Seven Patterns for Software Startups

JORGE MELEGATI, Universidade de Sao Paulo
ALFREDO GOLDMAN, Universidade de Sao Paulo

Startups are human organizations focused on innovation, generally, under lack of resources. Then, a highly uncertain environment is created that influences how software is developed. This set of patterns tries to capture the expertise found in literature and from practioners to help founders and employed developers to optimize software development process under the unique circumstances that a startup faces. We present seven new patterns: Practice agility, On giants' shoulders, Simple tools, Early validations, Empower, In the clouds and Celebrate failure and discuss the relations between them and other patterns already presented in literature.

Categories and Subject Descriptors: K.6.3 [**Computing Milieux**]: Management of computing and information systems—*Software Developement*

General Terms: Startups

Additional Key Words and Phrases: Software development, startup development, startup patterns

ACM Reference Format:

Melegati, J. and Goldman, A. 2015. Seven Patterns for Software Development Startups. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (October 2015), 11 pages.

1. INTRODUCTION

Startups are present in innovation context since the 1970s. Nevertheless, like agile methodologies, their suggested best practices are being discussed after being applied in industry. As a matter of fact, only after the "dot com bubble", at the beginning of years 2000s, which led several young companies to close and investments in the sector to fall abruptly, efforts to reach best practices on developing startups came to light like Customer Development [Blank 2006] and Lean Startup [Ries 2011].

This set of patterns seeks to formalize several practices described in literature to improve software development, specifically inside software startups, focusing on team mindset, limited resources handling and fast learning. They can be used by founders, software development managers and developers to create an environment to foster innovation. In section 2, we review works in literature related to the subject, in section 3, we describe a set of patterns and, finally, we present a final conclusion in section 4.

2. RELATED WORK

Besides patterns focusing directly on startups, some other pattern languages found in literature indicate good practices that could be used in this context, either with some adaptations or even the way that were proposed for mature teams or companies. Then, in this section, we start by discussing some patterns related to software development process in general, then we present works that are not directly related but could be used on startups like on enterprise architecture and design thinking. At last, we finally present the work on patterns for startups.

Author's address: J. Melegati, A. Goldman, Deparment of Computer Science, University of Sao Paulo, Rua do Matao, 1010, Cidade Universitaria, Sao Paulo, SP, 05508-090, Brazil; email: melegati@ime.usp.br, gold@ime.usp.br

Hecksel [2004] proposes a set of patterns on software development in general focusing on the following topics: methodology, methodology selection, team composition, management and project management. Several patterns presented could be used for problems in startups, some with minor modifications and others in the same way they are presented. In this set, some represent good advices to startups like *Utilize Tools Over Growth*, *Set Senior Developer Ratio* and *Breadth Over Depth*. Under methodology category, *Utilize Tools Over Growth* gives preference to use productivity tools "when there are low hanging fruits manual tasks that can provide large return on investments if automated". This is especially true in a startup since financial resources to hire are scarce. This pattern could use our *On giants' shoulders* and *Simple Tools* to select their productivity tools. *Set Senior Developer Ratio* foresees a trap that should be avoided. When hiring, startup founders tend to try to save money. Then they search for junior developers, however this pattern tell us to advocate for a minimum rate of 25% of senior developers up to 50%, since, the authors argue, citing Pragmatic Programmer from Dave Thomas, that "adding a Senior Developer to the project team can have up to a 20x productivity contribution to the team compared to adding another inexperienced team member". But too many senior developers is also bad: it will create the "all chiefs no indians" effect where expectations about decision making authority would not be satisfied for all. This pattern is under team composition category. From the same category, we can discuss the *Breadth Over Depth* pattern that advocates hiring team members that understand the "big picture on both project requirements and selected technologies" instead of outstanding developers that only master a single technology. In a startup, this makes total sense since everything, from processes to the software itself, have to be built and help from everyone is very useful. Because of the high uncertain environment, some members could leave the company and, if any of them is the only one who dominate a technology important for the development, the progress will slow down. These last two patterns are closed related to our *Empower* since to empower a team there must be members with higher experience and they should have a bigger picture of the business.

