

The RIGHT model for Continuous Experimentation

Fabian Fagerholm^{a,*}, Alejandro Sanchez Guinea^b, Hanna Mäenpää^a, Jürgen Münch^{a,c}

^a*Department of Computer Science, University of Helsinki, P.O. Box 68, FI-00014 University of Helsinki, Finland*

^b*University of Luxembourg, 4 rue Alphonse Weicker, L-2721, Luxembourg*

^c*Faculty of Informatics, Reutlingen University, Alteburgstraße 150, D-72762 Reutlingen, Germany*

Abstract

Context: 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 must continuously find out what customers want by direct customer feedback and usage behaviour observation.

Objective: This paper examines the preconditions for setting up an experimentation system for continuous customer experiments. It describes the RIGHT model for Continuous Experimentation (Rapid Iterative value creation Gained through High-frequency Testing), illustrating the building blocks required for such a system. **Method:** An initial model for continuous experimentation is analytically derived from prior work. The model is matched against empirical case study findings from two startup companies and further developed. **Results:** Building blocks for a continuous experimentation system and infrastructure are presented. **Conclusions:** 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, 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.

*Corresponding author

Email addresses: `fabian.fagerholm@helsinki.fi` (Fabian Fagerholm),
`alejandro.sanchezguinea@uni.lu` (Alejandro Sanchez Guinea), `hanna.maenpaa@cs.helsinki.fi`
(Hanna Mäenpää), `juergen.muench@cs.helsinki.fi`,
`juergen.muench@reutlingen-university.de` (Jürgen Münch)

Keywords: Continuous experimentation, Product development, Software architecture, Software development process, Agile software development, Lean software development,

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 delivered to customers. Many constraints that previously existed, particularly in terms of the behaviour of a product or service, can now be removed.

With this newfound 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 has been proposed. For example, the Lean Startup methodology [26] 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.

Methods and approaches for 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:

26 **RQ** How can Continuous Experimentation with software-intensive products and ser-
27 vices be organised in a systematic way?

28 To further scope the question, we split it into two sub-questions:

29 **RQ1** What is a suitable process model for Continuous Experimentation with software-
30 intensive products and services?

31 **RQ2** What is a suitable infrastructure architecture for Continuous Experimentation
32 with software-intensive products and services?

33 We give an answer to the research questions by validating an analytically derived
34 model against a series of case studies in which we implemented different parts of the
35 model in cooperation with two startup companies. The result is the RIGHT model
36 for Continuous Experimentation (Rapid Iterative value creation Gained through High-
37 frequency Testing). This model focuses on developing the *right* software, whereas the
38 typical focus of software engineering in the past has been on developing the software
39 right (e.g. in terms of technical quality). The model is instantiated in the RIGHT process
40 model and the RIGHT infrastructure architecture model. Together, these instantiations
41 address the need to integrate the requirements, design, implementation, testing, deploy-
42 ment, and maintenance phases of software development in a way that uses continuous
43 empirical feedback from users.

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

52 **2. Related work**

53 Delivering software that has value – utility for its users – can be considered a primary
54 objective for software development projects. In this section, we describe models for
55 systematic value delivery and approaches for using experiments as a means for value
56 testing and creation. In addition, we discuss related work with respect to experiments at
57 scale.

58 *2.1. Models for systematic value delivery*

59 Lean manufacturing and the Toyota Production System [22] has inspired the defini-
60 tion of Lean software development. This approach provides comprehensive guidance
61 for the combination of design, development, and validation built as a single feedback
62 loop focused on discovery and delivery of value [25]. The main ideas of this approach,
63 which have been emphasised since its introduction, are summarised in seven principles:
64 optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast,
65 engage everyone, and keep getting better [24].

66 Lean Startup [26] provides mechanisms to ensure that product or service develop-
67 ment effectively addresses what customers want. The methodology is based on the
68 Build-Measure-Learn loop that establishes learning about customers and their needs as
69 the unit of progress. It proposes to apply scientific method and thinking to startup busi-
70 nesses in the form of learning experiments. As the results of experiments are analysed,
71 the company has to decide to “persevere” on the same path or “pivot” in a different
72 direction while considering what has been learned from customers.

73 Customer Development [4] emphasises the importance of not only doing product
74 development activities but also to learn and discover who a company’s initial customers
75 will be, and what markets they are in. Customer Development argues that a separate
76 and distinct process is needed for those activities. Customer Development is a four-
77 step model divided into a search and an execution phase. In the search phase, a
78 company performs *customer discovery*, testing whether the business model is correct
79 (product/market fit), and *customer validation*, which develops a replicable sales model.
80 In the execution phase, *customer creation* focuses on creating and driving demand, and

81 *company building* is the transition from an organisation designed to learn and discover
82 to one that is optimised for cost-efficient delivery of validated products or services.

83 In light of the benefits that a methodology such as Lean Startup can provide, where
84 controlled experiments constitute the main activity driving development, Holmström
85 Olsson et al. [12] propose a target stage for any company that wishes to build a develop-
86 ment system with the ability to continuously learn from real-time customer usage of
87 software. They describe the stages that a company has to traverse in order to achieve that
88 target as the “stairway to heaven”. The target stage is achieved when the software organ-
89 isation functions as an R&D experiment system. The stages on the way to achieving
90 the target are: (i) traditional development, (ii) agile R&D organisation, (iii) continuous
91 integration, and (iv) continuous deployment. The authors first describe these four stages
92 and then analyse them through a multiple-case study that examines the barriers that
93 exist on each step on the path towards continuous deployment. The target stage is
94 only described; the authors do not detail any means to overcome the barriers. A main
95 finding from the case study is that the transition towards Agile development requires
96 shifting to small development teams and focusing on features rather than on compo-
97 nents. Also, it is relevant to notice that the transition towards continuous integration
98 requires an automated build and test system (continuous integration), a main version
99 control branch to which code is continuously delivered, and modularised development.
100 Holmström Olsson et al. found that in order to move from continuous integration to
101 continuous deployment, organisational units such as product management must be fully
102 involved, and close work with a very active lead customer is needed when exploring
103 the product concept further. The authors suggest two key actions to make the transition
104 from continuous deployment to an R&D experiment system. First, the product must be
105 instrumented so that field data can be collected in actual use. Second, organisational
106 capabilities must be developed in order to effectively use the collected data for testing
107 new ideas with customers.

108 Other works have studied some of the stages of the “stairway to heaven” individually.
109 Ståhl & Bosch [27] have studied the continuous integration stage, pointing out that there
110 is no homogeneous practice of continuous integration in the industry. They propose
111 a descriptive model that allows studying and evaluating the different ways in which

112 continuous integration can be viewed. Eklund & Bosch [7] present an architecture that
113 supports continuous experimentation in embedded systems. They explore the goals of
114 an experiment system, develop experiment scenarios, and construct an architecture that
115 supports the goals and scenarios. The architecture combines an experiment repository,
116 data storage, and software to be deployed on embedded devices via over-the-air data
117 communication channels. The architecture also considers the special requirements
118 for safety in, e.g., automotive applications. However, the main type of experiment is
119 confined to A/B testing, and the architecture is considered mainly from the perspective
120 of a software development team rather than a larger product development organisation.

121 Holmström Olsson & Bosch [13] describe the Hypothesis Experiment Data-Driven
122 Development (HYPEX) model. The goal of this model is to shorten the feedback loop
123 to customers. It consists of a loop where potential features are generated into a feature
124 backlog, features are selected and a corresponding expected behaviour is defined. The
125 expected behaviour is used to implement and deploy a minimum viable feature (MVF).
126 Observed and expected behaviour is compared using a gap analysis, and if a sufficiently
127 small gap is identified, the feature is finalised. On the other hand, if a significant gap is
128 found, hypotheses are developed to explain it, and alternative MVFs are developed and
129 deployed, after which the gap analysis is repeated. The feature may also be abandoned
130 if the expected benefit is not achieved.

