Building Blocks for Continuous Experimentation

Fabian Fagerholm, Alejandro Sanchez Guinea, Hanna Mäenpää, Jürgen Münch
Department of Computer Science, University of Helsinki
P.O. Box 68, FI-00014 University of Helsinki, Finland
fabian.fagerholm@helsinki.fi, azsanche@cs.helsinki.fi,
hanna.maenpaa@cs.helsinki.fi, juergen.muench@cs.helsinki.fi

ABSTRACT

Development of software-intensive products and services increasingly occurs by continuously deploying product or service increments, such as new features and enhancements, to customers. Product and service developers need to continuously find out what customers want by direct customer feedback and observation of usage behaviour, rather than indirectly through up-front business analyses. This paper examines the preconditions for setting up an experimentation system for continuous customer experiments. It describes the building blocks required for such a system. An initial model for continuous experimentation is analytically derived from prior work. The model is then matched against empirical case study findings from a startup company and adjusted. Building blocks for a continuous experimentation system and infrastructure are presented. A suitable experimentation system requires at least the ability to release minimum viable products or features with suitable instrumentation, design and manage experiment plans, link experiment results with a product roadmap, and manage a flexible business strategy. The main challenges are proper and rapid design of experiments, advanced instrumentation of software to collect, analyse, and store relevant data, and the integration of experiment results in both the product development cycle and the software development process.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Elicitation methods (e.g. rapid prototyping, interviews, JAD)*; D.2.2 [**Software Engineering**]: Design Tools and Techniques—*Evolutionary prototyping*; D.2.9 [**Software Engineering**]: Management

General Terms

Economics, Experimentation, Management, Measurement, Theory

Keywords

Continuous Experimentation, Product Development, Architecture, Agile Software Development, Lean Software Development, Lean Startup

RCoSE'14, June 3, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2856-2/14/06...$15.00
http://dx.doi.org/10.1145/2593812.2593816

1. INTRODUCTION

The accelerating digitalisation in most industry sectors means that an increasing number of companies are or will soon be providers of software-intensive products and services. Simultaneously, new companies already enter the marketplace as software companies. Software enables increased flexibility in the types of services that can be delivered, even after an initial product has been shipped to customers. Many constraints that previously existed, particularly in terms of the behaviour of a product or service, can now be removed.

With this new-found flexibility, the challenge for companies is no longer primarily how to identify and solve technical problems, but rather how to solve problems which are relevant for customers and thereby deliver value. Finding solutions to this problem has often been haphazard and based on guesswork, but many successful companies have approached this issue in a systematic way. Recently, a family of generic approaches have been proposed. For example, the Lean Startup methodology proposes a three-step cycle: build, measure, learn.

However, a detailed framework for conducting systematic, experiment-based software development has not been elaborated. Such a framework has implications for the technical product infrastructure, the software development process, the requirements regarding skills that software developers need to design, execute, analyse, and interpret experiments, and the organisational capabilities needed to operate and manage a company based on experimentation in research and development. Kohavi et al. [10] note that running experiments at large scale requires addressing multiple challenges in three areas: cultural/organisational, engineering, and trustworthiness. The larger organisation needs to learn the reasons for running controlled experiments and the trade-offs between controlled experiments and other methods of evaluating ideas. Even negative experiments should be run, which degrade user experience in the short term, because of their learning value and long-term benefits. When the technical infrastructure supports hundreds of concurrent experiments, each with millions of users, classical testing and debugging techniques no longer apply because there are millions of live variants of the system in production. Instead of heavy up-front testing, Kohavi et al. report having used alerts and post-deployment fixing. The system has also identified many negative features that were avoided despite having support from key stakeholders, saving large amounts of money.

Continuous experimentation with software product and service value should itself be based on empirical research. In this paper, we present the most important building blocks of a framework for continuous experimentation. Specifically, our research question is:

RQ How can Continuous Experimentation with software-intensive products and services be organised in a systematic way?

We give an answer to the research question by validating an analytically derived model against a case study of a startup company that produces a novel video calling service with both hardware and software components.

The rest of this paper is organised as follows. In Section 2, we review related work on integrating experimentation into the software development process. In Section 3, we describe the research approach and context of the study. In Section 4, we first present our proposed model for continuous experimentation, and then relate the findings of our case study to it in order to illustrate its possible application and show the empirical observations that it was grounded in. In Section 5, we discuss the model and consider some possible variations. Finally, we conclude the paper and present an outlook on future work in Section 6.

2. RELATED WORK

Agile software development was introduced more than two decades ago [6]. Its proponents sought better ways to organize software development so that it would be able to meet the dynamic and unpredictable conditions that characterize the business environment for software intensive organizations. Many different Agile methods have been devised. Dynamic Systems Development Method (DSDM) [22] has been recognized by Larman and Basili as the first Agile method [11]. Extreme Programming (XP) [3] aimed to improve software quality and responsiveness to changing customer requirements. Scrum [20] was devised to manage software projects and product development. Despite their goals and benefits, Agile methods do not provide an integral framework that incorporates all the concerns and stakeholders of an organization towards developing software that can provide value to customers.

Lean manufacturing and the Toyota Production System [16] has inspired the definition of Lean software development. This approach provides comprehensive guidance for the combination of design, development, and validation built as a single feedback loop focused on discovery and delivery of value [18]. The main ideas of this approach, which have been emphasised since its introduction, are summarised in seven principles: optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone, and keep getting better [17].