Since startups use more often agile methodologies since they are guided by the product, based on small teams and embrace changes instead of avoiding them [Taipale 2010; Coleman and O'Connor 2008], Coplien and Harrison book [2004] about organizational patterns for agile software development is an important work and brings several patterns applicable to startups. Some of them are closely related to patterns proposed on this paper: *Early Validations* is an alternative to *Surrogate Customers* but it is inspired by *Build Prototypes*; to employ *Empower* it is important to have a *Community of Trust* and is possible when *Developer controls process*.

Rodin et al. [2011] created a pattern language for release and deployment process because, according to the authors, changes in software systems must occur in a controlled and disciplined way. The language would help software delivery be controlled and disciplined in an agile environment. For instance, *Obtain certificate of originality* is essential when using *On giants' shoulders*.

Van Hilst and Fernandez [2010] analyze theories behind most used practices in process improvement, among them CMMI, Lean and Agile, extracting a set of patterns to approach the theories that support these methodologies, even if these theories are not well understood yet. The set is composed of five patterns (*Have a Plan*, *Copy What Works or Better Practices*, *Remove Waste or Flow*, *Consider All Factors or System Thinking* and *Incorporating Feedback and Learning*) and four anti-patterns (*And a Miracle Occurs*, *Buy a Silver Bullet*, *All Problems are Nails*, and *Solutions must be General*). Between these, we can highlight *Remove Waste* and *Incorporate Feedback and Learning* that are closed related to our *Pratice Agility*, *Early Validations* and *Failure can be a good thing*.

The next two works have no close relation with ours but they could have important lessons to software startups. Ernst [Ernst 2008] uses patterns approach to discuss the enterprise architecture problem (EA) that aim to align information technology area with company's business. This is very important for startups since both areas development are done simultaneously and, many times, they are still building a new market. Rikner and West [2010] argue that developers are designers since "they design everything from user interface to program architectures to algorithms" and, besides that, receive little education on design discipline so, a set of patterns that represent designers way of thinking, called design thinking, is presented.

Finally, some works on patterns focus on *startups*. Hokkanen and Leppanen [2015] propose patterns that validate

your hypothesis though user involvement. They should be used when doing *Early Validations*. Eloranta [2014] tackled the same problem that we are discussing, patterns for software startups. Although on the same subject, patterns identified are different from our work. For the patterns he already identified, *Develop only what is needed now* is important when doing *Early Validations*.

3. PATTERNS

Figure 1 summarizes relations between patterns described in this paper. It also describes how these patterns connects to those found on literature. When you decide to create a startup company, generally, you have a product in mind. For a startup, its products or service represents an innovation: a new product, service, a product already established in a new market, a new technology or etc. But product acceptance uncertainty also represent a threat to company: you do not know if consumers (people or companies) will be interested in paying for it. You could spend a lot of time and money to build something that will not be what your market needs. In this scenario, it is better to use *Early Validations* and test your hypothesis about your potential market. Then, you will have highly changing environment: your hypothesis could be wrong and you must change your product development very fast. So, it is better to *Practice Agility* than to use traditional software development methodologies. To *Practice Agility*, it is better to *Empower* and it also facilitates *Early Validations* since more people will be able to do it. Besides that, through *Early Validations*, several hypothesis could be found to be false and a demotivating environment could be formed. So, it is important to remember that *Failure can be a good thing*.

You still do not know what to do, you are testing your hypothesis but there are already other companies building similar products. Besides that, you have to build, test and learn fast because you might be running out of money and your team is getting frustrated since your product is not getting traction. To gain speed, you cannot reinvent the wheel but build *on giants' shoulders*, that is, use already developed technologies and frameworks. You should also use infrastructure and/or supporting tools *In the clouds* what will free resources to build what it really matters to your company. You should also prefer *Simple Tools* to gain speed. The use of these three patterns will speed up your startup but it will represent dependency on others, like libraries and frameworks communities and cloud vendors.

