Creating Minimum Viable Products in Industry-Academia Collaborations

Jürgen Münch[1], Fabian Fagerholm[1], Patrik Johnson[1], Janne Pirttilahti[2], Juha Torkkel[2] and Janne Järvinen[2]

[1] Department of Computer Science, University of Helsinki,
P.O. Box 68, FI-00014 University of Helsinki, Finland,
{juergen.muench,fabian.fagerholm,patrik.johnson}@cs.helsinki.fi
[2] F-Secure Corporation,
P.O. Box 24, 00181 Helsinki, Finland,
{janne.pirttilahti,juha.torkkel,janne.jarvinen}@f-secure.com

Abstract. Customer value determines how products and services succeed in the marketplace. Early assessment of customer value is important for software startups, spin-off companies, and new product development in existing companies. Software technology often influences customer value and typically defines the main competitive advantage in both entrepreneurial and intrapreneurial settings. Value-related feedback from real customers is needed during software development and maintenance, and decision-making should be increasingly based on empirical evidence acquired through experiments. Getting such value-related feedback usually requires a so-called minimum viable product (MVP), i.e., an artefact that may be incomplete in functionality or quality, but displays characteristics that allows determining its customer value. In this article we report on a case study which used industry-academia collaboration for creating such an MVP. Our goal was to identify strengths and weaknesses of such an approach to creating MVPs while providing practical recommendations for improvement. The process followed in the case study was found to be very suitable for creating MVPs, reducing company-specific risks when testing customer-value, and advancing university education.

Key words: Minimum viable product; prototyping; software start-ups; entrepreneurship; intrapreneurship; Lean Startup; Software Factory; case study

1 Introduction

Software engineering experimentation aims at advancing the knowledge on how software development processes and methods in specific environments impact results [1]. According to Basili et al., experimentation is performed to help us better evaluate, predict, understand, control, and improve the software development process and product [2]. As with any other experimental procedure, experimentation in software engineering follows a cycle of building models for

development processes or products, defining and testing related hypotheses, and refining models and hypotheses based on experimental results.

Traditionally, the main focus of software engineering experimentation has been on the technical and managerial aspects; the developer and project manager perspectives have been emphasized. However, since software engineering is a field which is often associated with innovative high technology, a vibrant global community of entrepreneurs, and a customer base with increasing expectations, it should also take into account the customer-perceived value of its products and services. While customer value may be simply defined as "whatever the customer is willing to pay for", it is not simple to assess or measure this concept while designing a product or service. In software projects, multiple stakeholders may have several different needs that affect how they perceive the value of the end product [3]. Rather than attempting to fix a single assessment or measure of customer value, we emphasize the importance of a process of experimentation and learning which allows empirical discovery of customer value in a specific context, guided by analytically derived hypotheses.

This perspective is also important in software engineering education. Students are entering a field which demands not only programming skills, but also the ability to dynamically consider customer value. They should be equipped with suitable knowledge of how to participate in software projects where value considerations drive project activities. It is particularly important that students are exposed to realistic, current problem settings and learn to analyse and understand value-related experiments.

Developing innovative software technology requires early testing of customer-related hypotheses. To make this possible, experimental objects, such as product prototypes, need to be created. Prototyping is used heavily in Agile software development. For example, prototyping is a core technique in the Dynamic Systems Development Method (DSDM) [4]. DSDM recommends four categories of prototypes: business, usability, performance and capacity, and capability prototypes. Early in the development process, business prototypes provide opportunities to conceptualise and communicate the business processes being automated, while usability prototypes allow definition, refinement, and demonstration of the system from a user's perspective. These two types of prototypes are closest to the notion of prototyping used in this paper. However, DSDM does not explicitly consider prototypes as experimental objects for testing the business value of a product idea during development. In Lean Startup terminology, a value hypothesis and a minimum viable product (MVP) need to be developed. Following Ries [5], a value hypothesis "tests whether a product or service really delivers value to customers once they are using it". An example of a value hypothesis is that customers of a specific customer segment will choose to sign up for a service based on a given set of features being offered. An MVP is an experimental object that allows for empirical testing of value hypotheses. According to the Lean Startup method, it should be built with a minimum amount of effort and development time [5].