131 2.2. *Systematic value creation through experimentation*

132 The models outlined above all aim to make experimentation systematic in the
133 software development organisation. One important conceptual concern is the definition
134 of experimentation. Experimentation has been established in software engineering since
135 the 1980s. Basili et al. [3] were among the first to codify a framework and process for
136 experimentation. Juristo et al. [14] and Wohlin et al. [31] present more recent syntheses
137 regarding experimentation in software engineering. Taken together, these works show
138 that “experimentation” in software engineering can be considered in a broad sense,
139 including both controlled experiments but also more explorative activities which aim
140 at understanding and discovery rather than hypothesis testing. For the purposes of
141 this article, we consider experimentation to be a range of activities that can be placed

142 within a spectrum including controlled experiments as well as open-ended exploration.
143 However, we emphasise that regardless of the placement within this spectrum, all
144 methods require rigorous study designs and have a defensible and transparent way
145 of reasoning and drawing conclusions from empirical data. They are not the same
146 method being applied more or less carefully. The logic of controlled experiments relies
147 on careful manipulation of variables, observation of effects, and analysis to test for
148 causal relationships. Quasi-controlled experiments relax some of the requirements
149 for randomised treatment. Case studies often include qualitative elements and their
150 logic is different from controlled experiments: they generalise analytically rather than
151 statistically [32]. Qualitative methods may also be used alone, such as through interview-
152 or observation-based studies.

153 Experimentation may also be considered in terms of goals, and goals may exist
154 on different levels of the product development organisation. On the product level,
155 experimentation may be used to select features from a set of proposed features. On the
156 technical level, experimentation may be used to optimise existing features. However,
157 the model presented in this paper links experimentation on the product and technical
158 level to the product vision and strategy on the business level. Experimentation becomes
159 a systemic activity that drives the entire organisation. This allows for focused testing of
160 business hypotheses and assumptions, which can be turned into faster decision-making
161 and reaction to customer needs. Depending on the specific method used, the results
162 of an experiment may suggest new information which should be incorporated into the
163 decision-making process.

164 *2.3. Considerations for running experiments at a large scale*

165 Previous works have presented case studies that exhibit different aspects concerning
166 continuous experimentation. Steiber [28] report on a study of the continuous experimen-
167 tation model followed by Google, analysing a success story of this approach. Tang et
168 al. [29] describe an overlapping experiment infrastructure, developed at Google, that
169 allows web queries in a search engine to be part of multiple experiments, thus allowing
170 more experiments to be carried out at a faster rate. Adams [1] present a case study

171 on the implementation of Adobe’s Pipeline, a process that is based on the continuous
172 experimentation approach.

173 Kohavi et al. [16, 17] note that running experiments at large scale requires ad-
174 dressing multiple challenges in three areas: cultural/organisational, engineering, and
175 trustworthiness. The larger organisation needs to learn the reasons for running controlled
176 experiments and the trade-offs between controlled experiments and other methods of
177 evaluating ideas. Even negative experiments should be run, which degrade user experi-
178 ence in the short term, because of their learning value and long-term benefits. When
179 the technical infrastructure supports hundreds of concurrent experiments, each with
180 millions of users, classical testing and debugging techniques no longer apply because
181 there are millions of live variants of the system in production. Instead of heavy up-front
182 testing, Kohavi et al. report having used alerts and post-deployment fixing. The system
183 has also identified many negative features that were avoided despite being advocated by
184 key stakeholders, saving large amounts of money.

185 Experimentation also has an important relationship with company culture. Kohavi
186 et al. [15] describe a platform for experimentation built and used at Microsoft, noting
187 the cultural challenges involved in using experiment results, rather than opinions from
188 persons in senior positions, as the basis of decisions. They suggest, for example, that one
189 should avoid trying to build features through extensive planning without early testing of
190 ideas, that experiments should be carried out often, that a failed experiment is a learning
191 opportunity rather than a mistake, and that radical and controversial ideas should be
192 tried. All these suggestions are challenging to put into practice in organisations that are
193 not used to experimentation-based decision-making. Kohavi et al. note the challenges
194 they faced at Microsoft, and describe efforts to raise awareness of the experimentation
195 approach.

196 The final stage of the “stairway to heaven” model is detailed and analysed by
197 Bosch [5]. The differences between traditional development and the continuous ap-
198 proach are analysed, showing that in the context of the new, continuous software
199 development model, R&D is best described as an “innovation experiment system” ap-
200 proach where the development organisation constantly develops new hypotheses and
201 tests them with certain groups of customers. This approach focuses on three phases:

202 pre-deployment, non-commercial deployment, and commercial deployment. The au-
203 thors present a first systematisation of this so-called “innovation experiment system”
204 adapted for software development for embedded systems. It is argued that aiming for
205 an “innovation experiment system” is equally valid for embedded systems as it is in the
206 case of cloud computing and Software-as-a-Service (SaaS), and that the process could
207 be similar in both cases. That is, requirements should evolve in real time based on data
208 collected from systems in actual use with customers.

209 Inspired by the ideas that define the last stage of the “stairway to heaven”, we
210 develop and propose the RIGHT model for Continuous Experimentation. In this model,
211 experiments are derived from business strategies and aim to assess assumptions derived
212 from those strategies, potentially invalidating or supporting the strategy. Previous works
213 have explored the application of a framework for linking business goals and strategies
214 to the software development activities (e.g., [2], [20]). However, those works have
215 not considered the particular traits of an experiment system such as the one presented
216 in this paper. The model presented also describes the platform infrastructure that is
217 necessary to establish the whole experiment system. The Software Factory [8] can serve
218 as infrastructure for the model proposed, as it is a software development laboratory well
219 suited for continuous experimentation. In a previous article, in which we presented a
220 study on creating minimum viable products [19] in the context of collaboration between
221 industry and academia, we showed the Software Factory laboratory in relation to the
222 Lean Startup approach and continuous experimentation. Some of the foundational ideas
223 behind Software Factory with respect to continuous experimentation have been studied
224 in the past, analysing, for instance, the establishment of laboratories specifically targeted
225 for continuous development [21] and the impact of continuous integration in teaching
226 software engineering.

227 The building blocks presented in this paper, although generalizable with certain
228 limitations, are derived from a startup environment where the continuous experimenta-
229 tion approach is not only well suited but possibly the only viable option for companies
230 to grow. Our work has similarities to the “Early Stage Startup Software Development
231 Model” (ESSSDM) of Bosch et al. [6] which extends existing Lean Startup approaches
232 offering more operational process support and better decision-making support for startup

233 companies. Specifically, ESSSDM provides guidance on when to move product ideas
234 forward, when to abandon a product idea, and what techniques to use and when, while
235 validating product ideas. Some of the many challenges faced when trying to establish a
236 startup following the Lean Startup methodology are presented by May [18] with insights
237 that we have considered for the present work.

238 **3. Research approach**

239 Our general research framework can be characterised as design science research [11],
240 in which the purpose is to derive a technological rule which can be used in practice to
241 achieve a desired outcome in a certain field of application [30]. The continuous experi-
242 mentation model presented in this paper was first constructed based on the related work
243 presented in the previous section as well the authors' experience. While a framework
244 can be derived by purely analytic means, its validation requires grounding in empirical
245 observations. For this reason, we conducted a holistic multiple case study [32] in the
246 Software Factory laboratory at the Department of Computer Science, University of
247 Helsinki, in which we matched the initial model to empirical observations and made
248 subsequent adjustments to produce the final model. The model can still be considered
249 tentative, pending further validation in other contexts. It is important to note that this
250 study investigates how Continuous Experimentation can be carried out in a systematic
251 way independently of the case projects' goals and the experiments carried out in them.
252 Those experiments and their outcomes are treated as qualitative findings in the context
253 of this study. In this section, we describe the case study context and the research process.