Lean Startup [19] provides mechanisms to ensure that what customers wants gets effectively addressed by the development. The methodology is based on the Build-Measure-Learn loop that establishes the learning about customers and their needs as the unit of progress. It proposes to apply scientific method and thinking to startup businesses in the form of learning experiments. As the results of the experiments are analysed, the company has to decide to "persevere" on the same path or "pivot" in a different direction while considering what has been learned from customers.

In light of the benefits that a methodology such as Lean Startup can provide, where controlled experiments constitute the main activity driving development, Holmström Olsson et al. [9] propose a target stage for any company that wishes to build a development system with the ability to continuously learn from real-time customer usage of software. They describe the stages that a company has to traverse in order to achieve that target as the "stairway to heaven". This target stage is achieved when the software organization functions as an R&D experiment system. The stages on the way to achieving the target are: 1. traditional development, 2. agile R&D organization, 3. continuous integration, and 4. continuous deployment. The authors first describe these first four stages and then analyse them through a multiple-case study that examine the barriers that exist on each step on the path towards continuous deployment. The target stage is only described, and the barriers to reach it are not addressed. A main finding from the case study is that the transition towards Agile development requires shifting to small development teams and focusing on features rather than on components. Also, it is relevant to notice that the transition towards continuous integration requires an automated build and test system, a main branch to which code is continuously delivered, and modularized development. The authors found that in order to move from continuous integration to continuous deployment, organizational units such as product management must be fully involved, and close work with a very active lead customer is needed when exploring the product concept further. The authors suggest two key actions to make the transition from continuous deployment to R&D as an "experiment system". First, the product must be instrumented so that field data can be collected in actual use. Second organisational capabilities must be developed in order to effectively use the collected data for testing new ideas with customers.

Other works have studied some of the stages of the "stairway to heaven" individually. Ståhl & Bosch [21] have studied the continuous integration stage, pointing out that there is no homogeneous practice of continuous integration in the industry. They propose a descriptive model that allows studying and evaluating the different ways in which continuous integration can be viewed.

The final stage of the "stairway to heaven" is detailed and analysed by Bosch [4]. The differences between traditional development and the continuous approach are analysed, showing that in the context of the new, continuous software development model, R&D is best described as an "innovation experiment system" approach where the development organization constantly develops new hypotheses and tests them with certain group of customers. This approach focuses on three phases: pre-deployment, non-commercial deployment, and commercial deployment. The authors present a first systematization of this so-called "innovation experiment system" adapted for software development for embedded systems. It is argued that just as in the case of cloud computing and Software-as-a-Service (SaaS) the "innovation experiment system" should be the aim, development for embedded systems should be guided by similar processes. That is, requirements should evolve in real-time based on data collected from systems in actual use with customers.

In this paper, we build upon the ideas that define the last stage of the "stairway to heaven", and propose a model for continuous experimentation. In this model, experiments are derived from business strategies and aim to assess assumptions derived from those strategies, potentially invalidating or supporting the strategy. Previous works have explored the application of a framework for linking the business goals and strategies to the software development level (e.g., [2], [14]). However, those works have not considered the particular traits of an experiment system such as the one presented in this paper. The model presented also describes the platform infrastructure that is necessary to establish the whole experiment system. The Software Factory [7] can serve as infrastructure for the model proposed, as it is a software development laboratory well suited for continuous experimentation. A previous work on creating minimum viable products [13] in the context of collaboration between industry and academia presented the Software Factory laboratory in relation to the Lean Startup approach and continuous experimentation. Some of the foundational ideas behind Software Factory with respect to continuous experimentation have been studied in the past, analysing, for instance, the establishment of laboratories specifically targeted for continuous development [15] and the impact of continuous integration in teaching software engineering.

Previous works have presented case studies that exhibit different aspects concerning continuous experimentation. Steiber [23] report on a study of the model of continuous experimentation followed by Google, analysing a success story of this approach. Adams [1] present a case study on the implementation of Adobe's Pipeline, a process that is based on the continuous experimentation approach.

The building blocks presented in this paper, although generalizable with certain limitations, are derived from a startup environment where the continuous experimentation approach is not only well suited but possibly the only viable option for companies to grow. Our work can be seen as a low level model of the "Early Stage Startup Software Development Model" (ESSSDM) of Bosch et al. [5] which extends existing Lean Startup approaches offering more operational process support and better decision-making support for startup companies. Specifically, ESSSDM provides guidance on when to move product ideas forward, when to abandon a product idea, and what techniques to use and when, while validating product ideas. Some of the many challenges faced on trying to establish an startup following the Lean Startup methodology are presented in [12] with insights that we have considered for the present work.

One important conceptual concern is the relationship between exploration and discovery, and actually conducting experiments. Data mining and information retrieval techniques may be used to find interesting patterns in existing databases. Such databases commonly exist in medium and large-size companies and they can be a valuable source of information on customer behaviour. However, we specifically consider experiments to test hypotheses in this article. In many situations, the relevant data to answer specific questions may not be available. In other situations, the multitude of interesting patterns that can be found by analysing vast amounts of data make it difficult to make informed decisions on specific questions of interest. The model we present links experimentation on the product and technical level to the product vision and strategy on the business level. This allows focused testing of business hypotheses and assumptions, which can be turned into faster decision-making and reaction to customer needs.

3. RESEARCH APPROACH

Our general research framework can be characterised as design science research [8], in which the purpose is to derive a technological rule which can be used in practice to achieve a desired outcome in a certain field of application [24]. The continuous experimentation model presented in this paper was first constructed based on the related work presented in the previous section as well the authors' experience. While a framework can be derived by purely analytic means, its validation requires a grounding it in empirical observations. For this reason, we conducted a case study in the Software Factory laboratory at the Department of Computer Science, University of Helsinki, in which we matched the initial model to empirical observations and made subsequent adjustments to produce the final model. The model can still be considered tentative, pending further validation in other contexts. In this section, we describe the case study context and the research process.