3.1 Early Validations

Context. Before the "dot-com bubble" burst, the development of a startup followed a traditional project development process: find a problem, project a solution, build it and launch your product [Blank 2006]. This whole cycle could take years to be completed and, finally, get a real user feedback and any revenue. This approach was very criticized [Blank 2006] by not detecting what the user really wanted, that is, the company spent money and time building something that nobody wanted. Since then, several methodologies have risen, like Customer Development [Blank 2006], Lean Startup [Ries 2011] and Design Thinkign [Leavy 2010], that focus on user feedback and fast iterations.

Problem. Startups build brand-new products for incipient (in some cases, nonexistent) markets and they do not know exactly what to do or if what they are building is what the user is expecting. Technology related questions also emerge: which programming language should be used, which framework, which library?

Forces. In this innovative and competitive scenario, there is rush to build the company product and the startup development team could be using emergent technologies which they are not familiar with. They do not know if it is the best suit for their needs either. Wrong technology decisions can also create frustration on the team by the effect of wasting time on unnecessary features.

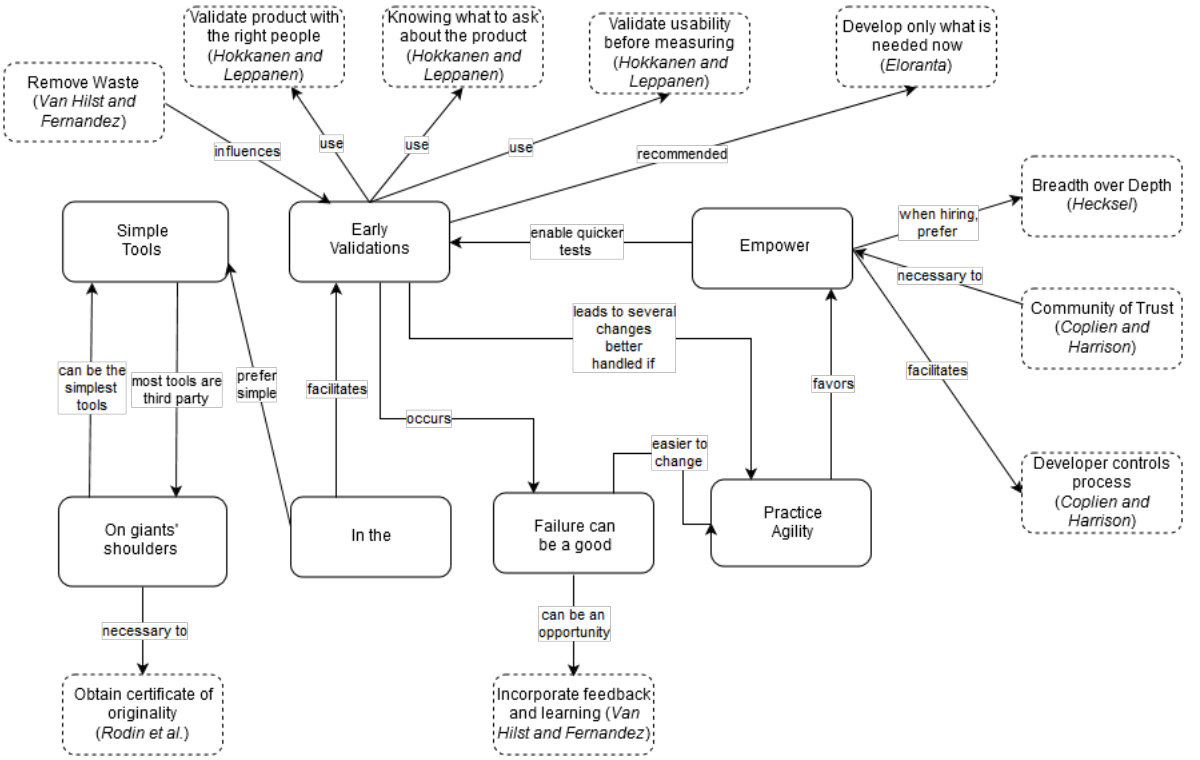

Fig. 1. Relation between patterns

Solution. Every assumption even one that seems obvious should be confirmed. Take your assumption as a hypothesis and develop an experiment that allow to test it. The experiment planning must include metrics to evaluate your hypothesis and it also try to isolate other variables that can influence the result. At the end of the experiment, you should be able to validate or reject your initial hypothesis and take actions based on your acquired knowledge. Your experiment could be a prototype or a Minimum Viable Product ([Ries 2011]) for product related hypothesis or a side project for technology related hypothesis.