254 *3.1. Context*

255 The Software Factory is an educational platform for research and industry collabora-
256 tion [8]. In Software Factory projects, teams of Master's-level students use contemporary
257 tools and processes to deliver working software prototypes in close collaboration with
258 industry partners. The goal of Software Factory activities is to provide students with
259 means for applying their advanced software development skills in an environment with
260 working life relevance and to deliver meaningful results for their customers [19].

261 During the case projects used in this study, two of the authors were involved as
262 participant observers. The first author coordinated the case projects: started the projects,
263 handled contractual and other administrative issues, followed up progress through direct
264 interaction with the customer and student teams, ended the projects, handled project
265 debriefing and coordinated the customer interviews. The third author also participated
266 as an observer in several meetings where the customer and student teams collaborated.
267 The researchers were involved in directing the experimentation design activities together
268 with the customer, and students were not directly involved in these activities. However,
269 the customer and students worked autonomously and were responsible for project
270 management, technical decisions, and other issues related to the daily operations of the
271 project.

272 3.1.1. Case Company 1

273 Tellybean Ltd.¹ is a small Finnish startup that develops a video calling solution for
274 the home television set. During September 2012–December 2013 the company was a
275 customer in three Software Factory projects with the aim of creating an infrastructure to
276 support measurement and management of the architecture of their video calling service.
277 Tellybean Ltd. aims at delivering a life-like video calling experience. Their value
278 proposition – “the new home phone as a plug and play -experience” – is targeted at late
279 adopter consumer customers who are separated from their families, e.g. due to migration
280 into urban areas, global social connections, or overseas work. The company puts special
281 emphasis on discovering and satisfying needs of the elderly, making ease of use the most
282 important non-functional requirement of their product. The primary means for service
283 differentiation in the marketplace are affordability, accessibility and ease of use. For
284 the première commercial launch, and to establish the primary delivery channel of their
285 product, the company aims at partnering with telecom operators. The company had made
286 an initial in-house architecture and partial implementation during a pre-development
287 phase prior to the Software Factory projects. A first project was conducted to extend
288 the platform functionality of this implementation. A second project was conducted to

¹<http://www.tellybean.com/>

289 validate concerns related to the satisfaction of operator requirements. After this project, a
290 technical pivot was conducted, with major portions of the implementation being changed;
291 the first two projects contributed to this decision. A third project was then conducted
292 to extend the new implementation with new features related to the ability to manage
293 software on already delivered products, enabling continuous delivery. The launch
294 strategy can be described as an MVP launch with post-development adaptation. The
295 three projects conducted with this company are connected to establishing a continuous
296 experimentation process and building capabilities to deliver software variations on
297 which experiments can be conducted. They also provided early evidence regarding the
298 feasibility of the product for specific stakeholders, such as operator partners, developers,
299 and release management.

300 *3.1.2. Product*

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

308 The company played the role of a product owner in three Software Factory projects
309 during September 2012–December 2013. The aim of the first two projects was to create
310 new infrastructure for measuring and analysing usage of their product in its real envi-
311 ronment. This information was important in order to establish the product's feasibility
312 for operators and for architectural decisions regarding scalability, performance, and
313 robustness. For the present research, the first two projects were used to validate the steps
314 required to establish a continuous experimentation process. The third project at Software
315 Factory delivered an automated system for managing and updating the STB software
316 remotely. This project was used to investigate factors related to the architecture needs
317 for continuous experimentation. Table 1 summarises the goals and motivations of the

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.	<p>... so that I can extract and analyse business critical information.</p> <p>... so that I can identify needs for maintenance of the product's technical architecture.</p>
Project 2	<p>As a Tellybean developer, I want to be sure that our product's system architecture is scalable and robust.</p> <p>As a Tellybean developer, I want to know technical weaknesses of the system.</p> <p>As a Tellybean developer, I want to receive suggestions for alternative technical architecture options.</p>	<p>... so that I know the limitations of the system.</p> <p>... so that I can predict needs for scalability of the platform.</p> <p>... so that I can consider future development options.</p>
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.

318 projects in detail. Each project had a 3–7 -person student team, a company representative
319 accessible at all times, and spent effort in the range of 600 and 700 person-hours.

320 *3.1.3. Project 1*

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

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

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

340 The first project constituted a first attempt at conducting continuous experimentation.
341 The goal of the experiment was to gain information about the performance of the system
342 architecture and its initial implementation. The experiment arose from operator needs
343 to monitor call volumes and system load – a requirement that Tellybean’s product
344 developers deemed necessary to be able to partner with operators. It was clear that there
345 existed a set of needs arising from operator requirements, but it was not clear how the
346 information should be presented and what functionality was needed to analyse it. From

347 a research perspective, however, the exact details of the experiment were less important
348 than the overall process of starting experimentation.

349 *3.1.4. Project 2*

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

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

360 The purpose of the experiment was a counterpart to the experiment in the first
361 project. Whereas the first project had focused on operator needs, the second focused
362 on their implications for developers. The initial system architecture and many of the
363 technical decisions had been questioned. The project aimed to provide evidence for
364 decision-making when revisiting these initial choices.

365 The team found significant performance bottlenecks in Tellybean's existing proof-of-
366 concept system and analysed their origins. Solutions for increasing operational capacity
367 of the current live system were proposed and some of them were also implemented.
368 Towards the end of the project, the customer suggested that a new proof-of-concept
369 call mediating server should be proposed by the Software Factory team. The team
370 delivered several suggestions for a new service architecture and composed a new call
371 mediator server. For the purposes of this study, we consider the second experiment to
372 be another round in the continuous experimentation cycle where findings from the first
373 cycle resulted in a new set of questions to experiment on.

374 *3.1.5. Project 3*

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

382 The project consisted of a team of five Master’s-level computer science students.
383 The team delivered a working prototype for rapid deployment of software updates. In
384 this project, the need for a support system to deliver new features or software variations
385 was addressed. We considered the architectural requirements for a continuous delivery
386 system that would support continuous experimentation.

387 *3.1.6. Case Company 2*

388 Memory Trails Ltd. (Memory Trails) is a small Finnish startup that develops a
389 well-being service which helps users define, track, and receive assistance with life goals.
390 During May–July 2014, the company was a customer in a Software Factory project
391 that aimed to develop a backend recommendation engine for the service, improve the
392 front-end user experience, and to validate central assumptions in the service strategy.
393 Memory Trails aims at delivering the service as an HTML5-based application which
394 is optimised for tablets but also works on other devices with an HTML5-compatible
395 browser. The service targets adults who wish to improve their quality of life and change
396 patterns of behaviour to reach different kinds of life goals.

397 Whereas the projects with the first case company focused mostly on establishing
398 a continuous experimentation process and building capabilities to deliver software
399 variations for experimentation, the project with the second case company focused on
400 some of the details of deriving experiments themselves. In particular, we sought to
401 uncover how assumptions can be identified in initial product or service ideas. These
402 assumptions are candidates for experiments of different kinds.

403 3.1.7. *Project 4*

404 Memory Trails provided an initial user interface and backend system prototype
405 which demonstrated the general characteristics of the application from a user perspective.
406 Users interact with photos which can be placed in different spatial patterns to depict
407 emotional aspects of their goals. Users are guided by the application to arrange the
408 photos as a map, showing the goal, potential steps towards it, and aspects that qualify
409 the goals. For example, a life goal may be to travel around the world. Related photos
410 could depict places to visit, moods to be experienced, items necessary for travel such as
411 tickets, etc. The photos could be arranged, e.g., as a radial pattern with the central goal
412 in the middle, and the related aspects around it, or as a time-line with the end goal to the
413 right and intermediate steps preceding it.

414 In the project, two high-level assumptions were identified. The customer assumed
415 that automatic, artificial intelligence-based processing in the backend could be used
416 to automatically guide users towards their goals, providing triggers, motivation, and
417 rewards on the way. Also, the customer assumed that the motivation for continued
418 use of the application would come from interacting with the photo map. Since the
419 automatic processing depended on the motivation assumption, the latter became the
420 focus of experimentation in the project. The customer used versions of the application
421 in user tests during which observation and interviews were used to investigate whether
422 the assumption held. For the purposes of this study, we used the project to validate the
423 link in our model between product vision, business model and strategy, and experiment
424 steps.