3.1 Context

The Software Factory is an educational platform for research and industry collaboration. In Software Factory projects, teams of Master's-level students use contemporary tools and processes to deliver working software prototypes in close collaboration with industry partners. The goal of Software Factory activities is to provide students a means for applying their advanced software development skills in a working life relevant environment and to deliver meaningful results for their customers [7].

Tellybean Ltd.[1] is a small Finnish startup that develops a video calling solution for the home television set. During September 2012-December 2013 the company was a customer in three Software Factory projects with the aim of creating an infrastructure to support measurement and management of the architecture of their video calling service. Tellybean Ltd. aims at delivering a life-like video calling experience as their single-product strategy. Their value proposition: "the new home phone as a plug and play -experience" is targeted at late adopter consumer customers who are separated from their families, e.g. due to migration into urban areas, global social connections, or overseas work. The company pays special emphasis on discovering and satisfying needs of the elderly, making ease of use the most important non-functional requirement of their product. The primary means for service differentiation in the marketplace are affordability, accessibility and ease of use. For the premiere commercial launch, and to establish the primary delivery channel of their product, the company aims at partnering with telecom operators. The company had made an initial in-house architecture and partial implementation during a pre-development phase. A first project was conducted to extend the platform functionality of this implementation. A second project was conducted to validate concerns related to the satisfaction of operator requirements. After this project, a technical pivot was conducted, with major portions of the implementation being changed. A third project was then conducted to extend the new implementation with new features related to the ability to manage software on already delivered products, enabling continuous delivery. The launch strategy can be described as an MVP launch with post-development adaptation.

3.1.1 Product

The Tellybean video calling service has the basic functionalities of a home phone: it allows making and receiving video calls and maintaining a contact list. The product is based on an Android OS based set-top-box (STB) that can be plugged into a modern home TV. The company maintains a backend system for mediating calls to their correct respondents. While the server is responsible for routing the calls, the actual video call is performed as a peer to peer connection between STBs residing in the homes of Tellybean's customers.

The company played the role of a product owner in three Software Factory projects during September 2012- December 2013. The aim of the first two projects was to create new infrastructure for measuring and analysing usage of their product in its real environment. Their third project at Software Factory delivered an automated system for managing and updating the STB software remotely. Table 1 summarises the goals and motivations of the projects in detail. Each project had a 3–7 -person student team, a company representative accessible at all times, and had a person-hour size of between 600 and 700 hours.

3.1.2 Project 1

The aim of Tellybean's first project at the Software Factory was to build means for measuring performance of their video calling product in its real environment. The goal was to develop a browser-based business analytics system. The team was also assigned to produce a back-end system for storing and managing data related to video calls, in order to satisfy operator monitoring requirements. The Software Factory project was carried out in seven weeks by a team of four Master's-level computer science students. Competencies required in the project were database design, application programming and user interface design.

[1]`http://www.tellybean.com/`

Table 1: Scope of each of the three Tellybean projects at Software Factory.

	High-level goal	Motivation
Project 1	As an operator, I want to be able to see metrics for calls made by the video call product's customers.	. . . so that I can extract and analyse business critical information. . . . so that I can identify needs for maintenance of the product's technical architecture .
Project 2	As a Tellybean developer, I want to be sure that our product's system architecture is scalable and robust. As a Tellybean developer, I want to know technical weaknesses of the system. As a Tellybean developer, I want to receive suggestions for alternative technical architecture options.	. . . so that I know the limitations of the system. . . . so that I can predict needs for scalability of the platform. . . . so that I can consider future development options.
Project 3	As a technical manager, I want to be able to push an update to the Tellybean set-top-boxes with a single press of a button.	. . . so that I can deploy upgrades to the software on one or multiple set-top-boxes.

The backend system for capturing and processing data was built on the Java Enterprise Edition platform, utilizing the Spring Open Source framework. The browser-based reporting system was built using JavaScript frameworks D3 and NVD3 to produce vivid and interactive reporting. A cache system of historical call data was implemented to ensure the performance of the system.

After the project had been completed, both students and the customer deemed that the product had been delivered according to the customer's requirements. Despite the fact that some of the foundational requirements changed during the project due to discoveries of new technological solutions, the customer indicated satisfaction with the end-product. During the project, communication between the customer and the team was frequent and flexible.

3.1.3 Project 2

The second project executed at Software Factory aimed at performing a system-wide stress test for the company's video calling service infrastructure. The Software Factory team of four Master's-level students produced a test tool for simulating very high call volumes. The tool was used to run several tests against Tellybean's existing call mediator server.

The test software suite included a tool for simulating video call traffic. The tool was implemented using the Python programming language. A browser-based visual reporting interface was also implemented to help analysis of test results. The reporting component was created using existing Javascript frameworks such as Highcharts.js and Underscore.js. Test data was stored in a MongoDB database to be utilized in analysis.

The team found significant performance bottlenecks in Tellybean's existing proof-of-concept system and analysed their origins. Solutions for increasing operational capacity of the current live system were proposed and some of them were also implemented. Towards the end of the project, the customer suggested that a new proof-of-concept call mediating server should be proposed by the Software Factory team. The team delivered several suggestions for a new service architecture and composed a new call mediator server.