In this paper, we argue that industry-academia collaborations are especially well suited for developing such MVPs and thereby support the rapid understanding

of customer value in software development. More specifically, we address the following research questions in the context of industry-academia collaborations:

RQ1: How should a product owner define a value hypothesis from which a minimum viable product is derived?

RQ2: How should an implementation team create a minimum viable product based on a value hypothesis?

RQ3: What are the main advantages and challenges for industry partners?

RQ4: What are the main advantages and challenges for academic partners?

The remainder of this paper is organised as follows. In Section 2, we describe work related to technology transfer and value-based software engineering. In Section 3, we describe our research and development infrastructure (i.e., the so-called Software Factory) for creating MVPs together with industry. In Section 4, we describe our case study, including the research design, method, and context, as well as the execution and results. In Section 5, we describe the limitations that apply when attempting to generalize the results. Finally, in Section 6, we summarize our findings and outline future directions for this work.

2 Related Work

As software is increasingly at the core of all kinds of innovations, the customer perspective comes more into focus. Rombach and Achatz [6] define innovation in software engineering as both invention and successful implementation. They present a 6-step generic process model for transferring an innovative technology from its creators to organizations which are able to make them a business success. There are many other similar technology transfer models proposed in the software engineering literature. Very often they are organized so that the technological invention comes first and the invention of a business case comes later. This is based on the implicit assumption that customers do perceive the invention as valuable or that the value can be identified later. In addition, it is often assumed that the link to business goals and strategies can be defined later or is obvious.

Although these assumptions might be true in some domains, they become questionable in domains of high uncertainty and highly competitive markets, such as start-up companies and companies entering new business segments. Here, business value is unclear or needs to be learned during the invention process. It is often more critical for business success to understand the perceived value for a real customer than to immediately bring an invention into technical perfection. Vice versa, perceived customer value might influence what innovative software technology is needed, i.e., the business drives the inventions. In such high-uncertainty environments, innovative software technologies and the understanding of their perceived customer value need to be integrated early in the software life cycle. Due to the fact that inventions often combine both innovative software technologies and innovative business models, it seems promising to analyse both experimentally. We argue that this approach is viable not only for new

development, but also for evolution of existing software-based products and services. We also argue that the approach is viable both for new, entrepreneurial start-up companies as well as for existing companies which face a need for renewal through internal entrepreneurship, or intrapreneurship.

Several approaches exist that integrate the business and the product development functions of organizations. One prominent approach is the "Lean Startup" [5] that leverages experimentation on the business level by rapid prototyping, testing value hypotheses, and very early feedback for product development. Another approach that focuses on considering value during the software process is "Value-based Software Engineering" (VBSE) [7]. This approach provides different elements, such as techniques for determining value by analytical means, and methods for including value considerations into management activities [8, 9]. Other related approaches exist, e.g. user-centered design [10], where design decisions are made based on empirical evidence gathered from user experiments and involvement. The approach presented in this article addresses the need to rapidly elicit and test value hypotheses through an empirical feedback cycle. The elements of VBSE and the experimental approaches in user-centered design can be seen as complementary to this approach.

3 Development and Usage of Minimum Viable Products in the Software Factory

Conducting experiments with customer value in the context of industry-academia collaborations requires a suitable infrastructure. The infrastructure we are using for this purpose is the co-called Software Factory. In this section, we sketch the concept of the Software Factory as an approach to MVP creation, outline some possible variations, and provide examples of how customer value can be operationalised and tested.

The Software Factory is a university laboratory that has been designed and implemented at the Department of Computer Science, University of Helsinki [11, 12, 13]. The main unique characteristics of the Software Factory are the integration of engineering- and business-level experimentation, a standardized laboratory infrastructure which supports conducting joint or comparative studies as well as the exchange of experience, the global network of Software Factories, which allows for running experiments in distributed settings, and the periodic conduction of projects several times a year, which allows efficient study set-up, rapid learning, and conducting longitudinal studies.

