The present work was submitted to the Research Group Software Construction

of the Faculty of Mathematics, Computer Science, and Natural Sciences

Master Thesis

A Large Scale Industrial Case Study of Continuous Delivery with JARVIS

Eine umfassende industrielle Fallstudie in Continuous Delivery mit JARVIS

presented by

Bastian Greber

Aachen, October 14, 2019

Examiner

Prof. Dr. rer. nat. Horst Lichter

Prof. Dr. rer. nat. Bernhard Rumpe

Supervisor

Dipl.-Inform. Andreas Steffens

Statutory Declaration in Lieu of an Oath

Eidesstattliche Versicherung

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Masterarbeit mit dem Titel

A Large Scale Industrial Case Study of Continuous Delivery with JARVIS

selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Aachen, October 14, 2019 (Bastian Greber)

Belehrung

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicher ung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen.

Aachen, October 14, 2019 (Bastian Greber)

Acknowledgment

I would first like to thank Prof. Dr. rer. nat. Horst Lichter for the possibility of writing my master thesis at his chair. I also thank him and Prof. Dr. rer. nat. Bernhard Rumpe for reviewing this thesis.

Special thanks go to my supervisor, Dipl-Inform. Andreas Steffens for his input and continuous feedback.

Furthermore I thank everybody at KITERS AG for their support during the case study, especially Lukas Schmidt and Andreas Kinderreich, who assisted me whenever possible.

Finally, I thank my parents for providing me with unfailing support throughout this thesis. This accomplishment would not have been possible without them. Thank you.

Bastian Greber

Abstract

Motivated by current problems in continuous delivery the JARVIS continuous delivery reference architecture was developed. Such problems are for example complex delivery processes that are not efficiently supported by the delivery system. This thesis evaluates the concepts of a JARVIS to cope with these problems in a real world productive scenario. Hereby the available functions and usability is assessed and compared to the underlying goals of JARVIS. In cooperation with the KISTERS AG, an extensive case study was executed in such a productive environment. This case study is divided into three individual smaller studies which each targeting specific aspects of JARVIS. First the basic functionality of JARVIS was evaluated by modelling real world continuous delivery solutions in the JARVIS system and comparing them to existing solutions. JARVIS was able to provide the desired functionality and produced equivalent artifacts, compared to the original solution. During the next study, the scope was extended to requirements which are currently not fulfilled by the continuous delivery solution at KISTERS. For this, first relevant continuous delivery stakeholders were identified. Here the whole business unit was considered as potential stakeholders. In connection the stakeholders were interviewed to identify their continuous delivery requirements. These requirements were then evaluated against JARVIS. It is possible to satisfy almost all of the gathered requirements with a JARVIS based system. Finally the third study investigated the experimental modelling features of JARVIS. This was done via an expert evaluation of those features which found these suitable for the desired tasks.

Contents

1. Introduction **1**
1.1. Thesis Structure 2

2. Background and Terminology **3**
2.1. Continuous Delivery 3
2.2. Deployment Pipeline 5
2.3. JARVIS 6
2.4. Empirical Software Engineering 9

3. Case Study Design **13**
3.1. Problem Statement 13
3.2. Goals and Contributions 14
3.3. Case Study Structure 15
3.4. Objectives and Research Questions 15
3.5. Case Studies 17
3.6. Summary 21

4. Related Work **23**
4.1. Case Study: Introducing Continuous Delivery of Mobile Apps in a Corporate Environment 23
4.2. Stakeholder Perceptions of the Adoption of Continuous Integration 24

5. Case Study 1: JARVIS In A Production Environment **27**
5.1. Study Design And Execution 27
5.2. KISTERS Reference Project 29
5.3. Findings o1.R1 31
5.4. Findings o1.R2 35
5.5. Findings o1.R3 36
5.6. Findings o1.R4 40
5.7. Classification of activity services 41
5.8. Composition of activities 43
5.9. Discussion and Summary 45

6. Case Study 2: CD Stakeholder And Requirements **49**
6.1. Study Design And Execution 49
6.2. Continuous Delivery Stakeholder 50
6.3. Continuous Delivery Requirements 53
6.4. Continuous Delivery Survey 63

6.5. Discussion and Summary 76

7. Case Study 3: Continuous Delivery Modelling **81**
7.1. Study Design And Execution 82
7.2. Process-Based modelling with BPMN 83
7.3. Process-Based modelling with DSL 84
7.4. State-Based modelling with DSL 85
7.5. Discussion and Summary 89

8. Summary and Future Work **91**
8.1. Summary 91
8.2. Future Work 93

A. Interview Guides **95**

Bibliography **101**

List of Tables

5.1. Developed Activities during Case Study 1 46

6.1. Preferred notification methods 65
6.2. Acceptable runtime for certain CD operations 75
6.3. Requirements mapped on the identified stakeholders 78
6.4. Requirements mapped to their Components from the Delivery Pipeline Model 79

List of Figures

2.1. Relationship continuous integration and delivery [LIL17] 4
2.2. Relationship between Delivery Model, System and Process [Dör17] 6
2.3. JARVIS activity classification [SLD18] 8
2.4. Holistic case study (left) and embedded case study (right) [RH09] 11

3.1. Decomposition of the thesis in sub-case studies 16
3.2. First study, multiple holistic case, mini case studies 18
3.3. Second study, embedded case study on general employee base 20
3.4. Third study, embedded case study with stakeholder 21

5.1. Design of the first case study regarding JARVIS general abilities 28
5.2. Reference project pipelines 30
5.3. Pipeline for the backend component of the reference project 31
5.4. The commit stage of the reference project 32
5.5. KISTERS reference project pipeline in JARVIS 39
5.6. Architecture remote-activity service 42
5.7. Architecture library-activity service 43
5.8. Architecture shell-activity service 44

6.1. Second study, embedded case study on general employee base 50
6.2. Organization chart of the KISTERS AG BU Energy 51
6.3. Results How importance is fast feedback for you if a pipelines fails? 64
6.4. Results What would be an acceptable feedback time by the CD system? . 64
6.5. Results How important is positive feedback for you? 65
6.6. Results How important is a reminder function by the CD system? 65
6.7. Importance visualize dependencies between different pipelines 67
6.8. Importance show pipelines as a sequence of steps 68
6.9. Importance provide information regarding pipeline runtime 68
6.10. Importance show the technical commands contained in a pipeline 68
6.11. Importance show trends regarding stability 69
6.12. Results How important are configurable dashboards? 69
6.13. Results How important is modelling system landscapes in CD? 71
6.14. Results How important is graphical modelling? 71
6.15. Results How important is textual modelling? 72
6.16. Results How important is self-service in CD for you? 72
6.17. Results How important are optional pipeline steps? 72
6.18. Results How important is multi-branch support through the CD? 73
6.19. Results How important are predictive builds? 73

6.20. Results How important is speed in the CD context? 74
6.21. Results How useful is caching artifacts to improve the pipeline runtime? . 74
6.22. Results How useful are APIs to test pipeline steps locally? 75

7.1. Third study, embedded case study with CD modelling as case 82
7.2. Unsolvable model for the smart planner 86
7.3. Pipeline for example application 87

A.1. Interview Guides page 1 of 5 95
A.2. Interview Guides page 2 of 5 96
A.3. Interview Guides page 3 of 5 97
A.4. Interview Guides page 4 of 5 98
A.5. Interview Guides page 5 of 5 99

List of Source Codes

7.1. Extract from a JARVIS delivery model 84
7.2. Pipeline Model for example project state-based 87

1. Introduction

Contents

1.1. Thesis Structure 2

The JARVIS research project was developed at the Research Group Software Construction (SWC) of the RWTH Aachen. This development was performed over the course of multiple theses and research papers. JARVIS is a continuous delivery reference architecture which tries to provide solutions for current continuous delivery problems. Such problems are for example the complexity of the pipeline descriptions or build scripts [LIL17]. The defined goals of JARVIS are as follows. Firstly, a delivery system, following the JARVIS blueprint, should be easy to maintain and operate. The second goal is to simplify the modelling and interacting with the delivery process for the user [Dör17].

Up until now, no detailed validation of those goals was performed. During the work of Döring a small case study with an example project was done, but the size of the study was very limited and the example project was also used for developing further JARVIS functionality. To improve on this situation, this thesis aims to investigate the performance and usability of JARVIS in a productive scenario. This will not only validate the concepts of JARVIS but also provide fresh feedback and ideas for future expansions. To ensure good quality results, it is important that the current case study is executed in an environment with no previous JARVIS work. By this we hope to achieve an unbiased evaluation of the characteristics and features of JARVIS. The KISTERS AG provides such an environment and is the cooperation partner for this work. KISTERS provides software solutions in the energy and water market and is a long lasting partner of the SWC. Additionally no previous research regarding JARVIS was done with their help. Their motivation to support this work was the detailed assessment of their continuous delivery solution. As a software company they already had a strong involvement with CD and are always interested in improving their solutions. This case study will use their existing productive projects and introduce them to a JARVIS system. During that process their existing delivery solution is also investigated and feedback or possible improvements can be identified. Furthermore, as part of this study continuous delivery stakeholder are searched and their requirements evaluated. These requirements can then be used us to test JARVIS concepts further and by the KISTERS AG to improve their system. Another aspect of this study will the assessment of experimental features of JARVIS. Features like automated delivery model optimization, creating delivery models in BPMN notations [Wil18] or regression test [Nun18] optimization are mapped against real world problems and their usefulness is tested.

1.1. Thesis Structure

This thesis is structured as follows. First needed background information and terminology is introduced in chapter 2. Next the goals and the overall design of the case study is developed in chapter 3. To put the objectives and research questions into context, the following chapter 4 introduces already existing research and information learned. This case study is divided into three smaller sub-case studies. In chapter 5 the first case study about JARVIS usability is executed. The next study is described in chapter 6 and evaluates continuous delivery stakeholders and their requirements towards CD. Lastly the third case study (see chapter 7) deals with the results of the third study. During this study the modelling methods of JARVIS were evaluated and a new idea for continuous delivery modelling found. This thesis then closes in chapter 3.6 with a summary of the found results and inspiration for potential future work.