3.1.4 Project 3

For their third project at Software Factory, Tellybean aimed to create a centralized infrastructure for updating their video calling product's software components. The new remote software management system would allow the company to quickly deploy software updates to already delivered STBs. The functionality was business critical to the company and its channel partners: it allowed updating the software without having to travel on-location to each customer to update their STBs. The new instrument enabled the company to establish full control of their own software and hardware assets.

The project consisted of a team of five Master's-level computer science students. The team delivered a working prototype for rapid deployment of software updates.

3.2 Research Process

The case study analysis was performed in order to ground the continuous experimentation model in empirical observations, not to understand or describe the projects themselves. Therefore, we collected information that would help us understand the prerequisites for performing continuous experimentation, the associated constraints and challenges, and the logic of integrating experiment results into the business strategy and the development process.

During the projects, we observed the challenges the company faced related to achieving the continuous experimentation system. At the end of each project, an in-depth debriefing session was conducted to gain retrospective insights into the choices made during the project, and the reasoning behind them. In addition to these sources, we interviewed three company representatives to understand their perception of the projects and to gain data which could be matched against our model.

The debriefing sessions were conducted in a workshop-like manner, with one researcher leading the sessions and the project team, customer representatives, and any other project observer present. The sessions began with a short introduction by the leader, after which the attendees were asked to list events they considered important for the project. Attendees wrote down each event on a separate sticky note and placed them on a time-line which represented the duration of the project. As event-notes were created, clarifying discussion about their meaning and location on the time-line took place. When attendees could not think of any more events, they were asked to systematically recount the progress of the project using the time-line with events as a guide.

The interviews with customer representatives were conducted either in person on the customer's premises or online via video conferencing. The interviews were semi-structured, having a mixture of predefined and free-form structure. A minimum of two researchers were present in the interviews to ensure that relevant information was correctly extracted. All participating researchers took notes during the interviews, and notes were compared after the interviews to ensure consistency. In the interviews, company representatives were first asked to recount their perception of their company, its

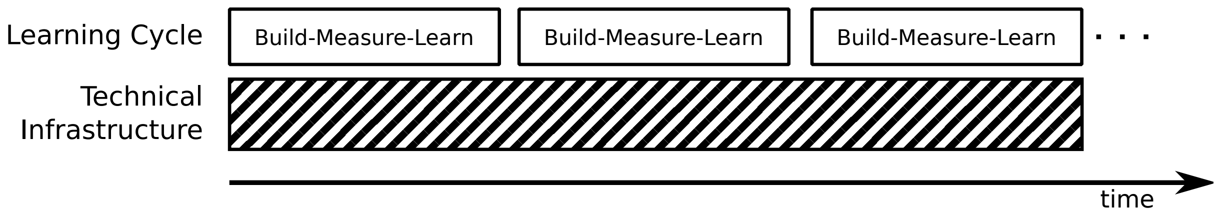

Figure 1: Continuous Experimentation System.

goals, and its mode of operation before the three projects. Then, they were asked to consider what each project had accomplished in terms of software outcomes, learned information, and implications for the goals and mode of operation of the company. Finally, they were asked to reflect on how the company operated at the time of the interview and how they viewed the development process, especially in terms of incorporating market feedback into decision-making.

During analysis, the project data was examined for information relevant to the research question. We evaluated and adjusted our initial model based on the understanding gained from the observations, retrospective sessions, and interviews.

4. RESULTS

In this section, we first describe our proposed model for continuous experimentation, and then report on the insights gained from the case study and how they inform the different parts of the model.

4.1 Model for Continuous Experimentation

By continuous experimentation, we refer to a software development approach that is based on field experiments with relevant stakeholders, typically customers or users, but potentially also with other stakeholders such as investors, third-party developers, or software ecosystem partners. The model consists of repeated Build-Measure-Learn blocks, supported by an infrastructure, as shown in Figure 1. Conceptually, the model can also be thought to apply not only to software development, but also to design and development of software-intensive products and services. In some cases, experimentation using this model may require little or no development of software.

The Build-Measure-Learn blocks structure the activity of conducting experiments, and connect product vision, business strategy, and technological product development through experimentation. Figure 2 illustrates the Build-Measure-Learn blocks. The general vision of the product or service is assumed to exist. Following the Lean Startup methodology [19], this vision is fairly stable and is based on knowledge and beliefs held by the entrepreneur. The vision is connected to the business strategy, which is a description of how to execute the vision. The business strategy is more flexible, and it consists of multiple assumptions regarding the actions required to bring a product or service to market that fulfils the vision and is profitable. However, each assumption has inherent uncertainties. In order to reduce the uncertainty, we propose to conduct experiments. An experiment operationalises the assumption and states a hypothesis that can be subjected to experimental testing in order to gain knowledge regarding the assumption. Once the hypothesis is formulated, two parallel activities can occur. The hypothesis is used to implement and deploy a Minimum Viable Product (MVP) or Minimum Viable Feature (MVF), which is used in the experiment and has the necessary instrumentation. Simultaneously, an experiment is designed to test the hypothesis. The experiment is then executed and data from the MVP/MVF are collected in accordance with the experimental design. The resulting data are analysed, concluding the experimental activities.

Once the experiment has been conducted and analysis performed, the analysis results are used on the strategy level to support decision-making. Again following Lean Startup terminology, the decision can be to either "pivot" or "persevere" [19]. If the experiment has given support to the hypothesis, and thus the assumption on the strategy level, a full product or feature is developed or optimised, and deployed. The strategic decision in this case is to persevere with the chosen strategy. If, on the other hand, the hypothesis was falsified, invalidating the assumption on the strategy level, the decision is to pivot and alter the strategy by considering the implications of the assumption being false. Alternatively, the tested assumption could be changed, depending on what the experiment was designed to test.