Consequences. Testing your assumptions early prevents your company from wasting a lot of time and money on something that will not be used by your user or will not bring other customers or a technology that is not well suited for the company needs. Instead of it, a little effort is spent to *test* something. But experiments should be planned with criterion. A not well design experiment could lead to wrong conclusions that can take to waste of time and money or to lose a business opportunity.

Known Uses. This pattern could be used in different areas of a startup like pointed out by Ries in his book Lean Startup [Ries 2011]. Besides using it to test hypothesis in what the startup product should be, an approach already described by Blank [2006], he extends this experimentation view to marketing and business areas of a startups.

Related patterns. Early Validations leads to several changes of software requirements and this is easier to handle when practice agilily (*Practice Agility*). This pattern can be seen as an alternative to Coplien and

Harrison's *Surrogate Customer* since they address the same problem: lack of customer presence but *Early Validations* seems to be a better option since a team member will hardly know what the user really wants (Ries [2011] presents good examples about it). Nevertheless, another pattern proposed on the same work was *Build Prototypes* and *Early Validations* has the same core but focus on hypothesis.

3.2 Practice agility

Context. During product development in your startup, you will have several points that you are not sure if what you are building is not the right thing. Then, you have to verify your hypothesis through *Early Validations*. As a result, you could change what is being built. In this context, since requirements change rapidly, traditional software development techniques are not adequate.

Problem. Traditional software development methodologies like waterfall are based on long planning and designing phases. That could slow down customer feedback or the response to a plan change. Besides that, it takes a longer time to get customer feedback to make *Early Validations*. For a startup to have an organized software development, it will need a methodology that supports change instead of avoiding it.

Forces. The following forces are present:

—A startup, by being in an innovative and uncertain environment, will have several changes during its product development, that is, software requirements will be volatile;

—the team must respond quickly to cope with customers expectations and competitors;

—several startups have inexperienced teams, so it could be tempting to not follow any methodologies nor principle and just do what each member wants.

Solution. Develop an agile mindset within your company, use agile practices. Agile methodologies, like XP, Scrum, Crystal and others, emerged from the criticism over traditional software management methodologies. As stated by Highsmith and Cockburn [2001], "agility, ultimately, is about creating and responding to change". In 2001, a group of authors of these new methodologies and others important figures in the area got together and wrote the Agile Manifesto [Beck et al. 2001], based on the following principles:

Individuals and interactions over processes and tools: cumbersome processes and tools could slow down responses to changes, prefer improve interactions between members and teams to favor creativity and speed.

Working software over comprehensive documentation: through working software, your startup will be trusted by stakeholders like customers and investors (that is very important to a young company) and a cumbersome documentation will not bring this and will slow down software development. Prefer a lighter documentation covering only architecture and software major points.

Customer collaboration over contract negotiation: for a software startup, most of the time, customer collaboration is made through the analysis of software use but it can also use interviews or surveys.

Responding to change over following a plan: this point is totally true in a startup environment since the plan is changing all the time and responsiveness is very important.

Software development startups, generally, are subject to product changes and these should be implemented quickly, so agile practices should be used, except when specific conditions related to certification processes are required like in pharmaceutical or defense markets.

Consequences. Through agile methodologies your company will have a better response time to changes in your market or other environmental changes. It should be avoided to take agile as another process that should

be followed strictly what could remove its advantages. For instance, if a two-week sprint is used and on the second day it is discovered that what is being built is not the right thing, everything should be stopped to rethink the strategy. The internal communication improvement could create a consensus environment taking to a false sensation of doing the right thing even without checking the user wills.