425 3.2. *Research process*

426 The case study analysis was performed in order to ground the continuous experi-
427 mentation model in empirical observations, not to understand or describe the projects
428 themselves, nor to assess the business viability of the case companies. Therefore, we
429 collected information that would help us understand the prerequisites for performing
430 continuous experimentation, the associated constraints and challenges, and the logic of
431 integrating experiment results into the business strategy and the development process.

432 We used four different sources of data in our analysis: (i) participant observa-
433 tion, (ii) analysis of project artefacts, (iii) group analysis sessions, and (iv) individual
434 interviews. We subsequently discuss the details of the data collection and analysis.

435 During the projects, we observed the challenges the companies faced related to
436 achieving the continuous experimentation system. At the end of each project, an in-
437 depth debriefing session was conducted to gain retrospective insights into the choices
438 made during the project, and the reasoning behind them. In addition to these sources, we
439 interviewed three company representatives from Tellybean to understand their perception
440 of the projects and to gain data which could be matched against our model. We also
441 conducted a joint analysis session with the project team and two representatives from
442 Memory Trails to further match insights on the experimentation process in their project
443 with our model.

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

454 The interviews with customer representatives were conducted either in person on the
455 customer's premises, online via video conferencing, or on the University of Helsinki's
456 premises. The interviews were semi-structured thematic interviews, having a mixture of
457 open-ended and closed questions. This interview technique allows participants to freely
458 discuss issues related to a focal theme. Thematic interviews have the advantage that
459 they provide opportunities to discover information that researchers cannot anticipate
460 and that would not be covered by more narrowly defined, closed questions. While they
461 may result in the discussion straying away from the focal theme, this is not a problem in

462 practice since the interviewer can direct the participant back to the theme and irrelevant
463 information can be ignored in the analysis.

464 A minimum of two researchers were present in the interviews to ensure that relevant
465 information was correctly extracted. All participating researchers took notes during
466 the interviews, and notes were compared after the interviews to ensure consistency. In
467 the interviews, company representatives were first asked to recount their perception
468 of their company, its goals, and its mode of operation before the three projects. Then,
469 they were asked to consider what each project had accomplished in terms of software
470 outcomes, learned information, and implications for the goals and mode of operation of
471 the company. Finally, they were asked to reflect on how the company operated at the
472 time of the interview and how they viewed the development process, especially in terms
473 of incorporating market feedback into decision-making.

474 During analysis, the project data were examined for information relevant to the
475 research question. We categorised the pieces of evidence according to whether they
476 related to the Continuous Experimentation process or to the infrastructure. We sought to
477 group the observations made and understanding gained during the projects with evidence
478 from the retrospective sessions and interviews so that the evidence was triangulated
479 and thus strengthened. Such groups of triangulated evidence was then matched with
480 our initial model, which was similar to the sequence shown in Figure 1, and included
481 the build-measure-learn cycle for the process, and a data repository, analysis tools, and
482 continuous delivery system as infrastructure components. We adjusted the model and
483 introduced new process steps and infrastructure components that supported the need
484 implied by the evidence. We strived for minimal models, and when more than one need
485 could be fulfilled with a single step or component, we did not introduce more steps or
486 components. When all the evidence had been considered, we evaluated the result as a
487 whole and made some adjustments and simplifications based on our understanding and
488 judgement.

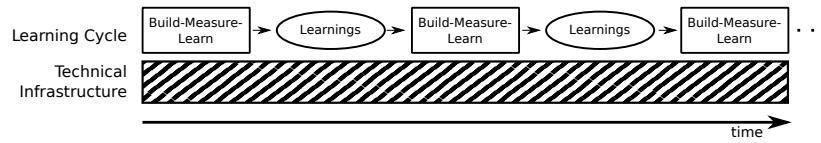


Figure 1: Sequence of RIGHT process blocks.

489 **4. Results**

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

493 *4.1. The RIGHT model for Continuous Experimentation*

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

503 The Build-Measure-Learn blocks structure the activity of conducting experiments,
 504 and connect product vision, business strategy, and technological product development
 505 through experimentation. In other words, the requirements, design, implementation,
 506 testing, deployment, and maintenance phases of software development are integrated
 507 and aligned by empirical information gained through experimentation. The model
 508 can be considered a vehicle for incremental innovation as defined by Henderson and
 509 Clark [10], but the model itself, as well as the transition to continuous experimentation
 510 in general, can be considered radical, architectural innovations that require significant
 511 new organisational capabilities.

512 *4.1.1. The RIGHT process model for Continuous Experimentation*

513 Figure 2 expands the Build-Measure-Learn blocks and describes the RIGHT process
514 model for Continuous Experimentation. A general vision of the product or service is
515 assumed to exist. Following the Lean Startup methodology [26], this vision is fairly
516 stable and is based on knowledge and beliefs held by the entrepreneur. The vision is
517 connected to the business model and strategy, which is a description of how to execute
518 the vision. The business model and strategy are more flexible than the vision, and
519 consist of multiple assumptions regarding the actions required to bring a product or
520 service to market that fulfils the vision and is sustainably profitable. However, each
521 assumption has inherent uncertainties. In order to reduce the uncertainties, we propose
522 to conduct experiments. An experiment operationalises the assumption and states a
523 hypothesis that can be subjected to experimental testing in order to gain knowledge
524 regarding the assumption. The highest-priority hypotheses are selected first. Once a
525 hypothesis is formulated, two parallel activities can occur. The hypothesis can optionally
526 be used to implement and deploy a Minimum Viable Product (MVP) or Minimum Viable
527 Feature (MVF), which is used in the experiment and has the necessary instrumentation.
528 Simultaneously, an experiment is designed to test the hypothesis. The experiment
529 is then executed and data from the MVP/MVF are collected in accordance with the
530 experimental design. The resulting data are analysed, concluding the experimental
531 activities.

532 Once the experiment has been conducted and analysis performed, the analysis
533 results are used on the strategy level to support decision-making. Again following Lean
534 Startup terminology, the decision can be to either “pivot” or “persevere” [26], but a
535 third alternative is also possible: to change assumptions in the light of new information.
536 If the experiment has given support to the hypothesis, and thus the assumption on the
537 strategy level, a full product or feature is developed or optimised, and deployed. The
538 strategic decision in this case is to persevere with the chosen strategy. If, on the other
539 hand, the hypothesis was falsified, invalidating the assumption on the strategy level,
540 the decision is to pivot and alter the strategy by considering the implications of the
541 assumption being false. Alternatively, the tested assumption could be changed, but not

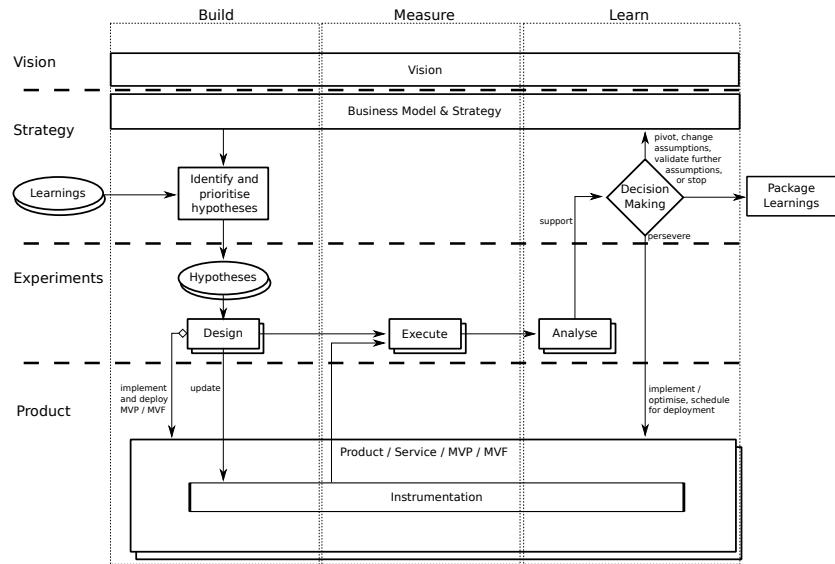


Figure 2: The RIGHT process model for Continuous Experimentation.

