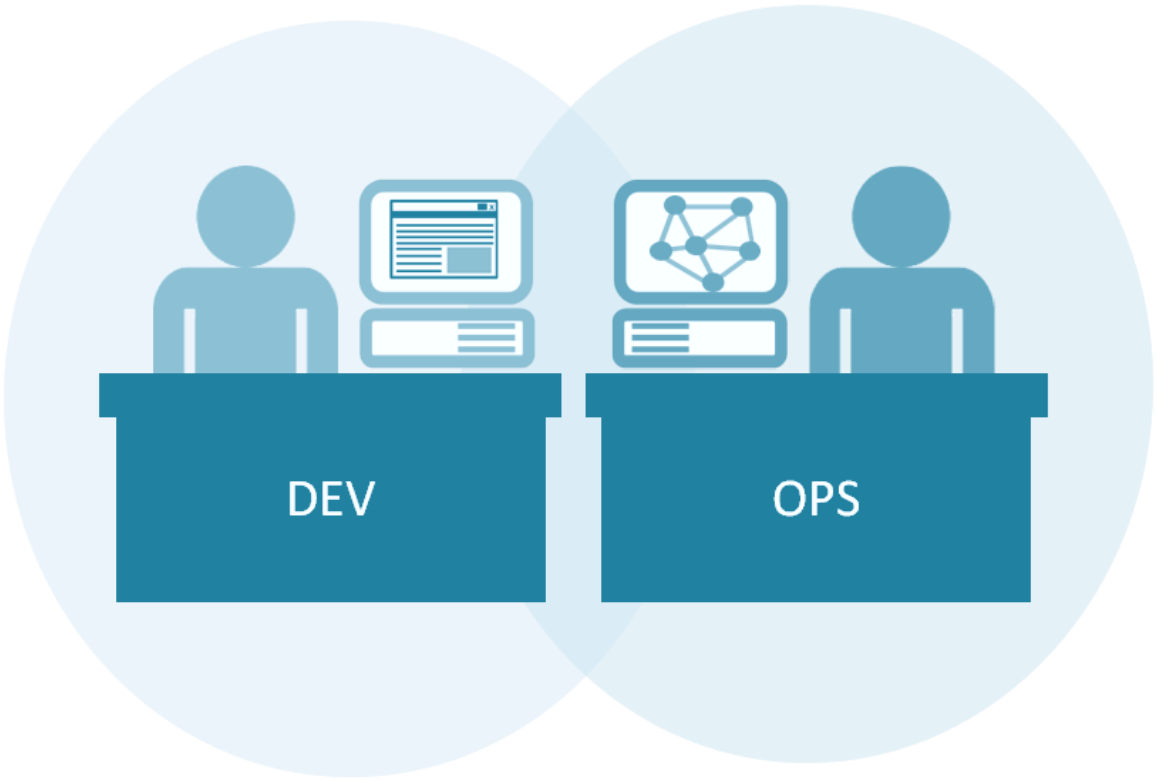

Extending the Agile Development Discipline to Deployment

The Need For a Holistic Approach

Master Thesis

Version: 1.0
Date: June 29, 2013

Onno Dijkstra
Master Business Informatics
Institute of Information and
Computer Science
Utrecht University

ORDINA

Universiteit Utrecht

Master Thesis

Thesis title: Extending the Agile Development Discipline to Deployment: The Need For a Holistic Approach

Student number: 3783936

Author: Onno Dijkstra
Master Business Informatics
Utrecht University
o.dijkstra@gmail.com

First supervisor: prof. dr. S. Brinkkemper
Department of Information and Computing Sciences
Utrecht University

Second supervisor: drs. K. Vlaanderen
Department of Information and Computing Sciences
Utrecht University

External supervisor: ir. M.J. Schudel
BUC OSD J-Technologies
Ordina

Above all, applying the DevOps approach is a change of mindset. It is essential to build a team of Devs and Ops, which is aligned with shared incentives.

- Hüttermann (2012)

Abstract

In the last decade information system development has made a shift towards the agile way of working. Agile teams are responsible for realizing requirements in a multidisciplinary set up, where business and development are represented in a single project. DevOps is a new movement to improve IT service delivery agility. DevOps fosters closer collaboration and communication between development and operations personnel. Until recently, operational issues and requirements remained underexposed in the project, which affected the quality of the software. DevOps aims to break down the functional silos and extends the scope of the project to the release deployment. In this thesis we propose an incremental method engineering approach to identify suitable process patterns for a Dutch IT organization, and be able to implement them into Scrum. The resulting method increments are sequentially implemented during an 8-week pilot project to demonstrate the effectiveness of DevOps practices on the experienced problem areas.

Version Information

Version	Date	Description	Author
0.1	December 3rd, 2012	Set-up	Onno Dijkstra
0.8	June 11th, 2013	First review version	Onno Dijkstra
0.8	June 14th, 2013	Review by Kevin Vlaanderen	Kevin Vlaanderen
0.8	June 14th, 2013	Review by Michel Schudel	Michel Schudel
0.8	June 17th, 2013	Review by Patrick Debois	Patrick Debois
0.8	June 21th, 2013	Review by Jan-Hein Bührman	Jan-Hein Bührman
0.9	June 24th, 2013	Second review version	Onno Dijkstra
0.9	June 26th, 2013	Review by Kevin Vlaanderen	Kevin Vlaanderen
0.9	June 28th, 2013	Review by Sjaak Brinkkemper	Sjaak Brinkkemper
1.0	June 29th, 2013	Final version	Onno Dijkstra

Contents

List of Figures

List of Tables

1. Introduction

1.1 Problem Definition

Nowadays, many software development methods are available to the IT organization. Some of them emerged due to deficiencies in the existing methods and others are created completely from scratch to meet a new development philosophy. IT organizations can freely choose a method they prefer based on the project's characteristics. Occasionally the need arises to improve the current method through new experiences. This is the case at a financial services company located in the Netherlands, where they use the Scrum method for their development projects. The project teams encounter several problems during development projects. At the case company there is a misalignment between the development team and operations staff. Most development teams build software in a high pace, but felt there are not responsible for the deployment process performed by operations. The result is the release is thrown over 'the wall' to operations, resulting in many production issues. Vice versa, operations personnel do not act as the owner of the system under development. Therefore, they are not attending project meetings and do not know what to expect with the upcoming release. Also, quality requirements and guidelines are not properly addressed during the development. These problems could be addressed with the use of DevOps practices, that attracted the attention of the company. The organization wants to adopt DevOps, starting with the implementation of practices to complement their current development method.

DevOps breaks the functional silos in the team to foster collaboration and focuses on business value by filling the gap between the development and operations departments (Hüttermann, 2012; Smith, 2011; Swartout, 2012). The term DevOps was first coined during the DevOps Days in 2009 and is supported by many practitioners in this area. Other methods often go by strict definition resulting in process fundamentalism, whereas DevOps is being maintained by the community and leaves room for your own interpretation. Therefore no official process definition is available, though the underlying theory suggests several modifications and additions to the agile method to gain advantage from the DevOps facilities. Since Scrum supports the agile methodology too, research needs to be performed on how the company can integrate DevOps into their Scrum development projects.

1.2 Case Company

The case company of subject is a Dutch firm in the financial sector with more than hundred subsidiaries which operate both nationally and internationally. Due to confidentiality issues we cannot provide the name of this organization, therefore we prefer to use the term CaseComp. The umbrella organization facilitates IT services and is responsible for compliance with laws for its subsidiaries. In terms of IT the company addresses the demand for information systems (IS) in the entire organization and is responsible for the development, testing and quality control, deployment, maintenance and management of the IS by considering its subsidiaries as customers. Since most IS are tailor made, CaseComp composed several development teams of hired industry experts. Each development team is assigned to a specific (part of a) product, e.g. mortgages and investments. The customer relationship management interface (CRMI) – a central service layer – binds all IS together.

1.3 Objective and Problem Statement

This research identifies the different paths that can be chosen to implement DevOps for companies that use Scrum as their current development method and experience the same issues. DevOps proposes solutions to the drivers in the interest of the company. The research shows what the rationale is of using DevOps by providing an in-depth explanation what drivers trigger the company to implement this extension. Also, the implementations paths are provided to help other companies that face the same problems with their development process.

The formal problem statement for this research project is formulated as follows:

> *How can the IT organization be supported in the implementation of DevOps?*

1.4 Scientific Relevance

In the last decade the area of method engineering (ME) got its attention in scientific studies. ME is the scientific discipline that focuses on the design, construction and improvement of software development methods, techniques and tools (Brinkkemper, 1996). Since ME is already a mature discipline, the underlying theory and techniques enables researchers and practitioners to adapt information system development methods (ISDMs). Since the study improves an existing method, it is directly related to the field of incremental method engineering (IME). This field focuses on evolving a method in time towards a higher maturity level by changing small parts of the method (Mirandolle, Van de Weerd, & Brinkkemper, 2011). Once method improvements are identified for the current method, method increments can be elaborated. Method increments are method fragments that improve the performance of a method (Van de Weerd, Brinkkemper, & Versendaal, 2007). However the implementation of method increments and the effect in practice is not extensively discussed. Therefore, knowledge and experiences on these implementations form a useful contribution to the field of IME.

This study applies the concepts and techniques of ME in practice by expanding an existing method in order to fulfill the companies' needs. This is realized by creating a situational method, a method that is tuned to the project-specific needs. A common technique that can be used for crafting methods and visualizing incremental differences is called meta-modeling, which supports the process of situational method engineering (Souer, Van de Weerd, Versendaal, & Brinkkemper, 2007). Currently, there is no way that supports decision-making in adopting method increments derived from the method base. Especially when a single method is used as a source. Also, there is no uniform solution that determines how method increments should be incorporated in practice. Thus this research elaborates on how the implementation of method increments can be supported.

1.5 Business Relevance

Large numbers of ISDMs exist to the IT organization. Based on the project's characteristics a suitable method can be selected. Over time projects may encounter that a method is performing poorly and the need arises to improve the method's performance in order to stay efficient.

Agile has become a popular development philosophy resulting in several methods that adheres to its main principles, such as extreme programming (XP) and Scrum. Its main concept is to let the

developers build software incrementally and to obtain feedback of the customer as soon as possible. This highly iterative process ensures that the developers build the software the customer actually wants and bugs are solved at a high pace.

A new movement focuses on DevOps, which streamlines the software delivery process by composing a multi-disciplinary team aiming to fill the gap between the development and operations departments (Edwards, 2010; Hüttermann, 2012; Swartout, 2012). In traditional development approaches these departments strive to achieve their own goal. For instance, development aims to frequently release new features and IT operations aims to provide stable applications. These contradictory goals impede the performance of the process. DevOps bridges the two departments by providing one overarching goal, namely to deliver value to the customer.

Since many IT organizations are using agile methods such as Scrum and there is increasing interest in DevOps, it would be helpful to support them in adopting DevOps into their agile development projects. Besides providing a practical framework for extending Scrum, the instruments provided by this research can also be used for all other kinds of method improvement implementations.

1.6 Challenges

There are several challenges in the execution of this research. First, the solutions proposed by DevOps are hard to discover. The method is maintained by the community and can best be characterized as a set of best practices. As a result there is a lack of a formal process definition. There is only a limited number of resources available, such as Internet blogs and a few books. Also, the solutions can be viewed through different perspectives resulting in different implementations. To ensure that the right solutions are implemented, only those will be identified that tackles the company's issues. This ensures the method's rationale is being preserved. Furthermore, it could be possible that not all problem areas can be covered by DevOps. Another challenge is to elaborate on an approach for selecting method increments in the process of creating a situational method. Also, since DevOps has strong ties with automation, adequate tools are needed to support the development, testing and deployment processes. The challenge is to find out how process improvements are dependent on tools. Finally, the last challenge is to find out how the method increments can be assessed in order to provide an optimal implementation schedule.

2. Research Method

2.1 Research Questions

The research questions are formulated by taking into account the problem statement and research objectives. By answering the main question we contribute to the field of incremental method engineering by showing how methods increments can be implemented with the use of integration scenarios. To answer the main question, sub questions need to be answered first. In this research each research activity answers one sub question.

First, the important drivers for an organization to implement DevOps are identified by the use of expert interviews (SQ1). These drivers ensure the method's rationale is preserved. Second, a procedure is elaborated for linking method fragments to drivers for improvement (SQ2). Third, based on the best practices described in literature, solutions are identified which result in several method increments (SQ3). Third, the proposed solutions are mapped to the main drivers which result in different scenarios to incorporate DevOps practices into Scrum (SQ4). Also, the optimal scenario is provided based on important considerations. These activities are supported by expert validations to ensure consistency and prevent common mistakes. The optimal scenario is validated in a DevOps pilot. Finally, the causal factors that shape the optimal integration scenario contribute to the knowledge base (SQ5). The relations between the research steps and sub questions are illustrated in Figure 1.

MQ. How can method engineering support the incremental implementation of DevOps?

SQ1. What are the main drivers and requirements for an organization to integrate DevOps into their Scrum development process?

SQ2. How can method fragments be linked to key problem areas?

SQ3. Which method fragments proposed by DevOps address the key problem areas?

SQ4. How can the selection of method increments in alternative integration scenarios be supported?

SQ5. How can the optimal integration scenario be executed in a real development project?

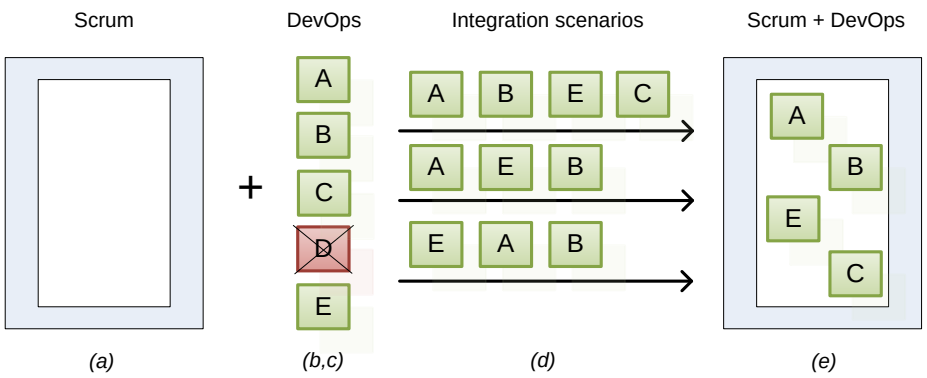

Figure 1. Research steps

2.2 Research Model

The research model is depicted below and is created using the research model method adopted from Verschuren & Doorewaard (2007). Rectangles in Figure 2 represent the research objects, where the most important ones have a solid border. Arrows indicate a conclusion between two or more objects resulting in a following object. The research questions are derived from the related arrows in this model.

A study on the topics of Scrum (a) together with expert interviews result in the process-deliverable diagram (PDD) of the current situation (d). The PDD is the blueprint of the process and is discussed in 4.2.1. The PDD is used as foundation for the development of a situational method (d) by implementing improvements proposed by DevOps. This activity is assisted by a study on the topics of DevOps and Method Engineering (b). Thereafter, method increments are identified (b). These provide input to determine the optimal integration scenario (c). The validation is twofold (c). First, interviews are held to ensure the consistency and to validate the method increments. Second, a pilot experiment is performed to demonstrate the effect on the experienced problem areas.

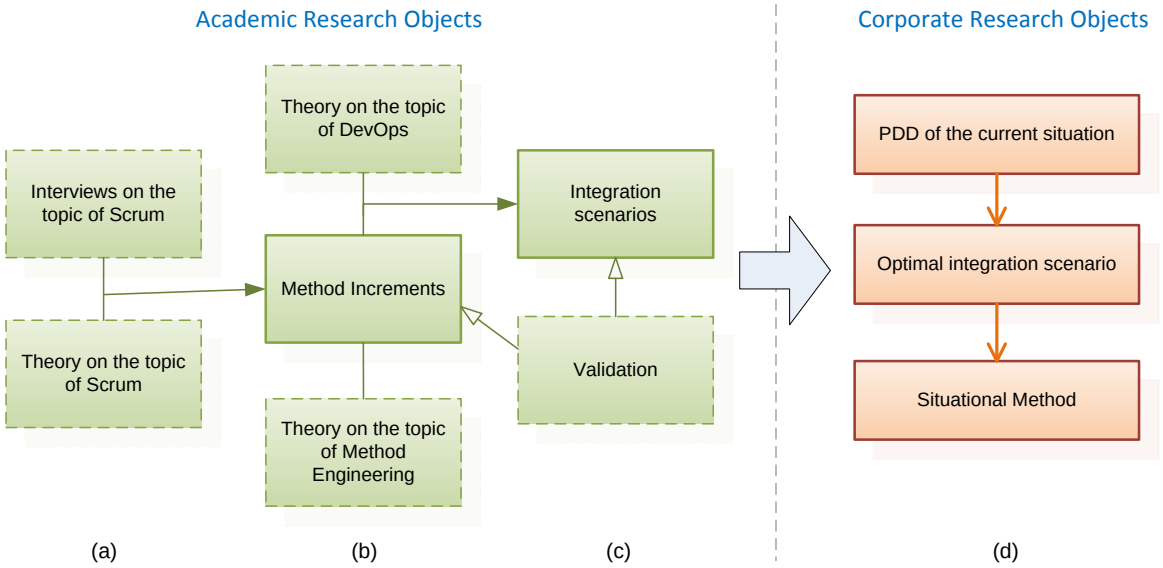

Figure 2. Research model

2.3 Concepts and Scoping

The conceptual model of this research is illustrated in Figure 3. The model was created by applying the input-environment-outcome (I-E-O) model developed by Astin (1993). Originally it is a guiding framework for assessments in higher education, but can be applied to other research areas as well. This framework states assessments are not complete unless the evaluation includes information on inputs (I), the environment (E), and outcomes (O) (Astin, 1993). The input variables indicate the independent variables of the research, which together with the business context are related to the dependent variables (output).

The input for this study is provided by the organization (i.e. development and operations personnel, processes, and documents), resulting in the identification of drivers for improvement, process difficulties, and business requirements for the process implementation. The organization resembles

the environment in which the case studies are performed. Part of the environment are project stakeholders and the project configuration (i.e. capacity, knowledge, experience). The loopback implies the obtained or created results are validated through iterative feedback (e.g. by the use expert validations, post iteration workshops). The output of the study is the set of improvement materials (e.g. method increment descriptions, process-deliverable diagrams, activity and concept tables) labeled as a situational method, integration scenario, and the perceived effectiveness by the team as well the quantitative feedback on the process metrics.

For the scoping of this research project we adhere to the concepts and constructs defined in Appendix I. The research is limited to enhancements to the Scrum process.

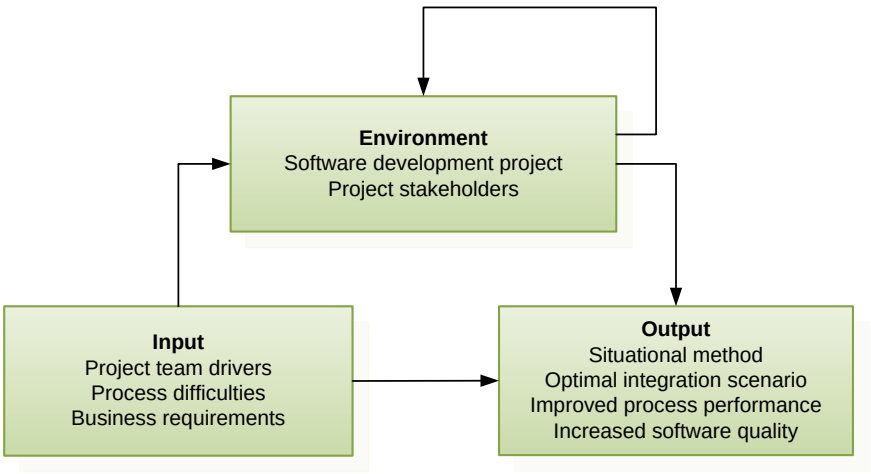

Figure 3. I-E-O conceptual model of study variables

2.4 Research Approach

In this section the research approach is discussed by describing the activities that are performed and the methods used during the execution of this research project. The research steps are described by the use of the process-deliverable diagram (PDD) technique. Since the study provides artifacts that contribute to the information systems (IS) discipline we first discuss the relation with the research area of design science.

2.4.1 Design Science Research

According to Hevner, March, Park, and Ram (2004), there exist two types of research in the IS discipline: behavioral science and design science. Behavioral science seeks to develop and verify theories that explain or predict human or organizational behavior. Design science seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management and use of information systems can be effectively and efficiently accomplished (Denning, 1997). In this thesis we describe the process to extend the agile development discipline to deployment. As this study produces viable artifacts we can relate this research to the design science problem-solving paradigm. The viable artifacts for this research include a situational method which represents the Scrum development method expanded by a set of method increments derived from DevOps. These method increments can be reused by other

organizations. The research also consists of a framework to set up integration scenarios, process alternatives in which method increments can be incorporated within a method to meet the situational method. The designed artefacts are important to the relevant business problems and are verifiable contributions to problem statement. The artifacts will be produced by using the design science research guidelines provided by Hevner et. al (2004). These seven guidelines ensure that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact. Part of it, is the use of design evaluation methods which are available in the knowledge base. For instance, the iterative improvement process (IIP) which prescribes post iteration workshops (PIWs) to obtain positive as well as negative experiences on the method. The designed artifacts are assessed on a predefined set of validation criteria. By using such methods the goodness and efficacy of an artifact can be rigorously demonstrated. The research framework of Hevner et al. (2004) is applied to this research project and is depicted in Figure 4.

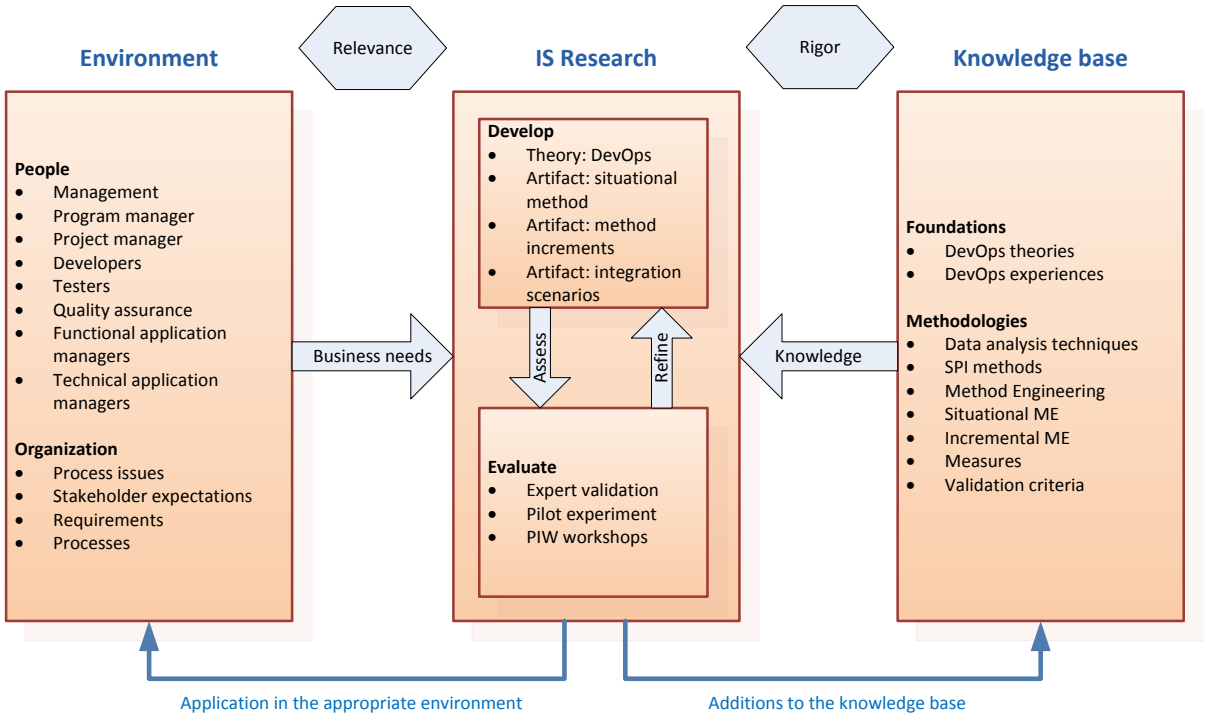

Figure 4. Design science research model applied to this research (Hevner et al., 2004)

2.4.2 Research Activities

In this subsection the research steps are provided accompanied with the PDD. The PDD shows both the activities and deliverables over the nine distinct phases. The research consists of a single case study for which the current situation is captured. Based on the identified drivers at the case company, the appropriate method increments of DevOps are selected to tackle their process issues. The outcome of the research is a situational method accompanied with the adequate steps to integrate DevOps into the Scrum development process.

The PDD of the research method is depicted in **Error! Not a valid bookmark self-reference.**. The phases in which the research activities are performed are discussed next.

Phase	Description
Prepare research	In this research phase the long proposal is created in accordance with the business and research needs. This document describes the relevant methods for the execution of this research project.
Perform literature study	The literature study is the foundation for this research as it provides the important concepts and constructs. Topics include Scrum, continuous integration (CI), DevOps and continuous delivery (CD) as these are interrelated to each other.
Capture current situation	This phase identifies the main drivers and requirements relevant to DevOps and elaborates on the PDD of the current situation. At the end of this phase, the first subquestion (a) of the research is answered.
Capture desired situation	The desired situation is captured by creating a situational method. First, the literature is consulted for the identification of DevOps patterns. The patterns that tackle the company's issues are assembled into the Scrum method. The PDD technique is used to formalize the method increments. Experts are asked to review the PDDs to ensure that they are correct and the actual integration is not at risk. At the end of this phase, the second and third subquestions of the research are answered (b,c).
Identify scenarios	In this phase the different paths are explored in how the method increments can be introduced in the current situation. The method increments are assessed and an optimal scenario is provided by taking into account the pros and cons. At the end of this phase, the fourth subquestion of the research is answered (d).
Integrate practices	This phase consists of a project experiment in which the method increments are implemented incrementally to the Scrum method according to the described scenario. Findings on implementing DevOps using this scenario are reported.
Improve process	This phase ensures the method increments are implemented correctly by evaluating and improving the method continuously. At the end of this phase, an answer is provided for the last subquestion of the research (e).
Finalize thesis and paper	These two parallel running activities ensure the research's artifacts are produced and revised.

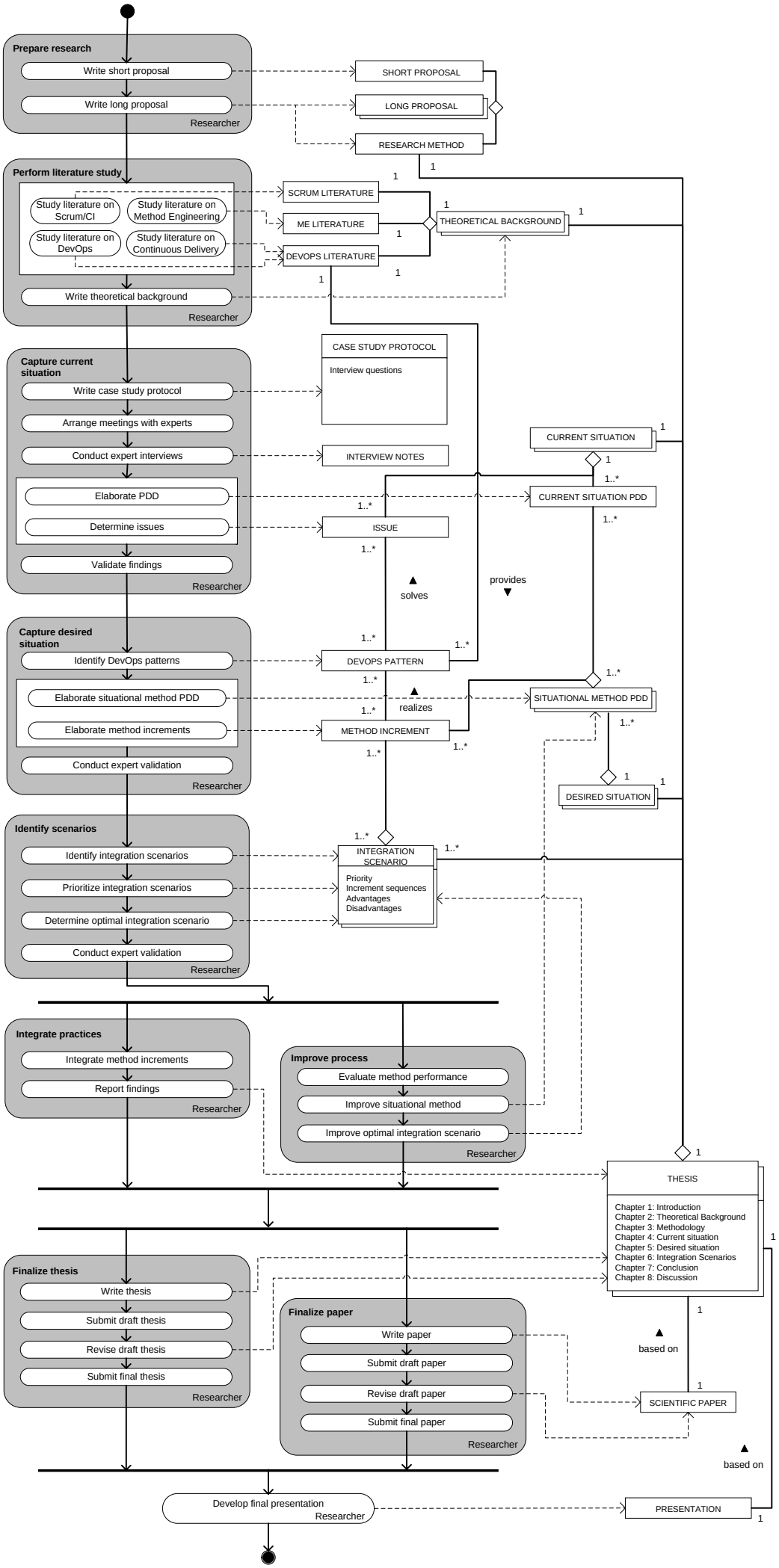

Figure 5. PDD of the research approach

3. Theoretical Background

In this chapter we provide the theoretical background for the topics of interest to this study. We start with the introduction of two system development practices as well their supporting techniques. Scrum is an information systems development method (ISDM) relevant for the identification of the current situation at CaseComp. Continuous integration is a technique applied within Scrum. Topics relevant for the desired situation include DevOps and continuous delivery, terms which are often used interchangeably (Pais, 2012; Smith, 2013). DevOps is a movement attempting to break down functional silos in organizations that need to deliver software (Phifer, 2011; Swartout, 2012). Continuous delivery elaborates on the technical aspects of implementing a so called deployment pipeline. Continuous delivery and DevOps have one goal in common, namely encouraging a greater collaboration between stakeholders involved in software delivery in order to release valuable software faster and more reliably (Hüttermann, 2012). In addition to these four topics, method engineering (ME) is discussed to support the process of integrating DevOps patterns into the Scrum method. The concepts and constructs are provided for this research as well an overview of the state of the art on incremental method engineering and its applicability in practice.

The relations between the five topics are illustrated in Figure 6. Note that topics with a solid border are extensively discussed in the thesis, whereas the topics on continuous integration and continuous delivery are only briefly discussed in this chapter to provide a coherent overview. These techniques support the ISDM in the left column, but a wide discussion about the implementation of these techniques is outside the scope of this research project.

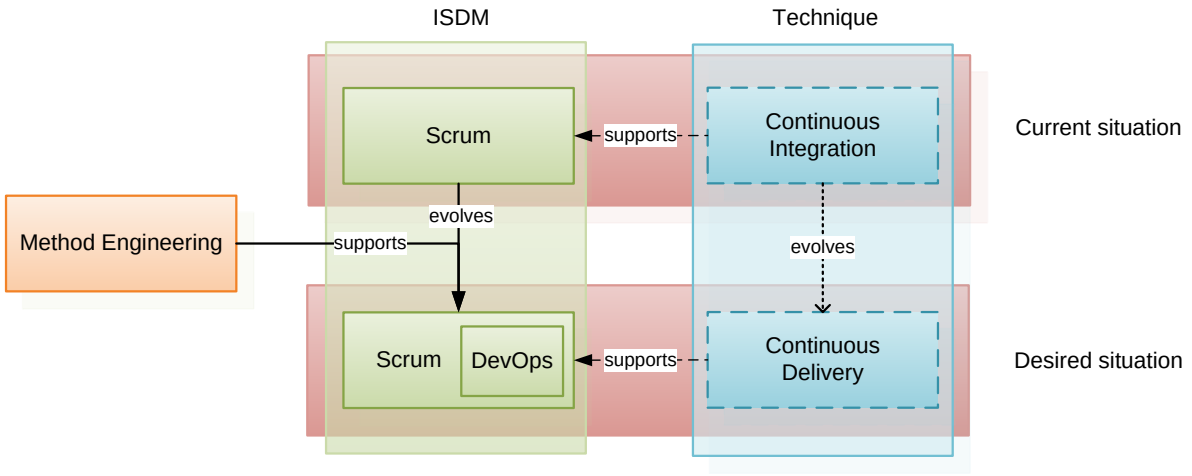

Figure 6. Topics relevant to this study

3.1 Software Development Practices

3.1.1 Scrum

Scrum is a specific approach of the agile development movement. Agile ISDMs should be carried out under agile values in order to answer the challenges of rapid development and changing requirements principles (Agile Manifesto, 2001). The manifesto states that agile development focuses on four core values:

I. individuals and interactions over processes and tools,
II. delivering working software over comprehensive documentation,
III. customer collaboration over contract negotiation,
IV. responding to change over following a plan.

Agile is now a mainstream development discipline and is adapted to the workplace of the organization (West & Grant, 2010). Unlike the traditional methods, agile methods deal with unpredictability by relying on people and their creativity rather than on processes (Cockburn & Highsmith, 2001). Agile enables rapid development and testing through multiple iterations, but does not prescribes procedures for the release deployment. As a result new features are not directly offered to the customer since the release is waiting for a manual, often slow deployment by IT operations. As the deployment process is not coordinated with the development, all new features stack up and wait to be released and thus, many advantages gained with agile are lost.

Over the last few years, surveys confirm the success of agile practices. The latest industry report by The Standish Group (2011) shows agile projects have a 42% success rate, compared to traditional waterfall projects at a dismal 14% success rate. Several methods have adopted the agile way of working, including XP, FDD, DSDM and Scrum.

Scrum is an agile framework for completing complex projects, originally developed for organizing software development projects, but suitable for any domain (ScrumAlliance, 2012). Schwaber (1995) defines Scrum as “a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems”. The method is used for the management, enhancement and maintenance of an existing system. Scrum assumes existing design and code and addresses totally new systems or legacy systems which are subjected to re-engineering (Schwaber, 1995). The name of the method is from rugby – a tight formation of forwards, who bind together in specific positions when a scrumdown is called. Initially, the approach was proposed by Takeuchi & Nonaka (1984) and is elaborated by Jeff Sutherland in 2003. Thereafter, Jeff Sutherland worked with Ken Schwaber to formalize the Scrum process. Nowadays it is by far the most popular method used in agile implementations worldwide (VersionOne, 2011).

Scrum has control mechanisms to deal with unpredictability and complexity that comes with the project. For example, the method uses an iterative approach to test the feasibility of a subsystem in the initial iterations. Scrum divides workload in sprints – cycles of 2-6 weeks containing user stories or functionalities that need to be ready at the end of the sprint. The project contains as many sprints are desired to evolve the system. Team members integrate their work frequently by applying continuous integration, which is discussed in the following section. In the last phase the release is prepared for deployment.

There are three phases in the Scrum method: planning, architecture design, and development. The planning phase involves the creation of a backlog, which contains functionality requirements that are not adequately addressed by the current product release (Schwaber, 1995). Afterwards, a release plan is created along with an estimate of its schedule and costs. The architectural design phase consists of the analysis on the domain models and architecture to check whether they are sufficient to support the user stories that are planned for the current release. Also, product standards are defined where the team adheres to when developing the system. During the development phase the functionalities described in the release plan are built and tested. This is done in an iterative cycle. Once all user stories are realized, they are put together to provide a fully integrated build. A visual representation of the features of the Scrum process is depicted in Figure 7.