Known Uses. Da Silva et al. [2005] reports the adoption of XP in a brazilian *startup*. The author concludes that "XP helped [them] adapt quickly to the constant changes in the economic reality of a developing country". Ambler [2002] uses the agile mindset to propose a new way to model an architecture based on his work in two startups. The first author is a software developer in a brazilian software startup and his team implement several agile practices like, for instance, fixed length sprints and collective code ownership. As a result, the team has a good responsiveness to change and a very low software downtime besides being small.

Related patterns. This pattern makes easier to change development directions caused by *Early Validations*. In an agile context, *Empower* is favored and more team members will think about business issues.

3.3 On giants' shoulders

Context. Startups build brand-new products disrupting some market or creating new ones. Generally, to develop all necessary software for the product (and tools needed to assist company operations) from scratch will be an unfeasible task.

Problem. Startups need software solutions, that support their software development, such as libraries and frameworks, of good quality and support and do not have human resources to develop them in-house either. These software resources should be easily and quickly obtained to not slow down the product development and, most of the times, it should run on several platforms to make wider the target audience.

Forces. Startups should make products of good quality to win the market which is composed of demanding clients expecting new products very often without flaws and available in the platform they have. Meanwhile, startups struggle to survive under lack of resources.

Solution. Startups should use libraries or other third party software to build their project so their development team can focus on what is important: build something to learn. It must be avoided the culture of reinventing the wheel. Open source solutions are good options since they can be obtained within few clicks, without bureaucracy, without initial monetary investment and, if well chosen, can have very good support through active communities. Although lack of resources could repel it, maybe spend some money for some library or tool already built, could be an wiser choice than build it in-house. Generally, startups use several thirdy party software, specially open source, but in some cases, caution is advised. Areas extremely sensible to flaws like health related could have strict restrictions about code (maybe some kind of certification is needed). When using open source, another concern would be licensing: some projects have strict licenses that prevent its code from use on a commercial license, this does not prevent the use of binary forms, although.

Consequences. Using third party solutions wisely will decrease your company time-to-market and the time to run a build-learn-measure cycle. But your company will be vulnerable to some risks: the development of a software can be discontinued or changed in a such way that would not be useful. In these cases, if it is very tied to the product core, it could lead to a slow down on your own company or even prevent something from

working. So, choose carefully where you should use third party tools/libraries and their providers.

Known Uses. Wall [2001] tells how open source software was used to handle low budget for his company, Yozons, which product would be a process for incorporating public-key cryptography into any document. Since the money to build the product would come from the founders, the so-called "bootstrapping", they did not have money to buy commercial tools. They used open source solutions through all their software stack: operating system, database system management and web server. Although his initial intent was to save money, Wall recognizes how good the solutions were: "open source solutions may be more than adequate and often run better than their commercial counterparts". He also concludes that these solutions have excellent support networks that can help when something goes wrong. Wood [2005] presents the use of open source on a company he helped to craete. Besides using it, they decided to release some of its product as open source. The product is described as "for the enterprise middleware market" that implemented some standards of the World Wide Web Consortium (W3C). The project part related to these standards were released as open source and the company was also involved and standards discussions. All this made the company famous and made it a natural market leader.

Related patterns. Sometimes, *Simple tools* are the best options when choosing third party solutions. It is important to be careful and *Obtain certificate of originality* to prevent license problems.

3.4 Simple tools

Context. Startups live with lack of resources, human and financial, then tools used for process management and the software development itself (e.g., CASE tools), cannot be complex as in consolidated companies because they demand more investment and time to implement.

Problem. Lack of human and financial resources and time prevent a startup to use complex tools that could leverage their development productivity.

Forces. Startups have to respond quickly to changes in their market and must use their resources the most optimized way, this lack of resources discourages the use complex productivity tools. Nevertheless, more elaborated tools could leverage team productivity and could free resources to develop other functions.

Solution. Prefer simple tools for both process management and internal communication, like white boards, cards and general use instant messaging software, and for software development like simple software environments over more complex tools (for instance, CASE tools). But, always appraise if a more complex tool could leverage your team efficiency without committing a major part of your budget.

Consequences. With simpler tools, less time will be used for training or learning how to use something, then more time will be left to focus on what is important: learn about the market and build the product. In the long run, simple tools may be changed because they will not be able to handle more complex processes and/or requirements what could take some time to implement.