In the Software Factory laboratory, a company brings the vision about a new product or service and an initial value hypothesis. The company has the responsibility to test the value hypothesis whereas the Software Factory has the responsibility to create the MVP. The interaction between the Software Factory and the company can be seen as a learning cycle (Figure 1). The company provides an initial value hypothesis as input to a Software Factory project and acts as product owner. The Software Factory then creates an MVP that serves as an experimental object for testing the value hypothesis. During this development,

the company provides feedback so that the MVP will be aligned with the value hypothesis. The MVP is used by the company to test the value hypothesis with real customers. Based on the results of this experiment, the company modifies the value hypothesis if necessary. Subsequently, the Software Factory develops a new version of the MVP that aims at testing the modified value hypothesis. The same applies for a set of multiple hypotheses to be tested simultaneously. One might consider this as a process consisting of developing first and testing afterwards, but actually the process is in the reverse order: the value hypothesis initially defines what needs to be developed in the Software Factory. Results from the evaluation guide further iterations of the MVP.

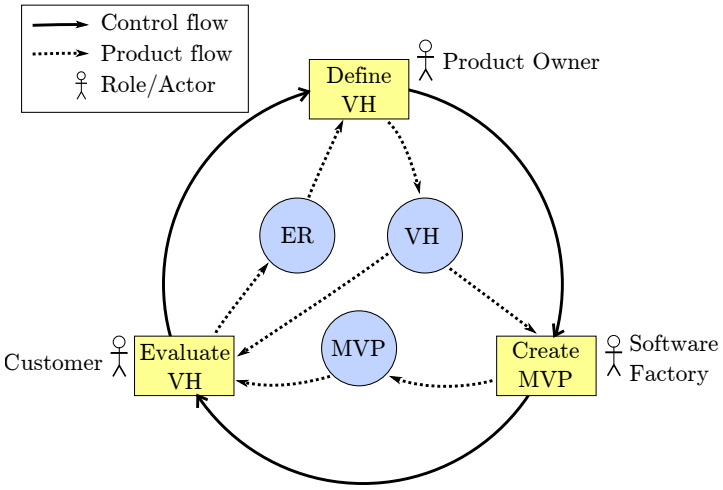

Fig. 1. Learning cycle: the Product Owner defines a Value Hypothesis (VH), which is used as input for the Software Factory to create a minimum viable product (MVP). The MVP is used to evaluate the VH in a customer test, resulting in an evaluation result (ER) that is fed back to refine or redefine the VH for the next cycle.

3.1 Variations in the learning cycle

The interaction between the company and the Software Factory is a crucial element of the experimental process, and there are several ways in which the interaction can occur. Depending on the context and constraints of the project, a company might replace testing the value hypothesis with qualitative expert feedback. This may occur especially in initial iterations when the customer is not ready to conduct potentially costly user experiments – but caution should be exercised here in order not to develop the product too far without empirical feedback. Another option could be to use the Software Factory project for the creation of the first MVP without testing the value hypothesis during this project.

In this case, experimentation with customers would start after project completion. It could also be possible that the company needs to pivot and start again with a new value hypothesis and a new MVP in case the initial value hypothesis has been proven significantly wrong.

3.2 Techniques for customer-related experiments

There are several techniques by which to perform customer-related experiments to validate value hypotheses of software-based products and services. Suitability of techniques depend on how far the software has been developed. Simpler techniques can yield coarse-grained insights quicker early on in the process, while more mature products and services can benefit from techniques which permit fine-grained analysis and linkage to business strategies. While this study does not cover the validation of the value hypothesis, we briefly explain three examples of techniques for conducting customer-related experiments: cohort analysis, A/B testing, and GQM$^+$Strategies.

Cohort analysis refers to studies comparing groups of people on one or several attributes over time (e.g. [14, 15]). A cohort is a group of people who share a common characteristic over a certain period of time. In medical research, cohort studies are used to assess the association between an event, such as the presence of a disease risk factor, and some outcome, such as actually developing the disease. Analysis of subgroups within a cohort can yield important insights that may confirm existing hypotheses or create new ones. Cohort analysis may be used, e.g., to determine whether users of different age groups signing up for a software service at a specific time are more or less likely to be frequent users of the service.

A/B testing is a technique where subjects are randomly assigned to two groups receiving different treatments (e.g. [15]). The groups are then compared with respect to some outcome. For example, group A may be shown a certain kind of feature in a mobile application, while group B is shown another feature. The groups are then compared to see which feature is more likely to attract the user to make a purchase decision. Such information can support the decision to focus implementation on one feature or the other.