542 completely rejected, depending on what the experiment was designed to test and what
 543 the results were.

544 *4.1.2. The RIGHT infrastructure architecture for Continuous Experimentation*

545 To support conducting such experiments, an infrastructure for continuous experimen-
 546 tation is needed. Figure 3 sketches the RIGHT infrastructure architecture for Continuous
 547 Experimentation, with roles and associated tasks, the technical infrastructure, and in-
 548 formation artefacts. The roles indicated here will be instantiated in different ways
 549 depending on the type of company in question. In a small company, such as a startup, a
 550 small number of persons will handle the different roles and one person may have more
 551 than one role. In a large company, the roles are handled by multiple teams. Seven roles
 552 are defined to handle four classes of tasks. A business analyst and a product owner, or a
 553 product management team, together handle the creation and iterative updating of the
 554 strategic roadmap. In order to do so, they consult existing experimental plans, results,
 555 and learnings, which reside in a back-end system. As plans and results accumulate and
 556 are stored, they may be reused in further development of the roadmap. The business

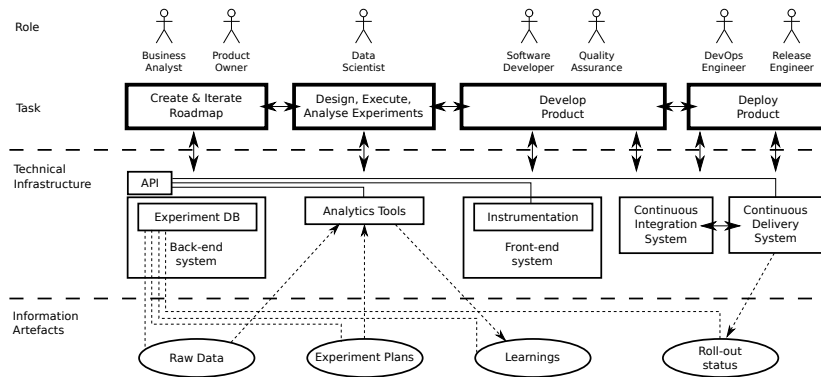


Figure 3: The RIGHT infrastructure architecture for Continuous Experimentation.

557 analyst and product owner work with a data scientist role, which is usually a team with
 558 diverse skills, to communicate the assumptions of the roadmap and map the areas of
 559 uncertainty which need to be tested.

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

564 The data analyst also communicates with a developer and quality assurance role.
 565 These roles handle the development of MVPs, MVFs, and the final product. They first
 566 work with the data analyst to produce proper instrumentation into the front-end system,
 567 which is the part of the software which is delivered or visible to the user. In the case of
 568 a persevere-decision, they work to fully develop or optimise the feature and submit it
 569 for deployment into production. MVPs, MVFs, and final products are deployed to users
 570 after first going through the continuous integration and continuous delivery systems. A
 571 DevOps engineer acts as the mediator between the development team and operations,
 572 and a release engineer may oversee and manage the releases currently in production.
 573 Importantly, the continuous delivery system provides information on software roll-out
 574 status, allowing other roles to monitor the experiment execution and, e.g., gain an
 575 understanding of the conditions under which the software was deployed to users and of
 576 the sample characteristics and response rate of the experiment. Cross-cutting concerns

577 such as User Experience may require additional roles working with several of the roles
578 mentioned here. To simplify the figure, we have omitted the various roles that relate to
579 operations, such as site reliability engineer, etc. Also, we have omitted a full elaboration
580 of which information artefacts should be visible to which roles. In general, we assume
581 that is beneficial to visualise the state of the continuous experimentation system for
582 all roles.

583 The back-end system consists of an experiment database which, conceptually, stores
584 raw data collected from the software instrumentation, experiment plans – which in-
585 clude programmatic features of sample selection and other logic needed to conduct
586 the experiment – and experiment results. The back-end system and the database are
587 accessible through an API. Here, these parts should be understood as conceptual; an
588 actual system likely consists of multiple APIs, databases, servers, etc. The experiment
589 database enables a product architecture where deployed software is configured for ex-
590 periments at run-time. Thus it is not always required that a new version of the software
591 or the accompanying instrumentation is shipped to users prior to an experiment; the
592 experimental capability can be built into the shipped software as a configurable variation
593 scheme. The shipped software fetches configuration parameters for new experiments,
594 reconfigures itself, and sends back the resulting measurement data, eliminating the need
595 to perform the Develop Product and Deploy Product tasks. For larger changes, a new
596 software version may be required, and the full set of tasks performed.

597 *4.2. Model instantiations and lessons learned*

598 In this subsection, we describe how the RIGHT models were instantiated in the four
599 projects, and we describe the lessons learned. We include illustrative examples from our
600 interview data. We note that the model was initially quite simple, similar to the sequence
601 described in Figure 1 with a build-measure-learn cycle, a data repository, analysis tools,
602 and continuous delivery system. We also note that not all parts of the models were
603 instantiated in all projects. We assume that this will be the case in other projects as well.
604 In the first two projects, we focused on problem validation: developing an understanding
605 of the needs in real situations that a model for continuous experimentation should
606 address. In the two latter projects, we already had most of the model in place and

607 focused more on validating our solution, using detailed findings from the projects in
608 order to adjust the model.

609 Each of the four case projects relate to different aspects of continuous experimen-
610 tation. The case findings support the need for systematic integration of all levels of
611 software product and service development, especially when the context is rapid new
612 product and service development. The key issue is to develop a product that customers
613 will buy, given tight financial constraints. Startup companies operate in volatile markets
614 and under high uncertainty. They may have to do several quick changes as they get
615 feedback from the market. The challenge is to reach product-market fit before running
616 out of money.

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

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

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

633 “We had to pivot when it comes to technology and prioritising features. But
634 the main offering is still the same: it’s the new home phone and it connects
635 to your TV. That hasn’t changed. I see the pivots more like springboards to

636 the next level. For example, we made a tablet version to [gain a distributor
637 partner].” (Tellybean CTO)

638 Also, although an experiment design is, at best, self-evident when viewed in hind-
639 sight, developing one based on the information available in actual software projects,
640 especially new product or service development, is not an easy task. There are multiple
641 possibilities for what to experiment on, and it is not obvious how to choose the first
642 experiment or each next experiment after that. Our case projects showed that initiating
643 the continuous experimentation process is a significant task in its own right and involves
644 much learning. This strengthens the notion that a basic and uncomplicated model to
645 guide the process in the right direction is needed.

646 *4.2.1. Project 1*

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

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

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

669 Table 2 shows the parts of our model that were instantiated in Project 1. The project
670 instantiated a few basic elements of the RIGHT process model. The chosen business
671 model and strategy was to offer the video calling service through operator partnerships.
672 In order for the strategy to be successful, the company needed to demonstrate the
673 feasibility of the service in terms of operator needs and requirements. This demon-
674 stration was to operators themselves but also to other stakeholders, such as investors,
675 who assessed the business model and strategy. The hypothesis to test was not very
676 precisely defined in the project, but could be summarised as “operators will require
677 system performance management analysis tools in order to enter a partnership”. The
678 experiment, which was obviously not a controlled one but rather conducted as part
679 of investor and operator negotiations, used the analytics instrument developed in the
680 project to assess whether the assumption was correct, thus instantiating an MVF, and
681 making a rudimentary experiment execution and analysis. Based on this information,
682 some decisions were made: to start investigating alternative architectures and product
683 implementation strategies.