2. Background and Terminology

Contents

2.1. Continuous Delivery 3
2.1.1. Principles of Continuous Delivery 4
2.2. Deployment Pipeline 5
2.3. JARVIS 6
2.3.1. Architecture 7
2.3.2. JARVIS Activities 7
2.3.3. Modelling with Smart Planner 8
2.4. Empirical Software Engineering 9
2.4.1. Case Studies 9
2.4.2. Semi-Structured Interviews 10

In this chapter multiple topics, which represent foundational work for this thesis, are introduced. These topics are intended to help the reader get a better insight into the subjects and provide a quick overview. The focus will be put exclusively on aspects relevant for this thesis. For a more detailed understanding, further research in external literature is advised.

2.1. Continuous Delivery

Continuous Delivery is software engineering approach which allows the user the release of reliable software in short increments. One major factor of continuous delivery is the ability to produce releases at almost any given time and still ensure a high quality [Che15]. Multiple popular process models like rapid prototyping, incremental and iterative software development or agile development [JH07] in general, are benefiting from continuous delivery. To achieve the specified goals, the release process of a given software, is executed automatically on every change (e.g. commit in a source code repository). Analogous to Humble and Farley, this section defines the general release process as four steps. The first step is the build step in which the artifact is created. The second step is the deployment step to install the created artifact to a test system or artifact-repository. This is then followed by the testing step. During this step the artifact is tested against predefined quality gates it has to satisfy. Finally the software is released which marks the end of the delivery process. To properly understand continuous delivery it is important to know its relationship with continuous integration [LIL17] [Vir15]. Figure 2.1 shows that relationship. During continuous integration changes are merged nonstop into the main code-base. Hereby it is important to always maintain a runnable artifact. Continuous

Delivery now extends continuous integration with testing and automated deployment into productive environments [DMG07]. Other stages can be included as well.

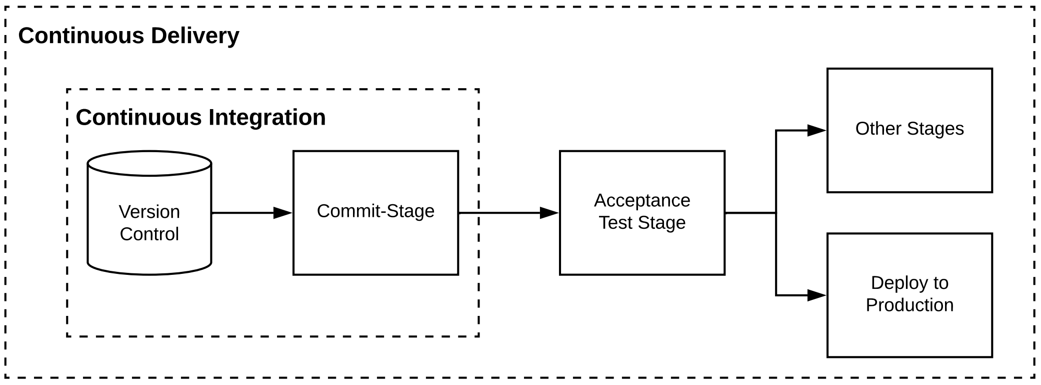


Figure 2.1.: Relationship continuous integration and delivery [LIL17]

Following the argumentation of Chen, the user of continuous delivery try to achieve the following benefits by following the CD approach.

1. Accelerate time to market
2. Building the right product
3. Improved productivity and efficiency
4. Reliable releases
5. Improved product quality
6. Improved customer satisfaction

2.1.1. Principles of Continuous Delivery

Based on the work of Humble and Farley, the basic principles of continuous delivery can be formulated as follows [HF10].

Suitable Process As mentioned before, continuous delivery relies on the delivery process of the software product. To get the most out of it, the process must be suitable and optimized for continuous delivery. Mainly this means the process has to be repeatable and executable by a machine. It must be predictable what the process does and every run must yield the same results (if the input has not changed). This also means the process has to be reliable.

High Automation The automation of tasks is key for continuous delivery. If it is possible, everything should be automated. This comes with two major advantages. First the process is executed in a repeatable fashion. Every time the task is performed, it is done so in the same way. This grants the ability to fix errors easily and optimize the process. The other aspect is the reduction of errors. By letting the machine do the monotonous and repetitive tasks, the error rate decreases significantly. By combining these aspects, a high degree of automation does increase the trust into the delivery solution.

Version Control Everything related to the software project should be kept under version control. This holds especially true for the source code itself. It allows the tracking of changes and allowing a revert in case problems arise. In most cases the version control is the starting point for the delivery process. The process itself is triggered by any changes in the corresponding repository.

Frequent Execution Continuous Delivery is meant to be executed frequently. Therefore it should be applied on every change to the code basis. If one step of the process is problematic or error prone, it only means it is not executed often enough.

Build Quality In This principle aims at multiple aspects. Firstly errors should be found as soon as possible. If the search for them is integrated into the process this can be achieved more easily. The second aspect is the direct integration of tests into the delivery process. Testing is not a separate action after the artifact is build but a continuous activity.

Done means released An important aspect of continuous delivery is the definition of done. A feature is completed if it was successfully released and shipped to the client. Many developers do consider a feature "done" after the initial development is completed but a finished feature requires the whole release process to be executed.

Responsibility In a continuous delivery scenario everybody is responsible for the delivery process. The input of anybody related to the development is gathered and considered in the delivery process.

2.2. Deployment Pipeline

In literature and research the term "Delivery Pipeline" gets used in manifold ways. Some define it as process, other see it as model for a process or even as a software system. To avoid confusion, the definition of those terms was adopted for this thesis from the work of Steffens, Lichter, and Döring. The research paper [SLD18] splits the term delivery pipeline in three parts and puts those parts in relation to each other. This relationship is shown in figure 2.2.

Delivery Model The delivery model is an abstract representation of the delivery process. It is maintained by the user and serves as input for the delivery system.

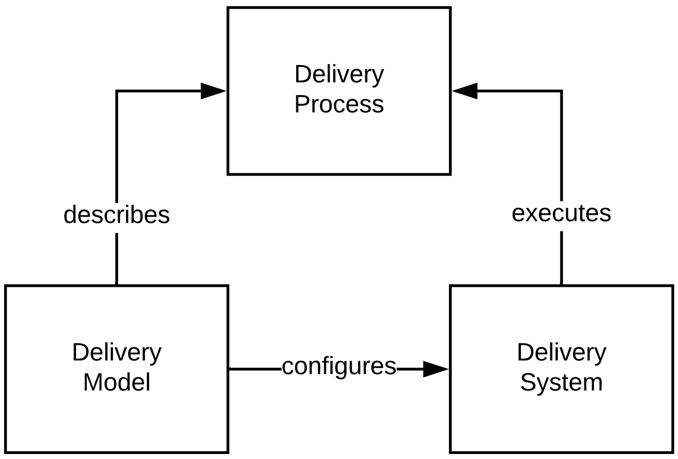


Figure 2.2.: Relationship between Delivery Model, System and Process [Dör17]

Delivery System The delivery system is a software product which is used to execute the delivery process. To be able to do so, the delivery model is used as configuration for the system.

Delivery Process The delivery process is the manifestation of the delivery model and gets executed by the delivery system.

2.3. JARVIS

This section provides an overview about the concepts and ideas behind the JARVIS continuous delivery architecture. JARVIS was developed during the work of Döring at the research group for Software Construction at RWTH Aachen University ([SWC]). Since JARVIS is foremost a reference architecture for a delivery system, a reference implementation was also developed to test the concepts. Over the course of two years, JARVIS was continuously extended with new functionality. For example the integration of a regression test framework [Nun18] or modelling delivery processes in BPMN notation [Wil18]. As previously mentioned, JARVIS is a reference architecture for a continuous delivery system. It serves as a blueprint for a real software system that utilizes the JARVIS concepts. The reference architecture aims to provide the following benefits for the user [Dör17].

Integration of heterogeneous technology To successfully develop software and solve problems, it is essential to use the right tool for the job. The delivery system must be able to support your technology decisions and allow easy transitions between different solutions.

Modularity Modularity allows the user of the delivery system to add or subtract individual features and functions as needed. This keeps the system small and allows for a flexible evolution.

Activity Abstraction Continuous delivery systems provide actions (activities) which represent the delivery process steps. To add or adapt actions, without influencing the remaining infrastructure, an abstraction is necessary. This principle goes hand in hand with the modularity requirement.

Self-Organizing The delivery system must provide support for the user and organize/optimize delivery models automatically. This reduces the knowledge needed by the user of the delivery system.

Custom PDLs The delivery system must provide different approaches to describe the delivery models. For example this can be a DSL or a even graphic modelling.

2.3.1. Architecture

To accomplish the modularity, as required in section 2.3, JARVIS utilizes a special architecture. The architecture is composed as an multi-layered architecture with five individual layers. Additionally each component of the architecture is developed self-contained. This micro-service approach allows the user to easily adapt the system with new functionality, without influencing the rest of the system. Every component is connected to the infrastructure through a central messaging-system and components in the top layer offer a variety of endpoints for external use. The bottom layer is the activity-layer. Its purpose is to house the activity-services which provide the activities that can be included in the delivery models. One layer above is the Delivery Process Management layer. All functionality needed, from a system management perspective, is located here. This layer (in combination with the infrastructure components) represents the central command unit of a JARVIS system. Following the architecture upwards, next is the access-layer. Components in this layer serve as gateways to external tooling and aggregate information, which they then also publish. The top-layer is the visualization layer. Here the aggregated information from the access-layer are visualized and eventual tooling for the user, to interact with the system, is present.