For more elaborate needs, *GQM$^+$Strategies* [16] is an approach to systematically break down business-level goals into sub-goals and strategies for implementing them, and to link these with software measurement. GQM$^+$Strategies allows companies to streamline metrics collection to support business decisions. For customer-related experiments, GQM$^+$Strategies allows companies to express the desired goal of measurement (e.g. to increase sales), construct strategies for reaching those goals, and devise metrics that determine how the actions taken are contributing to the realization of the goals.

Besides the mentioned techniques, many other techniques can be used for conducting customer-related experiments such as multivariate testing, big data technologies, live customer feedback analyses, etc. All of these methods require significant expertise to properly select and apply.

4 Industry case

We use one of our projects in the Software Factory laboratory at the University of Helsinki to illustrate the experimental cycle. The main objective of the project was to validate a value hypothesis given by a customer. The project was initiated as a rapid-feedback development effort that would proceed from an initial value hypothesis to an initial MVP that could then be subjected to separate evaluation with real users. Consistently with the objective, there were no requirements to use any legacy code or perform potentially difficult integration into existing systems.

We focus here on the two first steps of the learning cycle shown in Figure 1 (definition of value hypotheses and creation of MVP). Intermediate versions of the MVP were subjected to expert evaluation and the value hypothesis and project priorities modified accordingly. The project completed two major cycles but did not include evaluation with specific customer experiments. We expect to study those separately.

4.1 Research Design and Method

This study can be characterized as a single-case study using multiple-researcher triangulation [17, 18]. Case studies can be said to study "contemporary phenomena in their natural context" [19] and to "generalise from specific contexts" [18]. There are several types of case studies [17], some of which have been used and described specifically for software engineering [19]. However, case studies do not constitute a homogeneous class of studies, but display considerable variation. Since cases can vary arbitrarily, researchers must adapt their designs and methods to truthfully represent the case, while balancing considerations of generalisability.

In this study, we took an open-ended, participatory approach. Two persons from the Department of Computer Science, University of Helsinki, and one person from the case company, were present and participated to varying degrees in the project. Data sources included participatory and direct observation, notes taken during the course of the project, and analysis of project artefacts such as produced software, documentation, and other materials. In particular, the project coach kept a diary of project events and provided substantial input for this study. The other participating researcher organised meetings and performed open-ended, interview-like discussions with the project participants in the beginning, middle, and end of the project. The company representative was the project's product owner and interacted closely with the student team.

Our analysis method can be described as narrative and inductive. From the data and experiences gathered, we build a chronological story of the project, including different perspectives and considerations, and attempt to trace causal relationships that allow us to abstract from the case material to more general findings.

4.2 Goal and Context

The main goal of the project was to develop an MVP that could be used to test the end-user value of a cloud service. The MVP was a game that generated metadata

regarding a set of objects in a cloud storage system as a by-product of playing. The overall value hypothesis was that the game would be satisfying to play and that users would consider the generated metadata to be valuable. To validate the overall value hypothesis, the customer wanted to find out i) whether users are motivated to use the software, ii) whether users perceive fun in using the software, iii) whether using the software creates valuable metadata, and more specifically, iv) whether the created metadata is usable for providing additional services. From an engineering perspective, the goal was to test v) if and how the game concept includes key elements of good game design. As a related goal, the execution of the project itself was expected to reveal valuable information regarding industry-academia collaboration. Finally, since the project team consisted of students, a separate goal of the project was to provide an educational experience.

The customer company was the Innovation and New Concepts division at F-Secure Corporation. The project team consisted of four students, with every student working 24-30 hours per week during the seven-week project. In addition to the students, a project coach was present with the team on a daily basis, and a customer representative in the product owner role was present at least once a week. The product owner was also available on demand over teleconference and email. The product owner was technically skilled, was one of the originators of the project idea, and was empowered to make all decisions regarding the project.

To determine which features would be implemented first to attempt to satisfy the value hypotheses, the product owner experimented with paper prototypes of the game. A question posed early in the project was whether to develop a single game with a tightly defined feature set, or to develop a configurable game platform. After some consideration, the team and product owner decided not to embark on the platform option, since it would lead to effort being spent on platform features which would not necessarily be needed for testing the value hypotheses. Instead, the decision was to make a game with a tightly defined feature set but with configuration options for testing the specific variations that the product owner had reason to believe would affect the playability of the game. These variations could then be exploited in subsequent testing of the value hypotheses.