684 4.2.2. *Project 2*

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

691 “The Software Factory project [. . .] put us on the path of ‘Lego software
692 development’, building software out of off-the-shelf, pluggable components.
693 It got us thinking about what else we should be doing differently. [. . .] We

Table 2: Model instantiations in Project 1.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	“Operators will require performance management analysis tools in order to enter a partnership”
Design, execute, analyse	Rudimentary
MVF	Analytics instrument
Decision making	Start architectural pivot (continued in Project 2) Start product implementation strategy pivot (continued in Project 2) Validate further assumptions (regarding architecture and product implementation)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical Infrastructure	Analytics Tools (MVF developed in project)
Information Artefacts	Learnings (not formally documented in project)

694 were thinking about making our own hardware. We had a lot of risk and
695 high expenses. Now we have moved to existing available hardware. Instead
696 of a client application approach, we are using a web-based platform. This
697 expands the possible reach of our offering. We are also looking at other
698 platforms. For example, Samsung just released a new SDK for Smart TVs.”
699 (Tellybean founder)

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

709 The high-level goals of the second project could be considered as defining and testing
710 a solution hypothesis that addresses the feasibility of the proposed hardware-software
711 solution. The project delivered an evaluation of the technical solution as well as
712 improvement proposals. The analysis showed that the initial architecture and product
713 implementation strategy were too resource-consuming to carry out fully. The results
714 were used by the company to modify their strategy. Instead of implementing the
715 hardware themselves, they opted for a strategy where they would build on top of generic
716 hardware platforms and thus shorten time-to-market and development costs. Table 3
717 shows the model instantiations in Project 2.

718 4.2.3. Project 3

719 In the third project, the capability for continuous deployment was developed. The
720 STBs could be updated remotely, allowing new features to be pushed to customers at very
721 low cost and with little effort. The implications of this capability are that the company
722 is able to react to changes in their technological solution space by updating operating

Table 3: Model instantiations in Project 2.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	“Product should be developed as custom hardware-software codesign” and “Architecture should be based on Enterprise Java technology and be independent of TV set (which acts only as display)”
Design, execute, analyse	Prototype implementation; evaluate current solution proposal
MVF	Alternative call mediator server; alternative system architecture
Decision making	Architectural pivot (Android-based COTS hardware and OS) Product implementation strategy pivot (do not develop custom hardware)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical Infrastructure	Analytics Tools (from previous project)
Information Artefacts	Learnings (not formally documented in project)

723 system and application software, and to emerging customer needs by deploying new
724 features and testing feature variants continuously.

725 The high-level goals of the third project could be considered as developing a capa-
726 bility that allows for automating the continuous deployment process. The prerequisite
727 for this is a steady and controlled pace of development where the focus is on managing
728 the amount of work items that are open concurrently in order to limit complexity. At
729 Tellybean, this is known as the concept of one-piece flow.

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

736 The third project instantiated parts of our infrastructure architecture model, shown in
737 Table 4. In particular, it focused on the role of a continuous delivery system in relation
738 to the tasks that need to be carried out for continuous experimentation, meaning that top
739 and rightmost parts of Figure 3 were instantiated, as detailed in the table.

740 4.2.4. *Project 4*

741 In the fourth project, it was initially difficult to identify what the customers consid-
742 ered to be the main assumptions. However, once the main assumptions became clear,
743 it was possible to focus on validating them. This highlights the finding that although
744 it is straightforward in theory to assume that hypotheses should be derived from the
745 business model and strategy, it may not be straightforward in practice. In new product
746 and service development, the business model and strategy is not finished, and, especially
747 in the early cycles of experimentation, it may be necessary to try several alternatives and
748 spend effort on modelling assumptions until a good set of hypotheses is obtained. We
749 therefore found it useful to separate the identification and prioritisation of hypotheses
750 on the strategy level from the detailed formulation of hypotheses and experiment design
751 on the experiment level. Table 5 shows the instantiated model parts in Project 4. We

Table 4: Model instantiations in Project 3.

Process model instantiation	
Vision	Video calling in the home
Business model and strategy	Offer video calling through operator partnerships (+ assumptions about architecture and product implementation strategies)
Hypotheses	“Capability for automatic continuous deployment is needed for incremental product development and delivery”
Design, execute, analyse	Project focused on instantiating parts of infrastructure architecture model and did not include a product experiment
MVF	Prototype for rapid deployment of software updates
Decision making	Persevere
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students), DevOps engineer, release engineer (played by company CTO and other technical representatives; also represented by user stories with tasks for these roles)
Technical infrastructure	Continuous integration system, continuous delivery System (MVF developed in project)
Information artefacts	Roll-out status

752 note that some of these parts were introduced into the model because of our findings
753 from Project 4.

754 In this project, there were two assumptions: that interaction with the photo map
755 would retain users, and that an automated process of guiding users towards goals
756 was feasible. The assumption that continued use of the application would come from
757 interacting with the photo map was shown to be incorrect. Users would initially create
758 the map, but would not spend much time interacting with it – by, e.g., adding or changing
759 photos, rearranging the map, adding photo annotations, etc. Instead, users reported a
760 desire for connecting with other users to share maps and discuss life goals. Also, they
761 expressed a willingness to connect with professional or semi-professional coaches to
762 get help with implementing their life goals. The social aspect of the service had been
763 overlooked. Whether this was due to familiarity with existing social media applications
764 was left uninvestigated. In any case, the assumption was invalidated and as a result, the
765 assumptions regarding automated features for guiding users towards goals were also
766 invalidated. The investigation indicated that users were motivated by the potential for
767 interaction with other users, and that these interactions should include the process of
768 motivating them to reach goals. It is important to note that the two hypotheses could be
769 invalidated because they were dependent. The process of identifying and prioritising
770 hypotheses separately from detailed formulation of hypotheses and experiment design
771 makes it possible to choose the order of experiments in a way that gains the maximum
772 amount of information with the minimum number of experiments. Testing the most
773 fundamental assumptions – the ones on which most other assumptions rely – first, allows
774 the possibility of eliminating other assumptions with no additional effort.

775 The fourth project also revealed challenges involved with instrumenting the ap-
776 plication for data collection. It was difficult to separate the process of continuous
777 experimentation from the technical prerequisites for instrumentation. In many cases,
778 substantial investments into technical infrastructure may be needed before experiments
779 can be carried out. These findings led to the roles, the high-level description of the
780 technical infrastructure, and the information artefacts in the infrastructure architecture
781 (see Figure 3).

Table 5: Model instantiations in Project 4.

Process model instantiation	
Vision	Well-being service for defining, tracking, and receiving assistance with life goals
Business model and strategy	Product and service recommendations, automated recommendation engine for motivating progress towards goals
Hypotheses	“Motivation for continued use comes from interacting with photo map” and “Automatic recommendation engine will automatically guide users to reach goals” (depends on first hypothesis)
Design, execute, analyse	User tests with observation and interviews
MVF	HTML5-based, tablet-optimised application
Decision making	Product implementation strategy pivot (focus on social interaction rather than automated recommendations)
Infrastructure model instantiation (only applicable parts)	
Roles	Business analyst, product owner (played by company leadership), software developer (played by Software Factory students)
Technical infrastructure	Instrumentation, Front-end system
Information artefacts	Learnings

782 However, many experiments are also possible without advanced instrumentation.
783 The fourth project indicates that experiments may typically be large, or target high-level
784 questions, in the beginning of the product or service development cycle. They may
785 address questions and assumptions which are central to the whole product or service
786 concept. Later stages of experimentation may address more detailed aspects, and may
787 be considered optimisation of an existing product or service.

788 **5. Discussion**

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

799 Each project could have instantiated the RIGHT models in different ways. In the
800 first project, the experiment could have been carried out using mockup screens to
801 validate what metric data, visualisation, and analysis tools would have been sufficient to
802 convince the stakeholders. However, this would have been detrimental since it would not
803 have revealed the shortcomings in the initial architecture and implementation strategy.
804 Although the design of the experiment left much to be desired, carrying it out using a
805 real, programmed prototype system made it possible to discover the need to reconsider
806 some of the previous strategy choices.