2.3.2. JARVIS Activities

The activities represent the individual steps and actions a delivery process consists of. For each step in a process an individual activity is developed in JARVIS and integrated into an activity-service. An activity-service clusters activities which belong together content-wise. Nevertheless each activity operates alone and provides it function separate. In the following the three activity types, present in JARVIS, are presented. They are also illustrated in figure 2.3.

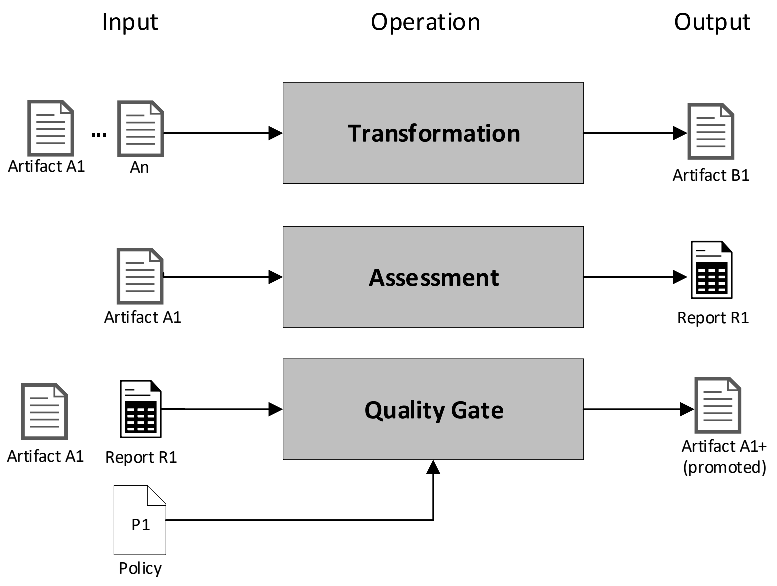


Figure 2.3.: JARVIS activity classification [SLD18]

Transformation This activity takes an input artifact (e.g. source code) and transforms it into another artifact (e.g. compiled classes). The transformation activity is the backbone activity of JARVIS.

Assessment The assessment is used to generate information about an artifact. Therefore the input is an artifact (produced by some transformation activity) and the result is a report about that artifact. An assessment could be for example the execution of unit tests, the report being the number of failed tests.

Quality Gate The quality gate is used to ensure certain policies are uphold. If not, the pipeline run is stopped. A policy can be something like a mandatory passed/fail ratio for unit-tests. Therefore the quality gate get an artifact with the corresponding report and compares this report to the applied policy. If the policy is uphold, the artifact get promoted.

2.3.3. Modelling with Smart Planner

The smart planner was introduced to enable JARVIS systems to self-organize (see 2.3). It works in combination with the orchestrator component and is responsible for supporting the user in his modelling efforts. To accomplish this, the smart planner utilizes the model abstraction present in JARVIS. The whole JARVIS infrastructure works on the so called "internal model" which represents the delivery model in a way all components understand. Additionally an external model exists, which is used by the user to encode the delivery-model. The external model is then translated into the internal model and can be processed by JARVIS. With this foundation in place, the smart planner is able to plan an optimize almost all delivery models. It takes the delivery model (in its internal representation) and detects dependencies between encoded process steps. This extraction (of the delivery process) is paired with an optimization regarding execution speed and a maximum of parallel task execution. The optimized delivery process is then executed by the orchestrator. Smart planning reduces the process knowledge the user is required to have. By declaring input and output artifacts for needed artifacts, the planner can construct a delivery process automatically.

2.4. Empirical Software Engineering

This thesis is a case study in the field of continuous delivery. Therefore the foundations of case study design are explained in this chapter. Furthermore a general overview about interviewing case study participants is given.

2.4.1. Case Studies

Following the argumentation of Runeson and Höst, research methods can be classified into four general categories [RH09]. The first type is the exploratory research method. Its purpose is to discover new insights and ideas for future work. Next is the descriptive research. Here a situation or scenario is investigated and described. This description can then be used in other research as a foundation or additional material. The third method is the explanatory research method. During explanatory research a situation is assessed and an explanation for that situation is searched and evaluated. Lastly there is the improving research. It is targeted at improving certain aspects of the studied subject. The case study, as research method, can be classified as exploratory [Fly06] and descriptive [RH09]. During the course of the thesis it is used in an exploratory way, since it is used to gather new insights into the usability of JARVIS (see 5) or continuous delivery in general (see 6). Before the case study can be executed, a plan must be made. This plan should contain at least the following elements [Rob02].

The Objective The objective is the overall goal of the study and answers the question "what to achieve" [RH09]. It is important to specify this objective very clearly to avoid ambiguity and misunderstandings during the study.

The Case The case is the research object. It is the entity under investigation and the central component of each case study.

The Theory This element specifies the frame of reference for the case study. What boundaries are there and in which context can the results be evaluated.

Research Questions These questions are a more concrete formulation of the objective. These question specify the desired knowledge the researcher tries to gain. Other than the objective, this is not seen in the greater context. A research question has to be precise and only focused on one piece of information.

Methods The methods define how the researcher wants to answer the research questions and reach the objective. For example if interviews are used to gather information.

Selection Strategy The element specified how the correct (or relevant) data is chosen and how it is processed. Is lays the foundation for reporting of the results.

After the described elements are planned, the case study can be executed. For this, Runeson and Höst splits the case study process in five individual steps [RH09]

Design In the first step the case study is designed and planned

Data Collection The data collection is planned and the collection procedure are specified

Execution In this step the case study is executed and the required data is gathered

Analysis The collected data gets analysed and structured

Reporting Data which was gained during the previous steps is prepared for presentation

The last relevant aspect in this short overview are the different types of case studies. Following Yin, a case study can be modelled in two ways. In figure 2.4 the two designs are illustrated. The first type is the holistic case study. This can be considered the classical design which most people are familiar with. The context defines the environment in which the study is executed and the boundaries to other areas. In this context a specific case is under investigation. This is then further concretized by defining the "unit of analysis" which produces the desired research data. In the embedded case study context and case are the same as before. The only difference is the use of multiple "unit of analysis" entities, to gain information.

2.4.2. Semi-Structured Interviews

As mentioned in the previous section, it is essential to chose a suitable data collection method to successfully complete a case study. One of these methods is conducting interviews with relevant test subjects and analyzing their feedback. Interviews are considered an exploratory method of research [RH09], which aligns with exploratory nature of this case study. Furthermore, it is suitable to gather stakeholder requirements [Zsu00], which is one of the main goals during this thesis (see 6). The process of conducting semi-structures interviews consists of six individual activities or phases [HA05]. These phases are as follows.

Scheduling The first activity is concerned with the scheduling of the interviews. This includes selecting the right participants and making appointments with them.

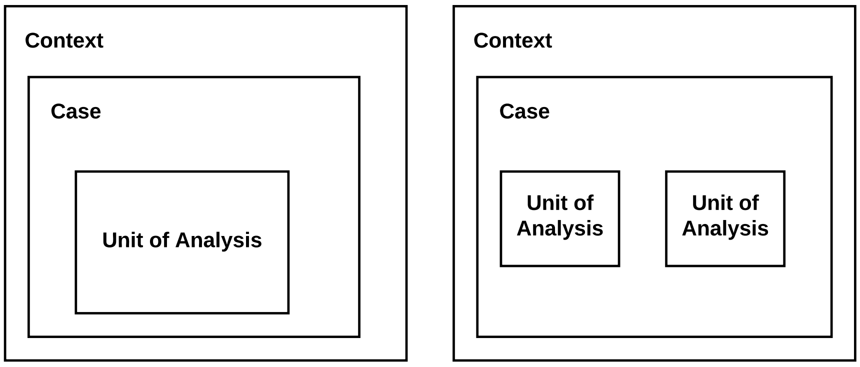


Figure 2.4.: Holistic case study (left) and embedded case study (right) [RH09]

Collecting of background information This activity can be optional and must not be executed for every interview scenario. If a scenario demands for background information about the participant (e.g. technical knowledge or CV), this is performed during this activity.

Preparing interview guides This activity is used to prepare the interview and specify which information are targeted. In some instances it can be useful to cluster the participants and design individual guides for each target group [RD03].

Discussion/Meeting During this activity, which can be before or after the interviews, the researchers discuss procedure or results of the interviews.

Summary Writing This activity is used to summarize the findings and information, gained during the interviews.

Transcribing This activity is not performed in every scenario. Since working with text is easier and faster than audio recordings, the recording is transcribed during this activity to have textual representation. This is especially beneficial in larger research groups.

Another important aspect of using interviews as data collection method, are the skills, the interviewer needs to have. To receive as many information as possible (and in good quality) three main skills need to be present in the person conducting the interviews [HA05]. Firstly the interviewer needs to encourage the participant to talk freely and make them feel comfortable. Additionally the participant needs to realize that her/his opinion matters and is taken seriously. The next skill is related to background knowledge of the researcher. It is important to have enough general knowledge about the topic to ask relevant and insightful questions [HA05]. This is strongly coupled with the ability to recognize interesting topics and follow up with them. The advantage of an interview is the ability to integrate new information as they come and not be limited to a fixed set of questions. Lastly it is important to make sure the participants know, no negative consequences will arise if they answer the questions truthfully. Here especially an anonymous approach is beneficial. Also letting the participant sign off on the final transcript can improve confidence and thereby information quality. Multiple tools can be used to conduct interviews. To close of this section the most important are explained here. One tool is an audio recorder. Taping the interview and later evaluating it is good practice because the attention of the interviewer is on the participant. Additionally the taping can be listened to on a later date if some information went missing. As mentioned above, the second tool ist the interview guide or questionnaire. It helps to stay on topic and no important question is forgotten. Finally providing visual aids during the interview can increase the value of the information because the participant has a concrete object/representation to talk about.