Known Uses. Ambler [2002] cite a *startup* company that he worked with that used white-boards on modelling sessions what make them "achieve the benefits of modeling without incurring the costs of writing and maintaining cumbersome documentation".

Related patterns. On giants' shoulders: several tools used by startups are thirdy party.

3.5 Empower

Context. Startups are, generally, inserted in a highly uncertain environment where requirements could change very fast. Respose time is intrinsically related to how decisions are made inside the organization and how these are implemented. In the meantime, these companies generally contract younger professionals since they represent lower cost and could adapt easier to a high demand but casual environment but they have less experience.

Problem. Modifications on products and/or strategies are often necessary and these modifications should be done quickly. A traditional hierarchy which separates managers who take decisions from employees who implement them could lead to high response times to those changes. Besides that, if more people have autonomy to create, run, analyze experiments without managers approval, more *Early Validations* can be done.

Forces. The application of this pattern should balance two forces: startups should empower their team members to respond quicker to market changes since, with more power and less bureaucracy, they can do more by themselves, taking decisions and, in last instance, learning. Nevertheless, given the lack of experience, some important errors could happen like inappropriate actions. This could be minimized by a careful team members selection focusing on finding "talents".

Solution. Cut bureaucracy and empower team members. They could be able to take some decisions by their own, leveraging company responsiveness. Besides that, this culture creates a better work environment increasing productivity and quality.

Consequences. More independent team members creates a better atmosphere in the company and people work better in this environment but, if not well implemented, could lead to problems. A lack of understanding about what motivate this culture could take people to do what they want to, instead of what should be done, threatening the whole company. Besides that, when not so interesting things must be done, some members could feel demotivated.

Known Uses. May [2012] describes lessons learned by her while developing a mobile app called Minidates inside a *startup*. One of the practices to optimize a team, she enforced that "must be able to trust each of your team members to do his/her job fully, completely, very well and independently".

Related patterns. This pattern enables quicker tests for *Early validations* and it is favored by the use of *Practice agility*. Again, we can relate this to patterns of Coplien and Harrison: it is necessary a *Community of Trust* and the *Developer controls process* as the authors stated: "successful organizations work in a organic way with a minimum of centralized control."

3.6 In the clouds

Context. The emergence of solutions to outsource technology infrastructure, also known as "cloud computing", where is possible, for instance, to run servers with few clicks, store as much data as you want and pay proportionally to your use, made the effort to build a company technology infrastructure decrease and, also, its cost.

In this category, we can cite:

IaaS (Infrastructure as a Service). Vendors that provide infrastructure (servers, data storage).

PaaS (Platform as a Service). A service that handle all infrastructure to run determined application. The common example is a solution that you can upload your code for a web site and the solution will create all necessary servers to run it and will handle increases on traffic, creating more servers if necessary, for example.

SaaS (Software as a Service). Instead of acquiring licenses and installing them on your own servers, softwares are provided through web. For instance, code repositories or business inteligence tools.

Problem. Most of software development startups depends on infrastructure or other information technology tools to build or distribute their product: run highly available web servers, store a lot of data and keep its backup, retain its code history, etc. But building an in-house computing center is expensive and complex, demanding people and expertise to handle common problems like backup, failing machines, etc.

Forces. Cloud computing is an opportunity to accelerate company creation and to save resources (again, human and financial). It also facilitates scale your company as changing size, for instance, of a machine, is as easy as a click. But it could be not a good solution for other stakeholders. For example, startups operating, or whose clients operate, on information sensitive markets like pharmaceuticals or biotechnology could prefer have control of data. Even investors could not feel safe having market sensitive information in the cloud. A possible minimization of this threat should be a careful vendor selection, that is, prefer well-known vendors with good reputation.

Solution. Do not spend time and money building your own technology infrastructure. Take advantage of cloud services as much as you can from compute to store data eliminating the overhead of managing this company area.

Consequences. Using cloud solutions, your startup will be able to build faster and, then, learn faster. But as you grow, you can be tied to a vendor and have some limitations because of this vendor's solution capabilities. For instance, you may need some solution or a customization not available. Besides that, when your business get bigger, you can end with huge bills that would be smaller if you had your own infrastructure.