807 In the second project, the learnings could have been better used to define a more
808 precise set of hypotheses after a careful analysis of the shortcomings of the previous
809 system architecture. However, this was not necessary since the purpose was not a point-
810 by-point comparison but rather an either-or comparison between one general approach

811 and another. This highlights an important notion regarding continuous experimentation:
812 it only seeks to produce enough information for a decision to be made correctly.

813 In the third project, only the capability for continuous delivery was instantiated. The
814 project could also have addressed the components that are necessary to carry out actual
815 experiments. Due to project time constraints, this was left uninvestigated in the third
816 project, but was considered in the fourth project instead. In that project, one cycle of
817 the full RIGHT process model was carried out, and the software was instrumented for
818 experimentation although using ready-made services such as Google Analytics.

819 While our ultimate aim is for our models to cover the entire breadth of continuous
820 experimentation, we assume that not all real-life projects will need to instantiate all parts.
821 For instance, experiments can be conducted without an MVP, especially in an early
822 stage of product development. It may also not be necessary in all cases to have a heavy
823 infrastructure for the experimentation – this becomes relevant if experimentation is
824 conducted in very large volumes or when the purpose is to maintain a set of experiments
825 that are run continuously to collect trend information while the product is incrementally
826 changed.

827 In addition to the project-specific observations, we consider some more general
828 concerns. Having several experiments run in parallel presents a particular challenge.
829 The difficulty of interpreting online experiments has been convincingly demonstrated by
830 Kohavi et al. [16]. Statistical interactions between experiments should be considered in
831 order to assess the trustworthiness of the experiments. For this reason, it is important to
832 coordinate the design and execution of experiments so that correct inferences are drawn.
833 More generally, the issue of validity becomes important when the entire R&D organisa-
834 tion is experiment-driven. Incorrectly designed or implemented experiments may lead
835 to critical errors in decision-making. Threats to validity can also stem from a failure to
836 consider ethical aspects of experiments. Not only may unethical experiments damage
837 company reputation, but they may cause respondents to knowingly or unconsciously
838 bias the experimental results, leading to errors in decision-making.

839 Other challenges include the difficulty of prioritising where to start: which assump-
840 tion should be tested first. In Project 4, we identified a dependency between assumptions
841 regarding the backend recommendation logic and the assumption of what motivates users

842 to keep using the application. By invalidating the latter, we automatically invalidated
843 the first assumption. This highlights the importance of identifying critical assumptions,
844 as testing them first may save several unneeded experiments. We see a need for further
845 research into this area. Also, in hardware-software co-design, illustrated by the first
846 three projects, setting up the experimental cycle quickly is a major challenge due to both
847 the longer release cycle of hardware and the potential synchronisation problems between
848 hardware and software development schedules. Based on the findings presented in this
849 paper, it may be beneficial to test a few strategic technical assumptions first, such as the
850 viability of a certain hardware-software platform. As our case demonstrates, choosing
851 the correct platform early can have a significant impact on the ability to proceed to
852 actual service development.

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

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

871 Another question is whether continuous delivery is a strictly necessary precondition
872 for continuous experimentation. In the beginning of the product development cycle,

873 experimentation must occur before much software is written at all. At that stage,
874 continuous delivery may not be necessary. Also, not all experiments require new
875 software to be delivered to users. While a continuous delivery system may exist, the
876 software itself may be architected for variability so that it can reconfigure itself at
877 run-time. In such cases, no new version of the software needs to be delivered for new
878 experiments to run. However, not all experiments are possible even with a very flexible
879 architecture that allows for run-time reconfiguration. Continuous delivery is a good
880 vehicle for delivering experiments to users and to ensure quality in the development
881 process. The model presented here is based on iterative, evolutive optimisation of
882 product features and an incremental model of innovation. To carry out revolutionary
883 innovation, the process needs to be extended with other means of discovering customer
884 value. These may profoundly invalidate the business model or strategy, and may even
885 have an impact on the overall vision.

886 Finally, experimentation may be conducted with several kinds of stakeholders. Apart
887 from customers and end users, experiments could be directed towards investors, suppli-
888 ers, sales channels, or distributors. Companies whose product is itself a development
889 platform may want to conduct experiments with developers in their platform ecosystem
890 to optimise the developer experience [9] of their tools, methods, and processes. These
891 experiments may require other kinds of experimental artefacts than the MVP/MVF,
892 including, e.g., processes, APIs, and documentation. Research on the types of experi-
893 mental artefacts and associated experimental designs could lead to fruitful results for
894 such application areas. Also, an open question is who should primarily lead or conduct
895 the experimentation, especially when the development organisation is separate from
896 the customer organisation. Some training may be needed for customers in order to
897 ensure that they can interact with the continuous experimentation process running in
898 the development organisation. Similarly, the development team may need additional
899 training to be able to interact with the customer to derive assumptions, plan experiments,
900 and report results for subsequent decision-making. Another possibility is to introduce
901 a mediating role which connects the customer and development organisations. More
902 generally, increasing the capability to perform experimentation and continuous software
903 engineering requires consideration of human factors in software development teams [23].

904 Further research is needed to determine how the experimental process works across
905 organisational borders, whether within or outside a single company.

906 A particular limitation of this study is the use of relatively short projects with student
907 participants. Students carried out the technical software development and analysis tasks
908 in the projects, while the researchers handled tasks related to identification of assump-
909 tions, generation of hypotheses, and higher-level planning tasks together with customer
910 representatives. While it is reasonable to expect that professional software developers
911 would have reached a different level of quality and rigour in the technical tasks, we
912 consider it likely that the findings are applicable beyond student projects since the focus
913 of this paper is not on the technical implementation but on the integration of experiment
914 results in the product development cycle and the software development process. The
915 length of the projects means that at most one experimental cycle could be carried out
916 in a single project. Thus the first case company completed three, and the second case
917 company one experimental cycle. In a real setting, multiple experimentation rounds
918 would be carried out over an extended period of time, proceeding from experiments
919 addressing the most important assumptions with the highest impact towards increasing
920 detail and optimisation. The findings of this study should be considered to apply mostly
921 in the early stages of experimentation.

922 **6. Conclusions**

923 Companies are increasingly transitioning their traditional research and product de-
924 velopment functions towards continuous experiment systems [12]. Integrating field
925 experiments with product development on business and technical levels is an emerg-
926 ing challenge. There are reports of many companies successfully conducting online
927 experiments, but there is a lack of a systematic framework model for describing how
928 such experiments should be carried out and used systematically in product development.
929 Empirical studies on the topic of continuous experimentation in software product devel-
930 opment is a fruitful ground for further research. Software companies would benefit from
931 clear guidelines on when and how to apply continuous experimentation in the design
932 and development of software-intensive products and services.

933 In this paper, we match a model for Continuous Experimentation based on analysis
934 of previous research against a multiple case study in the Software Factory laboratory
935 at the University of Helsinki. The model describes the experimentation process, in
936 which assumptions for product and business development are derived from the business
937 strategy, systematically tested, and the results used to inform further development of the
938 strategy and product. The infrastructure architecture for supporting the model takes into
939 account the roles, tasks, technical infrastructure, and information artefacts needed to
940 run large-scale continuous experiments.

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

953 In future work, we expect the model to be expanded as more use cases arise in the
954 field. Domain-specific variants of the model may also be needed. Furthermore, there are
955 many particular questions with regard to the individual parts of the model. Some specific
956 areas include (i) how to prioritise assumptions and select which assumptions to test first;
957 (ii) how to assess validity and determine how far experimental results can be trusted
958 – especially how to ensure that experiments are trustworthy when running potentially
959 thousands of them in parallel; (iii) how to select proper experimental methods for
960 different levels of product or service maturity; and (iv) how to build a back-end system
961 for continuous experimentation that can scale to the needs of very large deployments, and
962 can facilitate and even partially automate the creation of experimental plans. Particular
963 questions regarding automation include which parts of the model could be automated or

964 supported through automation. Another question is how quickly a Build-Measure-Learn
965 block can be executed, and what the performance impact of the model is on the software
966 development process.