3. Case Study Design

Contents

3.1. Problem Statement 13
3.2. Goals and Contributions 14
3.3. Case Study Structure 15
3.4. Objectives and Research Questions 15
3.5. Case Studies 17
3.5.1. Study 1: JARVIS in a Production Environment 18
3.5.2. Study 2: CD Stakeholder and Requirements 19
3.5.3. Study 3: JARVIS Innovations 20
3.6. Summary 21

Following the general introduction in chapter 1 and a description of terms and concepts in chapter 2, this chapter outlines the objectives, this thesis aims to work on. Additionally the concepts, developed to achieve these objectives, are presented. This chapter is structured as follows. First the problem statement is given. In the next section objectives and research questions are specified. Finally the case study design is developed and explained.

3.1. Problem Statement

Since continuous delivery can offer large benefits in enterprise scenarios [Che15] the need for tailored CD towards stakeholder requirements is apparent. While the research community is highly active in improving CD as a discipline itself and developing ways to enhance CD for certain target groups (e.g. developer), the focus rarely favors the user base in its entirety. Laukkanen et al. started to tackle this issue in his paper "Stakeholder Perceptions of the Adoption of Continuous Integration - A Case Study" [LPA15]. Though this paper was concentrated on the perception aspect of CD in business environments and not general requirements in the field of CD. During the initial preparation of this thesis, no exhaustive analysis regarding general continuous delivery requirements, targeted at the whole organization as potential stakeholders, was found. Therefore executing such an analysis is one of the tasks of this thesis. The other task is evaluating the JARVIS continuous delivery system against these requirements. JARVIS was developed at the research group for Software Construction at RWTH Aachen University (SWC) and up to now, only tested in experimental scenarios. Til today, no study exists, which investigated all of JARVIS features in a productive context and compares them to actual user needs. Especially the user acceptance and general usability was not evaluated to this date. To close this information gap, evaluating JARVIS with the mentioned general CD requirements is sensible. Not only would this support the future development of JARVIS and point out eventual shortcomings of the system, a "hands on" examination of the architecture, through real world stakeholders, would also substantiate the claims made by the designers of JARVIS (see 2.3).

3.2. Goals and Contributions

Based on the formulated problems in section 3.1, there are three major goals which this thesis aims to achieve. These goals are specified and described in this section. All goals are dealt with in the context of a collaboration with an industry partner of the SWC chair. A detailed introduction of this partner and the relevant context is provided in chapter 3.5.

1. **Assess JARVIS as CD solution** During the development of JARVIS, respectively the reference architecture, a small case study was executed to verify basic concepts [Dör17]. This study was limited to only one project, which in addition was used as an example during development of JARVIS. To be able to make a more qualified statement about the usability and general abilities of JARVIS a larger and more detailed case study is needed. While the objective of JARVIS [Dör17] were more technical (e.g. evaluation of core domain and integrability of new technologies Descriptions), this study will focus on the ability to provide continuous delivery in a productive environment. The delivery solution of real world projects is modelled in JARVIS to test its functionality outside the scientific context. Furthermore the ability of JARVIS to fully replace existing CD solutions, like Jenkins [Jen], is evaluated on multiple projects with heterogeneous technologies.

2. **Identify stakeholders and requirements** As stated by Laukkanen et al. [LIL17] the empowerment of stakeholders is one of the key benefits of continuous delivery. To take advantage of this benefit, it is essential to identify all relevant stakeholder and their requirements towards CD. The second objective of this thesis is the identification of classical (e.g. developer) stakeholders towards CD in the given context. This objective is extended to investigate if the list of classical CD stakeholders is exhaustive or if the requirements of the classical stakeholders are sufficient and complete. Additionally JARVIS is evaluated against the found requirements and assessed regarding its capabilities to satisfy them.

3. **Assess JARVIS innovations** Since the initial development of the JARVIS reference architecture, additional innovative features like e.g. automated pipeline planning/optimization and CD modeling through BPMN [Wil18] were developed. Until now, these features were only tested and evaluated on dedicated test scenarios and projects. During this thesis these innovations will be evaluated in a large scale productive environment. Again, the base for this evaluation will be the stakeholders and their requirements identified during the analysis.

This section defined the goals of this thesis and the underlying challenges which led to those goals. During this thesis, a large scale stakeholder analysis will be conducted. Especially the requirements of non common stakeholders are focused. In a second step the developed JARVIS reference architecture ([Dör17]) is evaluated as a possible solution to satisfy the requirements found in the analysis. Additionally the general usability as a fully functional CD alternative is evaluated. Finally the new innovative features, introduced and made possible by the reference architecture, are evaluated in two separate contexts. The first context is the classical evaluation within the group of well-known CD stakeholders like developers and DevOps members. In addition the second context consists of more unusual stakeholder like security teams or enterprise architecture engineers.

3.3. Case Study Structure

To solve the problems listed in chapter 3.1, a suitable scientific research strategy needs to be chosen. Suitable in this context means able to produce the desired information and findings. Aligning this thesis objectives with common selection criteria for research strategies [Yin94], the case study was chosen to be a fitting method. A case study is able to generate requirements and the reasons for them, without having to control the behavioral events in the environment. This thesis understands itself as an exploratory case study in the field of continuous delivery.

To solve the challenges described in the previous sections, this thesis will be split into three different smaller case studies. Each study is dedicated to its own challenge and corresponding goals. This approach puts this work in line with the guidelines for case studies in software development, proposed by [RH09]. Like detailed in chapter 2.4.1, a case study needs an specified objective and clear boundaries. These specifications ensure the singular focus of each study regarding their respective challenge. Therefore, before the studies are presented in detail, the objectives and research questions need to be outlined. In a next step, the case and context of each study is defined. To complete the study design, the used methodology is explained.

3.4. Objectives and Research Questions

As stated by Runeson et al. [RH09], to successfully execute a case study, the definition of an overall objective in advance is necessary. This objective states the expected achievements and aim of the study. Later it is refined into research questions which are fed directly into the methodology. In this instance, there are multiple objectives present, which are derived directly from the goals of this thesis (see section 3.2).

o1 Make a qualitative assessment regarding JARVIS as substitution for established continuous delivery solutions

o2 Make a qualitative assessment regarding the usability of JARVIS on a comprehensive list on stakeholders

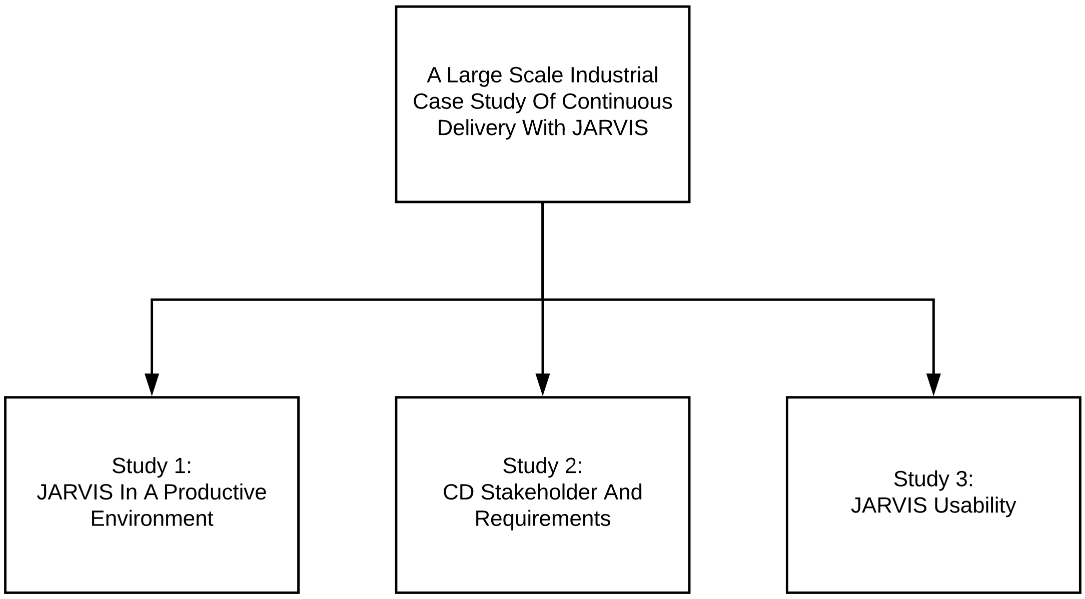

A Large Scale Industrial Case Study Of Continuous Delivery With JARVIS
Study 1: JARVIS In A Productive Environment
Study 2: CD Stakeholder And Requirements
Study 3: JARVIS Usability

Figure 3.1.: Decomposition of the thesis in sub-case studies

o3 Make a qualitative assessment if the experimental features of JARVIS solve common continuous delivery problems

After extracting the overall objectives for this thesis from the defined challenges, the concrete research questions, asked during the examination must be specified. Referring to Runeson and Höst, these questions need to satisfy two major demands. They need to be clearly and precisely formulated. Additionally they need to be relevant to the research objectives. The quality of the research questions is essential for the upcoming success of the study, since the questions are the basis for eventual interview questions [RH09] and they are used to evaluate the findings. Each research question is formulated to help achieve a specific objective. In the case of this study, this equates to three groups of questions, each assigned to one objective. The first objective **o1** is concerned with the general functionality of JARVIS and its abilities to perform as a productive continuous delivery solution. Therefore the questions are focused more on the technical aspect of the use of JARVIS in a business environment. The following research questions are under investigation for objective **o1**:

o1.R1 Is it possible to develop and integrate basic CD stages compile, test, package and deploy?

o1.R2 Is the integration of new technologies possible? Which obstacles arise from this integration?

o1.R3 Is it possible to model pipelines, extracted from real world scenarios?

o1.R4 How does JARVIS compare to existing CD solutions?

