how startups can efficiently recognize the difference between throw-away and evolutionary parts of MVPs to assess the first solution. Similarly, we need to study to which extent the idea of experiments as top-level requirements is already used and how it affects development. Also other ways that companies use to handle failure and waste are of interest. Instructions on how to write and manage such requirements should be developed e.g. with action research methodology. These two methods could be used to help solve the dilemma.

The Developer's Dilemma is a mismatch between the software developer's professional ambitions and the startup's need for fast business experimentation. As not responding to the dilemma can have grave consequences for the company, the dilemma should be alleviated as well as possible to ensure the growth. This is not just a problem for the management, but a challenge for the whole company on how to communicate their core targets.

Acknowledgments. The authors wish to thank Digile's Need4Speed program (http://www.n4s.fi/) funded by the Finnish Funding Agency for Innovation Tekes (http://www.tekes.fi/en/tekes/) for its support for this research.

References

1. Alvarez, S.A., Barney, J.B.: Discovery and creation: alternative theories of entrepreneurial action. Strateg. Entrepreneurship J. **1**(1–2), 11–26 (2007)
2. Baker, T., Nelson, R.E.: Creating something from nothing: resource construction through entrepreneurial bricolage. Adm. Sci. Q. **50**(3), 329–366 (2005)
3. Blank, S.: The Four Steps to the Epiphany. K&S Ranch, Pescadero (2013)
4. Blank, S., Dorf, B.: The Startup Owner's Manual. K&S Ranch, Pescadero (2012)
5. Da Silva, F.Q., França, A.C.C.: Towards understanding the underlying structure of motivational factors for software engineers to guide the definition of motivational programs. J. Syst. Softw. **85**(2), 216–226 (2012)
6. Eloranta, V.P., Koskimies, K.: Aligning architecture knowledge management with scrum. In: Proceedings of the WICSA/ECSA 2012 Companion Volume. pp. 112–115. WICSA/ECSA 2012. ACM, New York (2012). doi:10.1145/2361999.2362023
7. Floyd, C.: A systematic look at prototyping. In: Budde, R., Kuhlenkamp, K., Mathiassen, L., Züllighoven, H. (eds.) Approaches to Prototyping, pp. 1–18. Springer, Heidelberg (1984). doi:10.1007/978-3-642-69796-8_1
8. Maurya, A.: Running lean: iterate from plan A to a plan that works. O'Reilly Media Inc., Sebastopol (2012)
9. Ries, E.: The Lean Startup. Penguin, New York (2011)
10. Sandor, C., Klinker, G.: A rapid prototyping software infrastructure for user interfaces in ubiquitous augmented reality. Pers. Ubiquit. Comput. **9**(3), 169–185 (2005)
11. Stephens, M., Bates, P.: Requirements engineering by prototyping: experiences in development of estimating system. Inf. Softw. Technol. **32**(4), 253–257 (1990)
12. Terho, H., Suonsyrjä, S., Jaaksi, A., Mikkonen, T., Kazman, R., Chen, H.M.: Lean startup meets software product lines: survival of the fittest or letting products bloom? (2015)
13. Terho, H., Suonsyrjä, S., Karisalo, A., Mikkonen, T.: Ways to cross the rubicon: pivoting in software startups. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) PROFES 2015. LNCS, vol. 9459, pp. 555–568. Springer, Heidelberg (2015). doi:10.1007/978-3-319-26844-6_41