To support conducting such experiments, an infrastructure for continuous experimentation is needed. Figure 3 sketches the roles and associated tasks, the technical infrastructure, and the information artefacts of such an infrastructure. The roles indicated here will be instantiated in different ways depending on the type of company in question. In a small company, such as a startup, a small number of persons will handle the different roles and one person may have more than one role. In a large company, the roles are handled by multiple teams. Five roles are defined to handle three classes of tasks. A business analyst and a product owner, or a product management team, together handle the creation and iterative updating of the strategic roadmap. In order to do so, they consult existing experimental plans and results which reside in a back-end system. As plans and results accumulate and are stored, they may be reused in further development of the roadmap. The business analyst and product owner work with a data analyst role, which is usually a team with diverse skills, to communicate the assumptions of the roadmap and map the areas of uncertainty which need to be tested.

The data analyst designs, executes, and analyses experiments. A variety of tools are used for this purpose, which access raw data in the back-end system. Conceptually, raw data and experiment plans are retrieved, analysis performed, and results produced. The results are stored back into the back-end system.

The data analyst also communicates with a developer and quality assurance role. These roles handle the development of MVPs, MVFs, and the final product. They first work with the data analyst to produce proper instrumentation into the front-end system, which is the part of the software which is delivered or visible to the user. In the case of a persevere-decision, they work to fully develop or optimise the feature and deploy it into production. Cross-cutting concerns such as User Experience may require additional roles working with several of the roles mentioned here.

The back-end system consists of an experiment database which, conceptually, stores raw data collected from the software instrumentation, experiment plans, which include programmatic features of sample selection and other logic needed to conduct the experiment, and experiment results. The back-end system and the database are

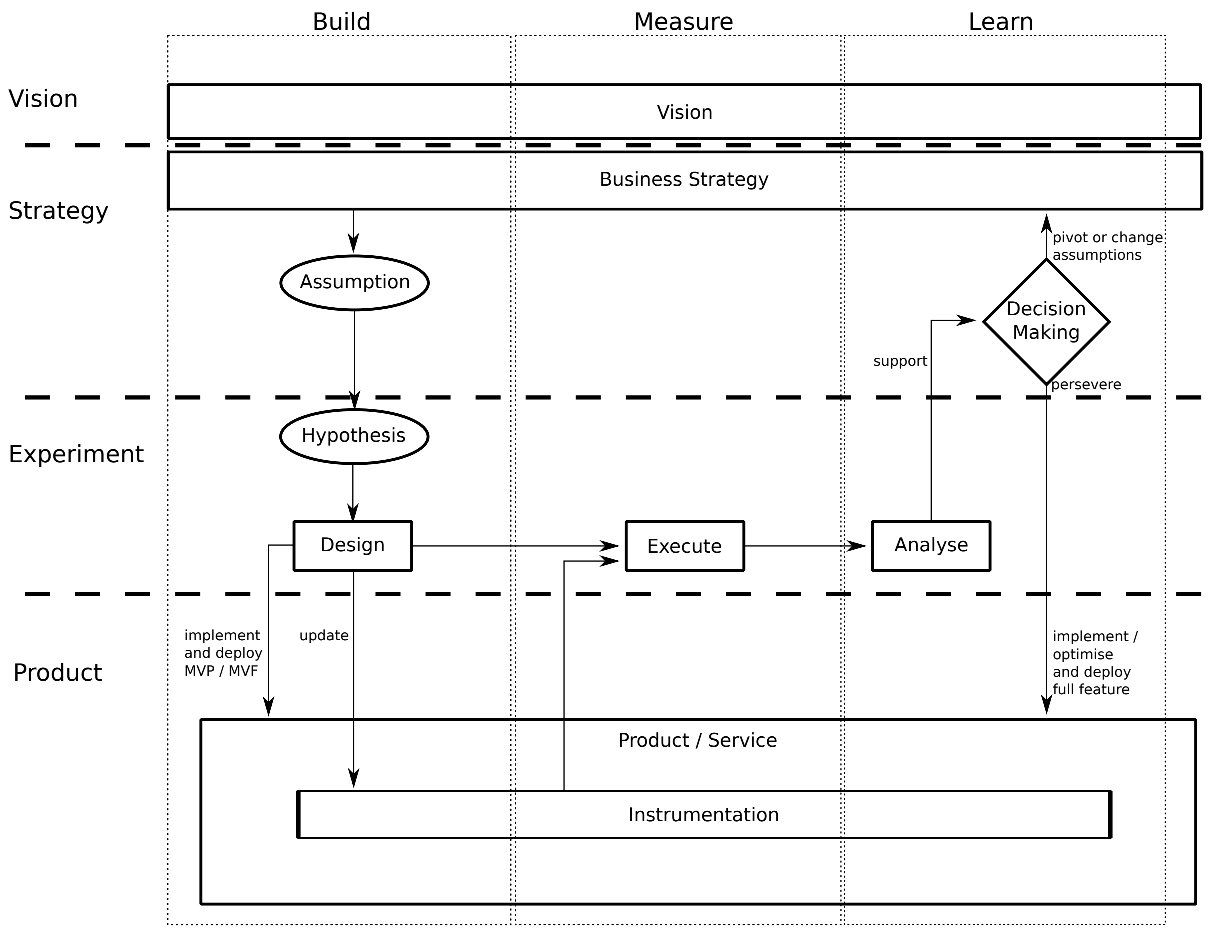

Figure 2: Build-Measure-Learn Block.