The next objective **o2** is engaged in the usability aspect of JARVIS. This time the user experience is the focused topic and how JARVIS deals with different stakeholder needs. To be able to answer that question, it is necessary to find relevant stakeholders first and work out their requirements in continuous delivery area second. After this is done, the found requirements can be evaluated against the provided experience in JARVIS. Hence the research questions for **o2** are as follows:

o2.R1 What are the stakeholder of continuous delivery?

o2.R2 What requirements do these stakeholders have regarding continuous delivery?

o2.R3 What concepts does JARVIS incorporate to deal with these requirements?

During the development of JARVIS, multiple experimental features were introduced with the goal to solve common continuous delivery problems. For example the automated optimization of pipelines. The task now is the evaluation of those features and determining if they add value to the CD domain. In accordance with that, the research questions for **o3** are listed below:

o3.R1 Can the experimental modelling features be mapped on common problems in continuous delivery?

o3.R2 Do the experimental features solve or improve on those problems?

3.5. Case Studies

Based upon the work of the last two sections, the questions to be answered during this study are now clear. The next step is putting them into a context in which they can be dealt with and also formulating a concrete case to satisfy the requirements for a formal case study [RH09]. With context and case specified, the design for the studies is then finished. Next the context for the studies is introduced.

Since the objectives of this study are praxis oriented, the investigation in a real-live context is necessary. Therefore the chosen context for this endeavor is the KISTERS AG. Founded in 1963 and now with over 500 employees the company provides leading software solutions in the energy, water and air quality market. With more than 80 active software development projects, greater than 500 pipelines and over 110000 builds since 2013, KISTERS presents itself as an ideal partner for the evaluation of a continuous delivery solution. In addition, the KISTERS AG is in an active cooperation with the

Research Group Software Construction (SWC) of the RWTH Aachen, at which this thesis is written. Varying from case to case the context is expanded to the whole organization or reduced to the energy software business unit.

After introducing the continuous delivery solution JARVIS in chapter 2 and KISTERS in this section, the context for the planned studies is specified. As a next step, the individual cases per study have to be defined and a complete study can be formulated. A case can generally be anything, which is a "contemporary phenomenon in its real-life context" [Yin94]. In this instance the cases vary between studies. To get an overview of the planned case studies and each of their goals, they are introduced on a study by study level in the following sections.

3.5.1. Study 1: JARVIS in a Production Environment

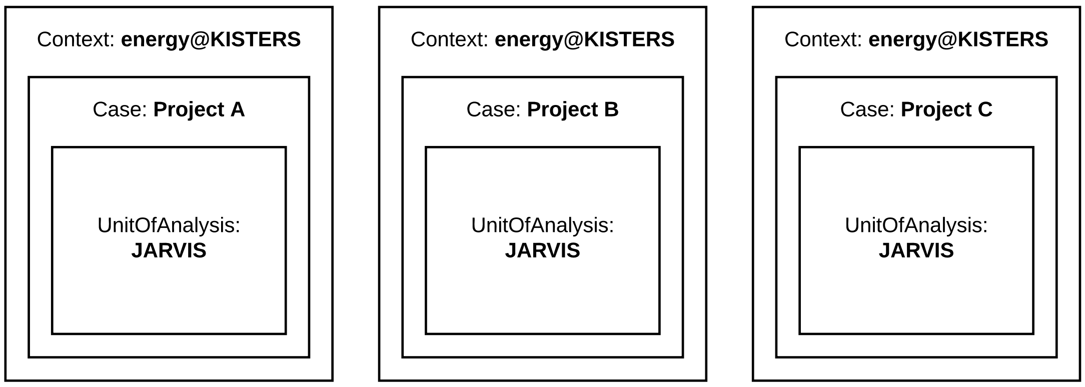

Context: **energy@KISTERS**
Case: **Project A**
UnitOfAnalysis: **JARVIS**
Context: **energy@KISTERS**
Case: **Project B**
UnitOfAnalysis: **JARVIS**
Context: **energy@KISTERS**
Case: **Project C**
UnitOfAnalysis: **JARVIS**

Figure 3.2.: First study, multiple holistic case, mini case studies

Shown in figure 3.2 the first case study is designed as an holistic [Yin94] case study which is divided into smaller sub-studies. As mentioned, the context is the energy software business unit of KISTERS, which will provide the appropriate software projects. The currently active continuous delivery solution of these projects is then modelled in JARVIS to evaluate its general CD capabilities. The overall goal of this study is the work on objective **o1** and the derived research questions **o1.R1 - o1.R4**. The projects are classified as the cases for each of the studies, since they represent the environment, the unit of analysis operates in. As unit of analysis the current implementation of JARVIS is chosen. A unit of analysis is regarded as the data source for the study [RH09] and in this instance the expected data includes integration results and upcoming integration challenges, while modelling the CD for these projects. To finalize the design of this study, a methodology needs to be defined. The first step is to make an appropriate choice regarding the projects under investigation. For this a set of requirements is needed to evaluate the candidates against. These requirements are as follows:

p.req.1 A project needs to be under active development. The active development is needed to evaluate the CD capabilities of JARVIS in a "living" environment where code is modified and requirements are changing

p.req.2 The selection of projects (or the components in a project) should use a set of heterogeneous technologies. This requirement derives from the research question **o2.R2** and is needed to evaluate technology flexibility

p.req.3 To increase the relevance for KISTERS, the architecture of the chosen project should reflect the current design decisions made by them. Thereby the abilities of JARVIS can be tested with the latest software architectures

p.req.4 To be able to compare JARVIS CD approach, there needs to be a counterpart in the current CD solution. Hence the project used in this case study needs to have a pipeline in the current KISTERS continuous delivery solution

After the project selection, the corresponding pipelines are modelled in JARVIS. During this process, JARVIS is expanded with new services and activities [Dör17]. Findings, regarding the defined research questions, are documented and subsequently evaluated. The modelling of the pipelines is performed by an external individual to keep the results unbiased.

3.5.2. Study 2: CD Stakeholder and Requirements

The second study is classified as an embedded case study [Yin94] and illustrated in 3.3. The purpose of this study is the identification of possible CD stakeholder and their requirements. Furthermore it is evaluated how JARVIS deals with these requirements. Therefore the context is defined as the whole KISTERS organization. Especially stakeholder which are not typical for CD requirements but do participate in some form in the software development process are examined. Here the case is continuous delivery itself, since the research questions **o2.R1** and **o2.R2** are the primary target for this part. The unit(s) of analysis are the individual employees of the KISTERS AG and the expected data they deliver are identified stakeholders, their CD requirements and the way JARVIS satisfies these requirements (or not).

To answer the research questions, associated with this study, the methodology consists of two parts. First there will be an investigation into the KISTERS business structure to

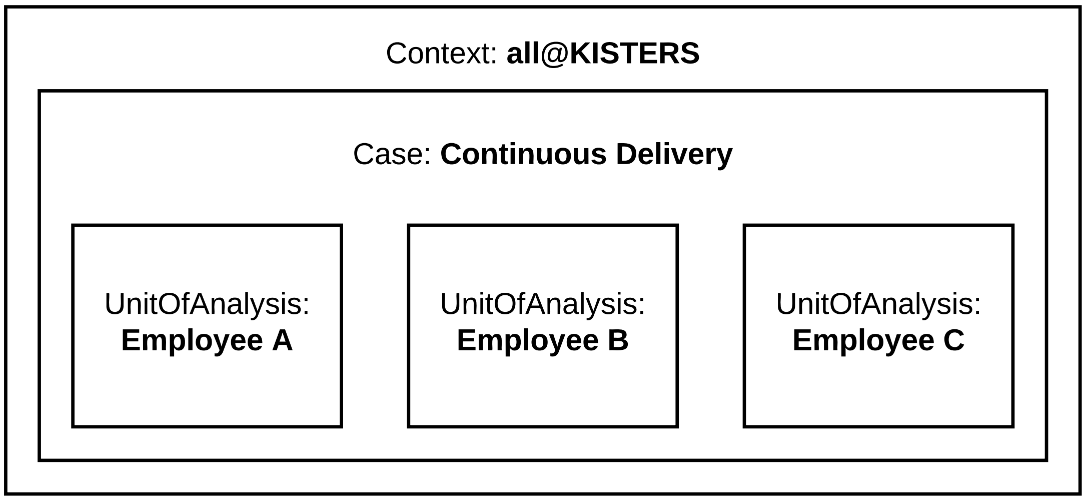

Context: **all@KISTERS**
Case: **Continuous Delivery**
UnitOfAnalysis: **Employee A**
UnitOfAnalysis: **Employee B**
UnitOfAnalysis: **Employee C**

Figure 3.3.: Second study, embedded case study on general employee base

identify departments and roles, which may have interests or requirements in continuous delivery. This results in a set of relevant stakeholders, which form the base of the following analysis. The next step is the gathering of requirements for this group of stakeholder. For this, qualitative interviews are performed. As basis for these interviews multiple questionnaires will be developed. Each specific for a certain type of stakeholder (e.g. active in software development or member in DevOps team). The interviews are recorded and afterwards transcribed. The final transcript is then given to the participant to check for correction and additional input. These results are handled and published anonymous, to ensure no consequences for the participants if existing internal problems were discussed. Finally the found requirements are compared to the functionality of JARVIS to make an qualitative statement regarding its overall usability as a continuous delivery system.