The project team used Scrumban, a combination of Scrum and Kanban [20, 21], to coordinate its work. The team was given a training session on the process during the first days of the project. Otherwise, no tightly controlled process was used, as the project was aimed to promote creativity and exploration. This is also in line with the process itself, as described by Kniberg [21], since it allows a team to develop and tune its own process.

4.3 Execution

The time-line shown in Figure 2 illustrates milestones and events during the project, and shows a classification of the overall focus of project activities. Since the development of the prototype had no initial technological restrictions, significant time was spent on deciding which technology the new prototype should use. Aside from organization and initial training, most of the first two weeks

of the project were spent evaluating technology choices. The team evaluated several options from multiple perspectives, including ease of use and speed of development, licensing, technological options, and prior experience of team members. The most promising alternatives were then further analysed by developing small proof of concept implementations of the product idea. During the second week, two platforms were developed in parallel until the final choice could be made with established knowledge and taking the whole project context into consideration. A modern HTML5-based browser platform was chosen, and the main implementation was made using JavaScript and HTML5 web technologies.

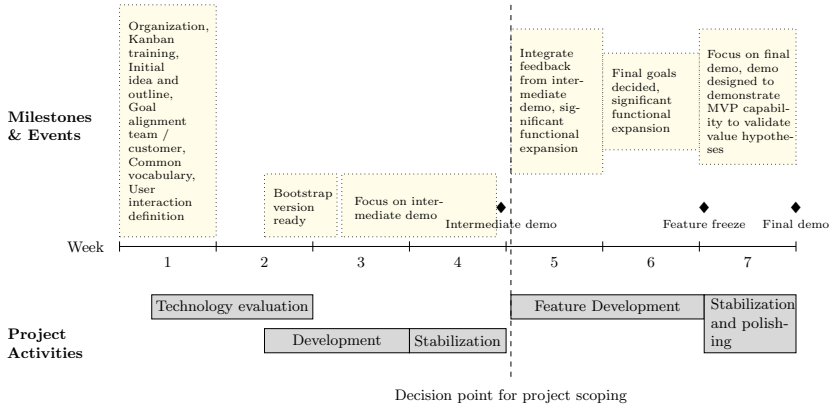

Fig. 2. Milestones, events, and activities per week.

Throughout the project, there was a clear backlog of features generated by the team in conjunction with the product owner. Goals of both the product owner and development team were aligned, which aided discovery of new features that could potentially contribute to the value hypothesis. The product owner offered an opinion on how to prioritize high level features, but the final choice was left to the team, as well as breakdown of features into practical tasks.

Since the product owner was available on demand, the team was able to apply rapid prototyping of the MVP and validate first draft solutions close to the time of inception. This "fail fast" approach allowed the product owner to immediately influence the direction of the project based on tangible evidence from a product increment.

The initial MVP was delivered, and the first major learning cycle completed, approximately two and half weeks after the technology choice had been made. An intermediate demo was given at the customer's premises roughly half-way through the project. The fact that this demo was scheduled already at the start of the project gave the team a clear focus for its efforts. The project then proceeded in very iterative expansions of the functionality, which meant that the customer was provided with product variations that could be used for testing the

value hypotheses. Code structure and architecture were expanded and developed iteratively, using several explorative solutions.

4.4 Results and Lessons Learned

In this section, we discuss the results of the study and list essential lessons learned from the project. We present each research question in turn, along with the lessons related to that particular question.

The most important lessons learned are concerned with i) the role of the product owner and ii) the close connection between the product owner and the team, but we also report some lessons regarding iii) technological choices and iv) demonstrations.

The role of the product owner *RQ1* asks how a product owner should define a value hypothesis from which a minimum viable product is derived. In our case, the product owner succeeded in defining the value hypothesis by not only presenting the hypothesis itself, but working with the implementation team to iteratively define the variation points in the MVP that would support testing the hypothesis. The result was a software prototype that did in fact support testing of the value hypotheses that the company had defined. We therefore propose that the product owner should define an overall value hypothesis on the business level, but be prepared to interpret the value hypothesis in increasingly refined and technical detail, in order to ensure that proper variation points are implemented for subsequent hypothesis testing.