967 **Acknowledgements**

968 This work was supported by Tekes – the Finnish Funding Agency for Technology
969 and Innovation, as part of the N4S Program of DIGILE (Finnish Strategic Centre for
970 Science, Technology and Innovation in the field of ICT and digital business).

971 **References**

- 972 [1] Rob J. Adams, Bradee Evans, and Joel Brandt, *Creating Small Products at a Big Company: Adobe's*
973 *Pipeline Innovation Process*, CHI'13 Extended Abstracts on Human Factors in Computing Systems,
974 2013, pp. 2331–2332.
- 975 [2] V Basili, J Heidrich, M Lindvall, J Münch, M Regardie, D Rombach, C Seaman, and A Trendowicz,
976 *GQM+ Strategies: A comprehensive methodology for aligning business strategies with software measure-*
977 *ment*, Proceedings of the DASMA Software Metric Congress (MetriKon 2007): Magdeburger Schriften
978 zum Empirischen Software Engineering, 2007, pp. 253–266.
- 979 [3] V. Basili, R. Selby, and D. Hutchens, *Experimentation in Software Engineering*, IEEE Transactions on
980 Software Engineering **12** (1986), no. 7, 733–743.
- 981 [4] Steve Blank, *The Four Steps to the Epiphany: Successful Strategies for Products that Win*, 2nd ed., K&S
982 Ranch, 2013.
- 983 [5] Jan Bosch, *Building Products as Innovation Experiment Systems*, Software Business, 2012, pp. 27–39.
- 984 [6] Jan Bosch, Helena Holmström Olsson, Jens Björk, and Jens Ljungblad, *The Early Stage Software Startup*
985 *Development Model: A Framework for Operationalizing Lean Principles in Software Startups*, Lean
986 Enterprise Software and Systems, 2013, pp. 1–15.
- 987 [7] U. Eklund and J. Bosch, *Architecture for Large-Scale Innovation Experiment Systems*, Joint Working
988 IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software
989 Architecture (ECSA), 2012, pp. 244–248.
- 990 [8] F. Fagerholm, N. Oza, and J. Münch, *A platform for teaching applied distributed software development:*
991 *The ongoing journey of the Helsinki software factory*, 3rd International Workshop on Collaborative
992 Teaching of Globally Distributed Software Development (CTGDSD), 2013, pp. 1–5.
- 993 [9] Fabian Fagerholm and Jürgen Munch, *Developer Experience: Concept and Definition*, International
994 Conference on Software and System Process, 2012, pp. 73–77.

- 995 [10] Rebecca M. Henderson and Kim B. Clark, *Architectural Innovation: The Reconfiguration of Existing*
996 *Product Technologies and the Failure of Established Firms*, *Administrative Science Quarterly* **35** (1990),
997 no. 1, 9–30.
- 998 [11] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, *Design Science in Information*
999 *Systems Research*, *MIS Quarterly* **28** (2004), no. 1, 75–105.
- 1000 [12] Helena Holmström Olsson, Hiva Alahyari, and Jan Bosch, *Climbing the “Stairway to Heaven” – A*
1001 *Multiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous*
1002 *Deployment of Software*, 39th EUROMICRO Conference on Software Engineering and Advanced
1003 Applications (2012), 392–399.
- 1004 [13] Helena Holmström Olsson and Jan Bosch, *The HYPEX Model: From Opinions to Data-Driven Software*
1005 *Development*, *Continuous Software Engineering*, 2014, pp. 155–164.
- 1006 [14] Natalia Juristo and Ana M. Moreno, *Basics of Software Engineering Experimentation*, Springer, 2001.
- 1007 [15] Ron Kohavi, Thomas Crook, and Roger Longbotham, *Online Experimentation at Microsoft*, Third
1008 Workshop on Data Mining Case Studies and Practice Prize, 2009.
- 1009 [16] Ron Kohavi, Alex Deng, Brian Frasca, Roger Longbotham, Toby Walker, and Ya Xu, *Trustworthy*
1010 *Online Controlled Experiments: Five Puzzling Outcomes Explained*, Proceedings of the 18th ACM
1011 SIGKDD International Conference on Knowledge Discovery and Data Mining, 2012, pp. 786–794.
- 1012 [17] Ron Kohavi, Alex Deng, Brian Frasca, Toby Walker, Ya Xu, and Ni Pohlmann, *Online Controlled*
1013 *Experiments at Large Scale*, Proceedings of the 19th ACM SIGKDD International Conference on
1014 Knowledge Discovery and Data Mining (KDD’13), 2013, pp. 1168–1176.
- 1015 [18] Beverly May, *Applying Lean Startup: An Experience Report – Lean & Lean UX by a UX Veteran:*
1016 *Lessons Learned in Creating & Launching a Complex Consumer App*, Agile Conference (AGILE) 2012,
1017 2012, pp. 141–147.
- 1018 [19] Jürgen Münch, Fabian Fagerholm, Patrik Johnson, Janne Pirttilahti, Juha Torkkel, and Janne Järvinen,
1019 *Creating Minimum Viable Products in Industry-Academia Collaborations*, Proceedings of the Lean
1020 Enterprise Software and Systems Conference (LESS 2013, Galway, Ireland, December 1-4), 2013,
1021 pp. 137–151.
- 1022 [20] Jürgen Münch, Fabian Fagerholm, Petri Kettunen, Max Pagels, and Jari Partanen, *Experiences and*
1023 *Insights from Applying GQM+Strategies in a Systems Product Development Organisation*, Proceedings
1024 of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA
1025 2013), 2013.
- 1026 [21] Jim Nieters and Amit Pande, *Rapid Design Labs: A Tool to Turbocharge Design-led Innovation*,
1027 *Interactions* (2012), 72–77.
- 1028 [22] Taiichi Ōno, *Toyota production system: beyond large-scale production*, Productivity press, 1988.

- 1029 [23] Efi Papatheocharous, Marios Belk, Jaana Nyfjord, Panagiotis Germanakos, and George Samaras, *Personalised Continuous Software Engineering*, Proceedings of the 1st International Workshop on Rapid
1030 Continuous Software Engineering, 2014, pp. 57–62.
- 1032 [24] Mary Poppendieck, *Lean software development: an agile toolkit*, Addison-Wesley Professional, 2003.
- 1033 [25] Mary Poppendieck and Michael A Cusumano, *Lean Software Development: A Tutorial*, IEEE Software
1034 (2012), 26–32.
- 1035 [26] Eric Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation To Create Radically
1036 Successful Businesses*, Crown Business, 2011.
- 1037 [27] Daniel Ståhl and Jan Bosch, *Modeling Continuous Integration Practice Differences in Industry Software
1038 Development*, Journal of Systems and Software (2014), 48–59.
- 1039 [28] Annika Steiber and Sverker Alänge, *A Corporate System for Continuous Innovation: The case of Google
1040 Inc.*, European Journal of Innovation Management (2013), 243–264.
- 1041 [29] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer, *Overlapping Experiment Infrastructure:
1042 More, Better, Faster Experimentation*, Proceedings 16th Conference on Knowledge Discovery and Data
1043 Mining, 2010, pp. 17–26.
- 1044 [30] Joan E. van Aken, *Management Research Based on the Paradigm of the Design Sciences: The Quest
1045 for Field-Tested and Grounded Technological Rules*, Journal of Management Studies **41** (2004), no. 2,
1046 219–246.
- 1047 [31] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén,
1048 *Experimentation in Software Engineering*, Springer, 2012.
- 1049 [32] Robert Yin, *Case study research: design and methods*, 4th ed., SAGE Publications, Inc., 2009.