3.5.3. Study 3: JARVIS Innovations

This last study is again an embedded study and based loosely on the requirements of study 2. Figure 3.4 illustrates the structure of case study three. The key difference to the previous is the chosen case (JARVIS Modelling approaches) and of course the underlying research questions **o3.R1** and **o3.R2**. This study is centered around the innovative ways JARVIS provides, to work with delivery models. These include the process-oriented

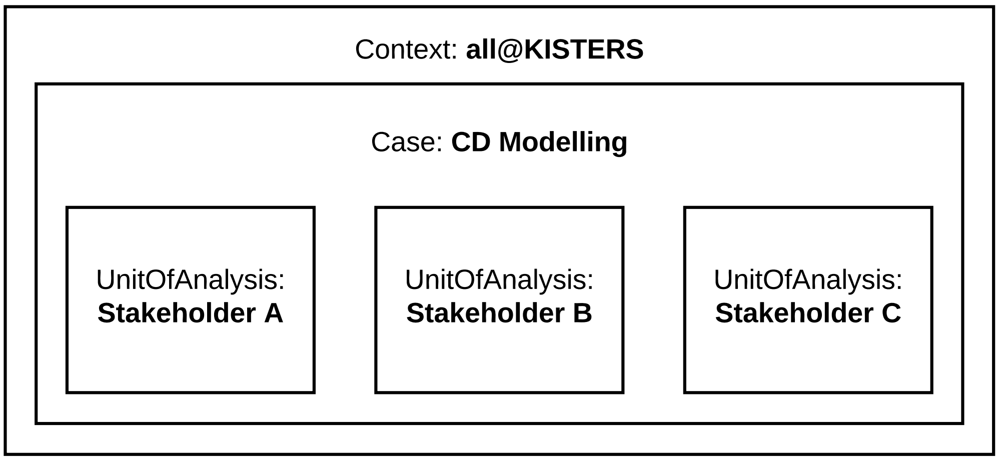

Context: **all@KISTERS**
Case: **CD Modelling**
UnitOfAnalysis: **Stakeholder A**
UnitOfAnalysis: **Stakeholder B**
UnitOfAnalysis: **Stakeholder C**

Figure 3.4.: Third study, embedded case study with stakeholder

modelling with BPMN and DSL, as well as a new modelling paradigm. This so called state-based approach was developed during this thesis and shall be also evaluated during this study. The methodology chosen was an experiment with expert CD users. These user will model an example delivery solution in all three ways. Their feedback represents the findings and results of this case study. It is used to make an qualitative assessment of the modelling capabilities of JARVIS.

3.6. Summary

This chapter provided the concrete problem statement, as well as the overall design of the superordinate case study. After stating the challenges, this thesis aims to solve (3.1), the decisions to execute three individual studies was formulated. In the first case study the capabilities of JARVIS in a productive environment are tested. This resulted in insights about the real world use of JARVIS. Additionally a statement regarding the possibility to use JARVIS as a replacement for existing delivery systems, will be possible. The results are presented in form of a review by the author of this study. For the first part of the second case study, the domain was switched to continuous delivery in general. Stakeholders for continuous delivery were gathered and their requirements evaluated. This resulted in a comprehensive list of stakeholders and their needs concerning continuous delivery. The third case study investigated a single aspect of JARVIS in detail. Since modelling is one of the key features in a delivery system, the methods present in JARVIS were evaluated. The results are given in form of a review.

4. Related Work

Contents

4.1. Case Study: Introducing Continuous Delivery of Mobile Apps in a Corporate Environment . . . 23
4.2. Stakeholder Perceptions of the Adoption of Continuous Integration . . . 24

The previous chapter introduced the objectives and the associated research goals for this thesis. This chapter describes the current state of research, related to those research questions. In the following, multiple studies in the context of continuous delivery are presented. To our knowledge, there is currently no other case study, which is directly targeted towards the research goals defined in chapter 3.4. Additionally no other evaluation of the JARVIS continuous delivery reference architecture is present. Nevertheless the presented studies can provide implicit information and insights towards the goals of this thesis.

4.1. Case Study: Introducing Continuous Delivery of Mobile Apps in a Corporate Environment

The first case study [Kle+15] was conducted by researchers from the TU Munich and about their developed release management workflow "Rugby". This process was introduced in the year 2014 and represents a lightweight process model for release management. It was designed to support SCRUM based, agile development. One of the goals of Rugby was the ability to ship an increased number of releases and handle feedback reports from customers. All of this with the focus on mobile applications like iOS and Android apps. Because of its lightweight design, it is also applicable for collaborative work in multiple teams. The overall goal of the study was the evaluation of an extended Rugby workflow. Rugby was modified for better supporting a continuous delivery approach with mobile applications. This evaluation was done in cooperation with Capgemini SE. Capgemini [Cap] is a consulting and IT service provider with multiple branches across the world. During the study eight of their mobile apps project were managed by the extended Rugby workflow and the findings documented. The chosen method to collect data was to survey eight project managers by using informal interviews. Interesting for the current case study are the implicitly formulated requirements towards continuous delivery and delivery systems during the work of [Kle+15]. This is especially important since they represent requirements in the context of mobile application development, which represents a major share of todays development effort. These requirements were extracted and are presented in the following.

Feedback Feedback in the context of this study means the feedback which is given by the customer regarding the application. This includes for example feature requests and bug reports. This kind of feedback was prioritized very highly during the case study. A continuous delivery solution must provide mechanics and concepts to help the developers cope with the increased amount of feedback. For example receiving crash reports and creating automated tickets for them.

Adaptibility It was made obvious during the study that the continuous delivery solution must be highly adaptable towards new and changing process activities.

Integrated Tooling The Rugby process has identified several essential tools which need to be integrated/available in a continuous delivery solution. This includes a ticketing system, a version control system and of course the continuous integration and delivery system.

Flexible In the context of several projects, the requirement for technology flexibility was identified. Since the applications were partly cross-platform projects, this must be supported by the continuous delivery solution (process as well as system).

Testing The need for automated testing during the continuous delivery process was identified.

Privacy Data privacy is always a concern for software developers. Therefore the continuous delivery system must be able to be hosted on premise and not in the cloud.

Distribution The development of some application was outsourced to offshore locations to reduce costs. This resulted in the need to execute different process activities in different physical location, which in turn requires the support of the continuous delivery solution.

Manual Steps It was identified that the manual execution of process steps is occasionally necessary. Hence the continuous delivery system must provide the ability to let the user execute steps manually.

Branching Support Capgemini is utilizing multiple different branches in their software development projects. A continuous delivery system should have the functions to support the work with multiple branches in one project.

4.2. Stakeholder Perceptions of the Adoption of Continuous Integration

This section introduces a case study [LPA15] which was executed by Laukkanen, Paasivaara, and Arvonen in cooperation with the communications company Ericsson [Eri].

Like with the previous case study, the focus of this work was not to gather requirements in the field of continuous delivery. The original focus was to adopt continuous integration (and delivery) in a productive scenario and investigate the perception of this adoption. Nevertheless it will be possible to extract continuous delivery requirements from the stakeholder feedback and use it as foundation for this thesis. The case of this study was a single XaaS platform with the related set of services. During the course of this case study, 27 stakeholders were interviews and asked to provide feedback, related to the introduction of continuous delivery in the project. Out of the four research questions formulated, the feedback to **RQ1 and RQ2** was used to extract relevant requirements for this work. The research questions are listed below.

RQ1 [LPA15] What actions were done to adopt continuous integration? We asked this question to understand the current perceptions of the adoption.

RQ2 [LPA15] What challenges were faced when adopting continuous integration? We asked this question to understand what kind of challenges had occurred during the adoption.

In the following the requirements, extracted from the feedback of the stakeholders, are presented.

Automated Testing It is required for the continuous delivery solution to enable automated testing for the software artifacts.

Automated Deployment It is required for the continuous delivery solution to enable automated deployment of the artifacts to different systems.

Distribution The development of this project was performed in multiple locations. A continuous delivery solution must be able to handle distributed teams and process activities.

Reduced Knowledge The development of automated tests was assessed as difficult and much knowledge is needed to perform it. A continuous delivery solution can support the user during the development of such tests.

Hidden Error During the adoption of continuous delivery the participants had problems deducting the source of error during builds. It was sometimes unclear if the problem was caused by failing test, problems with the testing framework or the delivery system itself. The continuous delivery system must provide the information about the source of an error in a clear way.

Speed The speed (especially for automated test execution) was assessed as important. A continuous delivery system must provide its functionality as fast as possible to increase productivity.

5. Case Study 1: JARVIS In A Production Environment

Contents

5.1. Study Design And Execution 27
5.2. KISTERS Reference Project 29
5.3. Findings o1.R1 31
5.3.1. Compilation of source code 31
5.3.2. Testing the artifacts 33
5.3.3. Packaging of artifacts 34
5.3.4. Deployment of artifacts 35
5.4. Findings o1.R2 35
5.5. Findings o1.R3 36
5.6. Findings o1.R4 40
5.7. Classification of activity services 41
5.8. Composition of activities 43
5.9. Discussion and Summary 45
5.9.1. Answers to Research Questions 45
5.9.2. Observations 46
5.9.3. Threads to Validity 47

As detailed in chapter 3, this thesis is divided into three individual case studies. This chapter now describes the execution and results of the first study. The overall goal was the evaluation of JARVIS regarding its general usability and performance in a productive environment. Additionally JARVIS is examined and assessed as a potential replacement continuous delivery system for the KISTERS AG. First the research goals and the study design is recapitulated in the next chapter. After that, the KISTERS reference project and its meaning for this thesis is explained. Following this, the results of the study are presented and the chapter then concludes with an overall evaluation and summary.

5.1. Study Design And Execution

Before the results are presented in the next chapter, a short recap regarding the research questions and the case study design is needed. Additionally this section describes the methods used during this study. As described in chapter 3 the research questions associated with this study are as follows:

o1.R1 Is it possible to develop and integrate basic CD stages compile, test, package and deploy?

o1.R2 Is the integration of new technologies possible? Which obstacles arise from this integration?

o1.R3 Is it possible to model pipelines, extracted from real world scenarios?

o1.R4 How does JARVIS compare to existing CD solutions?

These questions are examined in the context of a cooperation with the KISTERS AG. This partner was introduced in chapter 3. The overall design of the case study is shown in figure 5.1.