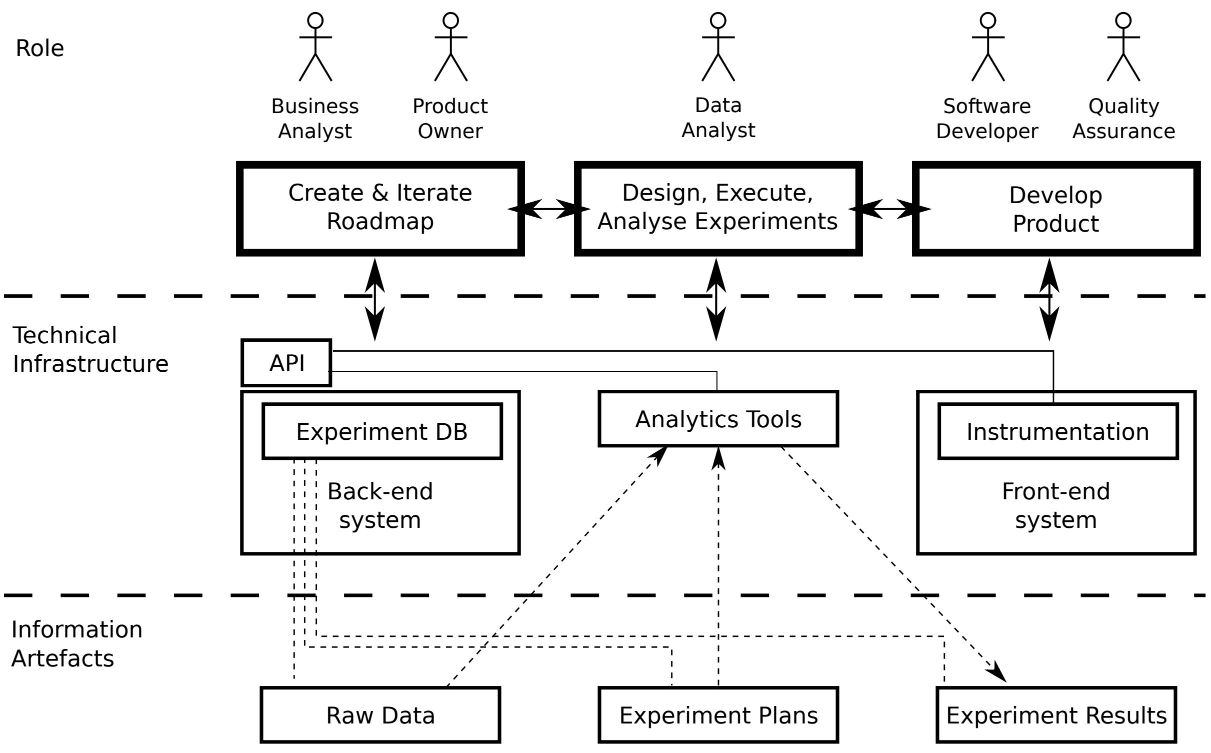

Figure 3: Continuous Experimentation Infrastructure.

accessible through an API. Here, these parts should be understood as conceptual; an actual system likely consists of multiple APIs, databases, servers, etc.

4.2 Lessons Learned from the Projects

Startup companies operate in volatile markets and under high uncertainty. They may have to do several quick changes as they get feedback from the market. The challenge is to reach product-market fit before running out of money.

> "You have to be flexible because of money, time and technology constraints. The biggest question for us has been how to best use resources we have to achieve our vision. In a startup, you are time-constrained because you have a very limited amount of money. So you need to use that time and money very carefully." (Tellybean founder)

When making changes in the direction of the company, it is necessary to base decisions on sound evidence rather than guesswork. However, we found that it is typically not the product or service vision that needs to change. The change should rather concern the strategy by which the vision is implemented, including the features that should be implemented, their design, and the technological platform on which the implementation is based. Although Tellybean has had to adapt several times, the main vision of the company has not changed.

> "The vision has stayed the same: lifelike video calling on your TV. It is very simple; everyone in the company knows it. The TV part doesn't change, but the business environment is changing. The technology – the hardware and software – is changing all the time." (Tellybean founder)

> "We had to pivot when it comes to technology and prioritising features. But the main offering is still the same: it's the new home phone and it connects to your TV. That hasn't changed. I see the pivots more like springboards to the next level. For example, we made a tablet version to [gain a distributor partner]." (Tellybean CTO)

In the first project, the new business analytics instrument allowed Tellybean to yield insights on their system's statistics, providing the company a means for feedback. They could gain a near-real-time view on call related activities, yielding business critical information for deeper analysis. The presence of the call data could be used as input for informed decisions. It also allowed learning about service quality and identifying customer call behaviour patterns. Based on the customer's comments, such information would be crucial for decision-making regarding the scaling of the platform. Excess capacity could thus be avoided and the system would be more profitable to operate while still maintaining a good service level for end users. The primary reason for wanting to demonstrate such capabilities was the need to satisfy operator needs. To convince operators to become channel partners, the ability to respond to fluctuations in call volumes was identified as critical. Potential investors would be more inclined to invest in a company that could convince channel operators of the technical viability of the service.

> "There were benefits in terms of learning. We were able to show things to investors and other stakeholders. We could show them examples of metric data even if it was just screenshots." (Tellybean CTO)

The high-level goal of the first project could be considered as defining a business hypothesis to test the business model from the viewpoint of the operators. The project delivered the needed metrics as well as a tool-supported infrastructure to gather the necessary data. These results could be used to set up an experiment to test the business hypotheses.