Known Uses. This pattern could be applied for any startup that needs some kind of information technology infrastructure since high dependent on servers like internet web sites to companies using IT only to support some kind of activity like code repository for a mobile app or even accounting software.

Related patterns. When choosing a vendor to implement this pattern, one must remember to use *Simple tools*, that is, vendors that have easier-to-use interfaces should be favored. The cost and easy-to-use of these tools also facilitate experiments for *Early Validations*.

3.7 Failure can be a good thing

Context. Startups develop new products with low resources and, usually, with not experienced teams. They do not know a lot about several areas of the company from business to technology. Using *Early Validations* could

try to solve this problem but leads to a lot of “perceived failures”, that is, still part of the process, a hypothesis proven wrong could demotivate people involved or even the whole team. Besides that, as stated in literature, most of startups fail [Blank 2006; Ries 2011; Paternoster et al. 2014].

Problem. In the context of uncertainty and inexperience, errors are common and failure is almost certain. How to cope with these failures, not lose faith and continue building a company?

Forces. Failures are an important mean to reach the learning goal of a startup but they could lead to demotivation issues in the team and also others stakeholders like consumers and investors.

Solution. Take failures naturally and use them to learn more about the business, the product or the technology. Try to understand why others startups failed and how to not make the same mistakes again. This point is well explored by Cotterill [2011]. Sometimes, a radical change of what the company is building is necessary, this is called a pivot by Ries [2011].

Consequences. Having this mindset, your team will be more resilient to failure, recovering faster what it is very important to make the necesssary adjustments to keep the business going. But precaution is advisable: people can get used too much with failures and not try to learn anymore from them. Without learning, not enough changes are made towards a better fit to the market.

Known Uses. Ries [2011] tells the story about his company IMVU. In the beginning, they created a product to handle several instant messages networks in only one software. After releasing it and not getting the expected number of users, they performed an experiment and realized that this was not what the users wanted. Users, actually, liked to have several instant messages software to split their social relations. This, definitely, was a failure. But this showed more insights about their target users and helped them to come up with their final product: a kind of game that emulates a parallel world where users control their own characters.

Related patterns. Failures are inherent when using *Early Validations* and when using *Practice agility* it is easier to change directions needed to handle these failures. It is a way to *Incorporate feedback and learning*.

4. CONCLUSION

A list of known patterns for startups that develop software was presented. In the case of *Early Validations*, the use of already know techniques of hypothesis test on startups was brought to technology related problems like choosing a programming language. The patterns *Practice Agility*, *Open source solutions* and *Simple Tools* emerged from literature research. *Empower* and *Celebrate Failure*, like *Early Validations*, were inspired by agile methodologies (related to *Practice Agility*) and well known methodologies of startup development like Customer Development [Blank 2006] and Lean Startup [Ries 2011]. Finally, this set was not intended to fulfil the wide range of problems related to creating a new company but aims to foster the discussion that, in a future work, can lead to a broader pattern language to guide entrepreneurs.

ACKNOWLEDGMENT

First of all, we would like to thank Michael Weiss for being our shepherd and helping us to give us a direction on this paper. A first version of this paper was presented at Writing Group of 10th Latin American Conference on Pattern Language of Programs (SugarLoafPLoP 2014). The authors would like to thank the participants of the

workshop including Fabio Kon, Joseph Yoder, Takashi Iba, Arisa Kamada and Masafumi Nagai. Last but not least we would like to thank our writer's workshop group at PLoP, Christian Kohls, Daniel Cukier, Mary Curtin, Philipp Bachmann and Russ Rubis.

REFERENCES

Scott W. Ambler. 2002. Lessons in agility from internet-based development. *IEEE Software* 19, 2 (2002), 66–73. DOI:http://dx.doi.org/10.1109/52.991334

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. 2001. Manifesto for Agile Software Development. (2001). http://www.agilemanifesto.org/

S. Blank. 2006. *The Four Steps to the Epiphany: Successful Strategies for Products that Win*. Lulu Enterprises Incorporated.