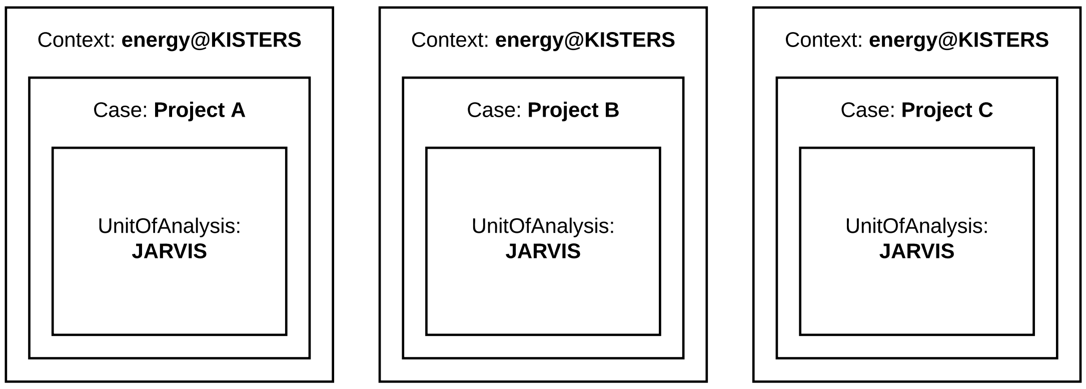


Figure 5.1.: Design of the first case study regarding JARVIS general abilities

To generate the required insights, the following approach was chosen: A subset of relevant projects was selected and their respective pipelines modelled in JARVIS. What qualifies as a relevant project is explained in chapters 3.5.1 and 5.2. The modelling is performed by an external researcher to keep the results unbiased, since no previous knowledge concerning the projects is present. The insights gained during this study can be divided into two groups. Firstly insights regarding the modelling aspect of JARVIS and secondly insights concerning the expandability (see 2.3) through development. During the modelling it was permitted to change the projects in some small aspects to accommodate for JARVIS specific requirements. But two rules were followed throughout the whole process. The resulting artifacts must be equivalent in function to the existing ones. In addition the project modifications were limited to changes in build related resources (e.g. adapting the POM.xml file for a maven project). With these boundaries in place, the next step is the selection of projects. This is described in the next section.

5.2. KISTERS Reference Project

As mentioned before, a set of projects is needed to serve as cases for the active case study (see 5.1). These projects must adhere to the four project selection criteria, defined in chapter 3.5.1. Since modelling the CD for almost 100 software projects was unfeasible during this thesis, an adequate clustering needed to be performed. Luckily the KISTERS AG utilizes a so called reference project. This project is used to document architectural designs, best practices and technology choices. It is also used as a template for future projects. Because of this properties, the reference project was the ideal candidate for the case study. By modelling the pipeline for this project, well over 60 percent of all software projects of the KISTERS energy business unit could be modelled.

Before the reference project, and its CD solution, is introduced in more detail, the project needs to be compared to the defined requirements (see 3.5.1) for project selection.

p.req.1 This requirement specifies that the project needs to be under active development. The reference project is constantly under development as the used technologies evolve and design decisions change. The goal for the reference project is always to represent the latest choices made by the company regarding software development. Hence this, the requirement is satisfied

p.req.2 This requirement demands a heterogeneous technology set for the project. The reference project incorporates almost all technology decisions made by the KISTERS AG and is therefore optimal for the current case study. The requirement is satisfied

p.req.3 This requirement focuses on the relevance for KISTERS, by choosing projects with current architecture design decisions. As mentioned already, the reference project represents the design state for many current and future projects. Therefore the requirement is satisfied

p.req.4 This requirements demands an already existing pipeline or CD approach for the selected projects. Since JARVIS needs a reference to compare to, an in place CD approach is needed. The reference projects does not only define the architectural as well as the technology decisions, it also is established as reference pipeline model for the whole business unit. This results in a detailed and comprehensive CD solution to use as a comparison. The requirement is thereby satisfied

After matching the specified requirements, it is time to present the reference project in more detail. The reference project is a web-application and utilizes a classic client-server architecture, which is shown in figure 5.2. To reduce the complexity during installation, the two components are then bundled into a single distribution package. By that, only one component needs to be rolled out. To further simplify the installation process, an installer is used to install the distribution on the target machine. The creation of the the installer is done via ChefIO [Che].

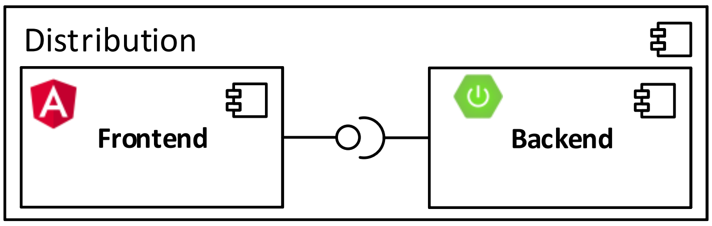


Figure 5.2.: Reference project pipelines

The client component is an Angular application, while the backend is written in Java Spring Boot. The project is build via Apache maven [Mav] incorporates unit-, integration- and system-tests. Currently the CD solution for the reference project is split into three individual pipelines. The first pipeline builds the client component as well as the distribution (the existence of the backend component is expected). Then the tests are executed and the distribution is installed on a demo-system. The second pipeline functions analogous to the first but builds the backend component. Lastly the third pipelines expects the existence of client and backend artifacts and only builds the distribution package (including tests and installation on the demo system). Each pipeline consists of multiple stages, which are composed of different activities (e.g. maven commands). On average each pipeline has around 14 stages with 1-5 individual commands per stage. The content of each step is described in the next section in more detail. Another important aspect of the CD solution is the chosen CD system itself and the way pipelines are modelled. In this case Jenkins CI [Jen] is chosen as a CD solution and a custom JobDSL was developed. The JobDSL works as follows. The user models a pipeline as text in the groovy language. For that certain skills are predefined (called traits). With these traits the whole pipeline is modelled and put into a generator. The generator then generates XML files which are used by Jenkins CI to create the pipelines. A trait can be a single command (e.g. git checkout action) or a composition of commands. Of course each command must be executable through Jenkins CI. By having this abstraction between the user and the CD system, KISTERS aims to gain multiple benefits. One benefit is the composition of multiple commands to a an entirely new one. This reduces the complexity of the models and increases the overall modelling speed. Another benefit is the reduced knowledge the user needs to know about the CD system itself. Everything the JobDSL offers is guaranteed to be valid for Jenkins CI and unnecessary functions are hidden. The last benefit is the possibility to guarantee certain governance rules while creating releases and artifacts. An example for this would be the dependency check, which is always executed when source code is getting compiled. This is encoded in the DSL as a composed trait and can not be deactivated. The full pipeline for the backend component is shown in figure 5.3. After this section introduced the reference project and establishes its relevance for this case study, the next section presents the findings related to the first research question **o1.R1**.

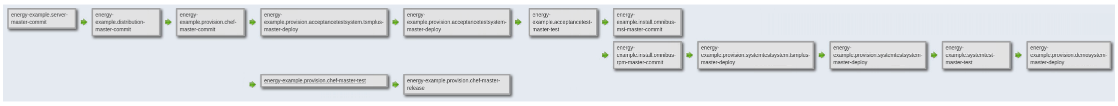

energy-example.server-master-commit
energy-example.distribution-master-commit
energy-example.provision.chef-master-commit
energy-example.provision.acceptancetestsystem.tsmplus-master-deploy
energy-example.provision.acceptancetestsystem-master-deploy
energy-example.acceptancetest-master-test
energy-example.install.omnibus-msi-master-commit
energy-example.install.omnibus-rpm-master-commit
energy-example.provision.systemtestsystem.tsmplus-master-deploy
energy-example.provision.systemtestsystem-master-deploy
energy-example.systemtest-master-test
energy-example.provision.demosystem-master-deploy
energy-example.provision.chef-master-test
energy-example.provision.chef-master-release

Figure 5.3.: Pipeline for the backend component of the reference project

5.3. Findings o1.R1

This section is about the first research question and the basic functionality of JARVIS. The research question specifies four basic functions JARVIS needs to provide to be able to be used in the productive context of the KISTERS AG. In the next sections the current manifestations of those functions in Jenkins CI is detailed and the corresponding implementation in JARVIS is presented. By that, the research question can be answered and the resulting activities can be used as a basis for the next question **o1.R2**.

5.3.1. Compilation of source code

The commit-stage is the first stage of each reference project pipeline. It performs a git-checkout and multiple maven commands to create an initial artifact. As the name suggests, the stage is triggered by a commit on the respective project repository. Next the commands are explained one by one and the realization in JARVIS is presented.

git:checkout This command is used to checkout the source code and get the workspace ready. The tooling used for this is the git version control system. Since this is a very basic function for a CD system, an activity providing the git-checkout command was already included in the JARVIS reference architecture. Nevertheless one modification was necessary since the reference project has a complex versioning scheme. The existing checkout command was expanded in regards to publishing more git information each time a checkout was performed. One of those information was the commit-ID and the shorten git commit hash. These information were later used for artifact naming and resolving and are explained in more detail in the upcoming command descriptions.

cd-helper:parse version The cd-helper plugin is a custom maven plugin, KISTERS developed to fullfil their versioning requirements. Used with the parse version goal, it scans the (maven-)project regarding its versions. It then increases these versions to the next major release version. This next version includes the current time stamp,