In the second project, Tellybean was able to learn the limitations of the current proof-of-concept system and its architecture. An alternative call mediator server and an alternative architecture for the system were very important important for the future development of the service. The lessons learned in the second project, combined with the results of the first, prompted them to pivot heavily regarding the technology, architectural solutions, and development methodology.

> "The Software Factory project [...] put us on the path of 'Lego software development', building software out of off-the-shelf, pluggable components. It got us thinking about what else we should be doing differently. [...] We were thinking about making our own hardware. We had a lot of risk and high expenses. Now we have moved to existing available hardware. Instead of a client application approach, we are using a web-based platform. This expands the possible reach of our offering. We are also looking at other platforms. For example, Samsung just released a new SDK for Smart TVs." (Tellybean founder)

> "Choosing the right Android-based technology platform has really sped things up a lot. We initially tried to do the whole technology stack from hardware to application. The trick is to find your segment in the technology stack, work there, and source the rest from outside. We have explored several Android-based options, some of which were way too expensive. Now we have started to find ways of doing things that give us the least amount of problems. But one really important thing is that a year ago, there were no Android devices like this. Now there are devices that can do everything we need. So the situation has changed a lot." (Tellybean CTO)

The high-level goals of the second project could be considered as defining and testing a solution hypotheses that addresses the feasibility of the proposed hardware-software solution. The project delivered an evaluation of the technical solution as well as improvement proposals. These results were used by the company to modify their strategy.

In the third project, the capability for continuous deployment was developed. The STBs could be updated remotely, allowing new features to be pushed to customers at very low cost and with little effort. The implications of this capability are that the company is able to react to changes in their technological solution space by updating operating system and application software, and to emerging customer needs by deploying new features. The high-level goals of the third project could be considered as developing a capability that allows for automating the continuous deployment process. The prerequisite for this is a steady and controlled pace of development where the focus is on managing the amount of work items that are open concurrently in order to limit complexity. At Tellybean, this is known as the concept of one-piece flow.

> "The one-piece flow means productisation. In development, it means you finish one thing before moving on to the next. It's a bit of a luxury in development, but since we have a small team, it's possible. On the business side, the most important thing has been to use visual aids for business development and for prioritising. In the future we might try to manage multiple-piece flows." (Tellybean founder)

5. DISCUSSION

The continuous experimentation model developed in the previous section can be seen as a general description. Many variations are possible. For instance, experiments may be deployed to selected customers in a special test environment, and several experiments may be run in parallel. A special test environment may be needed particularly in business-to-business markets, where the implications of feature changes are broad and there may be reluctance towards having new features at all. The length of the test cycle may thus have to be longer in business-to-business markets. Direct deployment could be more suitable for consumer markets, but we note that the attitude towards continuous experimentation is likely to change as both business and consumer customers become accustomed to it.

Having several experiments run in parallel presents a particular challenge. The difficulty of interpreting online experiments has been convincingly demonstrated by Kohavi et al. [10]. Statistical interactions between experiments should be considered in order to assess the trustworthiness of the experiments. For this reason, it is important to coordinate the design and execution of experiments so that correct inferences are drawn.

Other challenges include the difficulty of prioritising where to start: which assumption should be tested first. We see a need for further research into this area. Also, in hardware-software co-design, setting up the experimental cycle quickly is a major challenge due to both the longer release cycle of hardware and the potential synchronisation problems between hardware and software development schedules. Based on the case presented in this paper, it may be beneficial to test a few strategic technical assumptions first, such as the viability of a certain hardware-software platform. As our case demonstrates, choosing the correct platform early can have a significant impact on the ability to proceed to actual service development.

A further set of challenges have to do with the model of sales and supplier networks. Essentially all companies are dependent on a network of suppliers and sales channels. It may be necessary to extend the model presented here to take into account the capabilities particularly of hardware suppliers to supply the needed components in a timely fashion and with the needed flexibility to programmatically vary behavioural parameters in these components. Also, when the company is not selling its products directly to end users, several levels of intermediaries may interfere with the possibilities to collect data directly from field use. If a sales partner cannot grant access to end users, other means of reaching the audience are needed. We envision using early-access and beta-test programs for this purpose, a practice that is commonly used in the computer gaming industry. Other models are possible, and there is an opening for further research in this area.

In some cases, an experimental approach may not be suitable at all. For example, certain kinds of life-critical software or software that is used in environments where experimentation is prohibitively expensive, may preclude the use of experiments as a method of validation. However, it is not clear how to determine the suitability of an experimental approach in specific situations, and research on this topic could yield valuable guidelines on when to apply the model presented here.

Finally, experimentation may be conducted with several kinds of stakeholders. Apart from customers and end users, experiments could be directed towards investors, suppliers, sales channels, or distributors. Companies whose product is itself a development platform may want to conduct experiments with developers in their platform ecosystem. These experiments may require other kinds of experimental artefacts than the MVP/MVF. Research on the types of experimental artefacts and associated experimental designs could lead to fruitful results for such application areas.

6. CONCLUSIONS

Companies are increasingly transitioning their traditional research and product development functions towards continuous experiment systems [9]. Integrating field experiments with product development on business and technical levels is an emerging challenge. There are reports of many companies successfully conducting online experiments, but there is a lack of a systematic framework model for describing how such experiments should be carried out and used systematically in product development. Empirical studies on the topic of continuous experimentation in software product development is a fruitful ground for further research. Software companies would benefit from clear guidelines on when and how to apply continuous experimentation in the design and development of software-intensive products and services.