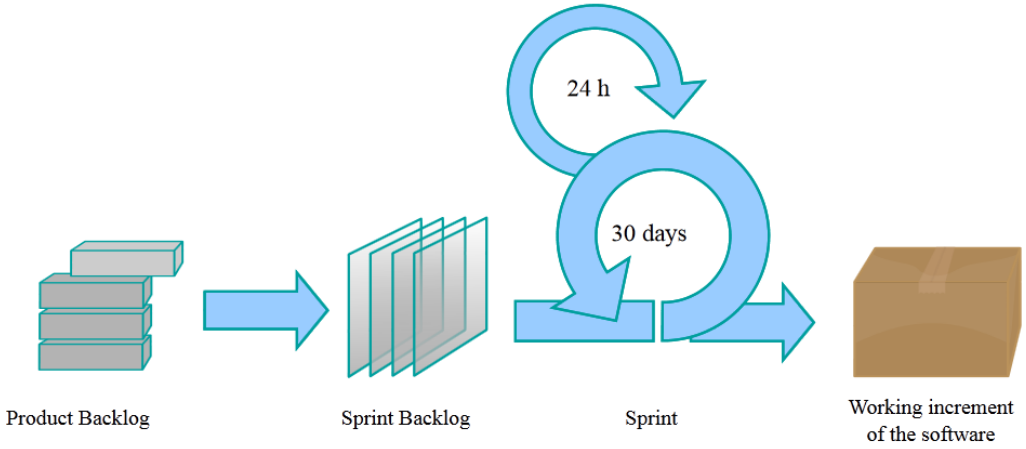

Figure 7. Overview of the Scrum process (from Lakeworks, 2009)

3.1.2 Continuous Integration

In the early days of the software industry the integration of a software project was often a painful and tense moment. Separate application modules where put together, resulting in major integration problems. As expected, the modules worked well individually. Solving these problems took lots of effort because over time the complexity of the system increased significantly. Yet in the last few years, integration problems largely disappear as it is diminished to a non-event. This is due to the introduction of the continuous integration (CI) technique. “Continuous integration is the practice of making small well-defined changes to a project’s code base and getting immediate feedback to see whether the test suites still pass” (Duvall, Matyas, & Glover, 2007; Fowler, 2006). CI was first named and proposed as part of extreme programming (XP), containing twelve agile development practices. Its aim was to prevent integration problems such as described above, referred to as “integration hell” (Jeffries, 2001).

The first perception of CI was to pass the unit tests before the system was committed to production, as described by the test-driven development approach part of XP (Janzen & Saiedian, 2005). Later elaborations of the concept introduced build servers, which run unit tests automatically or after every commit. In practice CI is supported by a version control system (VCS) of which the repository maintains the latest version of the systems source code (called the mainline or trunk). The developer extracts (or checks-out) a copy of the mainline to its local environment. Once adaptations have been made to the system, the used copy of the mainline is already out of date since other developers may have updated the code base already several times. Therefore before the so called commit is

performed – the action in which the working copy is transferred to the repository – the copy in the local environment first needs an update to include the recent adaptations of the mainline. These actions can easily be performed within an integrated development environment (IDE). The CI technique expects developers to check-in their code several times a day, so integration with other parts of the application proceeds smoothly. The features of the CI process include a connection to a source code repository, an automated build script, a feedback mechanism and a process for integrating source code (Duvall, 2010). This process prescribes frequent small changes as opposed to infrequent large changes. The four features are illustrated in Figure 8.

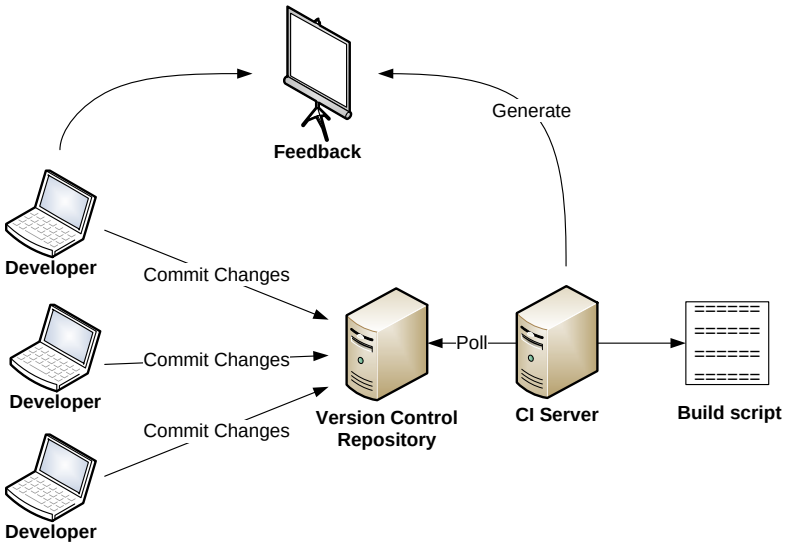

Figure 8. The process of Continuous Integration (redrawn from Duvall, 2010)

The following 10 key principles of Fowler (2006) should be considered for an effective continuous integration:

- Maintain a single source repository
- Automate the build
- Make the build self-testing
- Everyone commits to the mainline every day
- Every commit should build the mainline on an integration machine
- Keep the build fast
- Test in a clone of the production environment
- Make it easy for everyone to get the latest deliverables
- Everyone can see what's happening
- Automate deployment

Continuous integration addresses integration risks earlier by the use of smaller increments and increases opportunities for feedback (Duvall et al., 2007). Continuous delivery uses CI as foundation and ensures the mainline is always in a state to be deployed to users and makes the actual deployment very rapid (Humble & Farley, 2010). The improvements aim on achieving business value rather than providing functionality.

3.1.3 DevOps

Problems frequently occur in the deployment phase of the development process, where bugs and performance delays are detected once the software release is deployed to the production machine. Close cooperation between the development and operations departments may prevent that an application becomes unstable. Often the operations department has its own methods (such as ITIL, ASL, BiSL) and their release cycle is not aligned with the schedule of development. A common mindset in projects is people thinking and acting as functional silos – i.e. do nothing more than is required by their user role. Once software features are built, developers 'throw' the software build over the wall to operations, subsequently resulting in many issues.

DevOps is a portmanteau of development and operations, a term for practices that foster collaboration between these departments in order to help an organization rapidly produce software (Edwards, 2010; Hüttermann, 2012; Pant, 2009). The term DevOps was first coined by Patrick Debois during the DevOps Days in September 2009, but the philosophy itself is not entirely new to practitioners. DevOps has its origins in the proliferation of cloud services that changed the way of software development and the relationship between developers and operations (Jawalka, 2012; Smith, 2011; Yap, 2012). Earlier Debois conducted a research on the topic of agile infrastructures and states the current technology is mature to use as foundation to integrate the infrastructural work in the project (Debois, 2008). As we have discussed in the previous section, the ability to rapidly build and test new features satisfies only a small part of the overall development process. The demand grows towards a holistic approach that ties together every part of the delivery process and everybody involved in it. Therefore DevOps extends the definition of done or even banish the word done - so the process doesn't stop at the end of development. Instead the scope of the project includes the deployment of the actual software release. DevOps removes the barriers by composing multi-disciplinary teams that provide the information system as a team, so the overall process from inception to delivery proceeds more smoothly.

DevOps has emerged from the agile community and there is no concrete set of mandates or standards. Instead there are good practices for IT organizations which should be considered as a set of guiding principles to improve the agile development process. The term DevOps is associated with different types of content as it can be seen through different perspectives. Therefore, DevOps lacks of a formal definition. Gartner's definition of DevOps takes a broad perspective and formulates it as "an IT service delivery approach rooted in agile philosophy, with an emphasis on business outcomes, not process orthodoxy" (Smith, 2011). Practitioners more commonly agree to the definition of DevOps as a set of processes, methods and systems for communication, collaboration and integration towards a common goal between departments for development and technology operations (Pant, 2009; Rowe & Marshall, 2011; Swartout, 2012). Hüttermann (2012) elaborates on this definition – "DevOps describes practices that streamline the software delivery process, emphasizing the learning by streaming feedback from production to development and improving cycle time (i.e., the time from inception to delivery)". As the definitions address all aspects of DevOps, we reformulate the definition of DevOps to fit the scope and goal of the research project. We define DevOps as practices that embed operations knowledge into the project and foster bidirectional feedback between the development and operations departments in order to streamline the software delivery process.

Debois (2012) elaborated four key areas for DevOps to indicate which aspects are relevant to DevOps. The DevOps key areas are discussed below and visualized in Figure 9. The interaction flows between development (dev) and operations (ops) are bi-directional, resulting in knowledge exchange and feedback. As the research focuses on the expansion of the development process, suitable DevOps practices are identified in the fourth key area. However, this would not mean that the other areas are irrelevant for the case company and should be addressed at a later stage. The emphasis of the two central areas (1 and 2) is merely on tools rather than processes.

- *Area 1*: *Extend delivery to production.* The development and operations departments collaborate to improve the delivery process from project to production.
- *Area 2: Extend operations feedback to project*. The feedback flow which ensures all information from production is radiated back to the project.
- *Area 3*: *Embed project knowledge into operations*. The development team takes co-ownership of everything that happens in the production environment.
- *Area 4*: *Embed production knowledge into project*. Operators are involved from the beginning of the development project.

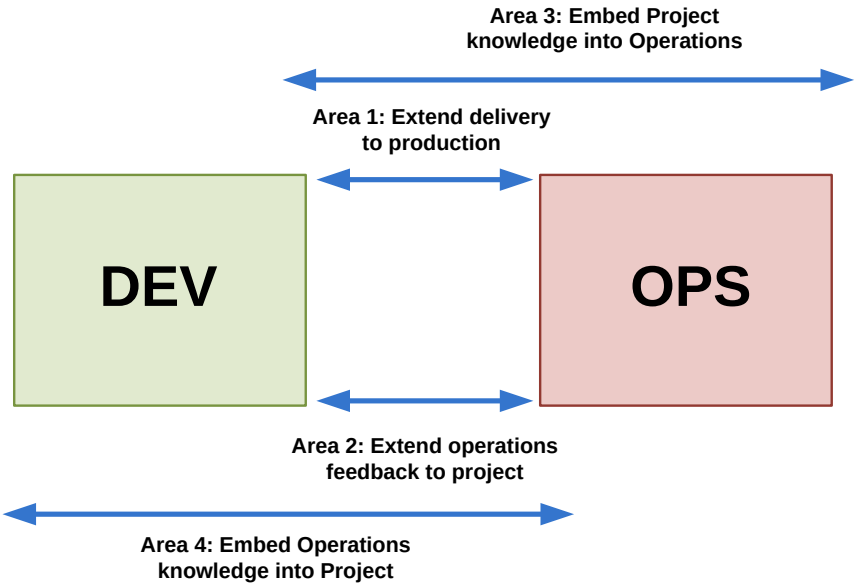

Figure 9. DevOps key areas (redrawn from Debois, 2012)

In the key areas from above, Debois (2012) makes a distinction between three different layers: (1) tools, to make things technically possible; (2) process, to show how it should be done; and (3) people or culture; to enable people to do something. The purpose of these layers is to assign the appropriate labels to DevOps practices to make clear where these are aimed at.

Another view of DevOps is by Damon Edwards and John Willis who proposed CAMS, an acronym representing the core values of DevOps: Culture, Automation, Measurement and Sharing (Willis, 2010). Jez Humble later added an L for Lean, to form CALMS (Willis, 2012). In this order the DevOps method should be introduced and improved. Culture is seen as the most crucial and hardest part of DevOps and forms the basis for the other core values. The cultural aspect addresses the longstanding tension between development and operations in order to compose a cross-functional delivery team. Without being aware of the culture, all automation attempts will be less effective. We are aware that

some of the proposed process changes which emerge from this study could be impossible without changing the culture in the firm. Therefore we will investigate requirements for the method improvements to make such change possible. A pre-condition is attached that determines whether a process change can be introduced. Automation is the first core value that needs to be addressed once a DevOps culture is established (Willis, 2010). Automation enables quicker feedback and more gradual deployment of software increments. Continuous delivery elaborates on setting up a deployment pipeline, a technique which is discussed in the next section. Measurement is of importance to continuously improve the workflow. Lean is a systems development paradigm which also applies to DevOps. Lean has its focus on creating value for the customer, eliminating waste, optimizing value streams, empowering people, and continuously improving (Ebert, Abrahamsson, & Oza, 2012). Sharing is the loopback in the CALMS cycle, enabling people to share ideas and problems.

Although there are many theories that support DevOps, the biggest challenge is to codify DevOps practices. Some attempts are done by Lee (2011); Debois (2012); Swartout (2012) and Hüttermann (2012). Debois (2012) created a template for codifying practices into patterns and principles. The template uses the aforementioned DevOps key areas and layers which can be assigned to a practice. Below we provide some examples of practices captured by Lee (2011):

- *Cross-functional teams*. Whereas agile integrates development and quality assurance into a single team, DevOps takes this further by integrating operational roles.
- *Develop for Production.* The artifacts (e.g. deployment scripts, release notes, etc.) needed to put a release into production are developed earlier. This increases the focus on delivering non-functional requirements.
- *Pushed Phased Releases.* Before the application is rolled out to all customers, the contents of releases are typically pushed to a small number of servers. Any problems could be addressed much faster, before the whole of a community is affected.

The findings from the latest industry survey indicate the DevOps adoption is accelerating since 2011 (PuppetLabs, 2013). Over 4000 IT operations and development professionals from over 90 countries participated in this survey. The outcome shows 63 % of respondents have implemented DevOps practices, a 26 % increase since 2011.

DevOps maintains the agile aspect of the project and enables incremental deployments by fostering automation and closer collaboration. DevOps is quite new in the field, but some of the described practices are not. However a common approach and term makes IT organizations aware of the shift towards a holistic development approach that embraces both departments aiming on delivering value to the customer.

3.1.4 Continuous Delivery

In the last few years continuous delivery (CD) has received attention by practitioners. CD is a set of good design practices within the field of software development, which can also be called a pattern language (Alexander, 1977). CD elaborates on the principles of continuous integration and automates the repetitive actions involved in information system (IS) development by means of a tool. The purpose of CD is to deliver software much faster. Originally, the term is derived from the first principle of the Agile Manifesto (2001) which states “our highest priority is to satisfy the customer

through early and continuous delivery of valuable software" and is elaborated and popularized by Humble & Farley (2010).

CD aims to deliver business value to the customer, whereas CI has its focus on getting new features quickly release-ready. Business value is achieved by means of stable applications, therefore CD focuses more on the technical aspects. CD emphasizes on the concept of staged builds, also called a deployment production line or deployment pipeline (Humble & Farley, 2010; Humble, Read, & North, 2006). During each stage the build is tested and obviously, when all tests are passed the release is finally distributed to the customer.

Continuous delivery rests on the three pillars configuration management, agile testing, and the deployment pipeline (Humble & Farley, 2010). First, in configuration management all artifacts relevant to the project are stored, retrieved, uniquely identified, and modified (Hass, 2003). Especially in CD, it is a synonym for version control. Second, agile testing relies on quality built into the delivery process by testing throughout the process. Third, the deployment pipeline refers to how information systems gets from the development phase to the release phase (Humble & Farley, 2010; Humble et al., 2006). Every change goes through the deployment pipeline where build generation, unit testing, performance testing, user acceptance testing and deployment are performed automatically (Humble & Farley, 2010; Mikita, Dehondt, & Nezlek, 2012).

According to Humble & Farley (2010) the following 8 principles should be considered for an effective software delivery process:

- Create a repeatable and reliable process for releasing software
- Automate almost everything
- Keep everything in version control
- If something is difficult or painful, do it more frequently
- Focus on built-in quality
- Done means released
- Everybody is responsible for the delivery process
- Improve continuously

Continuous delivery evolves both IS development as deployment to the next level. CD enables an efficient and highly automated delivery pipeline to provide stable applications. The process supports the common goal of DevOps targeting on business value and makes no distinction between departments. CD prescribes patterns to set up the deployment pipeline and to automate the various tasks involved in development, testing and deployment. The solutions proposed by CD are mainly technology-driven but give solid support to the DevOps processes.

3.2 Software Process Improvement

Methods exist in many variations to support the process areas of the IT organization, such as requirements management and change management. Over time project experience accumulates into knowledge which could lead to improvements for the method that is used. In the last decades lots of initiatives has moved its focus on improving processes of organizations. Business process management (BPM) was a response to the workflow wave of the nineties. As the workflow is oriented on enactment (i.e. to support the execution of operational processes), it was considered as

too restrictive (Van der Aalst, Ter Hofstede, & Weske, 2003). Aalst et al. (2003) define BPM as "supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information". An overview of the techniques that support business process modeling is presented by Aguilar-Savén (2004). Process improvement is also addressed by software process management (Florak, Park, & Carleton, 1997). Software process improvement (SPI) supports the IT organization to improve the processes related to information system (IS) development. The improvements enable the organization to raise their maturity level. In the past many case studies investigated the success factors in software process improvement. An overview of numerous SPI studies is presented by Dyba (2005). Despite the commonly recognized success factors such as management commitment and employee participation, operational measures are still unavailable (Dyba, 2005).

Macintosh (1993) defines five maturity levels for process improvement which are adopted by several SPI methods (e.g. CMMI, ITIL, and COBIT). The maturity level is achieved by implementing all processes in a certain maturity level including these of the underlying layers.

1. *Initial*. Setting up of processes.
2. *Repeatable*. Repeatable processes.
3. *Defined*. Documented processes standardized throughout an organization.
4. *Managed*. Measured and controlled processes.
5. *Optimizing*. Continuous process improvement.

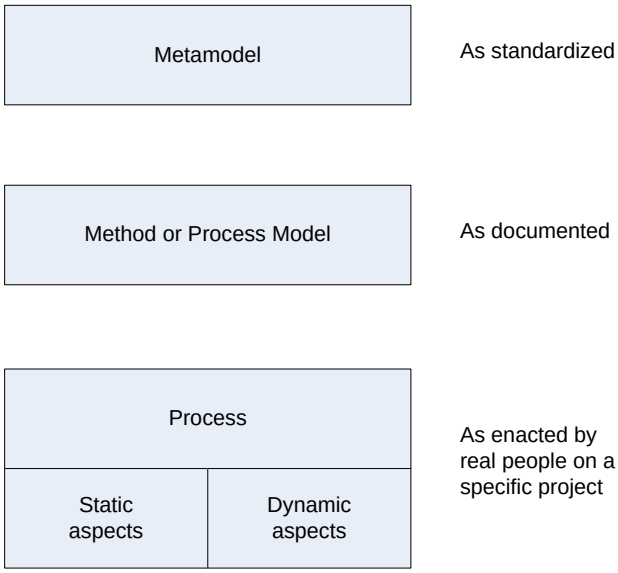

Figure 10. Abstraction layers (redrawn from Henderson-Sellers, 2006)

The process is a term for way of acting and describes what is done in real time with a real team on a real project (Conradi, 1993; Henderson-Sellers, 2006). It follows the prescribed steps of the method to produce an artifact. The process has both a static as dynamic aspect, in which the process steps represent the static aspect and the data (i.e. deadlines, deliverables) to create an instance of the process model represent the dynamic aspect. In contrast, the process model or method is an abstract entity that only exist in the mind of the user and needs to be captured in terms of some concrete artifact (Guizzardi, 2005). In concrete terms a method has documented the steps to execute. On the

other hand a process is the method applied in a real project environment. A meta-model is from a higher abstraction level and describes a method by representing the syntactical structures and provides formal statements about the model (Van de Weerd & Brinkkemper, 2008). The meta-model integrates the meta data model and meta process model (Harmsen et al., 1994). Meta-models are created using a meta-modeling technique, which we discuss later. The separation of the abstraction layers is illustrated in Figure 10.

3.3 Method Engineering

Method engineering (ME) supports the design, construction and adaptation of methods, techniques and tools for the development of information systems (Brinkkemper, 1996, p. 276). Brinkkemper (1996) defined a method as ”an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products”. The method describes instructions on how to perform development activities (i.e. the stages, activities and tasks to be carry out) and defines the structural requirements for the products (i.e. documents, models and diagrams), also called deliverables (Brinkkemper, 1996). Herein Brinkkemper (1996) distinguishes two types of method fragments, process fragments and product fragments.

A method fragment is defined as “any coherent product, activity, or tool being part of an existing generic or situational method” (Harmsen et al., 1994). The process and product fragments can each be subdivided into two subtypes, namely conceptual fragments and technical fragments. Conceptual fragments represent methods or part thereof, whereas technical fragments are required in order to include CASE tools in the engineered method (Harmsen et al., 1994).

Ralyté and Rolland (2001) address the notion of a method chunk. A method chunk integrates the process fragment and product fragment in one fragment to form a coherent module (Ralyté & Rolland, 2001). As a result, the method can be viewed as a set of loosely coupled method chunks expressed at different levels of granularity (Ralyté, 1999). Brinkkemper, Saeki and Harmsen (1998) provided five possible granularity levels for method fragments:

1. Method – addresses a comprehensive approach for performing a systems development project.
2. Stage – addresses solely a part of the information system life-cycle.
3. Model – addresses a perspective of an information system.
4. Diagram – addresses the representation of the model layer method fragment.
5. Concept – addresses the concepts and associations of the diagram layer method fragment.

The structure of a method component resembles that of the method chunk. According to Wistrand and Karlsson (2004) “each method component consists of method elements and their goals, which are anchored in the values of the method creator”. In this approach much attention is paid to the rationale of the component. Method rationale argues why and how the method has been established and is considered important as information on decisions that lead to a certain meta-model has been not so well codified in the past (Rossi, Tolvanen, Ramesh, Lyytinen, & Kaipala, 2000).

3.3.1 Situational Methods

Situational method engineering (SME) involves the creation of methods based on a set of project-specific requirements (Brinkkemper, Saeki, & Harmsen, 1999; Brinkkemper, 1996; Harmsen, 1997; Mirandolle, Van de Weerd, & Brinkkemper, 2011; Ralyté & Rolland, 2001). The SME discipline emerges as a reaction to the problems arisen in standardized methods. When a method is tuned at hand to meet the project-specific requirements, it is called a situational method (Brinkkemper, 1996). In situational method engineering route maps (i.e. scenarios for method fragments) are used to tune methods into situational methods (Slooten & Hodes, 1996).

Situational methods reuse method fragments, parts from the existing methods. The method database or method base is filled with reusable method fragments for this purpose. The method fragments are described in formal process definitions (e.g. books, manuals). Once identified, these are elaborated using a common notation (such as PDD), and finally stored in the method base.

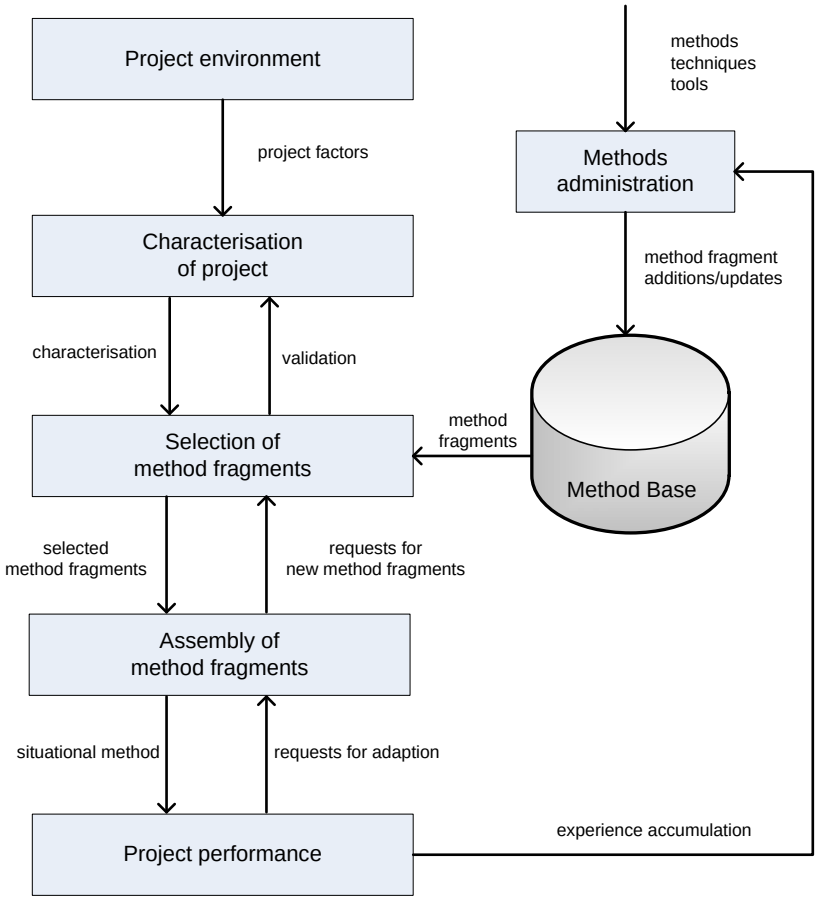

Figure 11. The configuration process for situational methods (redrawn from Brinkkemper, 1996)

The construction of a method depends on the objective of SME. Therefore Ralyté (2002) identified four objectives in order to aid the method engineer: (1) to define a brand new method to satisfy a set of requirements, (2) to add alternative ways of working in an existing method, (3) to extend a method by new functionality, or (4) to select only relevant functionalities. Several SME approaches exist to support the creation of a situational method, e.g. Brinkkemper (1996); Ralyté (2002); Ralyté, Deneckère, and Rolland (2003); Burns and Deek (2007); Luinenburg, Jansen, Souer, and Van de Weerd (2008). For example, the SME approach by Brinkkemper (1996) is depicted in Figure 11. A

generic SME approach is presented by Van de Weerd, Brinkkemper, Souer, and Versendaal (2006) which assumes the method base is already filled with method fragments or requires at least a set of methods selected for inclusion. The steps involved in this process are as follows:

1. Analyze project situation and identify needs.
2. Select candidate methods that meet one or more aspects of the identified needs.
3. Analyze candidate methods and store relevant method fragments in a method base.
4. Select useful method fragments and assemble them in a situational method by using route map configuration to obtain situational methods.

Part of the SME approach is to elaborate on method fragments (step 3 and 4). This is supported by a meta-modeling technique, which assists the method engineer in crafting a blueprint of the method in order to store relevant fragments to the method base. Also, for assessing the quality of a method, it is essential to explicitly describe the steps in high detail to find areas for improvement. Several meta-modeling techniques and their languages are discussed in an article of Harmsen and Saeki (1996). Van de Weerd et al. (2006) proposed a generic meta-modeling technique, based on work of Saeki (2003); Van de Weerd and Brinkkemper (2008). The technique combines UML activity diagrams and class diagrams in one diagram, called a process-deliverable diagram (PDD). A PDD or meta-model attaches semantic information to artifacts for measuring its quality (Saeki, 2003). The meta-model can also be used to analyze the method evolution of a company over the years (Van de Weerd et al., 2007).

Examples of situational method engineering applied in practice include the research of Van de Weerd et al. (2006), in which an implementation method is constructed for web-based CMS applications. The resulting method can be used for standard and complex situations by following the described routes in the route map. Coulin, Zowghi, and Sahraoui (2006) provide practitioners with a lightweight approach to requirements elicitation. The pre-constructed situational method can be tuned to the project at hand. Another example is from Seidita, Cossentino, and Gaglio (2007) who created their own SME approach for the construction of multi-agent systems design processes. In an experiment they adapted the Passi process by adding the requirements analysis phase from Tropos.

3.3.2 Method Increments

Methods are adapted over time to improve the process performance for information systems (IS) development. This is part of incremental method engineering (IME). IME is concerned with improving methods in an evolutionary way rather than in a revolutionary way by changing small parts of the method to obtain a higher maturity level (Mirandolle et al., 2011; Rossi et al., 2000; Tolvanen, 1998). The IME discipline is considered as a subtype of situational method engineering.

An adaptation of a method to improve the overall performance is called a method fragment increment, or simply method increment (Van de Weerd et al., 2007). A method snapshot is a method configuration valid at a particular time. By comparing two method snapshots it is possible to identify method increments. Van de Weerd et al. (2007) addressed the evolution of methods by the use of method increments and proposed 18 elementary method increment types. A method adaptation can either be an insertion, editing or removal of method fragments or its properties (Van de Weerd et al., 2007).

Currently, there are several IME approaches available to the method engineer. Ralyté, Rolland, and Ayed (2005) created the evolution-driven method engineering approach which captures various evolution ways as different strategies to create the product part of the model under construction. Brinkkemper, Van de Weerd, Saeki, and Versendaal (2008) proposed an approach for incremental method evolution by applying requirements engineering techniques to information system development methods (ISDMs). Van de Weerd (2009) provides an approach for incremental process improvement by assessing a company's maturity level and selecting method fragments based on situational factors and desired maturity level.

IME approaches are however, not widely applied in practice. A case study by Mirandolle et al. (2011) shows how a requirements prioritizing method can be adapted through marching situational factors. Their IME approach was based on comparing candidate methods and the case company method to visualize how a suitable method could be selected. Kevin Vlaanderen, Van Stijn, Brinkkemper, and Van de Weerd (2012) elaborated on the various implementation paths of the Scrum development method in the context of incremental method evolution. The study shows the implementation styles (e.g. disruptive or incremental) of Scrum method increments at several case companies. Another useful contribution to the field of IME is from Van Stijn, Vlaanderen, Brinkkemper, and Van de Weerd (2012), who provided a template for method increment case descriptions with the aim to structure improvement paths in a clear and concise manner.

In the field of method engineering lots of "big bang" method initiatives take place (e.g. Van de Weerd et al., 2006; etc.), however method improvements are incremental in nature. Therefore it is important to track the changes over time as the method rationale is a crucial part for the success of ME (Rossi et al., 2000). Incremental process improvement initiatives are often supported by a SME approach (e.g. Vlaanderen, Valverde, and Pastor, 2006; Mirandolle et al., 2011), by integrating new method fragments into the existing method.

4. Current Situation

This chapter elaborates on the development processes at the case company in order to find areas for improvement. First, we discuss the case study approach that is applied in this research phase. Second, by using a reference method we are able to validate the Scrum process at the case company. As the Scrum method is loosely defined by sets of core processes and optional processes, expert interviews are held to adapt the reference method to the situation of the organization. A process-deliverable diagram is provided together with their explanatory tables. Finally, based on the interview results, main drivers and requirements are elicited which allows us to answer the first question of the research:

> *SQ1. What are the main drivers and requirements for an organization to integrate DevOps into their Scrum development process?*

The answer on this research question is used as input for the next research phase, which enables us to search for solutions for the identified problem areas.

4.1 Case Study Approach

The case study approach is ideally suited for the purpose of the research. First, we want to identify the issues and main drivers in the information systems (IS) development process at CaseComp in order to find areas for improvement. Second, we want to make sure that the proposed improvements by DevOps are relevant for CaseComp. Third, we want to evaluate the elaborated paths for implementing the improvements. As the latter needs feedback from the environment, a case study is therefore ideally suited for these purposes. The case study protocol we employed for the current situation phase is provided in Appendix II. In this phase we applied the research methods for case studies by Runeson and Höst (2008); Miles and Huberman as both the principles for case study research by Yin (2009). The steps of the case study research method by Runeson and Höst (2008) are discussed below.

1. *Case study design*. Objectives are defined and the case study is planned. The research objective is formulated in accordance with the case company and the interviews have been planned in advance.
2. *Preparation for data collection*. Procedures and protocols for data collection are defined. To ensure all steps are carried out consistently, the case study design and data collection plan is developed using the protocol template for case study planning by Brereton, Kitchenham, Budgen, and Li (2008). Thereafter the protocol is validated by using the checklist of Runeson and Höst (2008).
3. *Collecting evidence*. Execution with data collection on the studied case. Interviews are held using a semi-structured interview technique. The interview questions are mapped to the main themes regarding to DevOps and the goals of the case study.
4. Analysis of collected data. Distill findings from the collected data. In order to answer the research question at the beginning of this chapter, we have to set up distinct groups of answers. For this step we followed the qualitative data analysis approach by Miles and Huberman (1994), which is discussed in section 4.1.3.

5. *Reporting.* Present the findings to the audience. The findings of the interviews are mapped to their main themes and presented in a matrix. The final results are discussed in section 4.3.2.

4.1.1 Case selection

In this case study we investigate the Scrum process of a project team with a high maturity, which is also available to participate in a pilot experiment. In this manner we try to avoid the problems are inherent to their Scrum process (i.e. the alignment between business and development) rather than DevOps (i.e. the alignment between development and operations). During a regular project, development teams are assessed by an external company that determines the Scrum maturity. Teams with scores above 3 are considered to be mature. For this case study we have selected a single team with an overall maturity score of 3.2 / 5. Therefore, the case study design consists of a single-case with a single unit of analysis. When the baseline (i.e. PDD and issues) is applicable to multiple teams, these teams will also be included for the case study pilot in the desired situation phase.

4.1.2 Data Gathering

This case study is provided with data from the following sources:

- *Interviews.* Main source for asking targeted questions in order to elicit main drivers and issues regarding to DevOps.
- *Documents*: process instructions, project documents, wiki, presentations, summaries of retrospective meetings.
- *Direct observations.* Since our research took place at a development team of CaseComp, we are able to attend all kinds of meetings (e.g. start and mid-sprint sessions, retrospective meetings, daily stand-up meetings). Important observations are documented.

4.1.3 Processing the Interview Results

A qualitative data analysis approach is used to extract suitable data from interview transcripts which are stored in the central case study database. We followed the three steps of Miles and Huberman (1994) which suggest that qualitative data analysis consists of three procedures:

1. *Data reduction*. Qualitative data is reduced and organized by discarding irrelevant data and assigning codes to relevant data.
2. *Data display*. In order to draw conclusions, good display of data is essential. Such as tables, charts, summaries and diagrams.
3. *Conclusion*. Develop conclusions based on the analysis, by comparing, contrasting, searching for patterns, triangulation etc.

An important aspect of the data reduction process is coding qualitative data. According Miles and Huberman (1994): “Codes are tags or labels for assigning units of meaning to the descriptive or inferential information compiled during a study. Codes are usually attached to ‘chunks’ of varying size – words, phrases, sentences or whole paragraphs”. In the context of the research, codes are oriented on the problems and issues arisen from the current development process. The procedure is supported by four stages for data coding:

1. *Open coding.* All statements related to the research question are identified and each is assigned a code, or category.
2. *Axial coding.* By using the developed codes, the researcher is able to search for statements that may fit into any of the categories.
3. The researcher analyzes the codes to look for patterns and explanation.
4. *Selective coding*. Raw data is analyzed again with the purpose to illustrate the analysis, or explain the concepts.

Using this procedure each interview transcript is scanned for internal forces, external forces and requirements that are relevant to the implementation of DevOps. All statements are placed in a spreadsheet document by assigning them to a category. Note that no frequency numbers are added to this table. As stated by Krane, Anderson, and Stean (1997): "Placing a frequency count after a category of experiences is tantamount to saying how important it is; thus value is derived by number. In many cases, rare experiences are no less meaningful, useful, or important than common ones. In some cases, the rare experience may be the most enlightening one".

4.2 Scrum Process Assessment

Part of the current situation phase is to develop process-deliverable diagrams for the Scrum process at CaseComp. The diagrams are used as baseline for implementing the proposed process improvements. The pilot case study in the desired situation phase only includes the development teams that comply with the baseline. A Scrum reference method is used to validate the Scrum method at CaseComp. Expert validations tuned the reference method to the project-specific situation of the case. We start with an explanation of the meta-modeling technique to provide the reader with basic knowledge for understanding a PDD. Then the reference method is provided and adaptations to this method are discussed. Finally the Scrum PDD for the case is provided accompanied with the concept and activity tables.

4.2.1 **Meta-Modeling Process**

The meta-modeling technique presented by Van de Weerd and Brinkkemper (2008) is used for crafting a baseline for the current situation. This technique is used to elaborate on the development process. The resulting diagram is called a process-deliverable diagram (PDD), which combines the UML process diagram and UML class diagram into a single diagram. In the PDD, each activity from the process side is linked to an artifact (e.g. document, requirement, etc.) on the deliverable side of the diagram. In this section we briefly discuss the basic concepts for understanding a PDD.

The process side of the PDD consists of activities and transitions. The activities may also embed sub activities to support hierarchical activity decomposition. For the creation of a meta-process model, there exist four types of activities (Figure 12):

- *Standard activity:* an activity that contains no further sub activities.
- *Complex activity:* an activity that contains several sub activities. A complex activity could either be an *open activity* or a *closed activity*.
- *Open activity:* a complex activity of whose sub activities are described. These activities may be described in the same diagram or in another diagram, therefore two notational variants exist.

- *Closed activity:* a complex activity of whose sub activities are not described since it is not known or not relevant in the specific context.

The transitions show the control flow between activities, of which four types exist: sequential, unordered, concurrent, and conditional activities. Sequential activities are connected with an arrow and need to be performed in a predefined order (Figure 13). Unordered activities do not have a predefined execution sequence and can be performed in any order. Concurrent activities are executed concurrently. A synchronization bar is depicted for the purpose of forking and joining the activities. Conditional activities are only performed if a predefined condition is met. The branch is depicted using a diamond and both an incoming and outgoing transition.

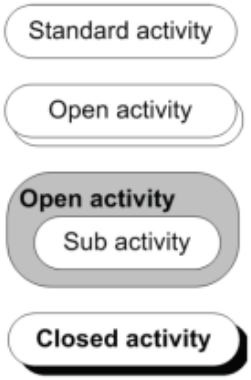

Figure 12. Activities types

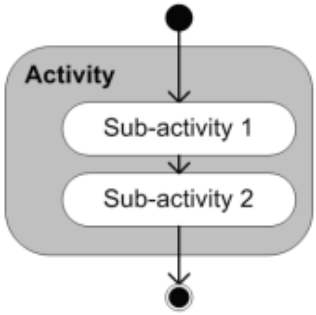

Figure 13. Sequential activities

The deliverable side of the process-deliverable diagram consists of a concept diagram, of which the important parts are discussed below. The diagram supports the following concept types (Figure 14):

- *Standard concept:* a concept that contains no further concepts.
- *Complex concept:* a concept that consists of several concepts. A complex concept could either be an *open concept* or a *closed concept*.
- *Open concept:* a complex concept of whose sub concepts are described. The aggregate structure may be described in the same diagram or in another diagram, therefore two notational variants exist.
- *Closed concept:* a complex concept of whose sub concepts are not described since it is not known or not relevant in the specific context.