We also found that being in close connection with the team allowed the product owner to correct misunderstandings and give further expansion on details that were not covered during active meeting time. This tight interaction with the team was essential for understanding and guidance, and therefore, the performance of the project. The product owner had both the technical and business-level skills to evaluate the product increments. The increments would not display stunning visual features immediately but would rather demonstrate some important behaviour in the software. The product owner's ability to see the essential characteristics of the feature under development was crucial. Being regularly present, the product owner facilitated continuous knowledge transfer, and no separate transfer process was required at the end of the project.

Technological choices *RQ2* asks how an implementation team should create a minimum viable product based on a value hypothesis. In our case, the team was able to interpret the value hypothesis and produce the desired MVP by efficient down-scoping of immediate next steps and by thoroughly experimenting with technology choices early in the project.

As previously noted, the team spent the first two weeks on evaluating technology choices. After the project, the team considered this appropriate, taking the project length into account. Also, the fact that the product owner knew the importance of making this choice helped the team to develop trust. The technology choice was not a simple feature list comparison, but was actually taken to the

point where the candidate platforms were used to develop simple proof-of-concept programs. This ensured that the team and product owner made decisions based on a real evaluation for this particular case, not general information given by a third party.

The project had a critical dependency on the customer's cloud storage platform. This posed an unnecessary development risk for the project, since the objective was not integration into the platform but production of the MVP. Therefore, the team and product owner decided very early that the storage API would be abstracted using a mock implementation. This had multiple positive effects for the project: not only was the prototype development technically independent from the platform, but the project also depended only on the product owner. The customer provided an API specification with the same characteristics as the real platform, ensuring that integration would be possible later. Besides serving as an MVP, the prototype revealed potential improvement needs in the platform API.

The team The small size of the team ensured little or no communication overhead, since team members could develop individual relations with each other, the customer, and other stakeholders. Task generation was initially difficult for the team, but became easier later in the project when high level features could be split into concrete development tasks for each part of the product (front end, game logic, back end). Clearly visible tickets on the Kanban board allowed for interactive prioritisation by the team together with the product owner.

Together, the observations on technological choices and the team link research questions 1 and 2: while the responsibility of the product owner is the definition of the value hypothesis, and the responsibility of the team is the creation of the MVP, both need to approach each other in interpreting how the value hypothesis should be operationalised and turned into a technological artefact. We propose that both product owners and team members require both special knowledge and skills to be able to function effectively when developing an MVP from a value hypothesis.

Demonstrations One way to facilitate the interpretation process between product owner and implementation team is to perform demonstrations. During the project, the team held weekly demonstrations for the product owner. In addition to this, an intermediate demo was scheduled for the middle of the project, and a final demo at the end. These were more comprehensive demonstrations held at the customer's premises and the audience included not only the product owner and team coach, but also experts from the customer organization.

The weekly demos provided clear intermediate deadlines for the team, and allowed it to focus on creating a tangible result. It also allowed the team to demonstrate functionality that had been defined during the development process in order to get feedback from the product owner. The demos made questions concrete and the product owner could base decisions on visible alternatives.

An interesting question arose after the intermediate demo: there was a trade-off between expanding the project scope and refining existing features (see Figure 2). For the customer, this decision highlighted the importance of defining what value was most important to test: whether the test should be more about comparing the viability of many features or about comparing the quality of a few features. The product owner chose to lean more towards comparing a larger set of features, since the concept was still in an early stage. In later stages, after the value proposition has been field-tested, the choice could lean more towards the details of a few promising features.

Advantages and challenges *RQ3* and *RQ4* ask what the main advantages and challenges of creating MVPs in industry-academia collaborations are for industry and academic partners, respectively. In our case, three particular advantages were clearly visible for industry partners. First, it is beneficial for industry partners to be able to conduct MVP creation in a relatively low-risk environment. Since the amount of company resources that are tied to the project is limited, the impact of project failure in terms of lost resources is also limited. Second, industry partners can benefit from the use of measurement experience that exists in academia. Researchers working in the Software Factory context are trained to design, conduct, and analyse experiments, and they can be utilised to assist in different stages of the project. Third, and more broadly, industry partners can utilise other kinds of research capabilities inherent in the academic environment. An MVP-creation project may be embedded in a larger research project, or it can be expanded into one.