In this paper, we match a model for Continuous Experiementation based on analysis of previous research against a case study in the Software Factory laboratory at the University of Helsinki. The model describes the experimental cycle, in which assumptions for product and business development are derived from the business strategy, systematically tested, and the results used to inform further development of the strategy and product. The infrastructure for supporting the model takes into account the roles, technical infrastructure, and information artefacts needed to run large-scale continuous experiments.

A system for continuous experimentation requires the ability to release minimum viable products or features with suitable instrumentation, design and manage experiment plans, link experiment results with a product roadmap, and manage a flexible business strategy. There are several critical success factors for such a system. The organisation must be able to properly and rapidly design experiments, perform advanced instrumentation of software to collect, analyse, and store relevant data, and integrate experiment results in both the product development cycle and the software development process. Feedback loops must exist through which relevant information is fed back from experiments into several parts of the organisation. A proper understanding of what to test and why must exist, and the organisation needs a workforce with the ability to collect and analyse qualitative and quantitative data. Also, it is crucial that the organisation has the ability to properly define decision criteria and act on data-driven decisions.

In future work, we expect the model to be expanded as more use cases arise in the field. Domain-specific variants of the model may also be needed. Furthermore, there are many particular questions with regard to the individual parts of the model. Some specific areas include 1) how to build a back-end system for continuous experimentation that can scale to the needs of very large deployments, and can facilitate and even partially automate the creation of experimental plans; 2) how to properly design experiments in order to reduce uncertainty in strategic assumptions; and 3) how to ensure that experiments are trustworthy when running potentially thousands of them in parallel. Particular questions regarding automation include which parts of the model could be automated or supported through automation. Another question is how quickly a Build-Measure-Learn block can be executed, and what the performance impact of the model is on the software development process.

7. REFERENCES

[1] R. J. Adams, B. Evans, and J. Brandt. Creating Small Products at a Big Company: Adobe's Pipeline Innovation Process. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 2331–2332. ACM, 2013.

[2] V. Basili, J. Heidrich, M. Lindvall, J. Münch, M. Regardie, D. Rombach, C. Seaman, and A. Trendowicz. GQM^{+}Strategies: A comprehensive methodology for aligning business strategies with software measurement. In *Proceedings of the DASMA Software Metric Congress (MetriKon 2007): Magdeburger Schriften zum Empirischen Software Engineering*, pages 253–266, 2007.

[3] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[4] J. Bosch. Building Products as Innovation Experiment Systems. In *Software Business*, pages 27–39. Springer, 2012.

[5] J. Bosch, H. Holmström Olsson, J. Björk, and J. Ljungblad. The Early Stage Software Startup Development Model: A Framework for Operationalizing Lean Principles in Software Startups. In *Lean Enterprise Software and Systems*, pages 1–15. Springer, 2013.

[6] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. *Information and software technology*, pages 833–859, 2008.

[7] F. Fagerholm, N. Oza, and J. Münch. A platform for teaching applied distributed software development: The ongoing journey of the Helsinki software factory. In *3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development (CTGDSD)*, pages 1–5, 2013.

[8] A. R. Hevner, S. T. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[9] H. Holmström Olsson, H. Alahyari, and J. Bosch. Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. *39th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 392–399, 2012.

[10] R. Kohavi, A. Deng, B. Frasca, R. Longbotham, T. Walker, and Y. Xu. Trustworthy Online Controlled Experiments: Five Puzzling Outcomes Explained. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 786–794, New York, NY, USA, 2012. ACM.

[11] C. Larman and V. R. Basili. Iterative and incremental developments. a brief history. *IEEE Computer*, pages 47–56, 2003.

[12] B. May. Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran: Lessons Learned in Creating & Launching a Complex Consumer App. In *Agile Conference (AGILE) 2012*, pages 141–147. IEEE, 2012.

[13] J. Münch, F. Fagerholm, P. Johnson, J. Pirttilahti, J. Torkkel, and J. Järvinen. Creating Minimum Viable Products in Industry-Academia Collaborations. In *Proceedings of the Lean Enterprise Software and Systems Conference (LESS 2013, Galway, Ireland, December 1-4)*, pages 137–151. Springer Berlin Heidelberg, 2013.

[14] J. Münch, F. Fagerholm, P. Kettunen, M. Pagels, and J. Partanen. Experiences and Insights from Applying GQM+Strategies in a Systems Product Development Organisation. In *Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2013)*, 2013.

[15] J. Nieters and A. Pande. Rapid Design Labs: A Tool to Turbocharge Design-led Innovation. *Interactions*, pages 72–77, 2012.

[16] T. Ōno. *Toyota production system: beyond large-scale production*. Productivity press, 1988.

[17] M. Poppendieck. *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

[18] M. Poppendieck and M. A. Cusumano. Lean Software Development: A Tutorial. *IEEE Software*, pages 26–32, 2012.

[19] E. Ries. *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation To Create Radically Successful Businesses*. Crown Business, 2011.

[20] K. Schwaber and M. Beedle. *Agile software development with Scrum*. Prentice Hall Upper Saddle River, 2002.

[21] D. Ståhl and J. Bosch. Modeling Continuous Integration Practice Differences in Industry Software Development. *Journal of Systems and Software*, pages 48–59, 2014.

[22] J. Stapleton. *DSDM: Business Focussed Development*. Pearson Education, 2003.

[23] A. Steiber and S. Alänge. A Corporate System for Continuous Innovation: The case of Google Inc. *European Journal of Innovation Management*, pages 243–264, 2013.

[24] J. E. van Aken. Management Research Based on the Paradigm of the Design Sciences: The Quest for Field-Tested and Grounded Technological Rules. *Journal of Management Studies*, 41(2):219–246, 2004.