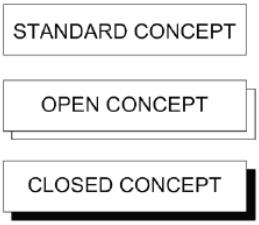

Figure 14. Concept types

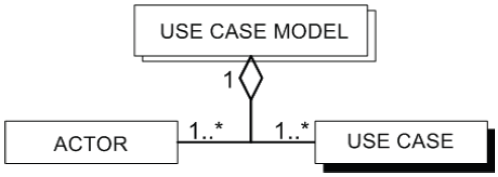

Figure 15. Example of standard, open and closed concepts

Other important concepts in meta-delivery modeling are: generalization, association, multiplicity and aggregation. Generalization is used to express a relationship between a general concept and a more specific concept. It is visualized by a solid arrow with an open arrowhead, pointing to the parent. An association is used to describe the structural relation between two or more concepts and is visualized with an undirected solid line. Multiplicity is a characteristic of a relationship between concepts, it states how many objects of a certain concept can be connected across an instance of association (e.g. *1..** corresponds to one-to-many). An aggregation represents the relation between a concept containing other concepts. An example of an aggregation and multiplicity are illustrated in Figure 15.

Finally, a PDD integrates the meta-process model and meta-data model by connecting a dotted arrow from the activities to the deliverables. More details about the syntactical structure of the PDD can be found in the paper by Van de Weerd and Brinkkemper (2008).

4.2.2 Scrum Reference Method

The Scrum approach consists of core practices and additional practices. The core practices have been grouped into a Scrum guide (Sutherland & Schwaber, 2011). Additional practices are maintained by the agile community. These practices come from practitioners who elaborate on new patterns, e.g. Välimäki and Kääriäinen (2008) proposed patterns that can be applied in distributed project teams. ScrumPLoP (2012) provides a comprehensive overview of the patterns that are available to Scrum practitioners. The patterns are divided into various categories, such as team patterns, retrospective patterns, and organizational patterns. Scrum is no one-size fits all approach, and thus, practices need to be tuned at hand to meet the project environment. For example the length of the sprint may vary from project to project. The ability to adjust Scrum to the situation is confirmed by Beedle, Sharon, Schwaber, and Sutherland (1999), who describe the Scrum method as an extension pattern language to the existing organizational pattern languages. An organization selects the Scrum patterns that are applicable to their specific situation (Beedle et al., 1999). Due to the customizability of Scrum, we cannot fully assume the general process description applies to CaseComp. Therefore we have to validate the Scrum process that is used within CaseComp. For this purpose we use a Scrum reference method.

The Scrum reference method is developed by Blijleven (2012) using the meta-modeling technique as described in the previous section. The method is based on the process definition of Schwaber (1995), the creator of the Scrum approach. A PDD of the reference method is depicted in Figure 16. This blueprint is used to validate the Scrum process at CaseComp. However, a fully comprehensive explanation of Scrum is outside the scope of this chapter and can be found in Blijleven (2012); and Schwaber (1995).

4.2.3 Adapting the Reference Method

The PDD suggests the development process uses a waterfall approach as the process proceeds from top to bottom. This contradicts with the iterative approach from the agile philosophy. Therefore, for the adapted reference method we make a distinction between two types of activities on the PDD, one-off and recurring activities. One-off activities are only performed at the beginning or end of the project, such as setting up the project plan or creating marketing materials. Recurring activities are performed in every sprint and address the iterative aspect of the process.

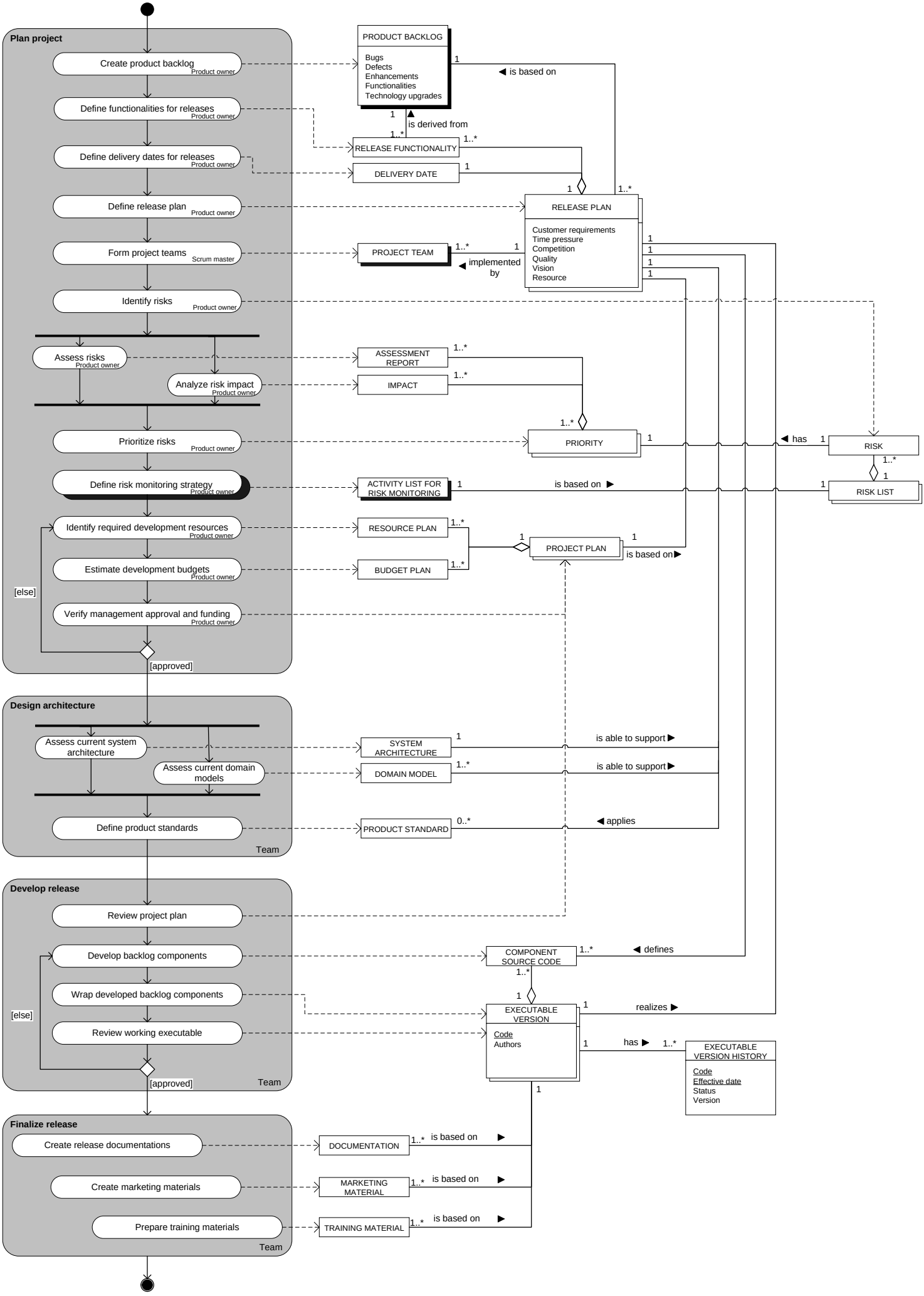

Figure 16. Process-deliverable diagram of the Scrum reference method (from Blijleven, 2012)

We considered to add arrows that flow back into previous steps, but found diagrams became hard to interpret as the PDDs contained lots of detail. For clarity purposes we introduced a cyclic icon (Figure 17) that is positioned right next to recurring activities, to make the distinction clearly visible.

Figure 17. Figure for indicating a recurring activity

Before we discuss the adaptations to the PDD, we explain how the incremental differences can be interpreted. The concepts and activities of which name or type is changed are colored light grey. Activities and concepts that are inserted to the reference method are colored dark grey. Crossed stripes indicate concepts or activities which are removed as they do not apply in the current situation. A legend is shown in Figure 18.

Insertion

Modification

Deletion

Figure 18. Legend for method increments

4.2.4 The Scrum Process at CaseComp

In this section we discuss the adaptations made to the reference method (Figure 16) in order to meet the project-specific situation of the case. In the end the final PDD is provided together with the activity and concept tables. When elaborating the Scrum process we have to take into account the feedback mechanisms that are inherent to Scrum, such as the daily stand-up, mid-sprint review, and retrospective meeting are omitted in the Scrum reference method as the results are intangible, meaning no specific deliverables are generated (Blijleven, 2012). Any changes to these meetings should therefore be textually described.

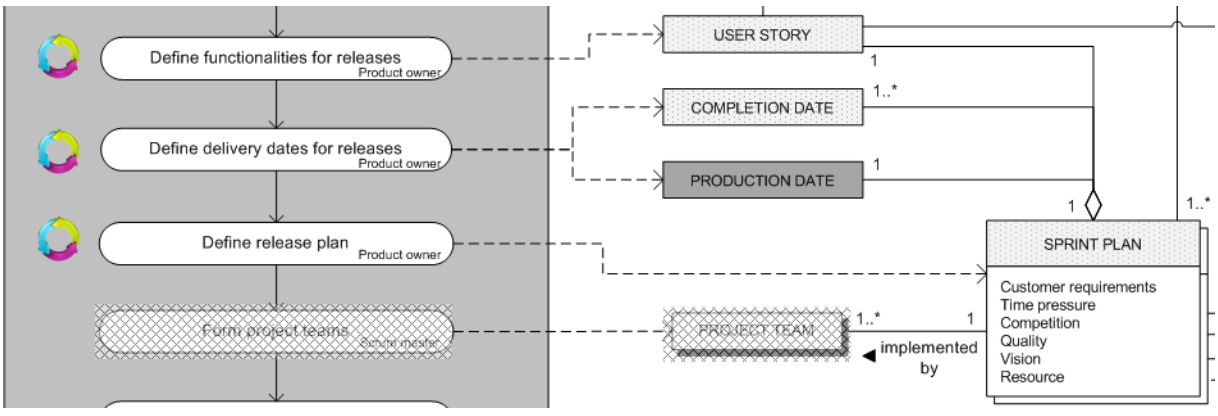

Figure 19. Reference method: deleted project team, inserted production date

To ensure the baseline is consistent with the development process at CaseComp, we use the names of the concepts and activities that reflect the actual process. Therefore the name of seven concepts has changed (consult Appendix III for more details). The PROJECT TEAM concept and its

corresponding activity are deleted, as the team is already established before the Scrum process takes place (Figure 19). At CaseComp a project team is assigned to an application module prior to the execution of the project.

Besides a COMPLETION DATE is set during the first Scrum phase, one indicating when the sprint is complete and the user stories are built, there is another date concept introduced. As the day of completion and the day of the actual deployment of the working software build are never on the same day, we need a new concept called PRODUCTION DATE (Figure 19). CaseComp has four release moments a year to deploy the information system (IS), while parts of the IS are developed in biweekly sprints or iterations. This means working builds stack up and wait for deployment. Because of this we also need a new concept called RELEASE, which aggregates all working builds into one singe release (Figure 22). Once the PRODUCTION DATE is met and all sprints are completed, the release is handed over to the operations department that deploys the IS.

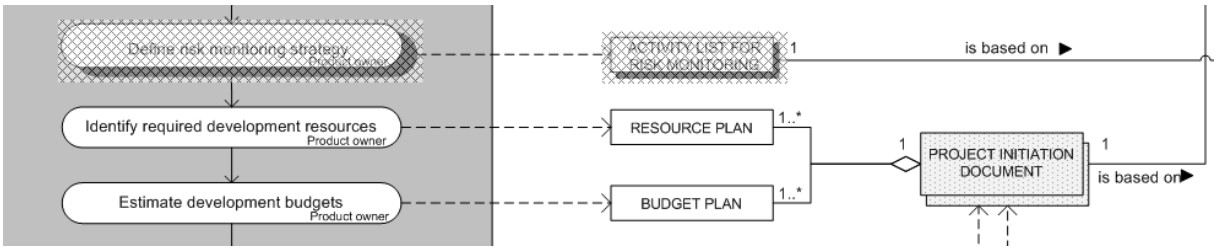

Figure 20. Reference method: deleted risk monitoring strategy

Instead of defining a risk monitoring strategy for the project, the risks are determined in a simple way and hence there is no need for a complex activity and concept (Figure 20). At CaseComp identified risks provide new input for existing user stories. For example if it appears that the application module is sensitive to memory leaks, there is added a new task to review the application for possible memory leaks. The consulted experts indicate that risks are not actively addressed. They argue that top prioritized risks should be eliminated at an early stage.

During the *Plan project* activity the PROJECT PLAN is elaborated and afterwards sent to the project initiators for approval. There is only need for one PROJECT PLAN at the beginning of the project, before the first sprint takes place. Whether there is given a formal agreement or not, the project still continues. Team members are hired on a project basis and therefore cannot be without work.

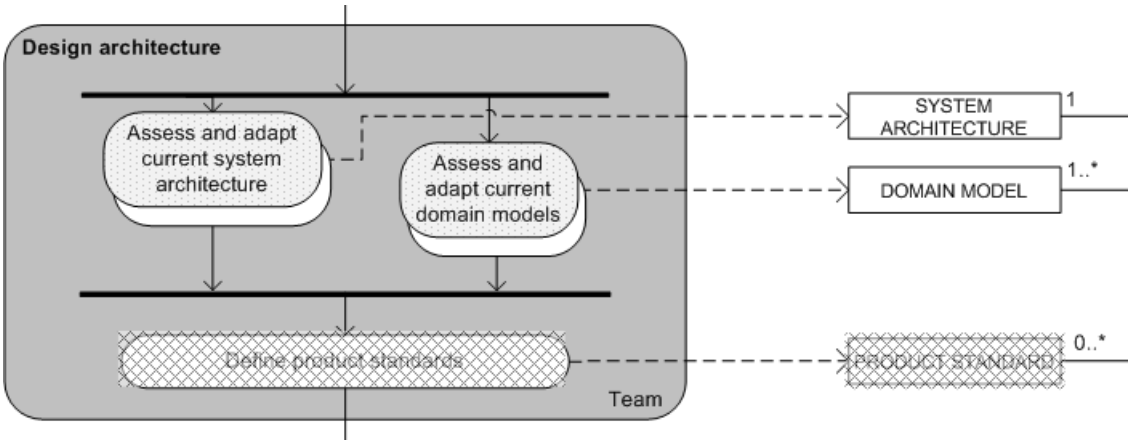

Figure 21. Reference method: updated assessment, deleted product standards

The parallel activities to assess the current system architecture and domain models are changed to an open complex activity (Figure 21). The activities now also include the refinement of the system architecture and domain models when these are insufficient to support the user stories. The activity *Define product standards* and its related concept are deleted since no product standards are defined for the project (Figure 21). Instead, there is a reference architecture available that is used by all project teams.

The *Review working executable* of the *Develop release* phase is renamed to *Test working build*, as it better reflects the underlying activities. The concept has also changed to a complex concept. Part of the test is also the sprint demo meeting as part of the mid-sprint review, in which the customer gives its commitment to the project by giving a formal approval.

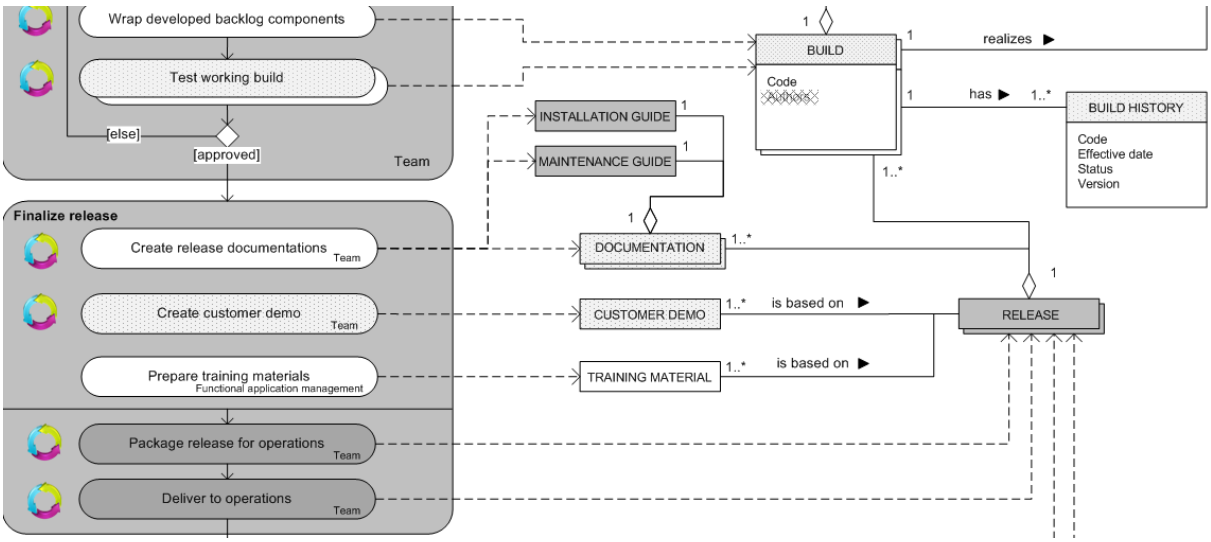

Figure 22. Reference method: changes to finalize release

The *Finalize release* activity is expanded by two sub activities (Figure 22). The first sub activity is *Package release for operations* which combines the builds into a single release together with the required software, installation files, and release documentations (such as release notes, installation guide, and maintenance guide). The second new sub activity is *Deliver release to operations*, in which the release package is handed over to the operations department. Afterwards, the team is standby to fix any errors that occur during the production acceptance test (PAT) and deployment to the production environment. The two latter activities are included in the *Provide support* activity.

So far we have discussed the adaptations to the Scrum reference method. An overview of the adaptations to the activities and concepts is provided in Appendix III.

4.3 Results

4.3.1 Process-Deliverable Diagram for the Current Situation

The final PDD is depicted in Figure 23 which illustrates the incremental differences as well. The PDD is used as baseline when implementing the process improvements in the next research phase. The corresponding activity and concept tables are provided in Appendix IV. These tables provide brief descriptions of the activities and concepts used in the PDD.

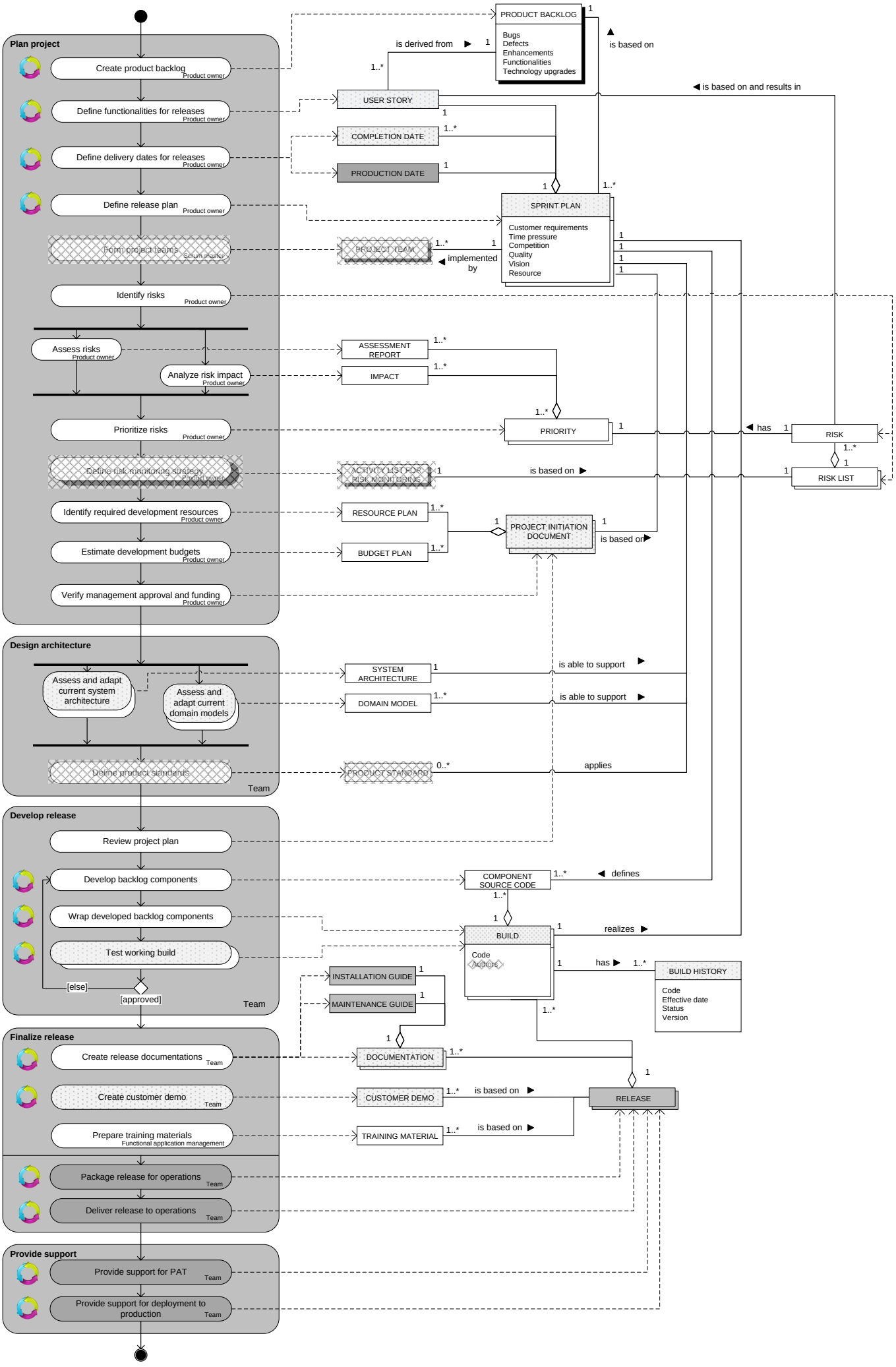

Figure 23. Adaptations to the Scrum reference method

4.3.2 Main Drivers and Requirements for DevOps

To come to a structured answer on the research question, we applied the qualitative data analysis approach as described in 4.1.3 for analyzing statements in interview transcripts. To recap, the research question was formulated as follows:

> *SQ1. What are the main drivers and requirements for an organization to integrate DevOps into their Scrum development process?*

The following user roles are subjected to an interview in order to answer this research question: Scrum master, developer, tester, product owner, functional application manager, technical application manager, and implementation manager. Each role is involved in the software delivery process and has experience with the team for at least one year. In case there are multiple persons available for a single role, the participants are randomly selected across the development team. According Miles and Huberman (1994) this enhances generalizability as well as deepening the understanding and explanation of a phenomenon.

We distinguish 8 unique codes or key drivers and implementation requirements at CaseComp that support the need for DevOps practices. The key drivers are related to internal and external driving forces (e.g. problems experienced in certain process areas). Implementation requirements refer to the organizational requirements for implementing adapted processes. The answer on the research question is summarized in Table 1 and is discussed below. We link the DevOps layers by Debois (2012) to key drivers so we are able to focus on the problem areas related to the process (i.e. code 1-5), which is the scope of the research. Note that the findings are unsorted and weighted equally.

Code / category	Type	Layer
1. The processes of the development and operations departments are not aligned with each other.	Driver	Process
2. Lack of standardization for quality guidelines.	Driver	Process, tools
3. IT Operations is not well represented in the project.	Driver	Process
4. Too comprehensive process for releasing information systems.	Driver	Process, tools
5. Moderate communication between development and operations.	Driver	Process, people
6. Dispersed or missing knowledge on development and operations.	Driver, requirement	People
7. Too complex IT infrastructure.	Driver	Tools
8. Tools are not aligned with the process.	Driver, requirement	Tools
9. For each process change, time and resources have to be estimated.	Requirement	-

Table 1. Identified drivers and requirements for DevOps

1. *The processes of the development and operations departments are not aligned with each other.*

The interview results indicate the processes of the development and operations departments are not aligned with each other. Development uses an agile way of working and teams are assigned to a project. This is in contrast with the operations department at CaseComp, which uses a less flexible waterfall method. According to developers “there are many people outside the process who want to believe they are stakeholder of the project, though they are not ready to work with agile“. The operations department is more business-oriented rather than project-oriented with the aim on

stable information systems. The development (sprint) schedules are not aligned with the release schedules. As the sprint length is two weeks and there are four release moments a year, there is a huge impact of each release that is deployed to production. Furthermore, when the development team is unable to meet the deadline – fixed time schedules for testing and deployment forces operations to perform the same activities within a smaller amount of time. As a result there is less time available to provide feedback for the development team. The processes regarding to both departments are not consequently executed. At the development side the Scrum implementation differs among the individual project teams so operations does not know what they can expect from each team. Also, the project team is not well informed on the processes at operations.

2. *Lack of standardization for quality guidelines.*

There is a lack of standardization for quality guidelines across the development and operations departments. For example, there are no proper requirements for logging mechanisms. Also, operations does do not provide a coherent checklist (e.g. for release notes) that can be used by all development teams. Current guidelines for obtaining approval to put an application into production are not adequately addressed as they are selectively monitored. According to one of the participants "one time they perform a syntactical check, whereas the other time they perform a quantitative check".

3. *IT Operations is not well represented in the project.*

Operators, such as functional and technical application managers are too late involved in the development. Developers have already begun developing the system without consulting operations for e.g. technical requirements. Hence operators are missing project-context and do not know what they can expect from the system. Initiatives have already tried to bridge the departments by inviting the operators to development meetings. During these meetings, the application module is demonstrated and the completed user stories are discussed. The operators indicate these demo meetings are a good attempt to improve communication, but turned out to be unnecessary to attend as only functionalities are communicated. Details regarding to infrastructure and application services were omitted. Despite the resources are made available, no attention is paid to the added value for operators during these meetings.

4. *Too comprehensive process for releasing information systems.*

To put an application module into production you have to go through a complicated change management process. Five types of approvals are required to put the system into production. The processes make the actual deployment more complicated and time-consuming. Also it slows the feedback for the system development. According to some interviewee "there is too much hassle to get something to production, e.g. release notes are not consistent or missing a comma somewhere. Technical application management is very strict in the release notes - a tough process."

5. *Moderate communication between development and operations.*

The involved parties are unsatisfactory about the way and frequency of communicating feedback to each other. Personnel from development indicate they are "dissatisfied with the way we communicate and the way in which feedback is given". The departments have also too little insight into each other's activities. For example development seldom knows which changes result in an increased number of reported incidents. Finally, the development and operations departments are

separated over two physical locations which make it impossible to walk along a colleague. Therefore they use e-mail as their main communication, which is an impediment for informal communication.

6. *Dispersed or missing knowledge on development and operations.*

Respondents notice a gap in the knowledge and experience of colleagues. Some argue that "the lack of knowledge sharing impedes the collaboration between development and operations". Due to the strict separation of the departments, only necessary information is exchanged. Users keep their own knowledge up to date, but they are not aware of new developments on the topics of its colleagues. Due to this knowledge gap people do not know what they can expect from each other. This issue is also considered an implementation requirement, as the basic understanding of jargon and know-how is a precondition for enabling close cooperation.

7. *Too complex IT infrastructure.*

At CaseComp there exist multiple environments for the development and testing activities. As the environments have their own configuration, requirements, and procedures and these are managed separately, we consider them as too complex. Examples of complex infrastructures include chain testing and the deployment of a fully furnished project environment. As the configuration of the test environments are not updated adequately, it ultimately leads to increased cycle time and creates obstacles to the customer acceptance test.

8. *Tools are not aligned with the process.*

This issue corresponds to the tools that provide insufficient information to the user. The tools are not aligned with the development process as the desired information could not be retrieved or does not match the actual situation. We explain this issue by providing brief examples provided by the interviewees:

I. Multiple releases are constructed in a single sprint but cannot be made visible in the project management system.
II. One of the principles for continuous integration aims that a failed build should become quickly visible for the team. This principle is not properly addressed.
III. The development team cannot retrieve the log files of their application module as they do not have the required access rights.
IV. Feedback or requests between the departments are mainly handled by e-mail due to the absence of a central tool for both departments. We consider this issue also an implementation requirement as the proposed process changes should be covered by tools, otherwise temporary workarounds are needed to cover them.

9. *For each process change, time and resources have to be estimated.*

A formal implementation requirement that is elicited during the interviews is the one stated above. The management should be timely informed on the amount of time and resources that is needed for the project team. An estimate for each process change has to be given. The estimations could be determined by the use of planning poker as applied in Scrum. In this way a unanimous consensus can be achieved between the involved parties.

5. Desired Situation

In this chapter we propose several process improvements that address the captured drivers from the previous chapter. Before we are able to incorporate the practices that are needed by the case company into the current situation, we first elaborate on the approach that links DevOps patterns to the identified problem areas as part of the situational method engineering (SME) approach. Finally, the situational method for the desired situation is captured using the meta-modeling technique. This chapter provides an answer on the second and third research subquestion:

> *SQ2. How can method fragments be linked to key problem areas?*
>
> *SQ3. Which method fragments proposed by DevOps address the key problem areas?*

5.1 The Creation of a Situational Method

For the construction of the adapted method we apply the situational method engineering (SME) approach by Brinkkemper (1996), which is based on work of Harmsen et al. (1994); Slooten and Brinkkemper (1993). In this section we elaborate on the SME approach which consists of the following steps:

1. Project characterization
2. Selection of method fragments
3. Assembly of the fragments
4. Validation of the situational method
5. Adaptation of the situational method

Method construction depends on the objective of SME, therefore for the research we identified the objective to extend a method by new functionality (Ralyté, 2002). The functionality is derived from DevOps patterns, which is discussed in the second step. According to Ralyté et al. (2003) the technique for extending a method by applying extension patterns is referred to as the extension-based strategy.

5.1.1 **Project Characterization**

Normally, the project characterization leads to suitable method fragments in the SME approach. Characteristics of the project describe the project-specific situation (e.g. level of innovation, expertise). Instead we use main drivers that gave rise to the demand for DevOps. In this point of view the existing method is refined based on the experienced problems, rather than constructing a method from scratch that fits in the situational context. This evolutionary approach is supported by incremental method engineering. The identified drivers for improvement are reported in section 4.3.2.

5.1.2 **Selection of Method Fragments**

The second step is probably the most challenging one in the process of creating a situational method. First of all, we need to codify the relevant DevOps practices and make them implementable for the current process. As we want to improve the current situation, we only include DevOps patterns that

tackle one or more problems. Only DevOps is used as source for identifying new method fragments. According to Harmsen et al. (1994) this is referred to as the uni method involvement (UMI) approach. The advantage of using one method as a source for method fragments is that it omits all kinds of integrity and consistency problems.

Practices, Patterns, and Principles

The second research subquestion relates to the selection of method fragments. For this we need to develop an approach that links DevOps patterns to the main drivers from Chapter 4. To recap, the research question is formulated as follows:

> *SQ2. How can method fragments be linked to key problem areas?*

In contrast to Scrum, DevOps patterns are not formally specified and stored on a single location. Sources include books, Internet blogs and conference presentations. The patterns provided by the community are generic applicable to any organization, therefore we need to make them tangible in order to incorporate them into the current process. To codify DevOps practices, Debois (2012) distinguishes practices, patterns, and principles. By grouping similar practices (either anecdotal or systematically described), patterns arise in the same manner as software design patterns. A pattern is commonly defined as “a description of a general solution to a common problem or issue from which a detailed solution to a specific problem may be determined” (Ambler, 1998).

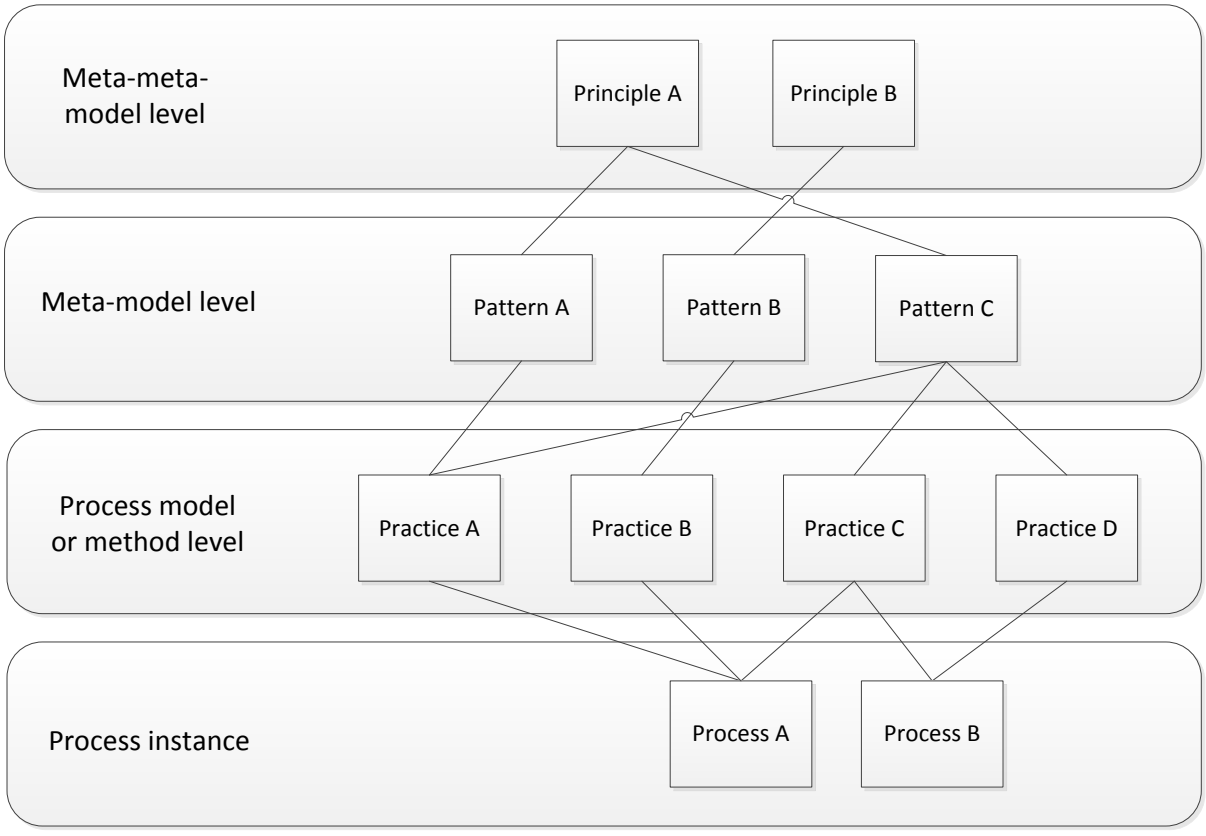

Figure 24. Practices, patterns, and principles

Patterns for system development exist in many variations, such as analysis patterns, design patterns, organizational patterns, and process patterns. The patterns rely on their underlying principles, just like Scrum that is guided by the agile process patterns (Tasharofi & Ramsin, 2007). In Chapter 3 we discussed that DevOps patterns are either focused on the tools, process, or people (culture) layer. The aim of the research is to extend the current process, therefore we only include patterns from the process layer. The hierarchy is made visible by placing the practices, patterns, and principles on the method abstraction levels (Figure 24). This figure clearly shows practices are distillated from process patterns to incorporate them in processes at IT organizations. This means the identified process patterns need to be codified as practices for the situational context at CaseComp in order to incorporate them on the same level (process model) as the baseline method.

Method Selection Techniques

In a regular SME project, the project characterization is input to the selection process (Brinkkemper, 1996). Some multi-criteria techniques for selecting method fragments are discussed by Kornyshova, Deneckere, and Salinesi (2007). Examples include simple addition, weighted sum, and outranking. The techniques aim on choosing the most appropriate method fragment (or chunk) from a collection of method fragments, based on a predefined set of criteria. However, problem-solving based selection techniques are very scarce. A MEMA-model is proposed by Punter (1996) which is based on an extensive investigation of the problem situation and method characteristics. The MEMA-model selects suitable modeling techniques that address the problem characteristics on different abstraction levels. In contrast to the regular selection process in SME, we start with the selection of method fragments and validate this choice against the main drivers from Chapter 4. Currently there is no approach that supports this purpose, therefore we propose the process pattern mapping approach.

Process Pattern Mapping Approach

The process pattern mapping (PPM) approach supports the selection of process patterns and the assembly of method fragments in the creation of a situational method. The PPM approach (Figure 25) maps process patterns to problem areas and transforms the required process patterns into method fragments in the following seven steps:

1. Collect the drivers for improvement.
2. Record all related process patterns by means of a brief description.
3. Set up a matrix by placing the main drivers on the vertical axis and the process patterns on the horizontal axis.
4. Indicate which process patterns address the drivers by ticking the corresponding cells.
5. Select the process patterns that cover at least one driver.
6. Describe in detail the selected patterns.
7. Elaborate on the method fragments for the given project context.