For academia, we found a number of benefits. Collaboration projects can provide real cases for research, as is evident in this article. Also, the benefit for students in the form of valuable learning experiences was clearly visible. As an example, one student who worked as a software developer remarked after the project that "I now understand much better what I'm doing at work". We also found that the project brought contemporary realism into the world of academia, and we were challenged to keep up to date with relevant problems from practice.

Based on our experience, a number of challenges apply when creating MVPs in industry-academia collaboration projects. A first challenge relates to synchronisation of schedules. University course calendars, in our case especially the Software Factory project schedule, needs to be positioned in an appropriate time window where a partner company has the willingness and available resources to test innovations. Second, technological infrastructure presents a challenge: MVPs must be deployable on either existing platforms, which may include legacy components, or on new platforms, which may be under development. In some cases, the MVP may need to be deployed on end-user systems, which requires careful consideration of platform compatibility and data security. Finally, properly conducting value-based experiments requires measurement competence. Appropriate training and competence is needed when defining value hypotheses, design appropriate experiments, derive necessary measures, conduct experiments, and reason from analysis results. Especially the customer should have adequate competences in defining value hypotheses and be able to draw conclusions from analysis results.

5 Limitations

Despite the case project having many realistic features, this study has a number of limitations. First, the produced MVP was not subject to field evaluation with real users. Therefore, lessons learned with respect to appropriate experimental instruments for customer validation are missing. On the other hand, this was precisely the intended scope of the project: to demonstrate how a concept idea can be developed into a prototype implementation for subsequent evaluation. We expect to evaluate the end result separately.

Second, since the project team consisted of students, the results may not be applicable to professional software development. Furthermore, the team was new, and with repeated cycles of execution, they may change their approach. Also, another team may have done things differently. However, we argue that these limitations are not as severe as they may first appear. It is well known that the programmer productivity can vary considerably [22]. The fact that this programming team was able to implement the prototype with unfamiliar technology in the given time-frame indicates that a professional team should be able to do the same in a similar amount of time, with the limitation imposed by varying individual degrees of productivity. In the formation of the team, a simple, single-task programming test was administered before team members were admitted to the project. There were no indications that the team members would all have been unusually skilled. In addition, the goal of of an MVP is to test value hypotheses. Therefore, the main criterion is that the MVP is suited for such testing, regardless of who has developed it. A more experienced team may have reached better results, but even at this baseline, the result was acceptable.

A final limitation relates to rapid prototyping. Many technologies are ill-suited to rapid prototyping and thus the proposed approach may not be applicable when using such technologies. However, it may be possible to set aside the less flexible tools and prototype with other technologies only for the purpose of testing a value hypothesis. Once the value has been determined, a production version can be implemented with another technology, thus avoiding potentially long and costly implementation projects when the value is uncertain. As demonstrated in this project, it is often possible to abstract away parts of the system that are not relevant for testing the value proposition, and address them at a later stage. A potential risk with this approach is that the technology to be used in the production version can cause significantly different cost of implementation than the prototyping technology for the chosen feature set. However, this question is different from the question of customer value. Also, the prototyping itself is likely to reveal important requirements for implementation technology.

6 Summary and Future Work

This paper presents experience with creating an MVP in the context of an industry-academia collaboration. The experience is based on a concrete case study. In summary, rapid prototyping for getting value-related feedback can

be systematically conducted using a simple framework process, in which the implementation team is given freedom to explore the concept design space in short, incremental cycles in close cooperation with the customer. Factors critical for success are i) the role of the product owner, who should come from the customer organization, have enough technical and domain knowledge to make correct design decisions, and be empowered to make those decisions, ii) taking the proper time to investigate technology options with the goal of selecting an implementation technology that supports rapid prototyping, iii) abstracting away parts of the target platform to keep the project focused on testing the value proposition, not the integration into the platform, iv) systematically employing a light-weight process in which the implementation team and the product owner can prioritize high-levels features and the team has autonomy to decide on feature decomposition into tasks, and v) having frequent demonstrations, including demonstrations with an audience that is not directly taking part in the project.