Gerry Coleman and Rory V. O'Connor. 2008. An investigation into software development process formation in software start-ups. *Journal of Enterprise Information Management* 21, 6 (2008), 633–648. DOI:http://dx.doi.org/10.1108/17410390810911221

James O. Coplien and Neil B. Harrison. 2004. *Organizational Patterns of Agile Software Development*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Keith Cotterill. 2011. Entrepreneurs' response to failure in early stage technology ventures: A comparative study. *Proceedings of the 1st International Technology Management Conference, ITMC 2011* (2011), 29–39. DOI:http://dx.doi.org/10.1109/ITMC.2011.5995927

Alexandre Freire Da Silva, Fábio Kon, Cicero Torteli, and Alexandre Freire. 2005. XP south of the equator: An eXPerience implementing XP in Brazil. In *Lecture Notes in Computer Science*, Vol. 3556. 10–18. DOI:http://dx.doi.org/10.1007/11499053_2

Veli-Pekka Eloranta. 2014. Towards a pattern language for software start-ups. In *Proceedings of the 19th European Conference on Pattern Languages of Programs*. ACM, 24.

Alexander M. Ernst. 2008. Enterprise architecture management patterns. *Proceedings of the 15th Conference on Pattern Languages of Programs - PLoP '08* (2008), 1. DOI:http://dx.doi.org/10.1145/1753196.1753205

David Hecksel. 2004. Software Development Patterns. In *Proceedings of 11th Pattern Languages of Program Conference*.

Jim Highsmith and Alistair Cockburn. 2001. Agile software development: The business of innovation. *Computer* 34 (2001), 120–122. DOI:http://dx.doi.org/10.1109/2.947100

Laura Hokkanen and Marko Leppanen. 2015. Three Patterns for User Involvement in Startups. In *Accepted to 20th European Conference on Pattern Languages of Programs*.

Brian Leavy. 2010. Design thinking âĂŞ a new mental model of value innovation. *Strategy & Leadership* 38, 3 (2010), 5–14. DOI:http://dx.doi.org/10.1108/10878571011042050

Beverly May. 2012. Applying lean startup: An experience report - Lean & lean UX by a UX veteran: Lessons learned in creating & launching a complex consumer app. *Proceedings - 2012 Agile Conference, Agile 2012* (2012), 141–147. DOI:http://dx.doi.org/10.1109/Agile.2012.18

Nicolò Paternoster, Carmine Giardino, Michael Unterkalmsteiner, Tony Gorschek, and Pekka Abrahamsson. 2014. Software development in startup companies: A systematic mapping study. *Information and Software Technology* (April 2014). DOI:http://dx.doi.org/10.1016/j.infsof.2014.04.014

Rebecca Rickner and Dave West. 2010. Design Thinking Patterns. *Proceedings of the 17th Conference on Pattern Languages of Programs - PLoP '10* (2010).

Eric Ries. 2011. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*. Crown Business. http://books.google.com.br/books?id=r9x-OXdzpPcC

Rick Rodin, Jon Leet, Maria Azua, and Dwight Bygrave. 2011. A pattern language for release and deployment management. *Proceedings of the 18th Conference on Pattern Languages of Programs - PLoP '11* (2011), 1–13. DOI:http://dx.doi.org/10.1145/2578903.2579147

Marko Taipale. 2010. Huitale–a story of a Finnish lean startup. In *Lean Enterprise Software and Systems*. Springer, 111–114.

Michael Van Hilst and Eduardo B Fernandez. 2010. A Pattern System of Underlying Theories for Process Improvement. In *Proceedings of the 17th Conference on Pattern Languages of Programs*. 8:1—-8:24. DOI:http://dx.doi.org/10.1145/2493288.2493296

David AE Wall. 2001. Using open source for a profitable startup. *Computer* 34, 12 (2001), 158–160.

David Wood. 2005. Open Source Software Strategies for Venture-Funded Startups. *MIND Laboratory, University of Maryland, Tech. Rep. TR-MS1287* (2005).

Received May 2015; revised September 2015; accepted January 2015