For step 1 we already identified the drivers for improvement in section 4.3.2. Since the sixth till the eighth business driver are not related to the process, we exclude them from the first step. Based on a thorough investigation of mixed sources, such as books, Internet blogs, and conference transcripts we identified a set of 30 DevOps patterns. An overview of the collected DevOps patterns is provided in Appendix V. In step 2 we only included patterns from DevOps area 4, which is the scope of the

research. To recap, DevOps area 4 embeds operational knowledge into the project by extending the development process. Step 4 is supported by literature, which prescribes process patterns to solve problems. Also, this step is validated by means of an expert interview. The *quality scenarios* pattern (P5) is not directly related to any of the problem areas, therefore the process pattern is excluded in step 5. *Quality scenarios* (P5) elaborate on quality requirements using scenarios. We consider this pattern as an extension for *integrate production stories* (P4), which introduces quality requirements to the product backlog.

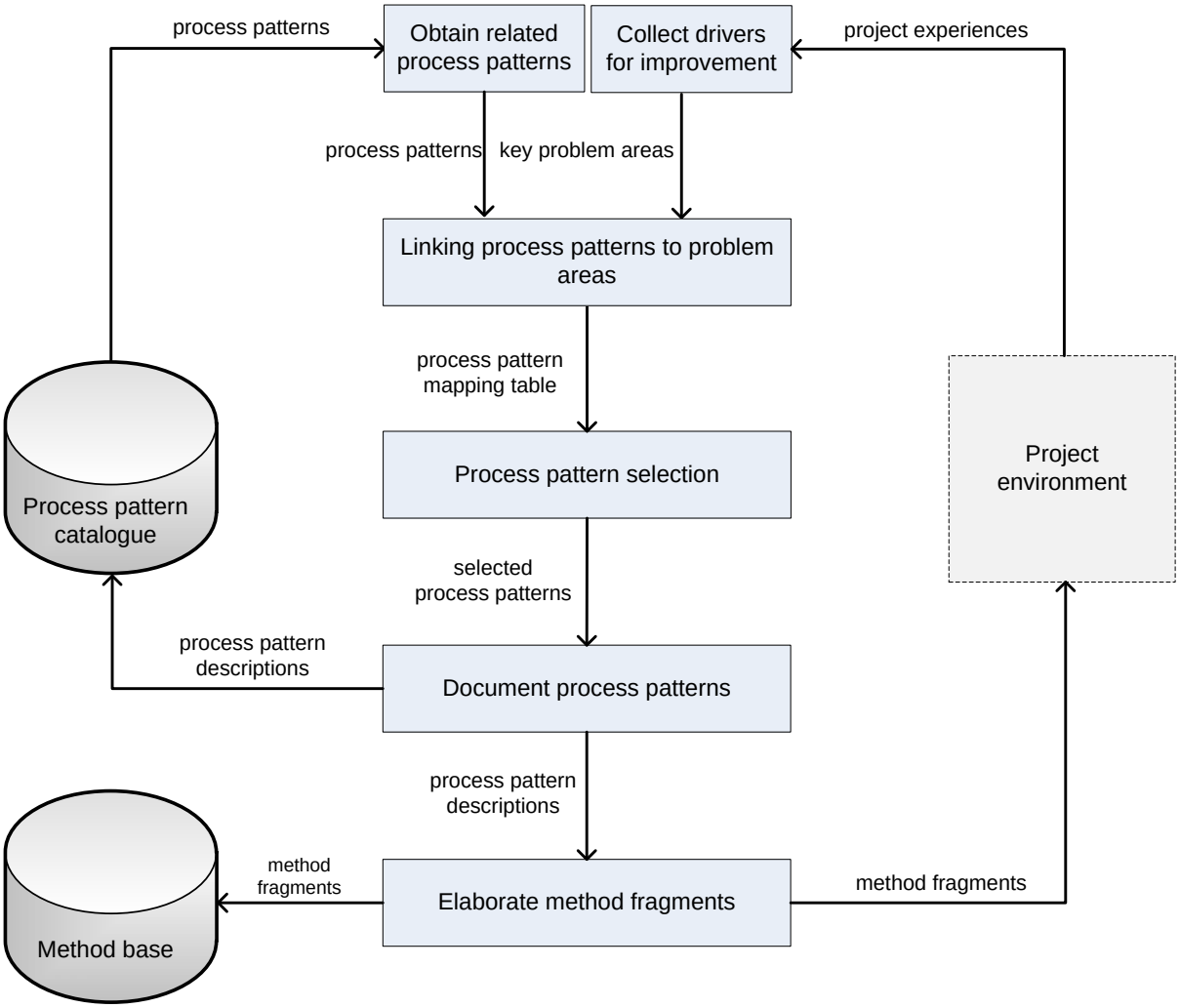

Figure 25. Visual representation of the process pattern mapping approach

Despite the tasks described by PPM are quite easy to perform, some of them are very time consuming. Such as obtaining all relevant patterns and documenting them. The former requires that the scope is determined for the SME effort. In this case study we were limited to DevOps process area 4 only. The latter needs an extensive investigation of mixed sources. The approach becomes harder to use when dozens of process patterns are included. Therefore, process pattern mapping can be helped when all process patterns are documented on forehand. The documented process patterns are stored for further reuse to the process pattern catalogue. The resulting method fragments may be different when assembled in the process for another organization.

PPM does not resembles an enhanced version of the situational method engineering (SME) approach by Brinkkemper (1996). Instead, PPM supports the SME approach in the codification of practices for the project context. The PPM approach is supported by the pattern based process model proposed by Ralyté et al. (2005). According to this model, PPM supports SME by two strategies: situation-based and goal-driven, which supplement each other. The goal-driven strategy identifies a set of atomic actions to be carried out in order to achieve the goal, whereas the situation-based strategy considers possible situations in which these goals are relevant (Ralyté et al., 2005).

The PPM approach can be incorporated by a computer-aided method engineering (CAME) tool to support problem-solving in SME. Such tool makes the pattern mapping table and manual selection irrelevant and the process pattern linking becomes less error prone. The CAME tool may come with a link to the method base, so elaborated method fragments can directly be stored into the method base for further reuse. A major challenge is to document drivers for improvement in a systematical way so the software application is able to propose suggestions for relevant process patterns.

Layers	Patterns / Drivers	P1. Cross-functional delivery team	P2. Develop for production	P3. Early feedback by operations	P4. Integrate production stories	P5. Sync meeting	P6. Quality scenarios
Process	D1. Departmental alignment	X		X	X	X	
Process, tools	D2. Lack of standardization		X				
Process	D3. Ops are not well represented	X		X	X	X	
Process, tools	D4. Complex release process	X	X				
Process, people	D5. Moderate communication	X		X	X	X	

Figure 26. Process patterns linked to the key drivers

The outcome of step 4 is depicted in Figure 26. The matrix shows patterns that are linked to the drivers from Chapter 4. We are now able to provide an answer on the third research subquestion:

> *SQ3. Which method fragments proposed by DevOps address the key problem areas?*

Based on the process pattern descriptions from Appendix VI, we identified a set of DevOps patterns that cover the process-oriented drivers from Chapter 4. The following process patterns are selected for the assembly of the fragments: cross-functional delivery team (P1), develop for production (P2), early feedback by operations (P3), integrate production stories (P4), and sync meeting (P5). As step 6 and step 7 are related to the assembly of method fragments (as in the SME approach), we discuss them in the next section.

5.1.3 Assembly of the Fragments

The selected process patterns need to be captured as method fragments in order to incorporate them into the current process. We apply the process framework of Gnatz, Marschall, Popp, Rausch, and Schwerin (2001), which is based on the process pattern approach by Bergner, Rausch, Sihling, and Vilbig (1998). This framework is a set of basic notions and definitions common for all process

models (or methods) and is described on the level of a meta-model (Gnatz et al., 2001). The process framework (Figure 27) implies that the problem is tackled by one or more process patterns. The process framework looks very similar to the method meta-model by Henderson-Sellers & Ralyté (2010), of which the product part and process model are based on a guideline. Other frameworks using process patterns in SME are the pattern meta-model by Ralyté et al. (2005) and the pattern based framework by Asadi and Ramsin (2009).

A process pattern describes and documents process knowledge in a structured, well defined, and modular way. The pattern is realized by an activity which represents the actual implementation to solve the problem in the situational context. Since DevOps process patterns are located on the meta-model level (Figure 24), we need to translate them to concrete activities in order to integrate them on the method level.

We elaborate on the DevOps process patterns by using the description template for process patterns by Gnatz et al. (2001). The purpose of a pattern description is to grasp the essence of a pattern immediately (Gnatz et al., 2001). The descriptions of the selected process patterns are provided in Appendix VI. Now the process patterns are fully documented, we are now able to elaborate on the method fragments in the project-specific context at CaseComp.

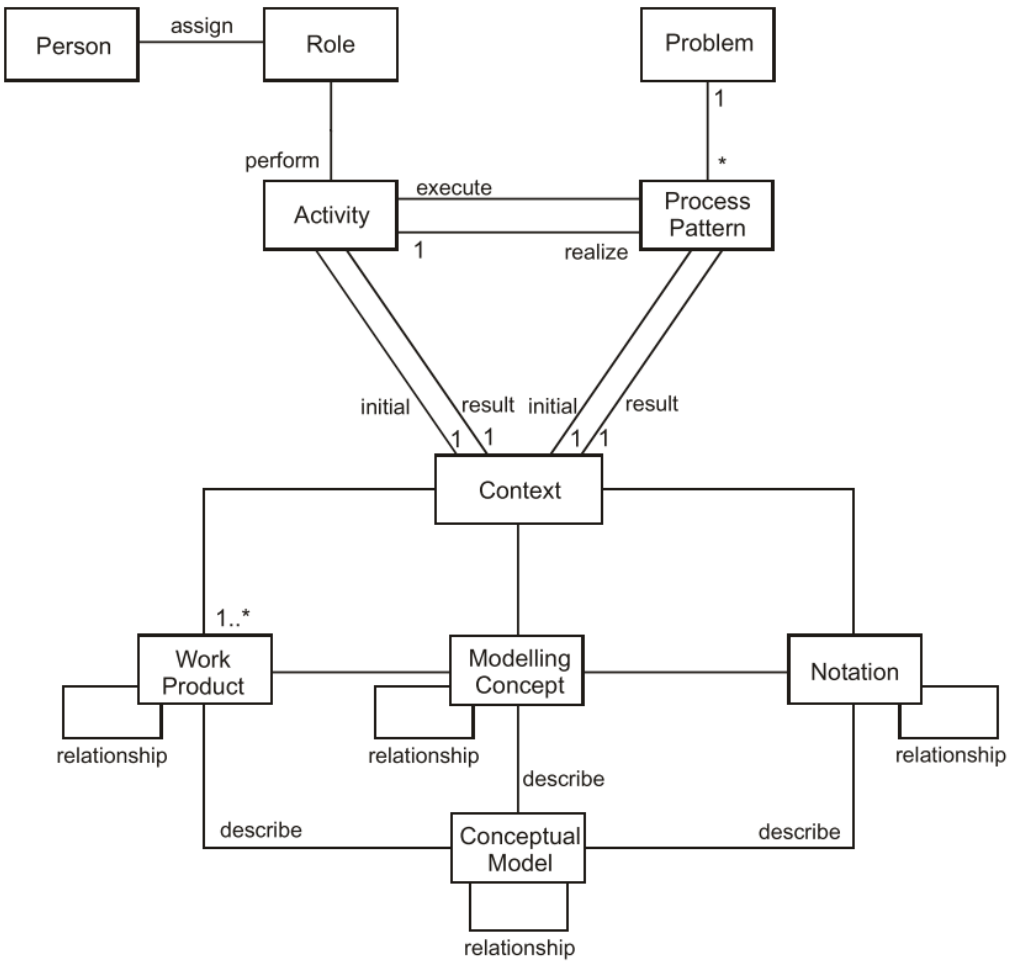

Figure 27. Process Framework (from Gnatz et al., 2001)

The method fragments are documented using the description template for method increments by Van Stijn et al. (2012). This template is based on UML's use case descriptions which enables

organizations to reflect on their software process improvement (SPI) initiatives. Instead of describing success or failed improvement scenarios, we use the template to propose incremental implementation paths for method increments. Therefore we make some minor changes to the template to fit this purpose. As the trigger, pre-conditions and post-conditions are already addressed by the pattern descriptions (labeled as problem, initial context, and result context, respectively), we omitted them from the descriptions. Also, the property *Failed paths* is omitted since no information on implementations is available yet. *Proposed incremental path* describes the incremental changes to the process by using the elementary increment types by Van de Weerd et al. (2007). *Unordered increments* are any additional steps which cannot be accommodated in the sequential implementation steps. A PDD is attached to provide more insight into the impact of the described changes. A legend is shown in Figure 28, which indicates how incremental differences should be interpreted.

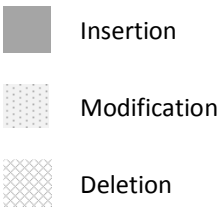

Figure 28. Legend for method increments

Cross-Functional Delivery Team

The purpose of this method fragment is to shape a cross-functional delivery team where team members act together to ensure the software delivery process proceeds smoothly. The major difference with the baseline method is that IT operators are also part of the team, so they are able to support developers in their activities to ensure the information system is production-proof. As the operations department of CaseComp is established in another location, the team would virtually exist (i.e. a geographically dispersed team). However, operators still need to attend the project meetings periodically to ensure both parties are aligned with each other. A prerequisite for this method fragment is that resources are made available for operators so that their regular activities are not at risk. A result of implementing this fragment is that the definition of done (DoD) changes for the project team. The updated definition includes the actual delivery to production. The sub activities are defined by Hüttermann (2012), who proposes activities on how to succeed in transforming the work group into a team. The activities aim on setting up shared definitions in a workshop session.

Name	**Cross-Functional Delivery Team**
Goal in context	Effective collaboration and smoother operations.
Scope	Entire Scrum process.
Primary and secondary stakeholders	Developers, testers, Scrum master, technical application manager, functional application manager
Proposed incremental path	1. Introduction of the activity to form a delivery team at the beginning of the *Plan project* phase. The delivery team is responsible for the entire software delivery process and exists either virtual or physical. The activity is executed before the project starts. During this activity a workshop session is held by the Scrum master where all team members elaborate on shared definitions for the team. - *Driver:* development and operations departments are

	disconnected from each other. - *Stakeholders:* Scrum master, technical application manager, developers, testers, product owner.
Unordered increments	- Developers should be educated to make it for operations easier in their work, e.g. they need to know how the technical guidelines related to the IT infrastructure can be properly addressed. - Also, developers and testers need to learn the basics of operations (at least need to know how the software is distributed to the production environment). - Testers should be adequately informed on technical requirements, e.g. how to cope with quality requirements, and how to test them. - All team members should work on their soft skills such as communication and writing skills, to ensure the barriers in communication are eliminated. - The technical application manager and functional application manager should have basic knowledge about development and testing.
Reference to PDD	Figure 30, Figure 29

Table 2. Method increment description: cross-functional delivery team

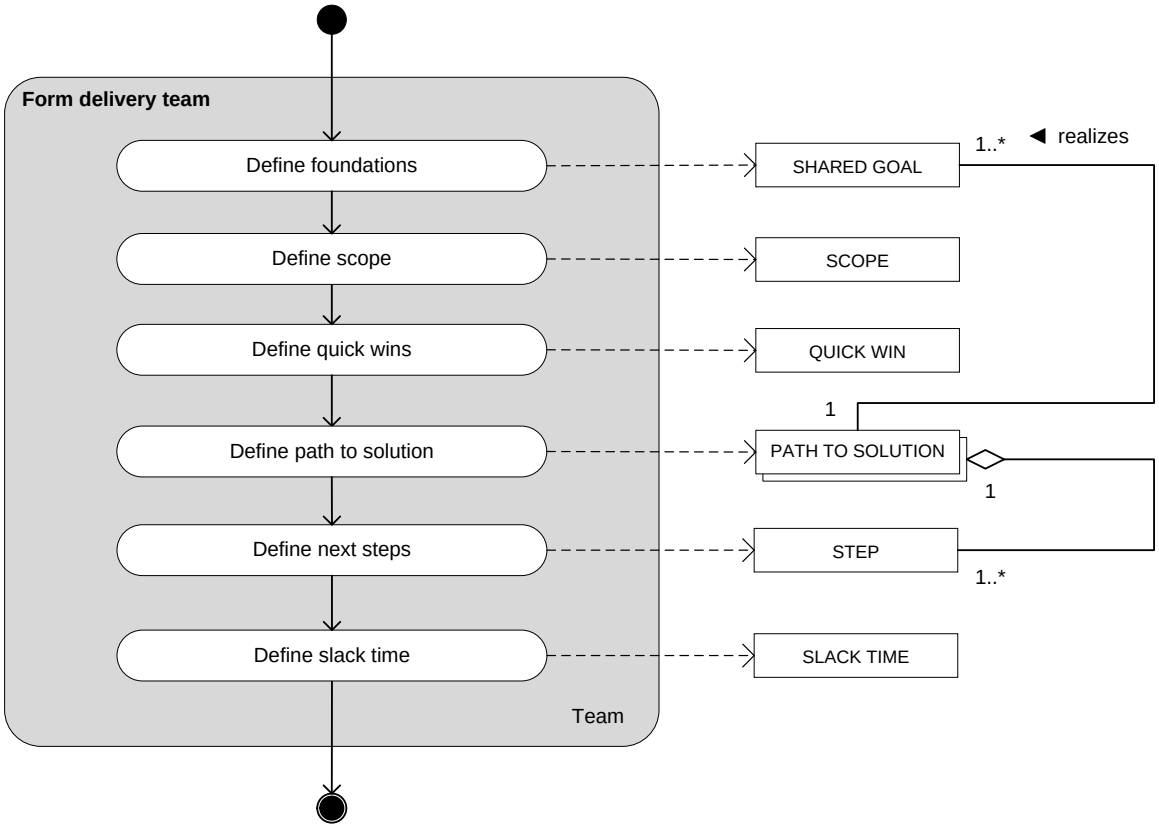

Figure 29. Method fragment: form delivery team

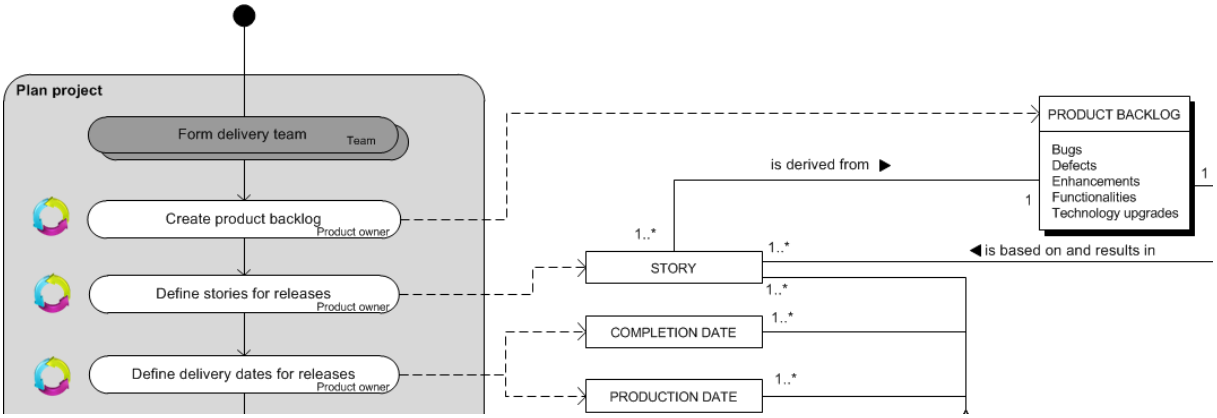

Figure 30. Method increment: cross-functional delivery team

Develop for Production

This method increment ensures the production artifacts are updated in an iterative way. Right after the development of a backlog component, the materials related to production are updated accordingly. These include release documentations such as the installation and maintenance guide, and two scripts. The health script is used to check whether the release is ready to put into production. This script tests the quality requirements. If needed, the database update script is updated accordingly. There has been added a new activity and corresponding concept for each script. As the documents and scripts are in sync with the system, the transition to production is less error prone.

Name	**Develop for Production**
Goal in context	Early creation of operational artifacts.
Scope	Fits into the phase *Develop release*.
Primary and secondary stakeholders	Developers, testers, technical application manager.
Proposed incremental path	1. The name of the activity *Develop backlog components* has changed to *Develop for production*. - *Driver:* The activity now entails more than just the development of backlog components. - *Stakeholders:* developers, testers, technical application manager. 2. Deletion of the activity *Create release documentations* from the *Finalize release* phase. - *Driver:* The operational artifacts were created post-mortem (i.e. when the development is done), which lead to problems in the software delivery process. - *Stakeholders:* developers, testers, technical application manager. 3. Introduction of the activity *Update health script* and its corresponding product concept within the *Develop for production* activity. - *Driver:* To ensure the quality requirements are properly addressed. - *Stakeholders:* developers, technical application manager 4. Introduction of the activity *Update database script* and its corresponding product concept within the *Develop for production* activity. - *Driver:* To ensure the production database scheme reflects the actual situation. - *Stakeholders:* developers

	5. Introduction of the sub activity *Update release documentations* and its corresponding product concepts in the *Develop backlog components* activity. The target activity switches to a complex concept. - *Driver:* to keep the release artifacts up to date during the development. - *Stakeholders:* developers, testers, technical application manager.
Unordered increments	- The project team need to know how the release process works, so developers know what the crucial details are when developing operational artifacts. - The project team should be informed on how the release scripts can be written and executed.
Reference to PDD	Figure 31, Figure 32

Table 3. Method increment description: develop for production

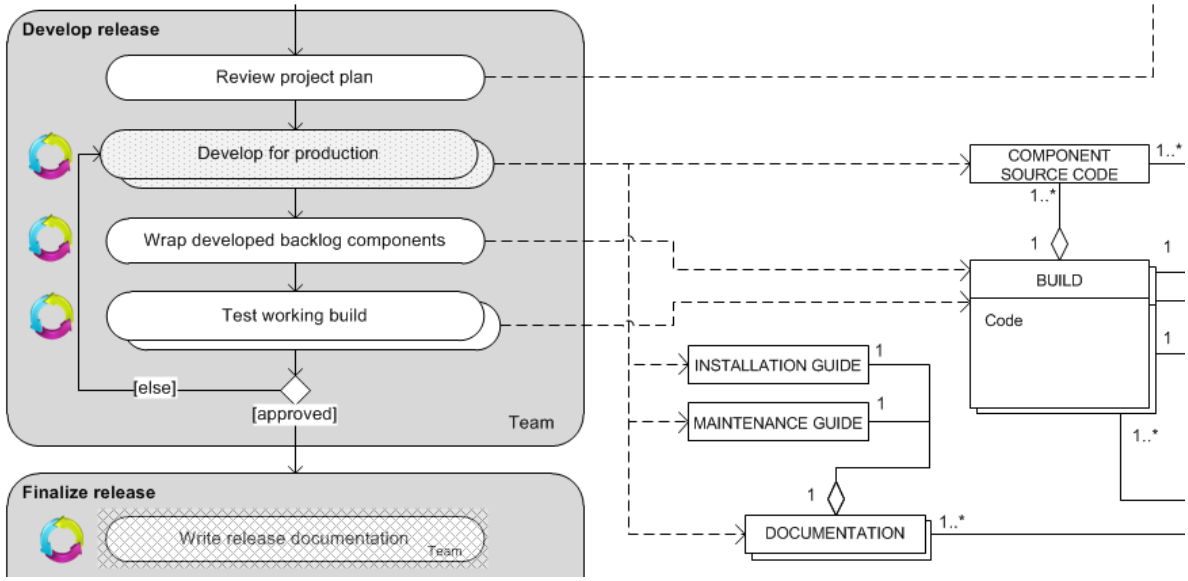

Figure 31. Method increment: Develop for Production

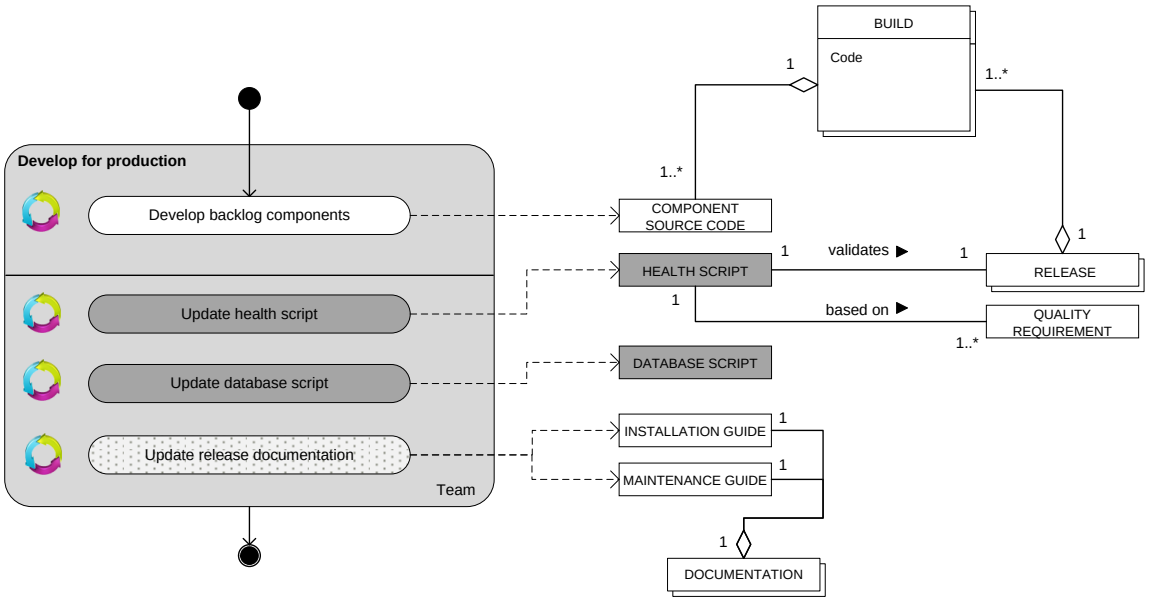

Figure 32. Method fragment: develop backlog components

Early Feedback by Operations

The goal of this method increment is to involve IT operations in the development of the information system so that early feedback is obtained about the design of the system. During the system development, IT operators review the system design in order to identify eventual problems in an early stage. In this manner the system is checked in time so that the system complies with the operational guidelines, such as logging, monitoring, security, etc. IT operations assesses whether the IT infrastructure is sufficient to support the realization of user stories and quality requirements planned for the current sprint. Below we provide a summary of the design documents that are produced during the project at CaseComp. The last four documents are selected for inspection by IT operations as the first two documents only address the creation of user functionalities, which are not relevant for operations. In this method fragment we use the term *SYSTEM ARCHITECTURE* to refer to these four documents.

- ***Project start architecture (PSA).*** This document provides a high level overview of the system architecture.
- ***High level solution (HLS).*** This document describes the required user stories as well the system interactions by the use of sequence diagrams.
- ***High level solution integration (HLSI).*** This document describes the application landscape and service calls across the systems.
- ***Project architecture constrains (PAC).*** This document describes the general architectural principles and the standards to be met. Also it provides a technological model with a detailed description of the systems and interfaces involved.
- ***System development and production environment (SOPO).*** This document describes the development and production environment.
- ***Interface specification & protocols (ISP).*** This document describes the input and output of the system interfaces.

Name	Early Feedback by Operations
Goal in context	IT operators provide feedback about the design of the application under development, early and often.
Scope	*Develop release* and *Design architecture* Phase
Primary and secondary stakeholders	Developers, technical application manager
Proposed incremental path	1. Introduction of the activity *Assess and adapt current infrastructure* and its corresponding concept. - *Driver:* Sometimes the infrastructure is not able to support the new features, therefore the infrastructure should be assessed before the development takes place. - *Stakeholders:* technical application manager (TAB) 2. Introduction of the following unordered activities with its corresponding concepts within the *Assess and adapt current system architecture* activity: *inspect HLSI, inspect PAC, inspect SOPO,* and *inspect ISP*. - *Driver:* The system design needs to be inspected by IT operators to ensure issues are found in an early stage. - *Stakeholders:* technical application manager (TAB) 3. Introduction of *Report findings* within the *Assess and adapt current system architecture* activity. - *Driver:* The inspection findings need to be presented so that issues can be corrected on time. - *Stakeholders:* technical application manager

	4. Introduction of *Adapt system architecture* within the *Assess and adapt current system architecture* activity. - *Driver:* The required change need to be incorporated in the system design. - *Stakeholders:* business analyst integration (BAI), architect, project manager
Unordered increments	- The technical application manager should be informed on both architecture of the system and domain models. - All documentation should be stored on a central location. - Technical application manager should use a formal technique or method to support the design inspection. - The developers are well informed on the current state of the IT infrastructure. - The team should be trained on communication skills. - Technical application managers should attend, at least start-sprint, mid-sprint and demo meetings to provide their feedback on the system.
Reference to PDD	Figure 33, Figure 34

Table 4. Method increment description: early feedback by operations

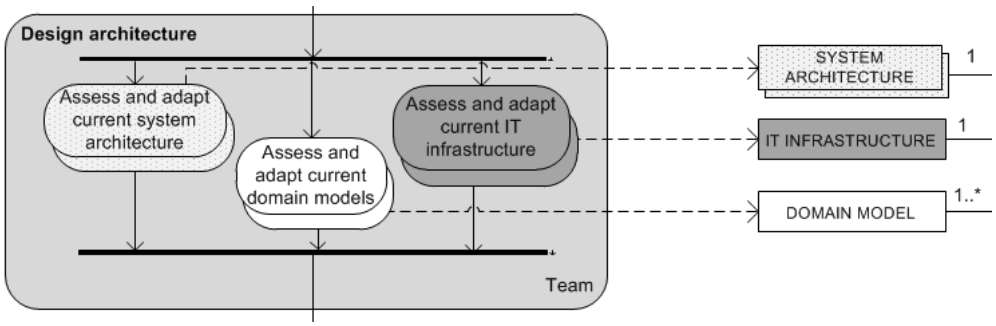

Figure 33. Method increment: early feedback by operations

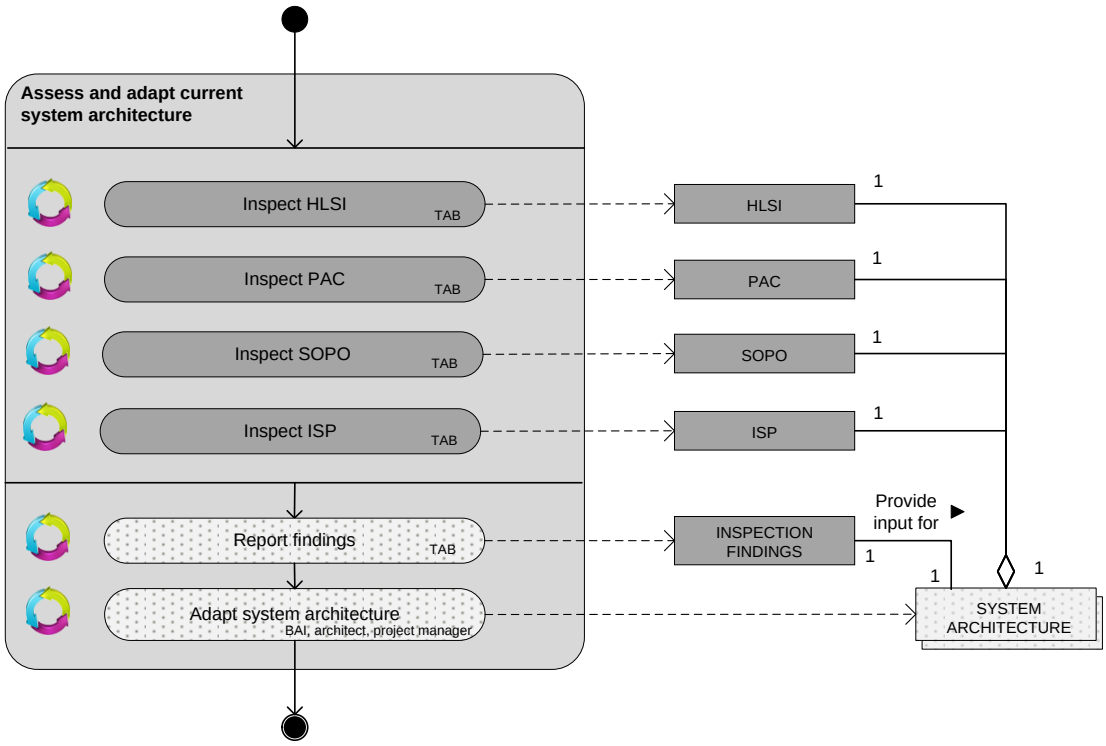

Figure 34. Method fragment: assess and adapt current system architecture

Integrate Production Stories

The following method fragment integrates production stories into the product backlog of the project. Production stories are based on quality requirements (e.g. the system should respond fast) and constrains (e.g. the system should be built on a Unix platform). According to Ambler (2012) there exist three strategies to implement this pattern, therefore we provide an incremental path for each variant. At the end we determine the approach that is most suitable for CaseComp. Ambler (2012) distinguishes production stories, acceptance criteria for individual user stories, and an explicit list for quality requirements. The technical or production stories strategy is identical to user stories. The strategy ensures that production stories are captured as a separate story that is meant to be addressed in a single sprint. Another strategy is attaching quality requirements to user stories. In this manner quality requirements are handled as acceptance criteria for existing user stories. A logical result is that the quality acceptance criteria becomes part of the definition of done (DoD) for the project. The last strategy uses an explicit list (i.e. a separate artifact) for capturing quality requirements. In addition to these strategies, Hüttermann (2012) proposed a combined approach of which the acceptance criteria are derived from the explicit list. The list contains high-level quality requirements for the system and is addressed when formulating acceptance criteria for individual user stories. The quality requirements list is filled with entries by operations just as intended by the pattern, however the list is maintained by the product owner so other stakeholders are also invited to provide their input.

Name	**Integrate Production Stories**
Goal in context	Eliminating the discrepancies between development and operations.
Scope	*Plan project* phase
Primary and secondary stakeholders	Product owner
Proposed incremental path for production stories	1. The name of the activity *Define functionalities for releases* has changed to *Define stories for releases.* - *Driver:* Quality stories are also defined. - *Stakeholders:* Product owner. 2. The introduction of the concepts PRODUCTION STORY and STORY. USER STORY has changed to a subtype of STORY. A STORY can either be a USER STORY or PRODUCTION STORY. - *Driver:* To make a distinction between user functionalities and quality (production) requirements. - *Stakeholders:* Product owner. 3. The insertion of an association relationship between PRODUCTION STORY and BUILD. - *Driver:* The BUILD is checked whether it meets the PRODUCTION STORY. - *Stakeholders:* developers, testers, technical application manager.
Proposed incremental path for acceptance criteria	1. The introduction of the concept QUALITY CRITERIA. USER STORY has changed to an open complex activity. - *Driver:* To attach quality requirements to user functionalities. - *Stakeholders:* Product owner. 2. The insertion of an association relationship between QUALITY CRITERIA and BUILD. - *Driver:* The BUILD is checked whether it meets the QUALITY CRITERIA. - *Stakeholders:* developers, testers, technical application manager.

Proposed incremental path for the explicit list	1. The introduction of the concept QUALITY REQUIREMENTS LIST. The list is created simultaneously with the product backlog. - *Driver:* To make a distinct requirements list for quality (production) requirements. - *Stakeholders:* Product owner. 2. The name of the activity *Define functionalities for releases* has changed to *Define contents for releases.* - *Driver:* Quality requirements need to be defined. - *Stakeholders:* Product owner. 3. The introduction of the concept QUALITY REQUIREMENT. - *Driver:* To make a distinct requirements list for quality (production) requirements. - *Stakeholders:* Product owner. 4. The introduction of an association relationship between QUALITY REQUIREMENT and USER STORY. - *Driver:* The USER STORY is checked whether it meets the QUALITY REQUIREMENT. - *Stakeholders:* developers, testers, technical application manager.
Proposed incremental path for the hybrid approach	1. The introduction of the concept QUALITY REQUIREMENTS LIST. The list is created simultaneously with the product backlog. - *Driver:* To make a distinct requirements list for quality (production) requirements. - *Stakeholders:* Product owner. 2. The introduction of the concept QUALITY REQUIREMENT. - *Driver:* To make a distinct requirements list for quality (production) requirements. - *Stakeholders:* Product owner. 3. The introduction of the concepts TECHINAL STORY and STORY. USER STORY has changed to a subtype of STORY. A STORY can either be a USER STORY or PRODUCTION STORY. - *Driver:* To make a distinction between user functionalities and quality (production) requirements. - *Stakeholders:* Product owner. 4. The introduction of the concept QUALITY CRITERIA. USER STORY aggregates the QUALITY CRITERIA, therefore USER STORY changed to an open complex activity. - *Driver:* To attach quality criteria to user functionalities. - *Stakeholders:* Product owner. 5. The name of the activity *Define functionalities for releases* has changed to *Define contents for releases.* - *Driver:* Quality requirements need to be defined. - *Stakeholders:* Product owner.
Unordered increments	- Technical application manager should be made responsible for providing and maintaining the quality requirements list as both the operational constrains. - The product owner should be made responsible for eliciting the production stories and quality criteria based on the quality requirements list. - The team should be informed on how quality requirements should be addressed during the project. Trainings are essential to ensure these requirements are efficiently processed by the team.
Reference to PDD	Figure 35 (production stories), Figure 36 (acceptance criteria), Figure 37 (explicit list), Figure 38 (hybrid approach)

Table 5. Method increment description: develop for production

At CaseComp a project management tool is used for the project administration. The tool records all project artifacts, such as the product backlog, sprint plan, and stories on a central location, which is accessible by all team members. It also maintains the relationship of work items to its parent products in order to foster traceability. The choice of which method variant is selected for implementation depends on the support and customizability of this tool. Since the research is limited to process improvements only, the changes should be supported by the current tool.