Our study provided several insights on how the development of MVPs with value-related feedback can be organized in short iteration cycles. The following questions illustrate some interesting future directions. First, testing value hypotheses in the field requires a suitable experimental infrastructure. What are the requirements for such an infrastructure in different contexts? How can software products and services be integrated into such an infrastructure? These questions are also relevant for supporting design decisions on how to evolve existing products or services. Second, an open question is how to integrate experimentation on the business level with experimentation on the technical level. Sustainable innovations require technological advantages that also need to be evaluated by experimental means. This warrants further research. Third, can experiment-based software development be aligned with organizational and legal constraints? For example, contractual issues may need special consideration in experiment-based software development. Finally, how should customer value be evaluated in cases where the value creation is part of an ecosystem? For example, the value could come from the number and type of apps available in an ecosystem.

Both start-up companies and innovative intrapreneurial divisions in established companies can benefit from insights into these questions. We expect more systematic experimentation with customer value in software development and evolution to significantly speed up the pace of innovation and help release more disruptive software-based solutions. Through industry-academia collaboration, companies can reduce the barriers to experimenting with MVP creation.

References

1. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., Wesslén, A.: Experimentation in software engineering. 2012 edn. Springer (2012)
2. Basili, V.R., Selby, R.W., Hutchens, D.H.: Experimentation in software engineering. IEEE Trans. Softw. Eng. **12**(7) (July 1986) 733–743
3. Boehm, B., Jain, A.: Developing a process framework using principles of value-based software engineering. Software Process: Improvement and Practice **12**(5) (2007) 377–385

4. Howard, A.: A new RAD-based approach to commercial information systems development: the dynamic system development method. Industrial Management + Data Systems **97**(5) (1997) 175–177
5. Ries, E.: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Publishing Group (2011)
6. Rombach, D., Achatz, R.: Research Collaborations between Academia and Industry. In: 2007 Future of Software Engineering. FOSE '07, Washington, DC, USA, IEEE Computer Society (2007) 29–36
7. Boehm, B.: Value-based software engineering. ACM SIGSOFT Software Engineering Notes **28**(2) (2003) 4–
8. Rönkkö, M., Frühwirth, C., Biffl, S.: Integrating value and utility concepts into a value decomposition model for value-based software engineering. Lecture Notes in Business Information Processing **32 LNBIP** (2009) 362–374
9. Raffo, D., Mehta, M., Anderson, D., Harmon, R.: Integrating Lean principles with value based software engineering. In: Technology Management for Global Economic Growth (PICMET), 2010 Proceedings of PICMET '10. (2010) 1–10
10. Greenbaum, J., Kyng, M.: Design at work: cooperative design of computer systems. Lawrence Erlbaum Associates, Inc. (1991)
11. Fagerholm, F., Oza, N., Münch, J.: A Platform for Teaching Applied Distributed Software Development: The Ongoing Journey of the Helsinki Software Factory. Collaborative Teaching of Globally Distributed Software Development Workshop (CTGDSD), 2013 (2013)
12. Software Factory: Software Factory Web Site. Online: `http://www.softwarefactory.cc/` Last visited: 2013-04-12.
13. Abrahamsson, P., Fagerholm, F., Kettunen, P.: The Set-Up of a Valuable Software Engineering Research Infrastructure of the 2010s. The 11th International Conference on Product Focused Software Development and Process Improvement (PROFES 2010) / Workshop on Valuable Software Products (VASOP 2010) (11) (2010)
14. Porta, M., ed.: A Dictionary of Epidemiology. Oxford University Press, New York, NY, USA (2008)
15. Croll, A., Yoskovitz, B.: Lean Analytics: Use Data to Build a Better Startup Faster. O'Reilly Media, Inc., Sebastopol, CA, USA (2013)
16. Basili, V., Heidrich, J., Lindvall, M., Münch, J., Regardie, M., Trendowicz, A.: GQM+Strategies – Aligning business strategies with software measurement. Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007 (2007) 488–490
17. Yin, R.: Case study research: design and methods. 4 edn. SAGE Publications, Inc. (2009)
18. Eisenhardt, K.M.: Building Theories from Case Study Research. The Academy of Management Review **14**(4) (1989) pp. 532–550
19. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2) (2009) 131–164
20. Ladas, C.: Scrumban – Essays on Kanban Systems for Lean Software Development. Modus Cooperandi Press (2009)
21. Kniberg, H., Skarin, M.: Kanban and Scrum – making the most of both. C4media (2010)
22. Endres, A., Rombach, D.: A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories. The Fraunhofer IESE Series on Software Engineering. Addison Wesley (2003)