The production stories approach (Figure 35) seems the most easiest solution. Production stories can be inserted right into the existing product backlog so they can be planned just like traditional user stories. The distinction between user stories and production stories can be made visible within the tool by selecting a type. A functional requirement is recorded as a user story, whereas a production story is labeled as a generic task. This approach seems very logical to CaseComp, however quality requirements may apply to multiple user stories so it could take a long time to finish a single production story. This issue complicates the sprint planning as the estimation for stories becomes less accurate.

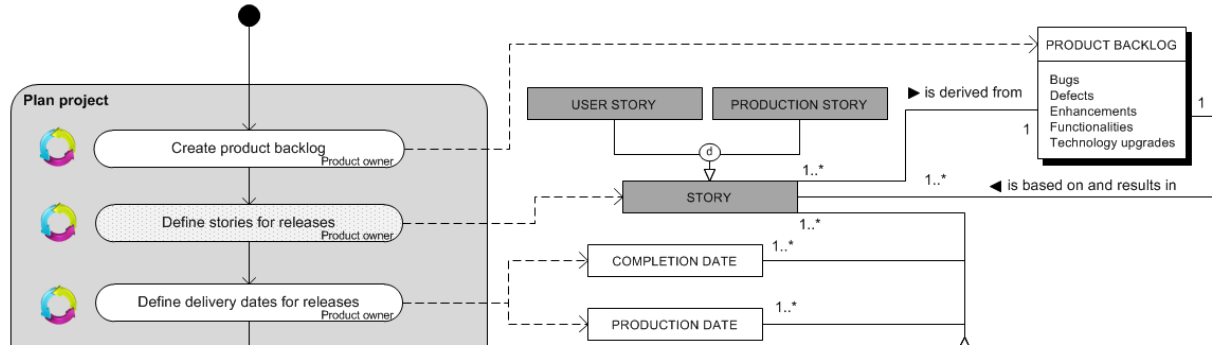

Figure 35. Method increment: develop for production (production stories)

The second approach (Figure 36), assigning quality criteria to individual user stories, is a practical solution when there are only a few quality requirements. The project team receives many cross-cutting quality requirements from external parties which probably result in the same quality acceptance criteria for multiple user stories.

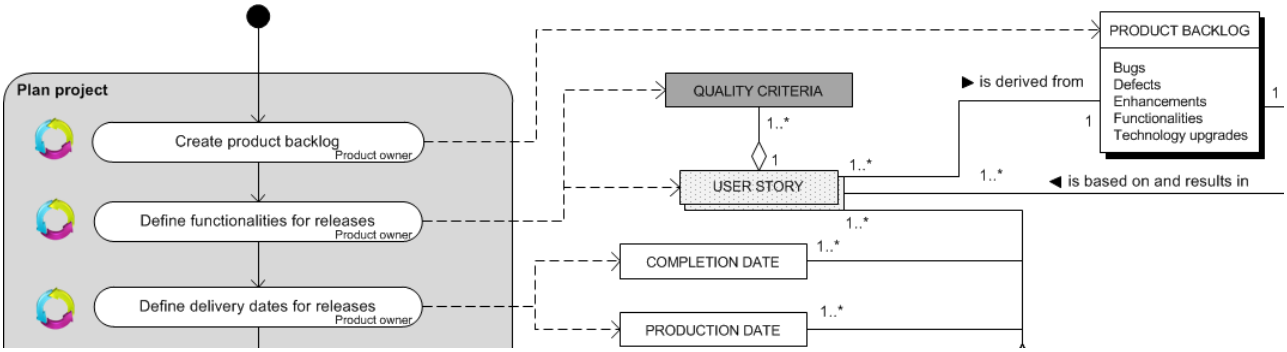

Figure 36. Method increment: develop for production (acceptance criteria)

The third solution (Figure 37), an explicit list for quality requirements is not of relevance as there is already a quality requirements list available for the system under development. This list is maintained and provided by external stakeholders. As the list is already available but it is not part of

the process (i.e. the list is not actively addressed during the development because no work items are linked to it), there is practical no difference when implementing this increment variant.

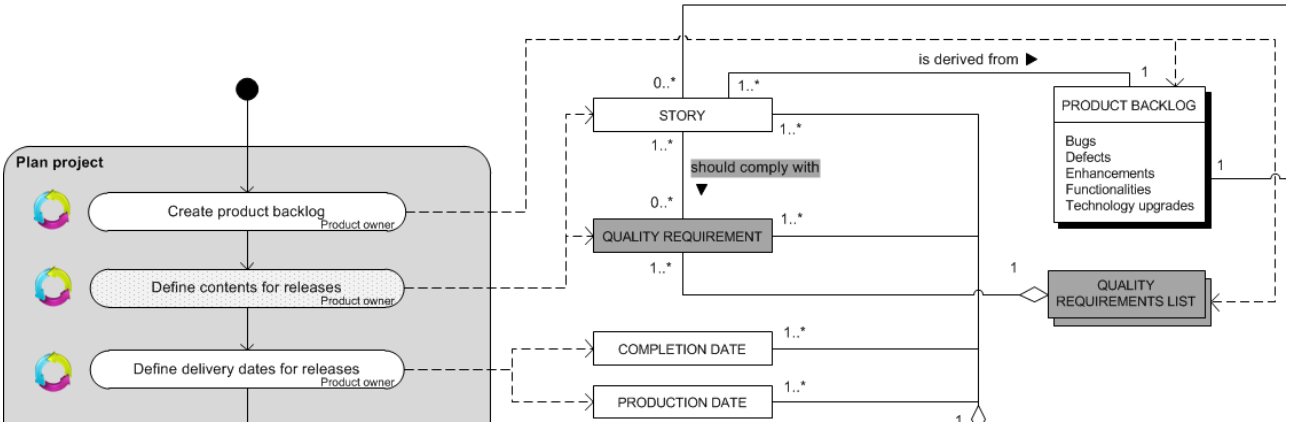

Figure 37. Method increment: develop for production (explicit list)

The final solution uses a hybrid or mixed approach (Figure 38). This approach combines the aforementioned approaches into a single solution. The quality requirements can either result in quality criteria for user stories or developable production stories. This solution is ideally suited to support both low and high-level quality requirements. As the original quality requirements list is unaltered, the responsibilities stay the same (i.e. the creation is still done by external stakeholders). The quality requirements are written in a different context (e.g. business or operations) at different levels, therefore the requirements need to be adjusted for the team. The hybrid variant is chosen as this approach supports the characteristics of quality requirements at CaseComp.

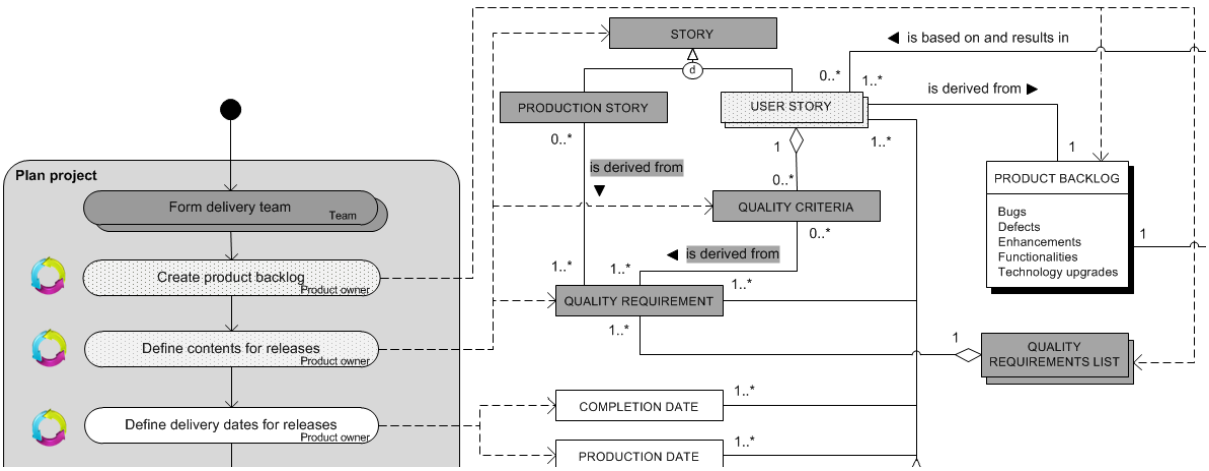

Figure 38. Method increment: develop for production (hybrid approach)

Sync Meeting

The purpose of this method increment is to have an evaluation meeting in which the team, especially development and operations, closely discuss the changes that need to be rolled out by the current release. The team also operational issues from the last deployment are discussed to learn from early experiences. Communication and alignment extend working relationships and, thus, foster collaboration.

Name	Sync meeting
Goal in context	Eliminate risks in the transition to production and learn from early experiences.
Scope	*Finalize release* phase
Primary and secondary stakeholders	Developers, testers, technical application manager, functional application manager
Proposed incremental path for production stories	1. The introduction of the activity *Evaluate release contents.* - *Driver:* To foster communication and increase the quality of the system. - *Stakeholders:* Developers, testers, functional application manager, technical application manager, product owner.
Unordered increments	- Operational issues from the last deployment should be collected by the functional application manager. - Developers provide a summary of the important changes in the current release. - Team members should have advanced skills in communication.
Reference to PDD	Figure 39

Table 6. Method increment description: sync meeting

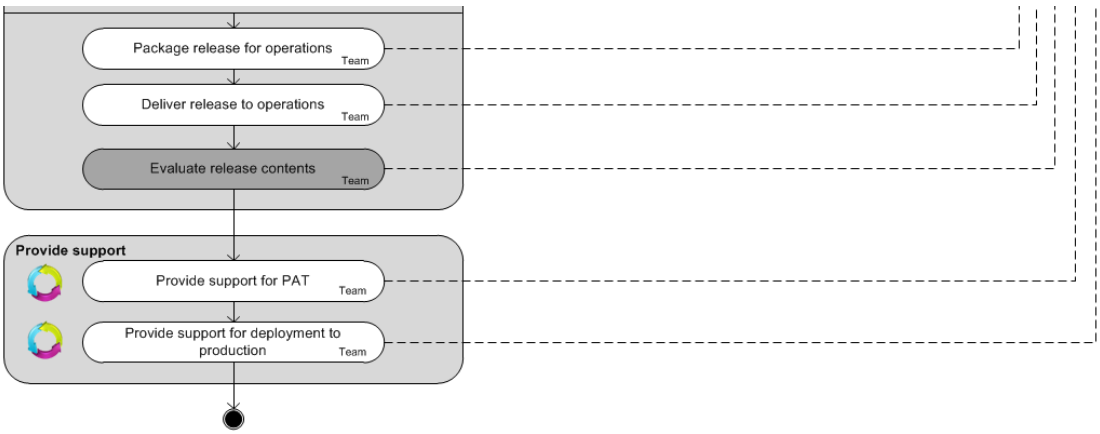

Figure 39. Method increment: sync meeting

5.2 Situational Method

In the previous sections the method fragments are elaborated for the situational context at CaseComp. At this moment we are able to assemble these fragments into the baseline from Chapter 4. The result is a situational method that describes the desired situation for the selected case. The process-deliverable diagram (PDD) of the situational method is depicted in Figure 41. The (open) complex activities are further elaborated in Figure 29 and Figure 32. The updated activities and concepts for the situational method are provided in Appendix VII. Descriptions from Appendix IV still apply to the unchanged parts of the PDD. Scrum meetings were initially omitted from the reference method (Blijleven, 2012), so we provide a model (Figure 41) to show the sequence of meetings during the entire project. The model includes the introduced meetings *team workshop* and *sync meeting* as well.

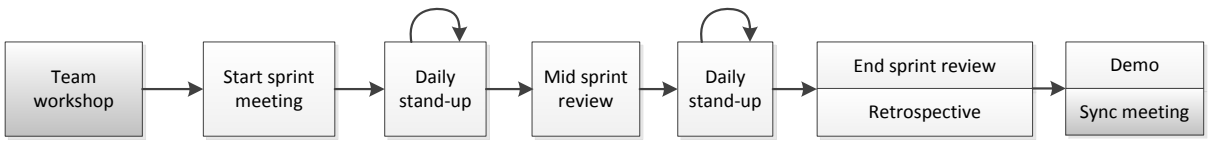

Figure 40. Meetings during the project

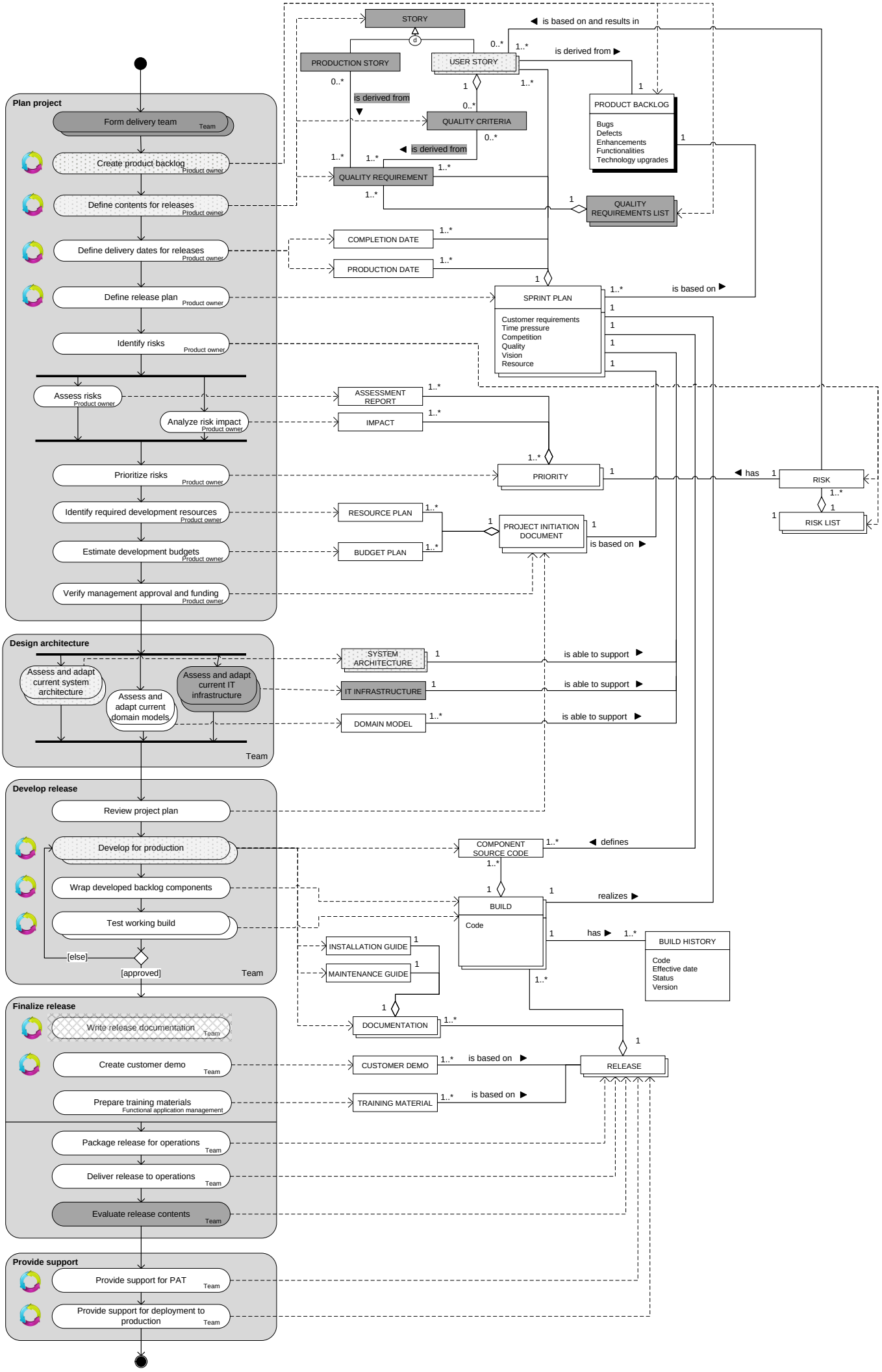

Figure 41. Situational method for the desired situation

6. Integration Scenario

This chapter elaborates on the optimal integration scenario, the path in which method increments are incrementally added to the baseline method. The small improvement packages focus on a limited number of issues, by taking small evolutionary steps. Implementing process changes in such evolutionary way, reduces both risks and implementation costs. The chapter has the aim to provide an answer on the fourth research subquestion:

> *SQ4. How can the selection of method increments in alternative integration scenarios be supported?*

We define an integration scenario as the process alternative to incorporate method increments into a baseline method. An integration scenario is considered fully executed when the incremental steps as described by the method increment descriptions from 5.1.3 are performed in the prescribed sequence. The scenario execution process is supported by a set of activities that are needed to shape the target situation (e.g. educating users, obtaining data).

6.1 Implementation Requirements

Before we elaborate on the integration scenario we need to make sure that implementation requirements are properly addressed. The requirements and fragment's characteristics aid us in shaping the optimal scenario for CaseComp. Ultimately, this integration scenario is executed and validated in a pilot experiment.

6.1.1 **Organizational Requirements**

In Chapter 4 we identified three implementation requirements that need to be addressed. The requirements for DevOps were captured as follows:

> *R1. The knowledge should be up to date.*
> *R2. Tools need to be aligned with the process.*
> *R3. For each process change the necessary time and resources have to be estimated.*

The quality of software is shaped by the controllable factors product, people, and technology (Paulish & Carleton, 1994). It becomes obvious that the first two implementation requirements are the logical result of people and technology factors. To address the first requirement, we identified knowledge themes and skills which team members must master for the updated process areas. The training and education needs for the improvements are discussed in the method increment descriptions in Chapter 5. The second requirement that states tools need to be aligned with the process, is already covered because method fragments are elaborated using input from the project team. This has ensured that method fragments are supported by the current tools and requires no further adaptations. Lastly, we address the third requirement by providing an estimation about the required time and resources for implementing the proposed improvements. This estimation is done by considering how much time is needed for each stakeholder to perform the activities prescribed by the patterns. Note that the SPI facilitator also needs time for preparation activities (e.g. education,

data collection, planning), the actual implementation, and coaching activities. The *cross-functional delivery team* requires a workshop session that takes at least a half day (Hüttermann, 2012). Therefore we assume 5 hours for each stakeholder. The review activities for *early feedback by operations* requires 3 hours to review the documentation by FAB and TAB. The feedback may result in adaptations of the system, of which 4 hours are planned for developers. The *integrate production stories* requires in total 4 hours by IT operations (i.e. FAB and TAB). The product owner has the responsibility to prioritize their input accordingly to ensure the ratio between functionalities and production stories is maintained. The pattern *develop for production* requires a minimum effort of 1 hour by developers and testers to update and validate the release artifacts in an early stage. The *Sync meeting* requires at least 3 hours for a meeting (of which 1 hour for preparations) in which development and operations discuss the lessons learned, release contents and expected issues for the current release.

The outcome is provided in Table 7 which shows the total estimated hours per stakeholder. The time is calculated by summing the required time for each pattern.

Time/resources	Developer (4x)	Tester (3x)	Business analyst	Product owner	FAB	TAB	CM	SPI facilitator	Project manager	Estimated hours
D1. Cross-functional delivery team	20	15	5	5	5	5	0	8	5	68
D2. Early feedback by operations	4	0	0	0	3	3	0	2	0	12
D3. Integrate production stories	4	3	0	3	2	2	0	3	0	17
D4. Develop for production	4	3	0	0	0	0	0	2	0	9
D5. Sync meeting	6	6	0	0	2	2	0	3	0	19
Total	38	27	5	8	12	12	0	18	5	125

FAB = Functional application manager
TAB = Technical application manager
CM = Change management

Table 7. Necessary time and resources for the SPI effort

6.1.2 DevOps Requirements

Beside organizational requirements for implementing DevOps, there are multiple requirements inherent to DevOps. According to Paulish and Carleton (1994), a well-established culture is seen as one of the success factors for adopting a software process improvement method. Therefore, the focus is first on the cultural aspects before we move on to any other efforts. During the DevOpsDays in Rome, Edwards (2012) stated that the best way to begin is to infuse cultural aspects (i.e. values, norms, language, systems, symbols, beliefs, and habits) with the DevOps vision. A DevOps vision is shaped in the following four steps:

1. *See the system.* End-to-end view of the system, from inception to technological implementation.
2. *Focus on flow.* Examine how to improve the flow in which business ideas are transformed into working products.
3. *Recognize feedback loops.* Learn from feedback on how to improve the system.
4. *Look for continuous improvement opportunities.* Monitor and enhance the process by the use of feedback loops.

According to Walls (2013) a DevOps culture is characterized by open communication, clear alignment of incentives and responsibilities, respect, and mutual trust. Edwards (2012) proposes a set of practices that successful DevOps organizations have adopted to create a DevOps culture. An overview of examples that support the organization in establishing a DevOps culture is provided by Pais (2012b). An important practice is to banish the word 'done' from the project, so the team is responsible for the system during the entire application lifecycle. Also, bottlenecks can be removed by realigning ownership and control:

- Development owns uptime for their code.
- Operations owns uptime for platform and tooling.
- Quality assurance owns standards, tooling, and enforcement.
- Everyone owns test writing and test coverage.

The culture requirement is addressed by applying the cross-functional delivery team pattern. The use of this pattern requires team members to participate in a workshop session to elaborate on the shared goals and vision for the team. This method fragment should be implemented as first as it is the foundation for any other improvement regarding to DevOps.

Another implicit requirement for adopting DevOps (as well for most other SPI initiatives) is commitment from management. Management should be adequately informed and engaged in setting up a DevOps pilot. As stated by McFeeley (1996), "without strong, informed, and steadfast commitment and sponsorship from senior management, the effort is doomed from start". Before the pilot experiment takes place, management is required to provide a formal approval for allocating the necessary resources and SPI budgets.

6.1.3 Optimal integration scenario

The kind of relationships between the method fragments such as precedence, deliverance, and requirements should be taken into account when composing an optimal integration scenario. We used the dynamic diagram box notation by Mullery (1997). This diagrammatic notation is part of the CORE method, a method that specifies requirements adequately. The notation uses a composite of widely used notations for expressing requirements or design, and is ideally suited to show how an integration scenario can be composed. The relationships that are inherent to the method fragment are built into the diagram, illustrating the integration process from left to right (Figure 42).

The *cross-functional delivery team* fragment is put in front as it is required by all method fragments. Without this, all implementation attempts related to DevOps will be fruitless. For the completeness we include all variations of the *integrate production stories* fragment. Variants use a circle in the upper right corner, which means that one of the alternatives needs to be chosen. This method fragment is put as second as the quality requirements need to be early addressed in the development process. Realizing quality requirements after the system is developed is very cost-inefficient. The third column provides an alternative, as either one of the two may be implemented as first, followed by the other. However, they also can be implemented in parallel. Finally, the *sync meeting* fragment should be implemented as last, since this meeting cannot be held as the development has not yet finished.

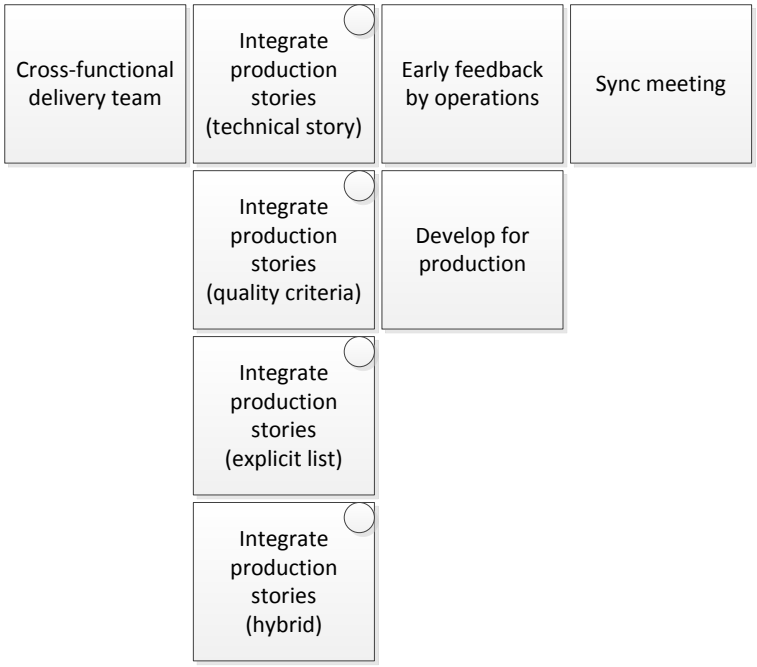

Figure 42. Composition of the integration scenario

We already determined to use the hybrid approach of the *integrate production stories* fragment in section 5.1.3, so we only have to determine the sequence of the two fragments in the third column in order to finish the composition. We argue to implement *early feedback by operations* before *develop for production,* because the assessment of the system is likely to be done before the backlog components are constructed. Based on this reasoning, the preferred implementation order of the method fragments is as follows:

1. Cross-functional delivery team (P1)
2. Integrate production stories (hybrid) (P4)
3. Early feedback by operations (P3)
4. Develop for production (P2)
5. Sync meeting (P5)

7. Pilot Experiment

In this chapter we discuss the results of the pilot experiment which was performed at CaseComp. During an 8-week pilot project we introduced the set of proposed improvements. Based on feedback we are able to make conclusions about the effectiveness of applying these method fragments in a Scrum project. First, we discuss the outline of the conducted case study approach. Second, we elaborate on the integration execution process. Third, we discuss the observations that were made during the implementation of the method fragments. Fourth, we reflect on the integration using PIW-sessions, and discuss any additional improvements to shape the situational method. Fifth, the measurements are presented and evaluated. These steps contribute to the answer on the fifth research subquestion:

> *SQ5. How can the optimal integration scenario be executed in a real development project?*

7.1 Scenario Execution Process

Earlier studies investigated how large improvements can be separated into small increments. For example, Van de Weerd et al. (2007) discusses how a method increment can be visualised. Van Stijn et al. (2012) provided a template for describing multiple method increments within one improvement effort. However, few methods exist that introduce these small changes into a living process in a real project. In this section we elaborate on the improvement planning procedures.

7.1.1 **Iterative Improvement Process**

Salo and Abrahamsson (2007) propose an iterative improvement process (IIP) for conducting SPI within agile software development projects. “The short development cycles of agile software development provide continuous and rapid loops to iteratively learn, to enhance the process and to pilot the improvements” (Salo & Abrahamsson, 2007). The iterative improvement process is founded on the principles of agile and is, therefore ideally suited for introducing improvements in the process and getting quick feedback on it. We apply this approach to support the implementation of method fragments in the preferred order.

After every sprint, a post-iteration workshop (PIW) is held in which the team exchanges experiences and results from the previous sprint (Figure 43). Each PIW session lasts about 2.5 hours, which requires additional 82.5 man-hours for the entire SPI effort. The PIW is carried out in the following steps:

1. *Preparation*. The appropriate metrics and techniques are selected for the PIWs.
2. E*xperience collection.* Problems and obstacles are identified by the project team.
3. *Planning of improvement actions*. Improvement actions are planned based on the negative experiences.
4. *Piloting*. The improvements are implemented according to the defined plans and measurement data is collected.
5. *Follow-up and validation*. Experiences of the team and analysed metrics provide feedback on the implemented improvements.
6. *Storing*. Agreements with the team are stored for the next PIW.

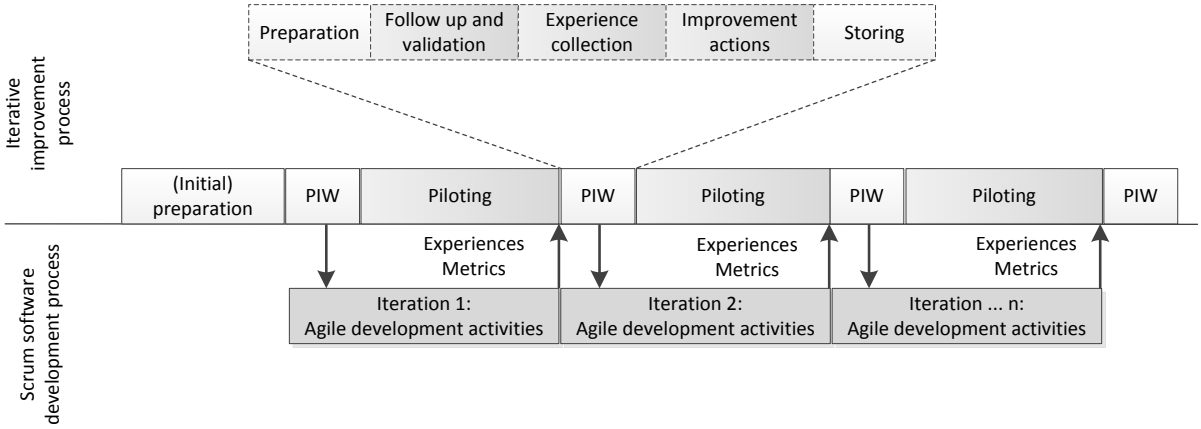

Figure 43. Iterative improvement process (redrawn from Salo and Abrahamsson, 2007)

7.1.2 Improvement Planning

The scenario execution process must adhere to the length of the pilot experiment, therefore we first determine the duration for the entire scenario execution process. The scenario should be executed within the allocated time for the software process improvement (SPI) effort. At CaseComp there are four, biweekly sprints reserved for the SPI effort, starting at the beginning of a new project. The integration scenario should be aligned to this project schedule. Based on the preferred order of the method fragments from section 6.1 we can map the improvements on the available sprints. Note that a single sprint may serve multiple improvements during a pilot.

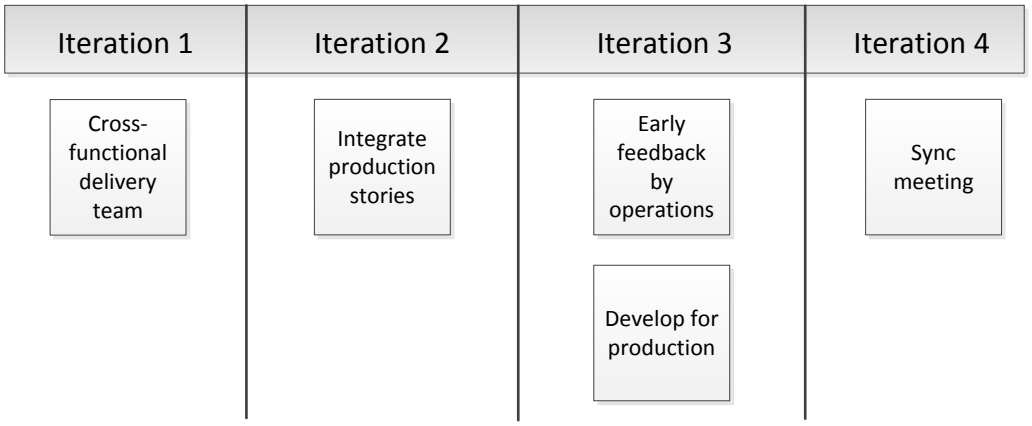

Figure 44. Improvements projected to the available time slots of the pilot experiment

7.2 Case Study Approach

An important guideline in design science is that the utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation (Hevner et al., 2004). The evaluation of the experiment was twofold. First, the method is empirically validated by the team. Second, measurements are compared with previous project results. We applied a pilot case study to validate the method fragments and integration scenario in a real project. “Case studies help industry evaluate the benefits of methods and tools and provide a cost-effective way to ensure that process changes provide the desired results” (Kitchenham, Pickard, & Pfleeger, 1995). During the pilot project, we introduced the improvements step by step and obtained feedback from the team. According to

McFeeley (1996) “the solutions will require some tailoring and refinement to fit them into projects across the organization, and the pilots will help determine the tailoring needs and guidelines for the rest of the organization”. The case study consisted of a single-case design with multiple units of analysis, which is also referred to as an embedded design (Yin, 2009). The aim is to demonstrate the usefulness of the method fragments and its corresponding process patterns, whether they affect the key problem areas from Chapter 4. The findings enable us to validate and improve the process pattern mapping table (Figure 26) in section 5.1.2. We followed the guidelines for case study experiments by Kitchenham et al. (1995), which support the execution of the pilot project in the following 7 steps:

1. Define the hypothesis
2. Select the pilot projects
3. Identify the method of comparison
4. Minimize the effort of confounding factors
5. Plan the case study
6. Monitor the case study against the plan
7. Analyze and report the results

To check whether the problem areas are tackled by the proposed method fragments, we provide a hypothesis for each key problem area (Table 8) that can be tested once the pilot project has ended. The hypotheses are justified in the case study protocol (Appendix IIX).

Code / category	Hypothesis
D1. The processes of the development and operations departments are not aligned with each other.	H_0. The standard deviation of the project velocity is equal. H_1. The standard deviation of the project velocity is decreased.
D2. Lack of standardization for quality guidelines.	H_0. The number of production acceptance testing (PAT) issues per release is equal. H_1. The number of production acceptance testing (PAT) issues per release is decreased.
D3. IT Operations is not well represented in the project.	H_0. The ratio of finished backlog items addressing quality requirements compared to the ratio of finished user stories is equal. H_1. The ratio of finished backlog items addressing quality requirements compared to the ratio of finished user stories is increased.
D4. Too comprehensive process for releasing information systems.	H_0. The time between the last PAT approval and the time of release is equal. H_1. The time between the last PAT approval and the time of release is increased.
D5. Moderate communication between development and operations.	H_0. The ratio of the number of quality defects per quality requirement is equal. H_1. The ratio of the number of quality defects per quality requirement is decreased.

Table 8. Hypotheses mapped to the main drivers

For this pilot case study we selected the same development team for whom we have elaborated the current situation in Chapter 4. The pilot results are compared with the results of the baseline, which

is the average of recent projects by the same team. For the baseline we included the last two projects which each had 8 sprints, with a total duration of 8 months. The pilot project is assisted by the iterative improvement process (IIP), a software process improvement method with an "integrated collection of procedures, tools, and training for increasing product quality, improving development-team productivity, or reducing development time" (Paulish & Carleton, 1994). For details regarding the case study planning and procedures consult Appendix IIX.

7.3 DevOps Integration

The pilot experiment is performed in parallel with a real information system development (ISD) project. The development project was an emergency branch for an application module that is already deployed to production. The process differs from a regular project as the team has never been discharged. This probably affects the implementation of some method fragments. The SPI facilitator was represented by the Scrum master and was assisted by the lead researcher. The Scrum master was also the supervisor of this research, so this person is familiar with the concepts of DevOps and is well informed on how to coach the team. In the following sections we discuss the observations during the implementation of the method fragments according to the method increment descriptions from 5.1.3.

7.3.1 **Cross-functional Delivery Team**

We started with the introduction of a cross-functional delivery team. Originally, the pattern suggests a kick-off meeting. Since the team has never been discharged, the SPI facilitator announced the meeting as an *update meeting* (rather than a project kick-off) in which the participants are informed on the project. The goal of the meeting remained the same as intended by the original process pattern, namely to create a shared vision for the team. The one-hour workshop consisted of two parts. The first part was to inform the participants on the user functionalities and release deficiencies planned for the subsequent sprints. The second part was a brainstorm session to elicit and discuss any operational needs. In total, 2 functional application managers (FAB), 2 technical application managers (TAB), 2 functional managers (FB), and 2 developers attended the workshop.

The session brought some useful discussions. For instance, a TAB correctly noticed that operations should have access to the project backlog to add any operational tasks and issues. This resulted in the creation of accounts for the project management system (PMS) by the Scrum master. One must note the method fragment realigns the responsibility of the product owner, who should manage operations as a new stakeholder. During the pilot, operations personnel was able to insert stories to the product backlog, however, any submission is approved and prioritized by the product owner to ensure the balance between production related tasks and user functionalities is maintained. Both product owner and Scrum master monitored the involvement of operations as they should not be discouraged to use the system at all. A downside is that they have to maintain a duplicate set of records for multiple management systems (i.e. project and operations). Another discussion was on guidelines for version numbering, which were not enforced by operations. These discussions would not have taken place without this meeting.

During the second part of the workshop participants placed notes on the wall to share their input to improve the software delivery process. The brainstorm session resulted in the identification 15

quality requirements, defects, and issues, which were used as input for the *integrate production stories* fragment.

Afterwards participants were asked to provide their opinion on the workshop. Generally, the participants were satisfied with the results that were obtained. However, some of the team members indicated they have experiences with numerous SPI projects. They told these initiatives create expectations among involved stakeholders, therefore they are somewhat reluctant towards the SPI pilot. The method fragment has ensured all (virtual) team members were committed to the same project. We argue this fragment plays an important role to establish a DevOps culture in near future since this meeting was the first initiative to shape the attitude and behavior of the team. New incentives may aim on knowledge sharing processes to encourage communication even more.

7.3.2 Integrate Production Stories

The *integrate production stories* fragment was implemented during the second sprint. The Scrum master decided to include only production stories that were strictly relevant for that iteration since the team has been forced to focus on the chain testing procedures. Due to this high workload the team was not able to realize all production stories. The stories that were inserted can be considered as general improvements related to IT rather than stories specific to the project. Operations personnel provided their feedback on the identified production stories. According to IT operators the improvements shall have more impact on the long term. Both FABs and TABs were positive on the collaboration and felt they were more involved in the project. They are however, not familiar with the program schedule for the Scrum teams which hinders their understanding for the project. Since operations staff operates software for multiple project teams, they notice huge differences in the alignment and collaboration with other teams. A critique on production stories is that they are often postponed when separated, whereas quality criteria can be easily integrated by definition of done in functional stories.

7.3.3 Early Feedback by Operations

The main purpose of this method fragment is to inspect the system design by human rather than machine. The software inspection process is employed by operations staff. According Fagan (2001) early inspections are “a formal, efficient, and economical method of finding errors in design and code”. An early study by Kitchenham et al. (1995) showed that inspections are a very cost-effective fault-detection method. However, we prevent the pitfall that is described by Kitchenham et al. (1995). During a pilot case study, the modules that were subjected to design inspection were not randomly allocated. In this study, only “difficult” modules were included and “easy” modules were not given design inspections. Due to this bias the researcher was unable to make valid conclusions on the effectiveness of the improvements. Based on this lessons learned we included all documentation on the system design. Note that the unordered increments of the method increment description (Table 4) require that all documentation is stored on a central location. This precondition cannot be met since there are multiple decentralized document management systems (DMS) at CaseComp. Therefore, during the pilot the required system documents are shared by e-mail to ensure IT operations is inspecting the appropriate materials.

Personnel from operations needed only a half hour to review the system documentation. It appeared that the SOPO and ISP documents were strictly relevant to the technical application manager (TAB).

The functional application manager (FAB) noticed that some document sections were frequently reused over time, thereby provide no added value for the reviewer. Ultimately, no issues on the system design were found. However, TAB found it useful to get access to such documents allowing him to provide early feedback. FAB argues that reviewing these documents is irrelevant part for its role and said the documents are relevant once major production issues occur (e.g. to find a root cause).

In near future the method fragment can be extended by applying a formal technique to support the design inspection. An example that can be used is the scenario-based analysis by Kazman, Abowd, Bass, and Clements (1996). “Scenarios are important tools for exercising an architecture in order to gain information about a system’s fitness with respect to a set of desired quality attributes” (Kazman et al., 1996). Furthermore, the company should use a single DMS that is accessible by all stakeholders rather than exchanging the documents by mail. Also, attention must be paid to improve the overall quality of the system design documents (i.e. to keep the documents up to date and remove irrelevant sections) to provide IT operations with value.

7.3.4 **Develop for Production**

This method fragment involves the creation of operational artifacts in an early stage to prevent any mistakes during the test and release phase of the project. Without good documentation the software must be operated by those who built it. The method fragment ensures the health script, database script, installation guide, and maintenance guide are updated according to the recent adaptations of the system. To make sure the activities are performed in time, the Scrum master attached tasks to existing user stories in the project management system. In this manner all necessary artifacts are updated before the status of a user story can be changed to done.

7.3.5 **Sync Meeting**

The purpose of the sync meeting is to align the activities of operations and development staff for the system deployment. This meeting is held with the expanded team (i.e. both personnel from development as operations), three days before the release is rolled out to production. In this manner both parties are focused on the finalization and deployment of the release. First, the issues and lessons learned from the last system deployment are discussed. For example, version numbers for updated system files were not always equally registered in two separate systems. To address this issue, agreements were made to stick to a version numbering convention. Second, the team evaluated the release contents by discussing the key changes of the system. Third, expected issues for the current release were shared with the team. Application time-out errors were expected, which would have to be reported to the team. Also, the team was uncertain if database update scripts are performed. Thus, procedures were provided to check if these scripts were executed correctly. Another expected issue, was that the release introduces new user roles which were not adequate tested during the production acceptance test (PAT). Participants indicated that user roles should be tested during the chain test (which is executed before the PAT). This issue was input to improve the chain test procedures. This deployment extra attention is paid to check whether problems occur as of the new user roles.

For the team it was not always clear which third parties were involved in the release process. As the sync meeting suggests a ‘pre-launch’ meeting were all representatives on the last release are

involved, it is important to figure out which other stakeholders need to participate during the sync meeting. This improvement is recorded by Scrum master to ensure that all representatives are present during the next sync meeting.

7.3.6 **Adaptations to the Situational Method**

The last step of the situational method engineering approach by Brinkkemper (1996) (as mentioned in section 5.1) is to adapt the situational method based on empirical validation. The pilot experiment provides us with new insights about the execution of the situational method in practice. We made the following observations:

- The multiplicity of QUALITY REQUIREMENT in relation with PRODUCTION STORY should be zero-to-many rather than one-to-many. In practice it appeared that PRODUCTION STORY(ies) didn't always derive from QUALITY REQUIREMENTs. Sometimes findings from production directly result in the identification of new PRODUCTION STORY(ies).
- For each activity regarding to *develop for production*, there is added a subtask to the STORY concept. All attached tasks should be performed before the status of a STORY can be set to done. Therefore, the TASK concept is inherent to the introduction of the *develop for production* fragment.

Also, during the pilot we have encountered the following issues:

- The QUALITY REQUIREMENTS LIST is maintained by IT operations and is invisible for the team, therefore cannot be easily retrieved by the product owner, who is responsible for managing stakeholder participation and requirements during the project. A new version of the list should be created and managed by the product owner, so quality requirements are elaborated in the correct context for the team.
- All design documents should be stored on a central location as IT operators have no access to the required documents. Since operations staff is granted access to the project management system (PMS), the documents should be stored into the PMS, so they can look up the documents themselves (rather than requesting them by e-mail). In the same way, IT operations should store the inspection findings on the system design on the same location ensuring all materials are available for the team.

7.4 Analysis

In this section we discuss the final step of the approach for case study experiments by Kitchenham et al. (1995), which evaluates the effects by testing the hypotheses from section 7.2. Based on the obtained data we are able to draw conclusions about the effectiveness of the proposed improvements on the key problem areas.

7.4.1 **D1. Poor alignment between projects and operations**

In a regular project the sprint velocity fluctuates between 70 and 80 story points, except for the last sprints in which IT operations gets involved to prepare the release for production. The collapse of the baseline velocity can be seen in Figure 45. The development project that took place during the pilot was a special branch with a different duration (4 sprints rather than 8 sprints). Therefore, we cannot draw conclusions by simply comparing the results from the chart. In the pilot project the release was

deployed during the fourth sprint, whereas the deployment in a regular project is typically scheduled for the eighth sprint. Hence we use the standard deviation of the project velocity to compare the results. For both the baseline as the pilot we used the corrected velocity to take into account the availability of team members. The corresponding formula is provided below and is discussed in Appendix IIX.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (V_i - \mu)^2}{n-1}}$$

$V_i = \sum$ of original estimates of all accepted work in period $_i$

The results on the metric are provided in Table 9. The pilot data shows an increase of 2,25 story points in the mean of the sprint velocity, which indicates the team was slightly more productive than in a regular project. Furthermore, the decrease of 10,22 in the standard deviation of the mean tells us the process efficiency is more stable. Based on this improvement we can reasonably conclude that hypothesis H_0 for D1 is rejected.

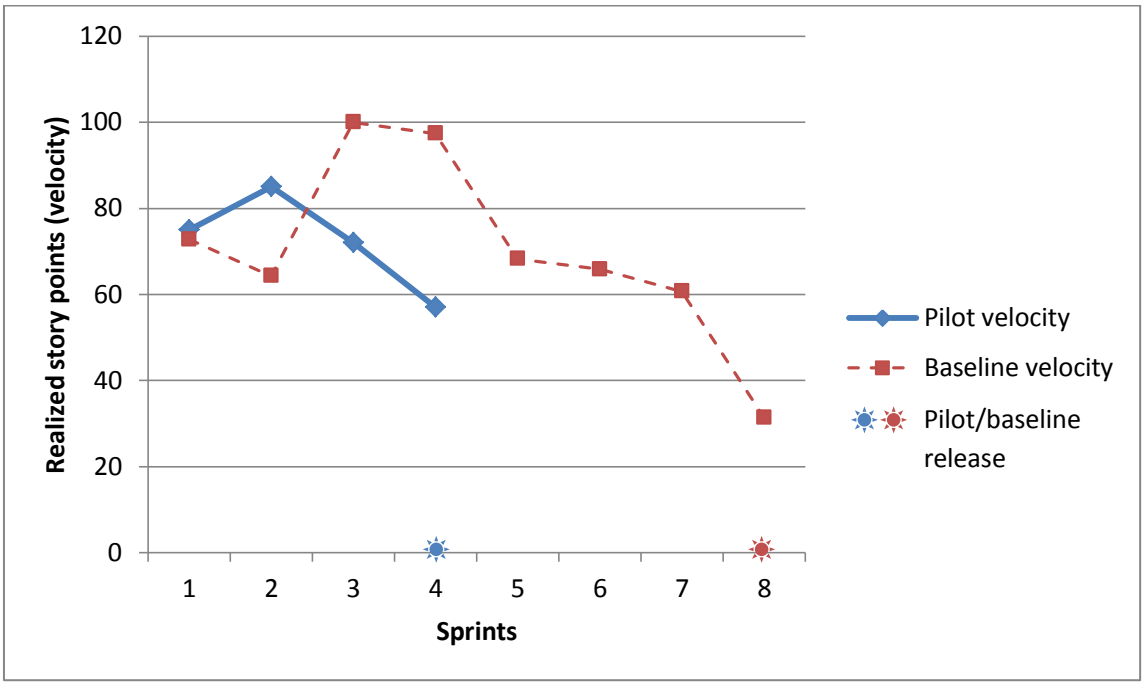

Figure 45. Quantity of project velocity

Comparison of the sprint velocity			
Baseline		**Pilot**	
Mean velocity	70	Mean velocity	72,25
Standard deviation	20,25	Standard deviation	10,03

Table 9. Comparison of sprint velocity

7.4.2 D2. Lack of standardization for quality guidelines

The pilot data tells us an improvement in the number of production acceptance testing (PAT) issues. These issues relate to the compliance with quality guidelines. The only PAT issue that occurred during the pilot was related to the product naming convention. Although this issue resulted in a redelivery of the system, it didn't affected the quality of the release process. For that reason we have omitted this PAT issue. We conclude that hypothesis H_0 for D2 is rejected as there is an improvement observed. Actually it is the optimal scenario as no relevant issues were found. Note that the issues per release ratio (IR) for the baseline is quite low, which means that there is no strong evidence for the results of the pilot.

The metric corresponds to the number of production acceptance testing (PAT) issues found (I_n) per number of releases (R_n) in order to pass through quality control. The formula is provided below and is discussed in Appendix IIX.

$$IR = \frac{I_n}{R_n}$$

Comparison of production acceptance testing (PAT) issues			
Baseline		**Pilot**	
PAT issues	6	PAT issues	0
# of releases	2	# of releases	-
IR ratio	*3*	*IR ratio*	*0*

Table 10. Comparison of PAT issues

7.4.3 D3. IT operators are not well represented

The baseline tells us that 6,04 % of the realized backlog items addresses quality requirements by IT operations, whereas the remaining part addresses user stories (Table 11). Based on this baseline data we can conclude IT operators are hardly represented as a stakeholder during a regular development project as the production issues are not planned in the backlog. The figures for the pilot project tell us an improvement of 6,66 %. The effect is relatively greater as the duration for the pilot project is only half of the duration for a regular project. Based on the pilot data we reject hypothesis H_0 for D3.

The metric corresponds to the total points of finished backlog items addressing quality requirements in ratio with the total points of finished user stories. The formula is provided below and is discussed in Appendix IIX.

$$QU = \frac{\sum QR}{\sum US}$$

Comparison of quality requirements			
Baseline		**Pilot**	
Finished quality requirements	53	Finished quality requirements	28
Finished user stories	877	Finished user stories	220,5
QU ratio	*6,04 %*	*QU ratio*	*12,69 %*

Table 11. Comparison of quality requirements

The proportions of quality requirements and user stories during the pilot are depicted in Figure 46. From the second sprint, the team identified production stories to include in the sprint plan. Notice a huge difference between the committed and finished user stories for the third sprint. This is due to a scope change of the project, however the team realized all planned production stories. From the third sprint, the team also introduced operational sub tasks for user stories to implement the *develop for production* increment. Since the workload for sub tasks cannot be estimated in the project management system, we projected the additional story points for the production stories in the graph below. In the fourth sprint the team was unable to realize the planned production stories, which can be explained by the low assigned priorities and absence of the Scrum master.

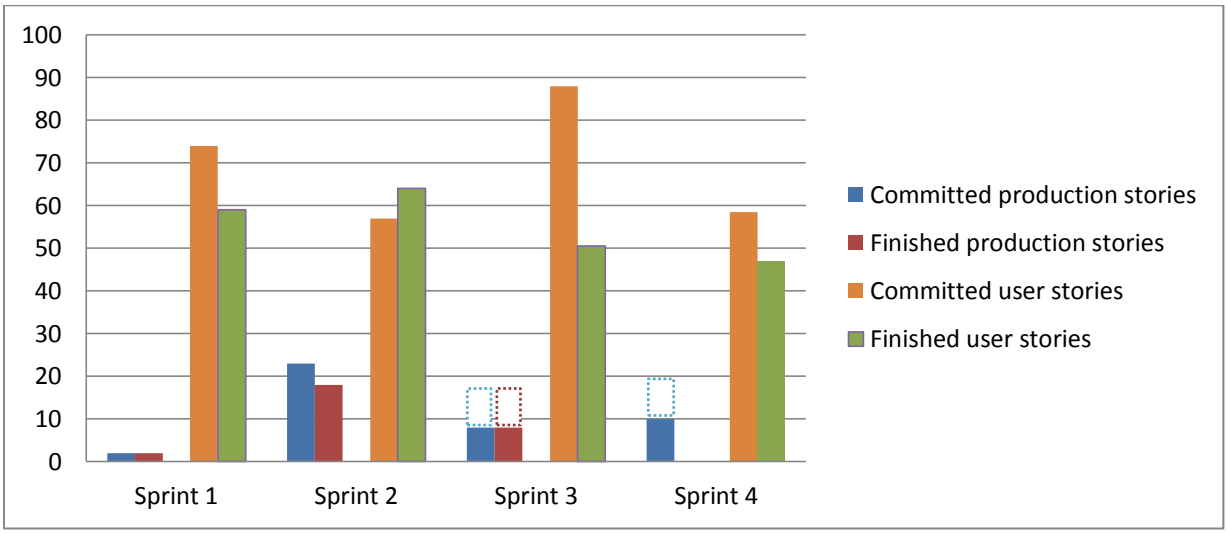

Figure 46. Quantity of realized quality requirements and user stories

7.4.4 D4. Complex release process

We stated that the required time to go through the release process is determined by slack time during office hours. The idle time indicates the project is already done and wait till the release can be deployed at the scheduled release date. Due to several approval moments that are built into the release process, the team sits idle for a quite long time till the moment the release can be deployed. How earlier the last approval is obtained, how better the release process can be quantified. This metric indicates an improvement of 55 % in the idle time of the pilot project, which means that that the release approval is obtained 11 hours earlier compared to an average project. Therefore, hypothesis H_0 for D4 is rejected. The formula of the metric is provided below and is discussed in Appendix IIX.

$$IT = 1 - (A_t - R_t)$$

Comparison of idle time (in hours)			
Baseline		**Pilot**	
Release date	-	Release date	16 June 2013, 9:00
Date last approval	-	Date last approval	11 June, 2013, 10:00
Idle time	*20 h*	*Idle time*	*31 h*

Table 12. Comparison of idle time

7.4.5 D5. Moderate communication between projects and operations

The post-release fault rates regarding to quality requirements are compared with those of the pilot project. For this experiment the number of quality defects are counted within the first five days after the system is deployed to production. The time limitation is due to the duration of this research, but does not have to a problem for this metric as production issues are usually captured within two days. The comparison of quality defects is provided in Table 13. The number of quality defects we measured within the timespan was zero. Comparing this figure with the baseline tells us this number is a huge improvement. Based on the pilot data we conclude that hypothesis H_0 for D5 is rejected.

The defects/quality requirement (DQR) ratio expresses the proportion of reported quality defects per quality requirement (QR_n). The formula of the metric is provided below and is discussed in Appendix IIX.

$$DQR = \frac{QRD_n}{QR_n}$$

Comparison of quality defects			
Baseline		**Pilot**	
Quality defects	6	Quality defects	0
Number of quality requirements	53	Number of quality requirements	38
DQR ratio	*11,32 %*	*DQR ratio*	*0 %*

Table 13. Comparison of quality defects

7.5 Findings

In this section we discuss the findings of the situational method as well the iterative improvement process. Based on both quantitative and qualitative feedback we are able to discuss the effectiveness of the situational method on the identified problem areas in a real development project. Finally, based on the analysis and experiences we are able to draw conclusions and improve the pattern mapping table.

7.5.1 Method Increment Case Descriptions

In section 5.1.3 we elaborated on the method increment descriptions for the method fragments at CaseComp. At this moment we are able to reflect on the template provided by Van Stijn et al. (2012). The template was originally intended to reflect on a process implementation, however it was used as a tool to prepare the process implementation schedule as well to maintain the method rationale. As the template was suited for this purpose, we argue to incorporate the *proposed paths* attribute in the original template to support the entire SPI effort (i.e. from planning to evaluation). Furthermore, we suggest to rename *unordered increments* to *implementation requirements*. These requirements are the preconditions of the environment that should be met before any of the proposed increments can be introduced. Also, we advocate to add acceptance criteria to each increment path to determine whether an increment step is executed correctly. The acceptance criteria should be checked by the SPI facilitator.

7.5.2 Iterative Improvement Process

The iterative improvement process (IIP) guided the SPI facilitators in the execution of the case study pilot. The process supported both the integration and evaluation of method increments. IIP enabled us to attach the proposed improvements to the available project time slots. During post iteration workshops (PIW) the SPI facilitator gathered feedback by placing notes on the wall. This activity was supported by the KJ method and semi-structured interviews. Sessions lasted for approximately 60 minutes and were held with the expanded team (i.e. including staff from operations). In the first sprint we applied a semi-structured interview technique to obtain feedback on the method. This sprint the team worked under high pressure, therefore not enough time was available to organize a workshop. Instead, each key stakeholder (i.e. TAB and FAB) is asked questions to gather experiences on the process changes in the same way as the PIW.

In all PIWs we counted the numbers of positive and negative experiences as both the number of improvement actions (Figure 47). Generally the participants had positive experiences with the implemented process changes. The average of positive experiences is above 9 for each sprint, which is quite positive for a SPI pilot with a relatively short duration. The negative experiences declined over the project, however each PIW session was focused on getting feedback on the realized improvements rather than the overall pilot project. For the second sprint (integrate production stories) there were relatively more improvement actions recorded. This mainly had to do with the fact that different version numbers were been allocated to quality requirements in the project management system. The sync meeting in sprint 4 was the only improvement that did not receive any negative feedback, as all parties were very satisfied with the obtained results and the purpose of this pattern.

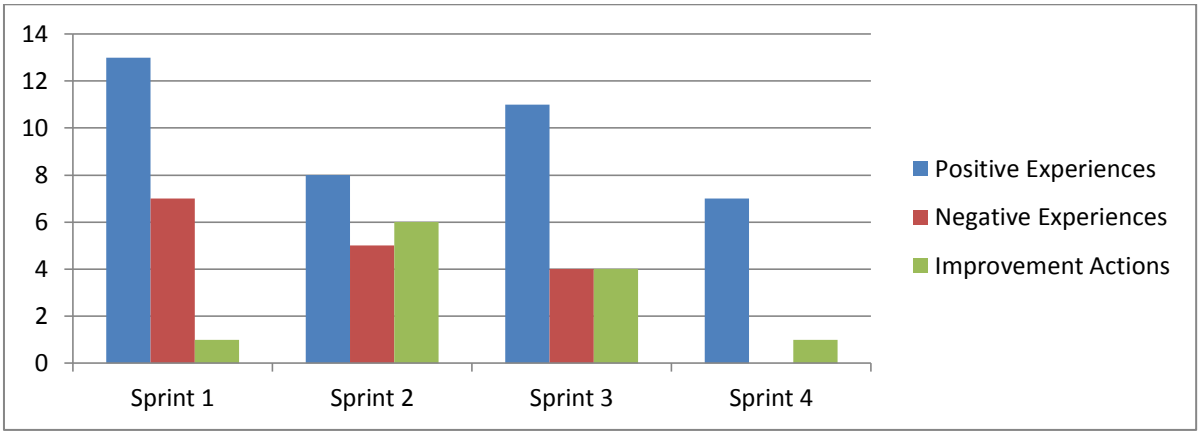

Figure 47. Quantity of positive and negative experiences, and improvement actions

The regular evaluation meetings were very effective in obtaining qualitative feedback on the method fragments. The advantage of employing IIP for a pilot project is that the team is already familiar with its procedures, and enables the team to stick to the process once the pilot has ended. In the original process, process improvements are continuously determined for the next iteration based on the negative experiences by the project team (Salo & Abrahamsson, 2007).

7.5.3 Effects on the Problem Areas

In this section we discuss the effects on the problem areas based on the qualitative and quantitative analysis. Based on the findings we improved the pattern mapping table (Figure 48). Initially, the

misalignment between operations and development processes (D1) was supposed to be tackled by the patterns of P1, P3, P4, and P5. The *cross-functional delivery team* fragment has ensured that both parties agreed on the activities for the upcoming sprints. By involving IT operations into the project, the focus was on an effective collaboration. According to developers and operations the project update (or kick-off) meeting was of added value to the project as it enhanced the alignment of the daily activities. The *develop for production* (P2) fragment has ensured the release artifacts were created in an early stage. As the materials for production were updated accordingly, fewer issues were found in transferring the release to production. Therefore, we established a link between P2 and D1. The design inspection initiated by *early feedback by operations* (P3) has ensured the system architecture and IT infrastructure are adequately reviewed by IT operations staff. Since operational needs are considered, we argue that P3 directly affects D1. The introduction of production stories into the sprint backlog (P4) has ensured IT operations needs are considered by the project. Also, IT operators are granted access so they were able to insert their quality requirements and issues into the backlog. The latter elicited discussions on the new product backlog entries. The *sync meeting* (P5) is of direct effect on the alignment, as the meeting aligns the release delivery by development and release deployment by operations. The lessons learned on the release process have contributed to a better understanding in each other and improved collaboration between operations and development personnel in the final project stage.

X	Positive effect
X+	Positive effect, missing in the initial table
X-	No effect, present in the initial table

Layers	Drivers \ Patterns	P1. Cross-functional delivery team	P2. Develop for production	P3. Early feedback by operations	P4. Integrate production stories	P5. Sync meeting
Process	D1. Departmental alignment	X	X+	X	X	X
Process, tools	D2. Lack of standardization		X	X+		
Process	D3. Ops are not well represented	X		X	X	X
Process, tools	D4. Complex release process	X-	X-			
Process, people	D5. Moderate communication	X		X	X	X

Figure 48. Improvements to the pattern mapping table

The need for standardized quality guidelines (D2) was supposed to be tackled by P2. The pattern has ensured the quality of the release process is maintained by providing up-to-date artifacts for the system release. However, P3 has also contributed to tackle this problem area. This method fragment proposed a design inspection as part of the development process, which had an impact on the quality guidelines. The other patterns (P1, P4, P5) did not affected D2, because no new quality guidelines were prescribed by these patterns. The representation of IT operations in the project (D3) is increased by the patterns P1, P3, P4, and P5. The resulting method fragments have ensure that IT operators have contributed to the project results (by means of input for production stories, feedback on the system design, and discussions and update meetings). By involving IT operations into the

project, their needs and wishes were balanced for the project. The *complex release process* (D4) is not tackled by any of the patterns. We expected that this problem area would be addressed by P1 and P2. The corresponding metrics showed an improvement, however the perceived value by the project team was that none of the patterns positively affected D4 (i.e. the error prone approval process remained untouched). Apparently, the method fragments did not prescribed alternative procedures or guidelines to support the release process. Therefore, the link with P1 and P2 is removed. One of the improvement actions by the team was to include personnel from change management (CM) in project meetings for a joined effort to address this issue. Another suggestion was to build a tool to automate the delivery e-mail messages, which were done manually. The *moderate communication* (D5) between development and operations personnel is enhanced by all patterns except for P2. Frequent communication was essential to perform the method fragments correctly. A side remark is, from the third sprint development personnel are temporary moved to the building of IT operations to assist them in the delivery process. The physical availability of staff would certainly play a role to foster communication. Therefore, the results can be different in a project when both parties are strictly separated from each other.

Except for D4 (i.e. *complex release process*), all problem areas encountered improvements to some extent (either quantitatively or qualitatively). A major challenge in the approach was that key problem areas cover a wide range of sub problems in underlying processes. The proposed improvements are pragmatic to solve a particular issue, but other issues in the same area may be underexposed. Although we have observed improvements by some metrics, we cannot conclude that these problem areas are completely resolved. However, the perceived value by project team members was of high value to demonstrate the effect of the method fragments on the experienced problem areas. We are aware that findings from this study are based on a single case study, therefore cannot be generalized to all project teams at CaseComp (Yin, 2009). However, we argue the pilot project was successful in the attempt to change the behavior of development and operations personnel at CaseComp by introducing DevOps practices. These practices can be seen as the first effort towards an organization-wide implementation of DevOps.

8. Conclusion

The research showed how process improvements for software development can be introduced using techniques provided by method engineering. The central research question “How can method engineering support the incremental implementation of DevOps?” is answered when its corresponding subquestions are answered. Below we discuss the key findings of this study.

The research was triggered by the case company that wanted to adopt DevOps practices in their Scrum development method. The rationale for these improvements is obtained using semi-structured interviews, which resulted in the identification of 8 distinct problem areas and 3 implementation requirements (SQ1). In the search for relevant practices, we found that DevOps patterns and practices are relatively immature and hard to find. DevOps patterns and practices are not stored on a single location, which complicates the search process. Based on a thorough analysis of Internet pages, books, and seminars we developed a list containing 30 DevOps patterns. The list served as a catalogue to select necessary improvements. We developed a process pattern mapping approach, of which outcome is a mapping table that ties DevOps patterns to key problem areas (SQ2). To fit the scope of the research (i.e. extending the Scrum development process) we only included patterns from DevOps area 4, which integrates operations personnel and feedback into the project. However, there are only a few process patterns available in DevOps area 4. Patterns on tools (e.g. deployment automation) and cultural aspects have received considerably more attention by literature. Each process pattern is codified as a method fragment in order to store them in the method base. First, the selected process patterns are elaborated using a standardized description template. Second, based on the description method fragments are constructed for the situational context of CaseComp. Ultimately, 5 DevOps process patterns were assembled into the Scrum process, which resulted in the creation of a situational method for CaseComp (SQ3). Based on the notation by Mullery (1997) we were able to provide a simple selection mechanism for method increment alternatives in composing an integration scenario (SQ4). We argued that the optimal scenario is shaped by both organizational requirements and requirements inherent to the method of choice (i.e. DevOps). Finally, a pilot experiment is conducted which introduced a series of improvements in a real development project. The experiment itself is assisted by the iterative improvement process by Salo and Abrahamsson (2007) which introduces process improvements in an incremental way and obtains feedback on it (SQ5). Researchers (e.g. Rossi et al., 2000) claim it is important to maintain the rationale behind any method improvement. The iterative improvement process (IIP) is considered as a useful basis for supporting evolutionary process change. Any driver for process change is documented and stored accordingly during the IIP, thus we can conclude IIP fulfills the needs of incremental method engineering. Based on both qualitative and quantitative feedback we were able to evaluate the effectiveness of the DevOps patterns on the key problem areas. Ultimately, 4 of the 5 problem areas were positively affected by the proposed increments.

This research has made an explicit contribution to the field of incremental method engineering as well the emerging field of DevOps. The first pillars are settled to provide researchers and practitioners with experiences on the implementation of DevOps practices into a development process. The procedures can be replicated for similar evolutionary SPI initiatives, where the described solutions need some tailoring for the organizations’ context and the effect of the improvement effort need to be measured.

9. Future Research

In this thesis we proposed a situational method based on the experienced problem areas at CaseComp. For them the method is the first attempt towards DevOps. However, we envision a single and shared process for both development and operations personnel to eliminate the functional silos from the project. We see several opportunities and issues that can be addressed by other researchers in this field.

First of all DevOps needs to be further researched and expanded as it does not prescribes a set of procedures and guidelines that IT organizations can use. What it especially does is defining the problem it proposes to tackle and describes the fundamental principles. Solutions are scarce at the moment and depend on project-specific implementations, therefore DevOps practices should be implemented and empirically validated on a large scale. To support this we developed an initial list with 30 DevOps patterns to help researchers to investigate the field of DevOps and its areas. Also, we see an opportunity towards a shared platform for storing and maintaining DevOps patterns. The method base should cope with the situational factors inherent to DevOps fragments, enabling the IT organization to construct a customized method for their projects. This may be supported by the use of the online method engine (OME) by Vlaanderen, Van de Weerd, and Brinkkemper (2011). OME is an online environment that provides advices based on process assessment. The goal of this approach is to align the tooling infrastructure with the method improvement by automatically configuring templates and documents (Vlaanderen et al., 2011).

A misperception by multiple practitioners (e.g. Mikita et al., 2012) is that a successful DevOps implementation depends on a solid deployment pipeline. DevOps is supported by tools, people and processes, therefore SPI initiatives should focus on an ideal mix of these. Practitioners commonly agree to introduce DevOps in the following order: culture, automation, measurement, sharing (CAMS). Tooling has garnered considerable attention in literature as DevOps practices are often mixed up or confused with continuous delivery (CD), a technological approach that focuses on deploying small pieces of working software. DevOps and CD have one goal in common (i.e. deliver valuable software to the business), however, DevOps amplifies collaboration and communication. In this study we identified multiple drivers related to the tools layer, but were outside the scope of this research. The drivers may lead to new studies on both DevOps area 2 and 3.

IT organizations should be aware that DevOps is a movement that does not takes into account the internal structure and methods of IT operations. In the Netherlands large-size companies widely use the management model by van Looijen, which makes a clear distinction between the application management, technical management, and functional management layer. Companies should address this issue properly when performing a SPI project regarding to DevOps.

In this study the dynamic diagram box notation by Mullery (1997) was useful in composing integration scenarios. However, the used set of patterns was very limited. Thus, further research should be done on how large numbers of method increments can be prioritized to determine the preferred implementation order.

Finally, we advocate a large-scale pilot experiment (i.e. multi-case) in which the method fragments can be tested in an isolated setting. The case studies of this research may be reproduced and configured to the need of the organization. Similar case studies may refine the method increments as described in this thesis.

Acknowledgements

I would like to thank my supervisor Michel Schudel, who guided me at CaseComp and provided me with valuable feedback. Also, I appreciated the support and feedback from Jan-Hein Bührman, who acted as a second supervisor during this research. Furthermore, I would like to thank my supervisors at Utrecht University. Sjaak Brinkkemper and Kevin Vlaanderen supported the research from the beginning and provided feedback on a regular basis.

I had very interesting discussions with experts from the agile competence center (ACC) at Ordina. People as Frank Verbruggen, Anko Tijman, and Jan Buurman I would like to thank for providing insights into DevOps as well giving their feedback on the research method.

I would like to thank the following people for participating in the interviews: Gijs Leussink, Ivo Woltring, Jan-Kees van Andel, Marcos Peralta, Marnix Bockhove, Peter van der Meer, Rick Zijlker, Frans Urgert, Gert Brugge, and Sven Petter.

Operatingdev.com enabled us to reuse their DevOps illustration for the cover of the thesis at no cost. Therefore credits go out to this website.

Last I would thank is guest reviewer Patrick Debois who provided us with valuable feedback and theories on the topic of DevOps.

References

Agile Manifesto. (2001). Manifesto for Agile Software Development. Retrieved from http://agilemanifesto.org/

Aguilar-Savén, R. S. (2004). Business process modelling: Review and framework. *International Journal of Production Economics*, *90*(2), 129–149. doi:10.1016/S0925-5273(03)00102-6

Alexander, C. (1977). *A pattern language: towns, buildings, construction* (2nd ed.). USA: Oxford University Press.

Ambler, S. W. (1998). *Process patterns: building large-scale systems using object technology*. Cambridge University Press.

Ambler, S. W. (2012). Strategies for Implementing Non-Functional Requirements. Retrieved March 29, 2013, from http://disciplinedagiledelivery.wordpress.com/2012/10/14/strategies-for-implementing-non-functional-requirements/

Ambler, S. W. (2013). Top 10 Practices for Effective DevOps. Retrieved March 21, 2013, from http://www.drdobbs.com/architecture-and-design/top-10-practices-for-effective-devops/240149363

Asadi, M., & Ramsin, R. (2009). Patterns of Situational Method Engineering. In *Software Engineering Research, Management and Applications 2009* (pp. 277–291). Springer.

Astin, A. W. (1993). *Assessment for excellence: The philosophy and practice of assessment and evaluation in higher education*. Phoenix: The Oryx Press.

Basili, V. R., Selby, R. W., & Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions on Software Engineering*, *7*, 733–743.

Bass, L., Jeffery, R., Wada, H., Weber, I., & Zhu, L. (2013). Eliciting Operations Requirements for Applications, 5–8.

Beedle, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). SCRUM: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design*, *4*, 637–651.

Bentley, C. (2012). *Prince2: A Practical Handbook*. Routledge.

Bergner, K., Rausch, A., Sihling, M., & Vilbig, A. (1998). A Componentware Development Methodology based on Process Patterns. In *Proceedings of the 5th Annual Conference on the Pattern Languages of Programs* (pp. 1–19). Citeseer.

Blijleven, V. (2012). Scrum Development Process from a Method Engineering Perspective. Retrieved December 11, 2012, from http://www.cs.uu.nl/wiki/bin/view/MethodEngineering/Scrumdevelopmentprocess20112012

Brereton, P., Kitchenham, B., Budgen, D., & Li, Z. (2008). Using a protocol template for case study planning. *Evaluation and Assessment in Software Engineering (EASE'08)*, 1–8.

Brinkkemper, S. (1996). Method engineering : engineering of information methods and tools. *Information and software technology*, *38*(4), 275–280.

Brinkkemper, S., Saeki, M., & Harmsen, A. F. (1998). Assembly Techniques for Method Engineering. *Advanced Information Systems Engineering*, *1413*, 381–400.

Brinkkemper, S., Saeki, M., & Harmsen, A. F. (1999). Meta-modelling based assembly techniques for situational method engineering. *Information Systems, Elsevier*, *24*(3), 209–228.

Brinkkemper, S., Van de Weerd, I., Saeki, M., & Versendaal, J. (2008). Process improvement in requirements management: A method engineering approach. In *Proceedings of Requirements Engineering: Foundation of Software Quality*.

Burns, T. J., & Deek, F. P. (2007). A Practitioner Based Method Tailoring Model for Information Systems Development. *Situational Method Engineering: Fundamentals and Experiences*, 15.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer*, *34*(11), 131–133.

Conradi, R. (1993). Concepts for Evolving Software Processes, 1–21.

Coulin, C., Zowghi, D., & Sahraoui, A. E. K. (2006). A situational method engineering approach to requirements elicitation workshops in the software development process. *Software Process: Improvement and Practice*, *11*(5), 451–464.

Debois, P. (2008). Agile Infrastructure and Operations: How Infra-gile are You? *Agile 2008 Conference*, 202–207. doi:10.1109/Agile.2008.42

Debois, P. (2012). Devops Areas - Codifying devops practices. Retrieved February 12, 2013, from http://www.jedi.be/blog/2012/05/12/codifying-devops-area-practices/

Denning, P. J. (1997). A new social contract for research. *Communications of the ACM*, *40*(2), 132–134.

Duvall, P. M. (2010). Continuous Integration: Patterns and Anti-Patterns. Aldon.

Duvall, P. M. (2012). *Agile DevOps: The flattening of the software release process*. Retrieved from http://www.ibm.com/developerworks/library/a-devops1/#N100F7

Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous Integration: improving software quality and reducing risk*. Addison-Wesley Professional.

Dyba, T. (2005). An Empirical Investigation of the Key Factors for Success in Software Process Improvement, *31*(5), 410–424.

Ebert, C., Abrahamsson, P., & Oza, N. (2012). Lean Software Development. *IEEE Software*.

Edwards, D. (2010). What is DevOps? Retrieved January 09, 2013, from http://dev2ops.org/2010/02/what-is-devops/

Edwards, D. (2012). You Can't Change Culture, But You Can Change Behavior. *DevOpsDays Rome 2012*. Retrieved April 19, 2013, from http://www.slideshare.net/dev2ops/you-cant-change-culture-but-you-can-change-behavior-and-behavior-becomes-culture

Edwards, D., & Thompson, L. (2011). Velocity 2011: Production Begins in Development. *Velocity 2011*. Retrieved March 26, 2013, from http://www.slideshare.net/dev2ops/velocity-2011-production-begins-in-development

Fagan, M. (2001). Design and code inspections to reduce errors in program development. *Pioneers and Their Contributions to Software Engineering*, 301–334.

Fitzgerald, G., & Avison, D. E. (2003). Where now for development methodologies? *Communications of the ACM*, *46*(1), 78–82.

Florac, W. A., & Carleton, A. D. (1988). *Measuring the software process: Statistical process control for software process improvement*. Addison-Wesley Professional.

Florak, W., Park, R., & Carleton, A. D. (1997). Practical software measurement: Measuring for process management and improvement.

Fowler, M. (2006). Continuous Integration. Retrieved January 04, 2013, from http://martinfowler.com/articles/continuousIntegration.html

Garlan, D. (2000). Software architecture: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering* (pp. 91–101). ACM.

Gnatz, M., Marschall, F., Popp, G., Rausch, A., & Schwerin, W. (2001). Towards a Living Software Development Process based on Process Patterns. *Software Process Technology*, 182–202.

Guizzardi, G. (2005). *Ontological foundations for structural conceptual models*. CTIT, Centre for Telematics and Information Technology.

Hamment, P. (2011). Make Large Scale Changes Incrementally with Branch By Abstraction. Retrieved March 15, 2013, from http://continuousdelivery.com/2011/05/make-large-scale-changes-incrementally-with-branch-by-abstraction/

Harmsen, A. F. (1997). *Situational Method Engineering*. Utrecht: Moret Ernst & Young.

Harmsen, A. F., Brinkkemper, S., & Oei, H. (1994). *Situational Method Engineering for Information System Project Approaches* (pp. 169–194). University of Twente, Department of Computer Science.

Harmsen, A. F., & Saeki, M. (1996). Comparison of four method engineering languages. *IFIP*.

Hass, A. M. J. (2003). *Configuration management principles and practice*. Addison-Wesley Professional.

Henderson-Sellers, B. (2006). Method engineering: Theory and practice. *Information Systems Technology and its Applications*, 13–23.

Henderson-Sellers, B., & Ralyté, J. (2010). Situational method engineering: state-of-the-art review. *Journal of Universal Computer Science*, *16*(3), 424–478. Retrieved from http://www.jucs.org/jucs_16_3/situational_method_engineering_state/jucs_16_03_0424_0478_henderson.pdf

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS quarterly*, *28*(1), 75–105.

Honor, A. (2010). Deployment management design patterns for DevOps. Retrieved March 12, 2013, from http://dev2ops.org/2010/02/deployment-management-design-patterns-for-devops/

Hossain, E., Babar, M. A., Paik, H., & Verner, J. (2009). Risk identification and mitigation processes for using Scrum in global software development: A conceptual framework. *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific*, 457–464.

Humble, J., & Farley, D. (2010). *Continuous delivery: reliable software releases through build, test, and deployment automation*. Addison-Wesley Professional.

Humble, J., Read, C., & North, D. (2006). The Deployment Production Line. *Agile 2006 (Agile'06)*, 113–118. doi:10.1109/AGILE.2006.53

Hüttermann, M. (2012). *DevOps for Developers* (pp. 1–184). Apress.

Janzen, D., & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer*, *38*(9), 43–50.

Jawalka, B. (2012). How DevOps supports the paradigm shift of the cloud. Retrieved January 09, 2013, from http://www.techrepublic.com/blog/datacenter/how-devops-supports-the-paradigm-shift-of-the-cloud/5698?tag=content;siu-container

Jeffries, R. (2001). What is extreme programming. *XP Magazine, Nov*.

Kazman, R., Abowd, G., Bass, L., & Clements, P. (1996). Scenario-based analysis of software architecture. *IEEE Software*, *13*(6), 47–55. doi:10.1109/52.542294

Kitchenham, B., Pickard, L., & Pfleeger, S. L. (1995). Case Studies for Method and Tool Evaluation. *Software, IEEE*, *12*(July), 52–62.

Kornyshova, E., Deneckere, R., & Salinesi, C. (2007). Method Chunks Selection by Multicriteria Techniques: an Extension of the Assembly-based Approach. *Situational Method Engineering*. Retrieved from http://www.springerlink.com/index/U73770553247RM46.pdf

Krane, V., Anderson, M., & Stean, W. (1997). Issues of qualitative research methods and presentation. In *Journal of Sport and Exercise Psychology* (pp. 213–218).

Lee, K. A. (2011). DevOps and Release Management. Retrieved March 01, 2013, from http://buildmeister.com/articles/devops_and_release_management

Luinenburg, L., Jansen, S., Souer, J., & Van de Weerd, I. (2008). Designing Web Content Management Systems Using the Method Association Approach. In *Proceedings of the 4th International Workshop on Model-Driven Web Engineering (MDWE 2008)* (pp. 106–120).

Macintosh, A. (1993). The need for enriched knowledge representation for enterprise modelling. In *AI (Artificial Intelligence) in Enterprise Modelling, IEE Colloquium on (Digest No. 078)*.

McFeeley, B. (1996). *IDEAL: A User's Guide for Software Process Improvement*.

Mikita, D., Dehondt, G., & Nezlek, G. S. (2012). The Deployment Pipeline. In *Proceedings of the Conference on Information Systems Applied Research ISSN* (pp. 1–10). New Orleans.

Miles, M., & Huberman, M. (1994). Qualitative data analysis: An expanded sourcebook.

Mirandolle, D., Van de Weerd, I., & Brinkkemper, S. (2011). Incremental Method Engineering for Process Improvement – A Case Study. *Engineering Methods in the Service-Oriented Context*, 4–18.

Moon, B. (2010). Developers should write code for production. Retrieved March 18, 2013, from http://brian.moonspot.net/develop-for-production

Mora, C., Menozzi, D., & Merigo, A. (2011). Exploring the Potential Competition in the Salmon Industry: a Scenario Analysis of Genetically Modified Fish Marketing. In *2011 International Congress, August 30-September 2, 2011, Zurich, Switzerland*. European Association of Agricultural Economists.

Mullery, G. P. (1997). CORE - A Method for Controlled Requirement Specification. In *Proceeding ICSE '79 Proceedings of the 4th International Conference on Software Engineering* (pp. 126–135).

Pais, M. (2012a). Is the Enterprise Ready for DevOps? *InfoQ*. Retrieved January 21, 2013, from http://www.infoq.com/articles/virtual-panel-entreprise-ready-for-devops

Pais, M. (2012b). Introducing DevOps Culture by Changing Behavior. Retrieved April 19, 2013, from http://www.infoq.com/news/2012/10/introduce-devops

Pant, R. (2009). Organizing a Digital Technology Department of Medium Size in a Media Company. Retrieved January 09, 2013, from http://www.rajiv.com/blog/2009/03/17/technology-department/#footnote_1_377

Paulish, D. J., & Carleton, A. D. (1994). Case Studies of Software Process-Improvement Measurement. *IEEE Software*, *27*(9), 50–57.

Phifer, B. (2011). Next-Generation Process Integration: CMMI and ITIL Do Devops. *Cutter IT Journal*, *24*(8), 28–33.

Puerta, A., & Eisenstein, J. (1999). Towards a general computational framework for model-based interface development systems. *Knowledge-Based Systems*, *12*(8), 433–442.

Punter, T. (1996). The MEMA-model: towards a new approach for Method Engineering. *Information and Software Technology*, *38*(4), 295–305. doi:10.1016/0950-5849(95)01087-4

PuppetLabs. (2013). 2013 State of DevOps Report. IT Revolution Press. Retrieved from https://puppetlabs.com/solutions/devops/

Ralyté, J. (1999). Reusing Scenario Based Approaches in Requirement Engineering Methods: CREWS Method Base. In *10th Int. Workshop on Database and Expert Systems Applications (DEXA'99), 1st Int. REP'99 Workshop*. Florence.

Ralyté, J. (2002). *Requirements Definition for the Situational Method Engineering* (pp. 1–22). Boston: Kluwer Academic Publishers.

Ralyté, J., Deneckère, R., & Rolland, C. (2003). Towards a Generic Model for Situational Method Engineering. In *Advanced Information Systems Engineering* (pp. 1029–1029). Springer.

Ralyté, J., & Rolland, C. (2001). An Approach for Method Reengineering. *Conceptual Modeling—ER 2001*, 471–484.

Ralyté, J., Rolland, C., & Ayed, M. B. (2005). An approach for evolution-driven method engineering. *Modeling Methods and Methodologies*, 80–101.

Rogowski, C. (2011). Introduction to DevOps: Agile Development and Operations Hand in Hand. Retrieved March 22, 2013, from http://agileelephant.blogspot.nl/2011/09/introduction-to-devops.html

Rossi, M., Tolvanen, J. P., Ramesh, B., Lyytinen, K., & Kaipala, J. (2000). Method rationale in method engineering. In *System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on* (Vol. 00, pp. 1–10). IEEE.

Rowe, M., & Marshall, P. (2011). *Spanning people, processes, and technologies: The business case for Collaborative DevOps*.

Runeson, P., & Höst, M. (2008a). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi:10.1007/s10664-008-9102-8

Runeson, P., & Höst, M. (2008b). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. doi:10.1007/s10664-008-9102-8

Saeki, M. (2003). Embedding metrics into information systems development methods: an application of method engineering technique. In *15th Conference on Advanced Information Systems Engineering* (pp. 374–389).

Salo, O., & Abrahamsson, P. (2007). An Iterative Improvement Process for Agile Software Development. *Software Process: Improvement and Practice*, *12*(1), 81–100. doi:10.1002/spip

Schwaber, K. (1995). Scrum Development Process. In *Proceedings of the Workshop on Business Object Design and Implementation at the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95)* (pp. 10–19).

ScrumAlliance. (2012). Scrum Alliance - What Is Scrum? Retrieved December 13, 2012, from http://www.scrumalliance.org/learn_about_scrum

ScrumPLoP. (2012). Scrum patterns. Retrieved February 20, 2013, from https://sites.google.com/a/scrumplop.org/published-patterns/home

Seidita, V., Cossentino, M., & Gaglio, S. (2007). Adapting passi to support a goal oriented approach: a situational method engineering experiment. In *Fifth European workshop on Multi-Agent Systems (EUMAS'07)* (pp. 1–15).

Slooten, K. van, & Brinkkemper, S. (1993). A method engineering approach to information systems development. *Information Systems Development Process. Elsevier Science Publishers (A-30)*, 167–186.

Slooten, K. van, & Hodes, B. (1996). Characterizing IS development projects. *Proceedings of the IFIP TC8, WG8*, *1*(8.2), 29–44.

Smith, D. M. (2011). *Hype Cycle for Cloud Computing, 2011* (pp. 1–74).

Smith, S. (2013). Continuous Delivery != DevOps. Retrieved March 01, 2013, from http://architects.dzone.com/articles/continuous-delivery-devops

Souer, J., Van de Weerd, I., Versendaal, J., & Brinkkemper, S. (2007). Situational requirements engineering for the development of content management system-based web applications. *International Journal of Web Engineering and Technology*, *3*(4), 420–440. Retrieved from http://inderscience.metapress.com/index/2704W84R2513L062.pdf

Sutherland, J., & Schwaber, K. (2011). The Scrum Guide. The Definitive Guide to Scrum: The Rules of the Game, (October). Retrieved from http://www.scrum.org/

Swartout, P. (2012). *Continuous Delivery and DevOps: A Quickstart guide* (p. 154). Packt Publishing.

Takeuchi, H., & Nonaka, I. (1984). The new new product development game. *Harvard business review*, *64*(1), 137–147.

Tasharofi, S., & Ramsin, R. (2007). Process patterns for agile methodologies. In *Situational Method Engineering: Fundamentals and Experiences* (Vol. 244, pp. 222–237). Springer. Retrieved from http://www.springerlink.com/index/l814257x33660874.pdf

The Standish Group. (2011). *The Chaos Manifesto*.

Tolvanen, J. P. (1998). *Incremental method engineering with modeling tools: theoretical principles and empirical evidence*. University of Jyväskylän.

Turnbull, P. D. (1991). Effective Investments in Information Infrastructures. *Information and Software Technology*, *33*(3), 191– 199.

Välimäki, A., & Kääriäinen, J. (2008). Patterns for Distributed Scrum – A Case Study. *Enterprise Interoperability III*, 85–97.

Van de Weerd, I. (2009). Advancing in software product management: An incremental method engineering approach. *SIKS Disseration Series*, (2009-34).

Van de Weerd, I., & Brinkkemper, S. (2008). Meta-Modeling for Situational Analysis and Design Methods. *Handbook of research on modern systems analysis and design technologies and applications*, *35*.

Van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., & Bijlsma, L. (2006). Towards a Reference Framework for Software Product Management. *14th IEEE International Requirements Engineering Conference (RE'06)*, 319–322. doi:10.1109/RE.2006.66

Van de Weerd, I., Brinkkemper, S., Souer, J., & Versendaal, J. (2006). A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice. *Software Process: Improvement and Practice*, *11*(5), 521–538. doi:10.1002/spip

Van de Weerd, I., Brinkkemper, S., & Versendaal, J. (2007). Concepts for incremental method evolution: empirical exploration and validation in requirements management. In *Advanced Information Systems Engineering* (pp. 469–484).

Van der Aalst, W., Ter Hofstede, A., & Weske, M. (2003). Business process management: A survey. *Business Process Management*, 1019–1019.

Van Stijn, P., Vlaanderen, K., Brinkkemper, S., & Van de Weerd, I. (2012). Documenting Evolutionary Process Improvements with Method Increment Case Descriptions. *Systems, Software and Services Process Improvement*, 193–204.

Verschuren, P. J. M., & Doorewaard, H. (2007). *Het ontwerpen van een onderzoek*. Lemma.

VersionOne. (2011). *State of Agile Survey 2011: The State of Agile Development* (pp. 1–12). Atlanta.

Vlaanderen, K., Valverde, F., & Pastor, O. (2006). Improvement of a web engineering method applying situational method engineering. *ICEIS (3-1)*, (1), 147–154.

Vlaanderen, Kevin, Van de Weerd, I., & Brinkkemper, S. (2011). The online method engine: from process assessment to method execution. *Engineering Methods in the Service-Oriented Context*, 108–122.

Vlaanderen, Kevin, Van Stijn, P., Brinkkemper, S., & Van de Weerd, I. (2012). Growing into Agility: Process Implementation Paths for Scrum. *Product-Focused Software Process Improvement*, 116–130.

Walls, M. (2013). Building a DevOps Culture.

Wells, D. (1988). Project Velocity. Retrieved May 14, 2013, from http://www.extremeprogramming.org/rules/velocity.html

West, D., & Grant, T. (2010). *Agile Development: Mainstream Adoption Has Changed Agility* (pp. 1–22).

Willis, J. (2010). What Devops Means to Me. Retrieved January 14, 2013, from http://www.opscode.com/blog/2010/07/16/what-devops-means-to-me/

Willis, J. (2012). Devops Culture (Part 1). Retrieved January 14, 2013, from http://itrevolution.com/devops-culture-part-1/

Wistrand, K., & Karlsson, F. (2004). Method Components – Rationale Revealed. In *Proceedings of the International Conference on Advanced Information Systems Engineering* (pp. 189–201).

Yap, J. (2012). Cloud driving DevOps transformation, importance. Retrieved January 09, 2013, from http://www.zdnet.com/cloud-driving-devops-transformation-importance-7000001783/

Yin, R. K. (2009). *Case Study Research Design and Methods Fourth Edition*. Sage Publications, Incorporated.

Appendix I. Definitions

#1 Method Engineering – Method Engineering is defined as "the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems" (Brinkkemper, 1996).

#2 Method – or Information Systems Development Method (ISDM). According to (Brinkkemper, 1996) a method is defined as "an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with corresponding development products".

#3 Method Fragment - Method fragments are defined as coherent pieces of IS development methods (Brinkkemper, 1996).

#4 Situational method – A method that is tuned to the project-specific needs at hand by reusing so-called method fragments (Harmsen et al., 1994). The steps of Harmsen et al. (1994) are used for the creation of a situational method.

#5 Situational Method Engineering – Situational Method Engineering is the area of Method Engineering focusing on situational methods (Harmsen et al., 1994).

#6 Method Increment – A method increment is basically any adaption in order to improve the overall performance of the method of subject (Van de Weerd et al., 2007; Van de Weerd, 2009) .

#7 Incremental Method Engineering - Incremental Method Engineering focuses on evolving a method in time towards a higher maturity level by changing small parts of the method (Mirandolle et al., 2011). It can be considered as a sub type of Situational Method Engineering.

#8 Integration Scenario – We define an integration scenario as the process alternative to incorporate method increments into a method.

#9 Process-Deliverable Diagram (PDD) – A meta-modeling technique that is based on UML activity diagrams and UML class diagrams. The activity diagrams represent the process-side that relates to the deliverable-side of the diagram, which shows the class diagrams (Souer et al., 2007).

#10 Systems Development Life Cycle (SDLC) – or software development process. An approach to build software applications that focuses on the identification of phases and stages that would improve the management of systems development and introduce discipline (Fitzgerald & Avison, 2003).

#11 Software Process Management (SPM) – Software Process Management (SPM) is the discipline aiming at controlling and managing all the resources involved in software development (Florac & Carleton, 1988).

#12 Agile Development – The group of system development methods that should be carried out under agile values and principles (Agile Manifesto, 2001) to answer the challenges of rapid development and changing requirements. The manifesto states that agile development should focus on four core values: (1) individuals and interactions over processes and tools, (2) working software over comprehensive documentation, (3) customer collaboration over contract negotiation, (4) responding to change over following a plan (Agile Manifesto, 2001).

#13 Scrum – Schwaber (1995) defines Scrum as "a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems".

#14 DevOps – For the research we define DevOps as practices that embed operations knowledge into the project and foster bidirectional feedback between the development and operations departments in order to streamline the software delivery process.

#15 Continuous Integration – According to Fowler (2006) Continuous Integration "is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day".

#16 Continuous Delivery – Continuous Delivery emphasizes on the concept of staged builds, also called deployment pipelines. The foundation of this approach is Continuous Integration, which was intended for the development cycle (Humble & Farley, 2010).

#17 Release – A release consists of a set of selected requirements. "Each requirement implies the addition of a technical or functional feature to the product" (Van de Weerd, Brinkkemper, Nieuwenhuis, Versendaal, & Bijlsma, 2006).

Appendix II. Case Study Protocol for the Current Situation

1. Introduction

The goal of this case study is to capture the current situation (or baseline) at CaseComp. Any process issues, business goals and requirements are elicited with the purpose to find appropriate DevOps patterns. The case study for the current situation phase of the research project focuses on answering the first subquestion:

> *SQ1. What are the main drivers and requirements for an organization to integrate DevOps into their Scrum development process?*

The goal of this case study is twofold. First to identify the main drivers at CaseComp for implementing DevOps. The findings result in a formal answer on the research question. Second to elaborate on the development process of the selected case. The resulting baseline is used to identify development teams which could be included in the DevOps pilot. Note the case study protocol is highly incremental in nature, so initial findings may result in adaptations of this document.

2. Design

The case study for the current situation consists of a single-case design with a single unit of analysis. This is also referred to as a holistic design according to the basic types of designs for case studies by Yin (2009). Holistic design considers the object being investigated as an interconnected whole that is also part of something larger. The object of study is the software development process of a development team at CaseComp. The process includes requirements identification, development, testing and quality control, deployment, and maintenance of the information system. The logical link between the research question and the case is to elicit process issues or main drivers (i.e. the rationale for improvements) for the software development process at CaseComp.

The research question can be divided into multiple subquestions. The resulting questions are used as main topics for which the interview questions are developed. Below we discuss the derived subquestions by providing a proper explanation for the keywords expressed in italics.

a) What are the *internal driving forces* related to DevOps?
b) What are the *external driving forces* related to DevOps?
c) What are the *requirements* for implementing DevOps?

Mora, Menozzi, and Merigo (2011) define driving forces as “key internal forces (such as knowledge and competence of management and workforce) and external forces (such as economy, competitors, technology) that shape the future of an organization”. For the case the internal forces are related to process issues, people’s knowledge and experiences and the corporate culture. External forces are the situations or events that occur outside the company and are largely beyond the control of the organization. Some examples addressed by driving forces might include the following: competition, customer behavior, industry outlook, demographics, economy, political movements, social environment, technological changes, and general environmental changes.

Regarding the third subquestion, requirements play an important role in the acceptance of the process changes proposed by DevOps. The organization may have documented preconditions or quality requirements which should be met before any improvements could be (widely) introduced. Other requirements for new (parts of) processes may require management commitment, user trainings, purchase of tools, or assessment by an external party.

To ensure that other researchers can repeat the steps of this research and the rigor of this research is guaranteed, we built traceability into the process to show how we came to the answers on the research question. The DevOps key areas by Debois (2012) are used as themes for developing relevant interview questions. We developed a matrix by placing the derived subquestions on the horizontal axis and placing the four DevOps key areas on the vertical axis. By doing so we ensure the interview questions are highly focused on the central themes and to gather data that is relevant for answering the subquestions. An example of the matrix is provided in Table 14. As the space in such table is quite limited, we elaborated the interview questions for each column below Table 14.

For establishing a baseline for the development process, expert interviews are held to craft and validate the Scrum development process. This iterative and incremental activity ensures the identified patterns will fit in the situational process at CaseComp.

The four DevOps key areas are as follows:

1. *Area 1*. Extend delivery to production.
2. *Area 2*. Extend operation to project.
3. *Area 3*. Embed project into operations.
4. *Area 4*. Embed production into project.

	A. Internal driving forces	B. External driving forces	C. Requirements
Area 1	Interview question 1 Interview question 2	..	..
Area 2	..	..	..
Area 3	..	..	..
Area 4	..	..	..

Table 14. Mapping table for interview questions (example)

As the interviewees are Dutch, interview questions are therefore elaborated in this language.

2.1 General questions

The following interview questions have the purpose to explore the background of the interviewee to create an informal atmosphere.

- Wat is uw functie?
- Wat zijn uw taken en verantwoordelijkheden?
- Met welke mensen werkt u (nauw) samen?
- Waar wordt u (of uw afdeling) op beoordeeld?
- Bent u tevreden met uw huidige werkwijze?
- Wat is uw rol in het algehele softwareproductieproces?
    - Bent u tevreden met dit proces?

    - Welke zaken gaan er goed en minder goed?
- Bent u bekend met DevOps?

2.2 Internal driving forces

The following interview questions are related to the following subquestion: '*What are the internal driving forces related to DevOps?*'. As described in the case study protocol, each of the four key areas of DevOps is addressed by the interview questions.

***Area 1*. Extend delivery to production.**

- Hoe verloopt de applicatieoverdracht van ontwikkeling naar beheer?
    - Gaat dit volgens een standaard proces?
    - Zijn er verbeterpunten voor dit proces?
- Wanneer worden beheerders in het ontwikkelproces betrokken?
    - Gebeurt dit pas zodra het systeem klaar is?
    - Heeft de huidige werkwijze nadelige consequenties voor de klant?
    - Ziet u ruimte voor verbetering?
- Hoe verloopt de samenwerking en afstemming tussen ontwikkeling en beheer?
    - Ziet u ruimte voor verbetering?
- Hoe verloopt de communicatie tussen ontwikkeling en beheer?
    - Ziet u ruimte voor verbetering?
- Hoelang duurt het voor de organisatie om een change uit te rollen welke bestaat uit 1 regel code?
    - Wordt dit gedaan op een herhaalbare en betrouwbare manier?
    - Wat zijn de vertragende factoren in het proces?
- Welke problemen ervaart u wanneer ontwikkeling en beheer samenwerken om de applicatie uit te rollen?
    - Wat merkt de organisatie als deze problemen niet worden opgelost?
    - Kunt u voor elk probleem aangeven hoe deze mogelijk kan worden verholpen?
- Welke andere veranderingen in het proces m.b.t. de afstemming tussen ontwikkeling en beheer zijn er reeds doorgevoerd?
    - Zijn deze veranderingen succesvol doorgevoerd?
    - Welke lessen heeft men hier uit geleerd?

***Area 2*. Extend operation to project.**

- Hoe wordt het ontwikkelteam geïnformeerd over belangrijke gebeurtenissen (b.v. fouten en bugs) op het productiesysteem?
    - Ziet u ruimte voor verbetering?
- Welke problemen ervaart u bij het verkrijgen van informatie van beheer?
    - Wat merkt de organisatie als deze problemen niet worden opgelost?

    - Kunt u voor elk probleem aangeven hoe deze mogelijk kan worden verholpen?
    - Zijn er verder nog verbeterpunten?
- Is het ontwikkelteam op de hoogte van de technische eisen m.b.t. de infrastructuur?
    - Hoe kan de afstemming worden verbeterd?
- Is het ontwikkelteam op de hoogte van de voorwaarden/richtlijnen waar releases aan moeten voldoen?
    - Hoe kan de afstemming worden verbeterd?
- Heeft beheer voldoende tijd beschikbaar om taken met ontwikkeling af te stemmen?
- Heeft het ontwikkelteam inzage in de activiteiten en gebeurtenissen die geregistreerd zijn door beheer?
    - Wordt hier (actief) iets mee gedaan?
- Ervaart u problemen of moeilijkheden in het krijgen van medezeggenschap over de systeemontwikkeling door beheer?
    - Wat merkt de organisatie als deze problemen niet worden opgelost?
    - Kunt u voor elk probleem aangeven hoe deze mogelijk kan worden verholpen?
    - Zijn er verder nog verbeterpunten?

***Area 3*. Embed project into operations.**

- Hoe wordt het beheer geïnformeerd over belangrijke wijzigingen voor de applicatie?
    - Ziet u ruimte voor verbetering?
- Welke problemen ervaart u bij het verkrijgen van informatie van ontwikkeling?
    - Wat merkt de organisatie als deze problemen niet worden opgelost?
    - Kunt u voor elk probleem aangeven hoe deze mogelijk kan worden verholpen?
    - Zijn er verder nog verbeterpunten?
- Is beheer op de hoogte van de technische vereisen/veranderingen m.b.t. de inrichting van het productiesysteem?
    - Hoe kan de afstemming worden verbeterd?
- Zijn er richtlijnen/eisen beschikbaar waar releases aan moeten voldoen?
- Heeft het ontwikkelteam voldoende tijd beschikbaar om taken met beheer af te stemmen?
- Heeft beheer inzage in de activiteiten en wijzigingen die geregistreerd zijn door ontwikkeling?
    - Wordt hier (actief) iets mee gedaan?
- Ervaart u problemen of moeilijkheden in het krijgen van medezeggenschap over het beheer door het ontwikkelteam?
    - Wat merkt de organisatie als deze problemen niet worden opgelost?
    - Kunt u voor elk probleem aangeven hoe deze mogelijk kan worden verholpen?
    - Zijn er verder nog verbeterpunten?

***Area 4*. Embed production into project.**

- Krijgt beheer de gelegenheid om project bijeenkomsten bij te wonen?

    - Met welke frequentie?
    - Wanneer zijn beheerders betrokken bij de ontwikkeling (b.v. aan de start van het project)?
    - Brengen zij iets bij aan de bijeenkomst?
    - Worden zij op tijd betrokken en geïnformeerd over deze bijeenkomsten?
- Vindt u het nodig om onderscheid te houden tussen incidenten (beheer) en systeem wijzigingen (ontwikkeling)?
    - Tot in welke mate moet er onderscheid blijven bestaan?
        - Beargumenteer uw mening op basis van uw ervaring in de praktijk.
    - Beheer en ontwikkeling gebruiken beide hun eigen methoden, hoe kijkt u hier tegen aan?
        - Beargumenteer uw mening op basis van uw ervaring in de praktijk.

2.3 External driving forces

The following interview questions are related to the following subquestion: '*What are the external driving forces related to DevOps?*'.

***Area 1*. Extend delivery to production.**

- Beheer en ontwikkeling zijn fysiek van elkaar zijn gescheiden (aparte locatie), ervaart u hier moeilijkheden mee?
- Hoe staat het management doorgaans tegenover verander initiatieven?
- Heeft de klant baat bij een snelle oplevering van functionaliteiten?
    - Wat heeft meer prioriteit en waarom: stabiliteit of functionaliteit?

***Area 2*. Extend operation to project.**

- Staat de technologie het toe om belangrijke informatie over het productiesysteem nauw te integreren in het project?
    - Is de infrastructuur hiervoor toereikend?
    - Zijn de gebruikte tools hiervoor geschikt?
- Is de klant tevreden met de huidige manier van werken bij fouten en verstoringen?

***Area 3*. Embed project into operations.**

- Hoe is de cultuur als het gaat om het vervullen van andermans taken of het uit handen geven van taken?
    - Denkt u dat mensen bereid zijn om hun taken uit te breiden?
    - Denkt u dat mensen bereid zijn om taken uit handen te geven?
    - Denkt u dat mensen kennis willen uitwisselen?
- Zijn er voldoende beheerders beschikbaar om ontwikkelaars te betrekken bij het uitrolproces?

***Area 4*. Embed production into project.**

- Hoe is de cultuur als het gaat om het vervullen van andermans taken of het uit handen geven van taken?
    - Denkt u dat mensen bereid zijn om hun taken uit te breiden?
    - Denkt u dat mensen bereid zijn om taken uit handen te geven?
    - Denkt u dat mensen kennis willen uitwisselen?
- Zijn er voldoende ontwikkelaars beschikbaar om beheerders te betrekken in het ontwikkelproject?

2.4 Requirements

The following interview questions are related to the following subquestion: '*What are the requirements for implementing DevOps?*'.

***Area 1*. Extend delivery to production.**

- Zijn er kwaliteitseisen waar procesveranderingen aan moeten voldoen?
- Welke randvoorwaarden of richtlijnen moeten in acht worden genomen bij de implementatie van DevOps practices?
- Zijn er technische eisen waar het proces rekening mee moet houden?
- Wat is de gewenste manier om aanpassingen door te voeren, op basis van uw kennis en ervaring?
    - Wat zijn eventuele aandachtspunten m.b.t. de invoering?

***Area 2*. Extend operation to project.**

- Welke tools dienen te worden afgestemd om feedback van beheer te kunnen verwerken?
- Waar moet de feedback van beheer aan voldoen om deze te kunnen verwerken?

***Area 3*. Embed project into operations.**

- Welke tools dienen te worden afgestemd om feedback van ontwikkeling te kunnen verwerken?
- Waar moet de feedback van ontwikkeling aan voldoen om deze te kunnen verwerken?

***Area 4*. Embed production into project.**

- Welke stappen zijn op basis van uw ervaring en kennis belangrijk voor een succesvolle DevOps implementatie?
    - Kunt u ook aangeven in welke volgorde?
    - Beargumenteer uw mening.

3. Case selection

Since all development teams at CaseComp use their own variant of the Scrum method, we merely focus on the development team's processes with the highest maturity for identifying issues. This ensures the focus is on the issues related to DevOps rather than Scrum, as issues or drivers of less mature teams may already solved or covered by the processes of this team. The elaborated process and corresponding issues are used as baseline for the desired situation phase of the research.

Suitable interview candidates from the case are selected. To form a coherent view of the current situation, all relevant roles involved in software development are subjected to an interview. One representative of each role is invited for an interview: *Product owner, Scrum master, Developer, Tester, Implementation manager, Quality manager, Program manager, Project manager, Technical Application Manager, and Functional Application Manager.*

4. Case Study Procedures and Roles

The case study is performed using a semi-structured interview technique in an informal setting. Using this approach the researcher is able to ask additional questions related to the answers of the interviewee. The first part of the interview focuses on the general activities of the user to create an informal atmosphere. The second part challenges a serious discussion about the components of DevOps, whether the problems in practice could be tackled by DevOps. In order to avoid bias due to previously identified problems, the interviewee is first asked to summarize the main problem areas to stay focused on the experiences of the interviewee. Once these are discussed in detail, the discussion can be continued on the untreated problems to cross-check previous findings.

An interview session lasts about an hour. During the interview notes are recorded and afterwards stored in the case study database. Also, the researcher has pre-announced to send any clarifying questions by e-mail when answers are insufficient or need additional validation in the case mutual responses of interviewees are contradictory or unclear to interpreted.

The case study is conducted by the lead researcher. For the rigor of the research, the supervisors of the research assess the case study protocol documents to ensure validity issues are consistently addressed.

5. Data Collection

In this case study we collect qualitative data regarding the problems and difficulties in the current development process at CaseComp. Documents and expert interviews are used as main source to elaborate and validate the Process-Deliverable Diagram.

Interview appointments are timely planned on a flexible basis - based on the presence of the participant. Since the relevant stakeholders for this research are spread over two physical locations, the interviews are held at the location of the interviewee so that the person can speak freely. A list with themes and main questions is consulted to support the semi-structured interview.

The collected data is stored in a case study database. The case study database is stored using a cloud storage service to ensure both availability and integrity.

6. Analysis

Once the data collection process has been completed, the main drivers and requirements for the DevOps implementation are summarized by subdividing these into distinct groups. This activity is performed using the data reduction method by Miles and Huberman (1994) which identifies important findings based on the interview data.

The steps of the method are summarized as follows:

1. *Data reduction*. Qualitative data is reduced and organized by discarding irrelevant data and assigning codes to relevant data.
2. *Data display*. In order to draw conclusions, good display of data is essential. Such as tables, charts, summaries and diagrams.
3. *Conclusion*. Develop conclusions based on the analysis, by comparing, contrasting, searching for patterns, triangulation etc.

The resulting codes or categories form the rationale to integrate DevOps into Scrum. The results also enable us to answer the research question for this case study. We expect the possible outcomes are related to the cooperation, coordination, communication, processes, methods, culture, and tooling of the development and operations departments. The purpose of this case study is to find the issues related to processes (the soft side) rather than technologies (the hard side). The analysis takes place as the case study research progresses.

7. Plan Validity

According to Yin (2009) there are four types of validity threats that apply to this case study: construct validity, internal validity, external validity, and reliability. With respect to construct validity, the case study protocol is developed using the template provided by Brereton, Kitchenham, Budgen, and Li (2008). The case study protocol is validated using the guidelines for case study design by Runeson and Höst (2008). The protocol ensures the interview sessions are focused on their primary goals. The internal validity is threatened by incorrect facts and incorrect results from the different sources of information. The interview sessions that are held consist of two parts, one to explore and elaborate, and one to cross-check documentation found in the document management system of CaseComp and to confirm facts stated in other interviews. With respect to external validity, a threat is that CaseComp is not representative for the Dutch IT organization. Despite CaseComp facilitates IT services, financial services are the core businesses of the company. Finally, to defend reliability, the case study procedures can be replicated for other cases in order to increase the generalizability of the results.

8. Study Limitations

An important limitation is that the research is limited to one object of study, namely the development team with a high mature development process. Since all development teams use their own implementation of the Scrum process, it could be possible that not all drivers and issues are included for analysis. Therefore, elaborated process improvements in the research address only a limited set of situational factors that apply to CaseComp. This problem can be tackled by replicating the case study procedures to other development teams, to elicit issues and drivers which were initially not identified.

9. Reporting

The case study protocol is iteratively improved once progress is being made. A template is used for case study planning and data collection procedures. The findings of the case study are reported in the ‘current situation’ chapter of the thesis. The results (e.g. process-deliverable diagram, issues) are used as foundation for the next phase of the research.

Appendix III. Alterations to the Activities and Concepts

Concept name (reference method)	Concept name (case)
RELEASE PLAN	SPRINT PLAN
RELEASE FUNCTIONALITY	USER STORY
DELIVERY DATE	COMPLETION DATE
PROJECT PLAN	PROJECT INITIATION DOCUMENT
PROJECT TEAM	*(deleted)*
ACTIVITY LIST FOR RISK MONITORING	*(deleted)*
(not present)	PRODUCTION DATE
EXECUTABLE VERSION	BUILD
EXECUTABLE VERSION HISTORY	BUILD HISTORY
(not present)	RELEASE
MARKETING MATERIAL	CUSTOMER DEMO
PRODUCT STANDARD	*(deleted)*
(not present)	MAINTENANCE GUIDE
(not present)	INSTALLATION GUIDE

Table 15. Changes to the concepts of the reference method

Activity name (reference method)	Activity name (case)
Define release plan	Define sprint plan
Form project teams	*(deleted)*
Define risk monitoring strategy	*(deleted)*
Review working executable	Test working build
Create marketing materials	Create customer demo
Assess current domain models	Assess and adapt current domain models
Assess current system architecture	Assess and adapt current system architecture
Define product standards	*(deleted)*
(not present)	Package release for operations
(not present)	Deliver release to operations
(not present)	Provide support for PAT
(not present)	Provide support for deployment to production

Table 16. Changes to the activities of the reference method

Appendix IV. Activity and Concept Tables for the Baseline

Activity	Description
Plan project	
Create product backlog	A PRODUCT BACKLOG is created by identifying features, functions, requirements, enhancements, and fixes that are not addressed by the current release.
Define functionalities for releases	The product owner selects the USER STORY(ies) that will be covered by the RELEASE PLAN for the current sprint.
Define delivery dates for releases	A COMPLETION DATE is determined. This is usually the date at which the sprint ends, after 2-6 weeks. Thereafter, the PRODUCTION DATE is defined at when the RELEASE that contains the current BUILD is put into production.
Define release plan	USER STORY(ies) are selected for inclusion in the current sprint. This results in the creation of a SPRINT PLAN. The SPRINT PLAN contains both a COMPLETION DATE as a PRODUCTION DATE.
Identify risks	The product owner identifies the risks that apply to the project. The RISKs are saved in a RISK LIST.
Assess risks	The identified RISKs are assessed by the product owner which results in the creation of an ASSESSMENT REPORT.
Analyze risk impact	The IMPACT of each RISK is determined by considering the threat level for the project.
Prioritize risks	Based on the ASSESSMENT REPORT and IMPACT of the risk a PRIORITY is determined and assigned to each risk.
Identify required development resources	A RESOURCE PLAN is elaborated by the product owner which describes the required resources needed to develop the system.
Estimate development budgets	A BUDGET PLAN is elaborated by the product owner which provides an estimation of the required budgets for the system development.
Verify management approval and funding	The product owner is responsible to request approval from management. For this a PROJECT INITIATION DOCUMENT is used that combines the SPRINT PLAN, RESOURCE PLAN(s) and BUDGET PLAN(s) into a formal document.
Design architecture	
Assess and adapt current system architecture	The team assesses the current SYSTEM ARCHITECTURE to check whether the architecture is sufficient to support the contents as described in the SPRINT PLAN. Once needed, adaptations are made to the SYSTEM ARCHITECTURE.
Assess and adapt current domain models	The team assesses the current DOMAIN MODEL(s) to check whether the architecture is sufficient to support the contents as described in the SPRINT PLAN. Once needed, adaptations are made to the DOMAIN MODEL(s).
Develop release	
Review project plan	The team reviews the PROJECT INITIATION DOCUMENT so that team members know what is expected.
Develop backlog components	Entries of the SPRINT PLAN are developed by the team which results in one or multiple COMPONENT SOURCE CODE(s).
Wrap developed backlog components	As soon as all the components are developed, the COMPONENT SOURCE CODE(s) are wrapped together by the team. This results in a BUILD, an executable version that realizes the SPRINT PLAN.
Test working build	In this activity a series of sub activities are sequentially performed to test the integrated BUILD. Any erroneous COMPONENT SOURCE CODE(s) are adjusted. Thereafter, the process can be continued as the BUILD is approved.

Finalize release	
Create release documentations	The team elaborates on the DOCUMENTATION(s) regarding to the RELEASE. The materials support a customer in understanding and using the RELEASE.
Create customer demo	A CUSTOMER DEMO is created by the team to present the features brought by the new RELEASE. The CUSTOMER DEMO is presented in a customer-intimate way.
Prepare training materials	TRAINING MATERIAL(s) are developed by the team which teaches customers on how to use the RELEASE.
Package release for operations	The BUILD(s) are wrapped together by the team to provide a coherent RELEASE. The result is a deployable version of the system.
Deliver to operations	The RELEASE is handed over to operations that will take further actions to put the RELEASE into production.
Provide support	
Provide support for product acceptation test (PAT)	The team sits standby to provide support for the product acceptation test. If there are changes needed to pass the test, the team adequately responds to fix the problems.
Provide support for deployment to production	Once problems arise in the production environment after the RELEASE is put into production, the team immediately provides support and solves eventual bugs.

Table 17. Activity table for the baseline

Concept	Description
PRODUCT BACKLOG	A PRODUCT BACKLOG contains product functionality requirements that are not adequately addressed by the current product release. Backlog items are bugs, defects, customer requested enhancements, competitive product functionality, competitive edge functionality, and technology upgrades (Schwaber, 1995).
USER STORY	A USER STORY is a product functionality requirement planned for a future release. It may concern a bug, a defect, a customer requested enhancement, a competitive product functionality, a competitive edge functionality, or a technology upgrade (Schwaber, 1995).
COMPLETION DATE	A COMPLETION DATE is the moment when the preliminary deliverable (BUILD) is finished. It is the date at which a particular sprint ends.
PRODUCTION DATE	A PRODUCTION DATE is the moment when the final deliverable (RELEASE) is deployed to the production environment (Schwaber, 1995).
SPRINT PLAN	A SPRINT PLAN describes the functionalities that are planned for the current sprint. It is based on the following variables: customer requirements, time pressure, competition, quality, vision, and resource (Schwaber, 1995).
ASSESSMENT REPORT	An ASSESSMENT REPORT describes an assessment of a RISK and appropriate risk control (Schwaber, 1995).
IMPACT	An IMPACT is the degree of negative influence a RISK can exert over the project (Hossain, Babar, Paik, & Verner, 2009).
PRIORITY	A PRIORITY is the perceived level of threat assigned to an identified RISK, based on its assigned IMPACT and ASSESSMENT REPORT (Schwaber, 1995).
RISK	A RISK is a perceived threat to the project, based on internal or external variables (Hossain et al., 2009).
RISK LIST	A RISK LIST contains the identified RISKs relevant for the development of the system (Schwaber, 1995).
RESOURCE PLAN	A RESOURCE PLAN describes the required resources (e.g. time, people, tools) for the project to realize a SPRINT PLAN (Schwaber, 1995).
BUDGET PLAN	A BUDGET PLAN describes the amount of funding required by the team in

	order to realize a SPRINT PLAN (Schwaber, 1995).
PROJECT INITIATION DOCUMENT	A PROJECT INITIATION DOCUMENT is the management product, the baseline against which progress and success will be measured (Bentley, 2012).
SYSTEM ARCHITECTURE	The SYSTEM ARCHITECTURE describes the gross structure of the system's architecture. This structure illuminates the top level design decisions, including things such as how the system is composed of interacting parts, where are the main pathways of interaction, and what are the key properties of the parts (Garlan, 2000).
DOMAIN MODEL	A DOMAIN MODEL defines the objects that a user can view, access, and manipulate through a user interface (Puerta & Eisenstein, 1999).
COMPONENT SOURCE CODE	A COMPONENT SOURCE CODE is a readable format of commands in a program before it is compiled or assembled into a BUILD, in this case of a developed component (Schwaber, 1995).
RELEASE	A RELEASE is a software version which is ready to make available to the end users. The RELEASE contains new features, bug fixes and improvements for the overall performance of the system.
BUILD	A BUILD integrates the source code of all the separately developed components, that can be executed as a computer program (Schwaber, 1995).
BUILD HISTORY	A BUILD HISTORY contains the chronological history of events related to a BUILD. The entries contain a copy of the source code of the BUILD, the date of when the record was updated, the status of the BUILD and the corresponding version number (Schwaber, 1995).
DOCUMENTATION	DOCUMENTATION describes the RELEASE both textually as visually (Schwaber, 1995).
INSTALLATION GUIDE	The INSTALLATION GUIDE describes how the RELEASE should be installed on the production machine as well the procedure to perform a rollback.
MAINTENANCE GUIDE	The MAINTENANCE GUIDE describes procedures for operations, such as how the RELEASE is kept fast and which logging mechanism is implemented.
CUSTOMER DEMO	The CUSTOMER DEMO is a presentation with the purpose of informing a customer about a certain RELEASE.
TRAINING MATERIAL	A TRAINING MATERIAL is the material to teach customers and users on how to use the RELEASE (Schwaber, 1995). Typically it only addresses the new USER STORY(ies).

Table 18. Concept table for the baseline

Appendix V. DevOps Patterns

Name	Layer	Area(s)	Source(s)
Cross-functional delivery team *Alternative name: active stakeholder participation, becoming a team*	Process	Area 3,4	(Ambler, 2013; Debois, 2012; Duvall, 2012; Lee, 2011; Rogowski, 2011)
Cross-functional skills *Alternative names: polyskilled engineers, DevOps culture*	People	Area 3,4	(Ambler, 2013; Duvall, 2012; Lee, 2011)
Develop for production	Process	Area 4	(Edwards & Thompson, 2011; Lee, 2011; Moon, 2010)
Automate for release *Alternative name: automated testing*	Tools	Area 1	(Ambler, 2013; Lee, 2011)
Consistent tooling	Tools	Area 1-4	(Lee, 2011)
Deployment pipeline *Alternative names: delivery pipeline, stages builds, build pipeline*	Tools	Area 1	(Duvall, 2012; Humble & Farley, 2010; Hüttermann, 2012; Lee, 2011)
Composable deployments	Process	Area 1	(Honor, 2010)
Adaptive deployment	Tools	Area 1	(Honor, 2010)
Code datasplit	Process	Area 1	(Honor, 2010)
Packaged artifact	Process	Area 1	(Honor, 2010)
Apply releases incrementally and iteratively	Tools	Area 1,2	(Hüttermann, 2012; Swartout, 2012)
Branch by abstraction	Tools	Area 1	(Hamment, 2011; Hüttermann, 2012)
Feature toggles	Tools	Area 4	(Hüttermann, 2012)
Dark launching *Alternative names: canary releases, pushed phased releases*	Tools	Area 1,2	(Hüttermann, 2012; Lee, 2011)
Blue-green deployment	Tools	Area 1	(Hüttermann, 2012)
Provision environments from versioned code *Alternative name: scripted environments*	Tools	Area 1	(Debois, 2012; Duvall, 2012; Hüttermann, 2012)
Provide monitoring and log files to development *Alternative name: application monitoring*	Tools	Area 2	(Debois, 2012; Hüttermann, 2012)
Set stability and capacity as development goals	Process	Area 3	(Hüttermann, 2012)
Integrate production stories *Alternative names: eliciting operations requirements,*	Process	Area 4	(Bass, Jeffery, Wada, Weber, & Zhu, 2013; Debois, 2012; Hüttermann, 2012)

integration of person and alignment of goals			
Developers wear pagers	People	Area 4	(Debois, 2012)
Version everything *Alternative name: integrated change management*	Tools	Area 1	(Ambler, 2013; Duvall, 2012)
Gatekeeper	Process	Area 3	(Hüttermann, 2012)
Check non-functional requirements	Tools	Area 4	(Hüttermann, 2012)
Integrated deployment planning	Process	Area 3	(Ambler, 2013)
Automated dashboards	Tools	Area 2	(Ambler, 2013)
Production support	Process	Area 3	(Ambler, 2013)
Task-based development	Tools	Area 4	(Hüttermann, 2012)
Early feedback by operations	Process	Area 4	(Hüttermann, 2012)
Quality scenarios	Process	Area 4	(Hüttermann, 2012)
Sync meeting	Process	Area 3,4	(Hüttermann, 2012)

Table 19. DevOps patterns

Appendix VI. Process Pattern Descriptions

Cross-Functional Delivery Team

Entry	Process Pattern Description
Name	Cross-functional delivery team
Author(s)	Debois (2012); Duvall (2012); Lee (2011); Rogowski (2011)
Version	1.0
Also Known As	Active stakeholder participation, becoming a team
Keywords	project team, communication, collaboration, delivery, sharing
Intent	Development and operations teams have historically been separate groups. By making operations part of the project, they can share their knowledge with other team members.
Problem	The developers and operators do not physically sit together and are mentally not on the same line. The operators are involved once the development is done and the system is ready for releasing. The primary task of operations is monitoring and incident handling.
Solution	"Teams work together in a dedicated fashion to deliver software consistently, without the time impediments inherent when teams communicate across the organization" (Duvall, 2012). Operations is part of a (virtual) project team from the very beginning of the project. A cross-functional delivery team makes every team member responsible for the software delivery process.
Realized Activity	Form delivery team
Initial Context	There are no corresponding work products required that allows the application of this process pattern.
Result Context	There are no new work products introduced.
Pros and Cons	Pros: ▪ Improves communication between developers and operators. ▪ Fosters collaboration and knowledge sharing between developers and operators. ▪ Enables faster feedback on the design of the system. ▪ Makes all team members responsible for the deliverables. Cons: ▪ The attitude towards each other should be mended. ▪ Mutual trust must be achieved. ▪ Not all people are willing to change their behavior. ▪ In the beginning more time is required to form a cross-functional team. ▪ The cultural gap between development and operations impedes this pattern
Example	The team at Rally Software evolved to DevOps because of some basic core values that are defended by everyone. "Placing people in a position to do work they are passionate about, embracing change, being respectful, and collaborating are fundamental things that lead to everything else" (Hüttermann, 2012). An anti-pattern is development, testing, and operations are not part of the same team. Some organizations implemented a distinct DevOps team as opposed to a cross-functional team.
Related Patterns	Cross-functional skills

Table 20. Process pattern description: cross-functional delivery team

Develop For Production

Entry	Process Pattern Description
Name	Develop for production
Author(s)	Lee (2011); Moon (2010); Edwards and Thompson (2011)
Version	1.0
Also Known As	-
Keywords	development, production, artifacts
Intent	The required artifacts that are needed to put the system into production are developed when the system is ready for releasing. The resulting errors could have been prevented if the artifacts were developed at an early stage.
Problem	The artifacts for operations are made when the release is already done, so there is no time left to review them. This results in unexpected deployment issues.
Solution	"Early creation of operational artifacts as part of the development process (for example, deployment and update scripts, automated database migration scripts, monitoring and reporting scripts)" (Lee, 2011).
Realized Activity	Develop for production
Initial Context	Documentations are made for the current release. There are two type of documents available, an installation guide and maintenance guide. Both documents are created after the development phase.
Result Context	In the result context, the aforementioned release documents are updated right after the development of a backlog component. Also, the team spends time on the creation of the health script and database update script.
Pros and Cons	Pros: ▪ Operational artifacts are kept up to date during the development. ▪ Timely feedback on operational artifacts from operations. ▪ Reduced errors during the deployment, so faster mean time to release (MTTR). Cons: ▪ The developer may not willing to spent time on tasks other than development.
Example	Moon (2010) tweaked the local DNS settings to simulate the production server address of the content delivery network (CDN) in the development environment. By doing this, the server address does not have to be changed for the development and staging environments and the code is production-proof. An anti-pattern is development, testing, and operations use their own scripts.
Related Patterns	Early feedback by operations

Table 21. Process pattern description: develop for production

Early Feedback by Operations

Entry	Process Pattern Description
Name	Operations provides feedback about the design of the application under development, early and often.
Author(s)	Hüttermann (2012)
Version	1.0
Also Known As	-
Keywords	operations, feedback, system, design, development
Intent	Operations give feedback about the feasibility of the system under development, so problems in the transition from development to operations are adequately tackled.
Problem	Sometimes the infrastructure is not sufficient to support the new system,

	therefore the system cannot directly put into production.
Solution	"The goal is to enable the development team to gain fast feedback about feasibility and to share knowledge across teams early and often" (Hüttermann, 2012).
Realized Activity	Assess and adapt current infrastructure, Review system design
Initial Context	In the current situation, IT operators review the system once the development has finished. The IT infrastructure is, however, never assessed before the system development takes place.
Result Context	There are no new concepts introduced, but the existing IT infrastructure is now part of the process. IT operators attend sprint and demo meetings to provide feedback on the systems design.
Pros and Cons	Pros: ▪ Less risks in the release process. ▪ Better alignment between development and operations. ▪ Reduces errors in the release process. ▪ Stimulates communication between development and operations. ▪ Fosters knowledge sharing. Cons: ▪ There should be additional procedures and guidelines for the assessment of the IT infrastructure.
Example	No practical example available.
Related Patterns	Cross-functional delivery team

Table 22. Process pattern description: early feedback by operations

Integrate Production Stories

Entry	Process Pattern Description
Name	Integrate Production Stories
Author(s)	Hüttermann (2012); Debois (2012); Bass et al. (2013)
Version	1.0
Also Known As	Eliciting operations requirements, integration of person and alignment of goals
Keywords	production, integration, project, backlog, stories
Intent	Production issues or quality requirements are too late addressed. Therefore, stories should be inserted into the product backlog in an early stage.
Problem	The development and operations department are originally siloed environments, where they have their own work items. This hinders the cooperation.
Solution	From the beginning of the project, all stories related to production (such as monitoring, security, etc.) are integrated into the product backlog. This eliminates the discrepancies between development and operations.
Realized Activity	Define contents for releases
Initial Context	The user stories are derived from the product backlog, and are thereafter assigned to the sprint plan for a particular sprint.
Result Context	The result context includes a new work product for production stories, which are derived from quality requirements. Alternatively, it is also possible to attach quality acceptance criteria to existing user stories or apply an hybrid solution.
Pros and Cons	Pros: ▪ Fosters collaboration between development and operations. ▪ Monitors quality requirements during the project. ▪ Involves operations in the project. Cons:

	▪ More time is needed to discuss and prioritize the production stories. ▪ Developers are expected to be unhappy with additional stories which were previously not part of the project.
Example	No practical example available.
Related Patterns	Develop for production

Table 23. Process pattern description: integrate production stories

Sync Meeting

Entry	Process Pattern Description
Name	Sync meeting
Author(s)	Hüttermann (2012)
Version	1.0
Also Known As	-
Keywords	sync, meeting, communication, alignment, release
Intent	Development and operations should be brought closer together so that they can discuss the upcoming release in order to prevent any pitfalls when placing the system into production.
Problem	There is no alignment on the transition to production.
Solution	For each new release there is held a DevOps sync meeting. During this meeting developers and operations come together and discuss operational issues that have occurred during the last release as well as planning for the upcoming release (Hüttermann, 2012).
Realized Activity	Sync meeting
Initial Context	Available work products are release documents such as the installation and maintenance guide.
Result Context	There are no new work products introduced. However, it is likely minutes are recorded and stored on a central location.
Pros and Cons	Pros: ▪ Early discovering of expected issues. ▪ Learn from early experiences. ▪ Fosters knowledge sharing. ▪ Stimulates communication. Cons: ▪ Additional time and effort is needed to synchronize both ‘silos’.
Example	At Rally Software the collaboration process is facilitated and built into their daily process (Hüttermann, 2012). “The team discusses new changes, maintenance and talks about areas that can be improved that are not necessarily architectural. In any given month they have weekly demos, which are opportunities for operations and development to get feedback on the work they are doing” (Hüttermann, 2012).
Related Patterns	Cross-functional delivery team

Table 24. Process pattern description: sync meeting

Appendix VII. Updated Activities and Concepts for the Situational Method

Activity	Description
Form delivery team	During this activity the team members form a cross-functional delivery team is. A workshop is held to formulate clear goals for the team (developers, testers, technical application manager). This workshop is facilitated by the Scrum master.
Define foundations	The team determines the SHARED GOALs for the team.
Define scope	The SCOPE as well boundaries and context are defined by the team. It becomes clear what is not in the scope of the definition of done (DoD).
Define quick wins	The team defines the quick achievable results for the project.
Define path to solution	The team elaborates on the path to come up with the shared goals.
Define next steps	The team rearrange and plan next steps to foster shared goals.
Define slack time	SLACK TIME is defined in order to improve the daily work, team collaboration, and the definitions of the shared goals.
Create product backlog	A PRODUCT BACKLOG is created by identifying features, functions, requirements, enhancements, and fixes that are not addressed by the current release. In addition, a QUALITY REQUIREMENTS LIST is made available for the team.
Define contents for releases	The product owner selects the STORY(ies) that will be covered by the RELEASE PLAN for the current sprint. A story can either be a USER STORY or a PRODUCTION STORY. QUALITY REQUIREMENTs may result in QUALITY CRITERIA for individual user stories or in new PRODUCTION STORY(ies).
Assess and adapt current system architecture	The SYSTEM ARCHITECTURE is assessed by the technical application manager (TAB) to check whether the system design is correct. If needed, adaptations are made by business analyst integration (BAI), architect, and project manager, who are responsible for these design documents.
Inspect HLSI	The HLSI is inspected by TAB. This document describes the application landscape and service calls across the systems.
Inspect PAC	The PAC is inspected by TAB. This document describes the general architectural principles and the standards to be met. Also it provides a technological model with a detailed description of the systems and interfaces involved.
Inspect SOPO	The SOPO is inspected by TAB. This document describes the development and production environment.
Inspect ISP	The ISP is inspected by TAB. This document describes the input and output of the system interfaces.
Report findings	Findings on the design inspection are reported by person that has performed the inspection.
Adapt system architecture	Based on the inspection findings, corrective actions are taken by business analyst integration (BAI), architect, and project manager to enhance the system architecture.
Assess and adapt current IT infrastructure	The IT INFRASTRUCTURE is assessed by the technical application manager to check whether the IT INFRASTRUCTURE is sufficient to support the realization of the STORY(ies). If needed, adaptations are made to the IT INFRASTRUCTURE.
Develop for production	The team develops the backlog components and afterwards the system

	is assessed by the team whether the QUALITY REQUIREMENTs are not at risk. Furthermore, operational artifacts and release documentation are updated.
Update health script	Based on the QUALITY REQUIREMENTs the team updates the health script to the actual state of the system.
Update database script	The team modifies the database update script so the database schemes reflect the actual state of the system.
Update release documentation	The team modifies the release documentation, such as the INSTALLATION GUIDE and MAINTAINANCE GUIDE to reflect the actual state of the system.

Table 25. Updated activities for the situational method

Concept	Description
SHARED GOAL	SHARED GOALs are taken into account by all team members during the project. It is used as foundation for all activities.
SCOPE	The SCOPE describes what is in the scope of the definition of done for the project.
QUICK WIN	A result that can be achieved quickly and is appreciated by all.
PATH TO SOLUTION	The way in how the team is able to achieve the SHARED GOALs.
STEP	STEP describes a single action towards the solution.
SLACK TIME	SLACK TIME enables thinking and analyzing the current working approach.
STORY	A STORY can either be a USER STORY or PRODUCTION STORY and are derived from the PRODUCT BACKLOG or QUALITY REQUIREMENTS LIST.
PRODUCTION STORY	PRODUCTION STORY(ies) are derived from QUALITY REQUIREMENTs and are written in the context of the project.
QUALITY REQUIREMENTS LIST	The QUALITY REQUIREMENTS list contains one or more QUALITY REQUIREMENTS for the system.
QUALITY REQUIREMENT	A QUALITY REQUIREMENT describes the non-functional behavior of the system (e.g. the system should respond fast) and constrains for the system (e.g. the system is developed on a Unix platform).
QUALITY CRITERIA	The QUALITY CRITERIA is derived from one or more QUALITY REQUIREMENTs and are written in the context of individual USER STORY(ies). The QUALITY CRITERIA should be met in order to ensure the QUALITY REQUIREMENTs are properly implemented.
SYSTEM ARCHITECTURE	The SYSTEM ARCHITECTURE is the composition of materials that record design decisions about the system under development.
IT INFRASTRUCTURE	The IT INFRASTRUCTURE includes the hardware, operating software, communications, other equipment and support required to enabled business applications (Turnbull, 1991). The design requirements for the information system must fit within the existing IT INFRASTRUCTURE.
HEALTH SCRIPT	The HEALTH SCRIPT assesses the system whether the QUALITY REQUIREMENTs are properly implemented.
DATABASE SCRIPT	The DATABASE SCRIPT ensures the database schemes of the production environment reflect the actual situation of the system under development.

Table 26. Updated concepts for the situational method

Appendix IIX. Case Study Protocol for the Pilot Experiment

1. Introduction

The goal of this case study is to set up a pilot experiment for implementing the selected process improvements at CaseComp and to obtain feedback on the solution for the identified problem areas. The case study aims to answer the last subquestion of the research:

> *SQ5. How can the optimal integration scenario be executed in a real development project?*

In Chapter 6 we determined the optimal integration scenario for the selected improvements. By executing this scenario in a real development project, we are able to provide feedback on the selection approach and elaborated implementation paths. The objectives of this case study are formulated as follows:

- Obtain feedback on the implemented process improvements.
- Validate the method fragments and situational method.
- Validate the scenario selection.
- Elicit factors that shape the DevOps integration.

The case study protocol is highly incremental in nature, so initial findings may result in adaptations of this document. During the pilot case study the progress and results are compared with the plan. Any changes are recorded, which leads to recommendations for changes in procedures.

2. Design

The case study for the pilot experiment consists of a single-case design with multiple units of analysis, also referred to as an embedded design according to the basic types of designs for case studies by Yin (2009). Basili, Selby, and Hutchens (1986) defined software engineering experiments in terms of a two-dimensional classification scheme: single-project studies, multiproject studies, replicated-project studies, and blocked subject-project studies. This case study is considered as a multiproject study, which examines objects across a single team and a set of projects (Basili et al., 1986).

The case study uses a pilot experiment (also referred to as pilot case study). “The pilot case can assume the role of a "laboratory" in detailing your protocol, allowing you to observe different phenomena from many different angles or to try different approaches on a trial basis” (Yin, 2009). This approach is therefore ideally suited for validating the integration scenario as both method increments by obtaining feedback from the environment. Also, mistakes or validity problems can easily rectified using a pilot case and thus, is cost-efficient. Instead of setting up a formal laboratory setting where the pilot does not affect the development of the information system, the included project teams actually develops working software during the pilot.

For the implementation of the process changes we use the iterative improvement process by Salo and Abrahamsson (2007). Originally, this process adapts the development process in an iterative way based on the experiences and context knowledge of software developers. As we have already

identified the problem areas, we use their approach to attach process improvements to sprints and validate them in the subsequent iteration. In this manner the method runs in sync with Scrum and method fragments are stepwise implemented according to the scenario.

To ensure the case study is scientifically sound, we use the guidelines for case study experiments by Kitchenham et al. (1995) who propose the following guidelines:

1. Define the hypothesis
2. Select the pilot projects
3. Identify the method of comparison
4. Minimize the effort of confounding factors
5. Plan the case study
6. Monitor the case study against the plan
7. Analyze and report the results

3. Case selection

The baseline method from Chapter 4 is used as main criteria for selecting a pilot case or cases. Possible cases are selected from a pool of available Scrum projects. The processes of the teams should match the baseline method in order to include them in the final selection, otherwise the integration scenario does not make sense for these projects. A prerequisite is that the pilot starts at the same time for all cases so the results can be processed within the time limitations of the research. The selected teams are timely informed on the start date of the pilot experiment. We aim to start the pilot in the first sprint of the project so the Scrum development process is executed from the beginning.

4. Case Study Procedures and Roles

4.1 Define the hypothesis

We start with defining the effect we expect the situational method to have. A hypothesis is defined for each process driver from Chapter 4 in order to measure the effort on the specific problem area. These measurements will be used to demonstrate the effect by implementing the proposed solutions. Hypotheses are linked to each method fragment according to Figure 26 in section 5.1.2. In the *Data collection* section of the case study protocol we elaborate on the metrics of choice and provide the reader with mathematical representations.

Code / category	Hypothesis
D1. The processes of the development and operations departments are not aligned with each other.	H_0. The standard deviation of the project velocity is equal. H_1. The standard deviation of the project velocity is decreased.
D2. Lack of standardization for quality guidelines.	H_0. The number of production acceptance testing (PAT) issues per release is equal. H_1. The number of production acceptance testing (PAT) issues per release is decreased.
D3. IT Operations is not well represented in the project.	H_0. The ratio of finished backlog items addressing quality requirements compared to the ratio of finished

	user stories is equal. H_1. The ratio of finished backlog items addressing quality requirements compared to the ratio of finished user stories is increased.
D4. Too comprehensive process for releasing information systems.	H_0. The time between the last PAT approval and the time of release is equal. H_1. The time between the last PAT approval and the time of release is increased.
D5. Moderate communication between development and operations.	H_0. The ratio of the number of quality defects per quality requirement is equal. H_1. The ratio of the number of quality defects per quality requirement is decreased.

Table 27. Hypotheses mapped to the main drivers

4.2 Select the pilot projects

The team of which the current situation is determined in Chapter 4 is already selected for participation. Since the Scrum implementation differs from project to project, there are no additional teams that meet the baseline. Due to time limitations of the research we are not able to elaborate on different baselines for other teams, so we do not know whether they experience the same problems. Therefore our pilot case study is limited to a single project.

Characteristic	Case project
Size of project (person months)	18
End product	Extension to an existing information system
Duration	8 weeks
Iteration length	4 x 2 weeks
Team size	9

Table 28. Characteristics of the case project

4.3 Identify the method of comparison

In order to compare the results of the new method with the existing method we need to choose a valid basis for assessing the results of the case study. Kitchenham et al. (1995) proposes three ways to facilitate this comparison: (Kitchenham et al., 1995)

- Select a sister project with which to compare.
- Compare the results of using the new method against a company baseline.
- If the method applies to individual components, apply it at random to some product components and not to others.

For this case study we chose the second approach in which the results of the new method are compared against the baseline (i.e. the results from a previous project).

4.4 Minimize the effort of confounding factors

There is one confounding factor for this case study that may affect the project performance. As we have selected a high mature Scrum team it is likely the team is very enthusiastic in improving the current method and may adopt changes more quickly than a team that is very skeptical about the new method. For example, the staff morale can have a large effect on productivity and quality. On

the other hand we can say, if the performance of the pilot project is negligible it is not due to the lack of team motivation. As we cannot eliminate this confounding factor we try to minimize the effect by telling the team in the beginning of the project that we expect from them to be critical. Also, we will not discuss the measures that will be used as baseline. The pilot facilitator should monitor the individual efforts during the pilot. Individuals should not waste excessive amounts of energy in these improvements, as we consider it as a team effort.

4.5 Plan the case study

The pilot experiment is planned over four iterations (Figure 49) in which the improvements from Table 29 are sequentially implemented. The experiment is supported by the iterative improvement process (IIP) by Salo and Abrahamsson (2007). At the end of each iteration a post-iteration workshop (PIW) is conducted to evaluate experiences and measurements from the previous iteration. The KJ method is used to structure the process for obtaining feedback. The KJ method focuses on the main question that need to be answered during a particular session (e.g. how can the method being improved). During the PIW session, the improvements planned for the current iteration are also implemented and communicated. The session has an estimated duration of 2 hours and is led by a facilitator, in this case the team's Scrum master. The facilitator is also responsible for the white areas of the process, whereas the grey areas are performed by the team. The researcher participates in the PIW meetings to record any observations. There is planned one improvement for each iteration, with an exception for a single iteration for which two improvements are planned. The case study is planned as of 1 May 2013.

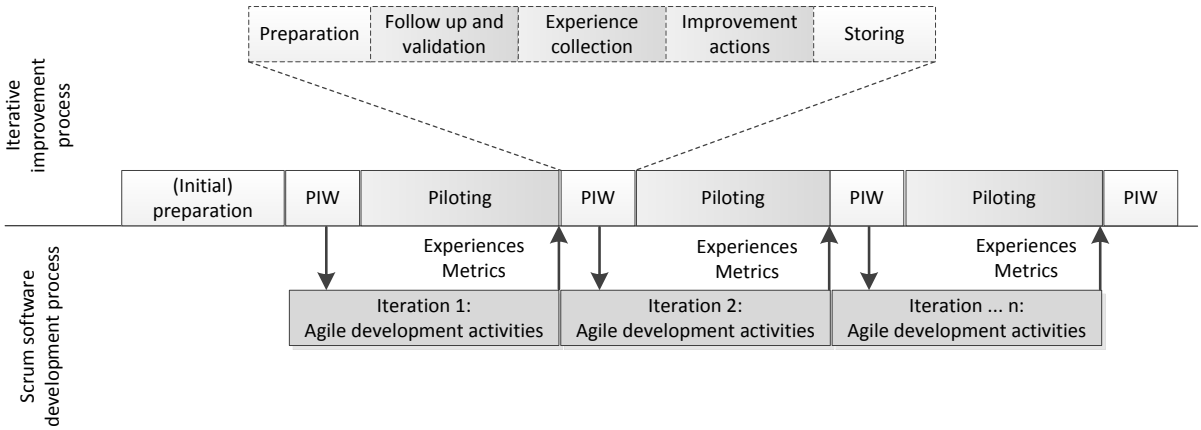

Figure 49. Iterative improvement process (based on Salo and Abrahamsson, 2007)

Iteration 1	Iteration 2	Iteration 3	Iteration 4
1. Cross-functional delivery team	2. Integrate production stories	3. Early feedback by operations	5. Sync meeting
		4. Develop for production	

Table 29. Improvements planned for the Scrum iterations

5. Data Collection

In this case study we collect both quantitative as qualitative data. The quantitative data relate to the hypotheses from section 4.1. In the preparation phase we collect the data on all metrics. These numbers will be used as baseline to compare with. After each iteration the project results are saved to the case study database. In addition, we obtain qualitative feedback from the team on the improvements and integration process itself during the post iteration workshops. Based on this feedback we are able to enhance the situational method and scenario selection process, and thus we can provide an answer on the research question stated at the beginning of this plan. The collected data is stored in a case study database. The case study database is stored using a cloud storage service to ensure both availability and integrity. Below we discuss the metrics for the quantitative analysis, which are derived from the hypotheses from section 4.1.

Metric for D1

We argue that the alignment of developmental and operational goals (D1) result in a productive team effort. The equally distributed effort is expressed by the standard deviation of the project velocity. Project velocity is measured by simply adding up the estimates of the user stories that were finished during the iteration. It is the key to keeping the project moving at a steady predictable pace (Wells, 1988). In the past the project velocity was subjected to a relatively high standard deviation, which was due to slack time (i.e. waiting time) at the end of the process. This slack time is caused by operational processes that were not in sync with development.

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n} (V_i - \mu)^2}{n-1}}$$

V_i = ∑ of original estimates of all accepted work in period $_i$

Metric for D2

The lack of standardization for quality guidelines (D2) is assessed by the number of production acceptance testing (PAT) issues found (I_n) per number of releases (R_n) in order to pass through quality control. One or more issues found during PAT result in a redelivery of the system. The result is the issue/release (IR) ratio which indicates fluctuations between the number of issues. The denominator is mainly intended to make the corrections for the baseline. In the case of the pilot, the number of releases will be 1.

$$IR = \frac{I_n}{R_n}$$

Metric for D3

The involvement of operations in the project (D3) is measured by the total points of finished backlog items addressing quality requirements (∑QR) (e.g. production stories, quality criteria) in ratio with the total points of finished user stories (∑US). We are referring to the finished story points to maintain the workload in the final ratio. The result is the quality requirement/user story (QU) ratio which indicates the proportion of quality requirements for the current release increment.

$$QU = \frac{\sum QR}{\sum US}$$

Metric for D4

In a typical project, the team needs to obtain approval by several parties. Any improvements in the software release process (D4) can be measured by the slack time or idle time (IT) after the last approval (A_t) is obtained. The slack time is due to the pre-set release date (R_t), at which the release is deployed to production.

$$IT = 1 - (A_t - R_t)$$

Metric for D5

We argue that the communication between development and operations (D5) is of direct effect on the quality of the system as the operational guidelines should be better monitored, and therefore affect the number of defects (e.g. bugs, outages) of quality requirements (QRD_n) in production. The defects/quality requirement (DQR) ratio expresses the proportion of reported quality defects per quality requirement (QR_n). According to the Scrum master, we only have to measure the effects in the first two days after the release is put into production. Most of the quality defects are usually found in this period.

$$DQR = \frac{QRD_n}{QR_n}$$

6. Analysis

The data analysis is twofold. First, we want to determine whether the process changes had a significant effect on the performance of the project. Since the data on the metrics is unavailable during the pilot, we compare the measures at the end of the pilot. As the measures provide one response value to compare with, no analysis technique is chosen. The results may confirm the method fragments as a suitable whole for solving particular problem areas. Also the findings can help to improve the pattern mapping table (Figure 26) from Chapter 5.

Second, we gather feedback on the situational method (e.g. learning curve for team members, suitability for tools) and the integration process. The recorded empirical observations are investigated by looking for patterns and phenomena that occur during the pilot experiments. The obtained feedback shapes the answer on the last research question.

7. Plan Validity

According to Yin (2009) there are four types of validity threats that apply to this case study: construct validity, internal validity, external validity, and reliability. With respect to construct validity, we establish operational measures for the concepts being studied. The case study protocol is developed using the template provided by Brereton, Kitchenham, Budgen, and Li (2008). The case study protocol is then validated using the guidelines for case study design by Runeson and Höst (2008). The protocol ensures the data collection and analysis procedures are focused on their primary goals. The

internal validity is threatened by results from wrong measures, or measures that are not only related to a single problem area. The research is originally intended to aid IT organizations in implementing method enhancements. With respect to external validity, a threat is that CaseComp is not representative for the Dutch IT organization. Despite CaseComp facilitates IT services, financial services are the core businesses of the company. Finally, to defend experimental reliability, the case study procedures can be replicated for other cases in order to increase the generalizability of the results. We use the guidelines for case study planning by Kitchenham et al. (1995) to ensure the criteria for research-design quality is adequately addressed. Additionally, the case study protocol is checked against the checklist for experimental case studies by Kitchenham et al. (1995).

The case study is conducted by the lead researcher. For the rigor of the research, the supervisors of the research assess the case study protocol documents to ensure validity issues are consistently addressed.

8. Study Limitations

An important limitation is the research is limited to one object of study, namely the development team with a high mature development process. Since all development teams use their own implementation of the Scrum process, it could be possible that not all drivers and issues are included for analysis. Therefore, elaborated process improvements in the research address only a limited set of drivers that apply to CaseComp. This problem can be tackled by replicating the case study procedures to other development teams, to elicit issues and drivers which were initially not identified. To replicate the integration process, project teams should adapt their process to comply with the baseline from Chapter 4.

Another important limitation are the measurements that can only be performed afterwards, when the method fragments are assembled into the process. The metrics use data which is only available after the development project has ended, or the release is deployed to production. The result is that a little can be said on the effectiveness of individual method fragments. Also, as the scheduled end date for the pilot is close to the deadline of the research, so we have a single week to collect and process the data for metric D5 (i.e. quality defects). This will impact the accuracy of this metric, as well the corresponding findings.

9. Reporting

The outcome of the case study is reported in Chapter 7 of the thesis. The target audience is the IT organization in general, as the results of this case study provide feedback on the established integration scenario as well the elaborated process patterns and the result of these practices in a real project simulation.