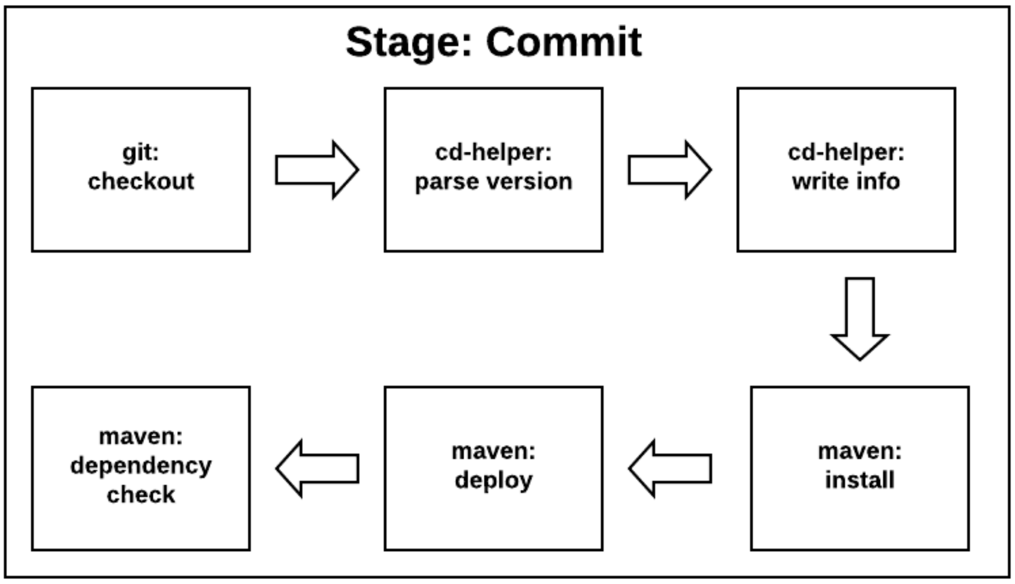


Figure 5.4.: The commit stage of the reference project

as well as the current git commit-hash. The git-hash (and timestamp) are provided by Jenkins CI through a custom groovy script which is executed before the stage starts. Since JARVIS does not provide these information, as previously mentioned, the git-checkout activity was modified to do so. To realize the cd-helper function a new service with the corresponding activity needed to be developed. The activity was called "VersioningCmd" and the service "maven-service". As input artifacts, this activity received the checkout repository and the git meta-information.

cd-helper:write info Again this command is provided by the custom KISTERS maven plugin. Its task is to write the calculated artifact version (and other meta information like the artifact name) into a property file, which in turn is injected into the build environment of Jenkins CI. From then on, these information is available to other commands via environment variables. In JARVIS this function was included in a new activity that received the workspace as input artifact and produced the property file as output. The activity was called "ProjectInfoCmd" and located in the maven-service.

maven:install This command is a basic maven goal to compile the project sources. In JARVIS a new transformation activity with the name "install" was developed. Since this is again a maven activity is was put into the maven-service. For the naming and versioning of the artifacts, the information from the "ProjectInfoCmd" are used.

maven:deploy Analogous to the last command this is a maven goal. Its purpose it to publish the build artifact to the nexus for subsequent stages to retrieve (as well as to archive the artifact for documentation purposes). To provide this functionality an activity named "deploy" was developed in the maven service.

maven:dependency check The dependency-check is a mandatory command, every time source get compiled at the KISTERS AG. It checks the external dependencies defined in the project POM.xml for eventually existing security vulnerabilities and exploits. This check is mandatory for KISTERS to keep their ISO certification. Hence it must be executed and can not be declared optional. Because of the importance of this command, an implementation in JARVIS must be possible. For this an assessment activity was developed and located in the maven-service. Like the maven activities before, the activity was designed to be able to execute terminal commands on the host system.

5.3.2. Testing the artifacts

Testing is an important aspect in software development. Therefore the KISTERS reference pipeline includes the stages to do so. Because the testing is spread over multiple stages and there are many very similar actions with only minor differences, not all actions are listed here. In the following the most relevant testing commands and their counterpart in JARVIS are explained.

unit-tests Java unit tests are not modelled explicitly in the reference pipeline. They are executed together with the maven install command. In the JARVIS environment this was implemented analogously.

acceptance-tests Before the acceptance tests can be executed, the current version of the reference project needs to be installed on a dedicated system. The command necessary for this is introduced in section 5.3.4. The testing command itself is performed via the maven verify goal. In JARVIS this is implemented as an assessment in the maven-service.

chef-unit-tests The application is packaged and shipped in form of an installer. This is created with the help of the ChefIO technology, which results in the development of a chef cookbook. To ensure the correct functionality this cookbooks testing is recommended. For that, unit-tests were implemented and are executed during the pipeline run. In Jenkins this was realized with a new service (called chef-service) and a new activity. Like the activities providing maven functions, this was designed as an activity that executes chef commands in the runtime-environment.

chef-integration-test Next to unit tests, the chef cookbook is tested via integration tests. Like the acceptance-tests from before, this requires an external system where the cookbook is deployed to. The deployment command in general is explained in 5.3.4. Analogous to the chef-unit tests another assessment activity was developed in JARVIS that received the cookbook as input and published the number of failing tests.

5.3.3. Packaging of artifacts

As mentioned in the previous sections, the backend and client are packaged into a distribution package and then packaged again in an installer. The distribution packaging is again realized by maven and the installer is created with the help of the ChefIO technology. To model the reference project pipeline the following commands were used.

maven:install As detailed before, the distribution package consists of backend and client component. It is modelled as maven project with an dedicated POM.xml file. Because of this, the maven command, to build the distribution, is very similar to the maven install command above. Only difference is the use of the meta-information that is produced by the cd-helper plugin. The information is used to resolve the correct client and backend artifacts from the nexus repository and name the distribution artifact correctly. In JARVIS the realization is also done in the same fashion. A transformation activity was implemented, which uses the property file from the "ProjectInfoCmd" to receive the required information. Since this is again a maven based activity, it was integrated into the maven-service.

chef-foodcritic Since the Chef cookbook is essentially infrastructure as code, certain coding guideline are useful. The KISTERS AG chose to govern these by using the Chef "foodcritic" function before releasing the cookbook itself. It checks if certain style-rules are followed. The JARVIS counterpart is an activity with the name "chef-foodcritic". Because the activity is promoting an artifact (in this case the chef cookbook), it was modelled as an assessment. The location of this command is the chef-service.

chef-rubocop This command is essentially the same as the previous foodcritic command. The major difference are the rules and styles that are checked. As above, a JARVIS activity was developed and introduced into the chef-service (again the assessment type was chosen).

rpm/msi-installer The rollout of the reference project is done by an installer, which is created with ChefIO. For this, the rpm/msi-installer commands are present in the reference pipeline. These commands execute a custom bash script (with ChefIO omnibus commands) on a dedicated external system which represents the target environment for the installer. This was mirrored in JARVIS with an transformation activity. To be able to connect to the remote system, the chef-service was refactored to an agent based services. The agent was run on the external system, while the chef-service provided the interface for the JARVIS infrastructure. The design pattern for this kind of service is presented in more detail in section 5.7.

5.3.4. Deployment of artifacts

The last set of actions that are required by the research question **o1.R1** are deployment actions. In the current reference pipeline these were used to transfer the final installer-artifact to the target system and execute the installation process. In the reference pipeline this is done by executing a generic bash script through Jenkins. Since there is no significant difference between deploying to the acceptance-test system or to the demo system, only the deployment to the acceptance system is presented in detail.

deployment-acceptance As already mentioned, the application needs to be installed on a remote system to execute the acceptance tests. In Jenkins CI this is realized via a custom script that is executed in a generic bash step. Since the script is tailor made for the remote system and the reference project, replacing it by an activity was not sensible. Instead a new service was introduced and named "deployment-service". Again an activity was developed, but this time with a more generic approach. The activity takes the already existing bash-script and executes it in the correct environment. By that, already existing logic and knowledge was reduces and the implementation time decreased significantly. The activity was classified as an transformation.

In this section the findings regarding the first research question were introduced. All required actions were successfully implemented, integrated in JARVIS and performed as expected. All in all 21 new activities and 4 new services were developed (see table 5.1) to represent the reference project pipeline. The development process was straightforward but revealed multiple aspects of JARVIS that need further exploration. One of these aspects is the identification of service classes. All developed activities can be divided into three classes and corresponding design patterns for these classes were introduced. These patterns are further explored in section 5.7. To sum up this section, the research question **o1.R1** can be considered answered. JARVIS can provide all necessary activities to model a productive pipeline. In the next section the integration of new technologies is explored in detail.

5.4. Findings o1.R2

The second research question is concerned with the integration of new technologies in JARVIS. Especially eventually occurring problems or challenges are focused. The findings regarding this research question were produced during the development of new activities in the last section. In sum, three new major technologies needed to be integrated, to model the pipeline for the reference project. The first new technology was Apache maven. It is used to compile the Java source code and execute plugins for versioning and code quality. During the creation of JARVIS, a maven-service with basic activities was already developed and used. Unfortunately this service could not be utilized because of compatibility problems. It was using an integrated library to provide maven commands in JARVIS. Because it was integrated, the Java version used, was the same as the Java version the maven-service was executed in. In this instance this was Oracle Java 8. Since the reference project was developed in Oracle Java 11 (and used version 11 features), a compilation was not possible. One solution to this problem would have been an upgrade of the maven-service to version 11. This was unfeasible, because the whole JARVIS reference implementation would have had to be updated. To reduce time and effort another solution was chosen. The maven service was redeveloped as an environment service (see section 5.7 for service classes). The new maven-service can now use the Java version of the execution environment and compile the reference project successfully. The consequences following this problem were the identification of a new best practice:

Always separate the technology of the JARVIS activity-service from the artifact technology. It should always be possible to change/switch the artifact version/technology without rewriting the whole service.

The second technology integrated was ChefIO to create the installer. This technology was not yet existing in the JARVIS eco-system. Therefore a new service was developed and the required activities integrated. With this, another type of service was needed. Because the installer needs multiple very specific environments to be build in, the utilization of external systems was necessary. Hence a service was developed as an agent-based service. More details about this type is provided in section 5.7. The last new technology was not a technology in the classical sense. The deployment of artifacts to test- and demo-systems was not yet present in JARVIS. Again a new service was created, which uses SSH access to the target systems to execute scripts on those. These scripts then provisioned the machine. Because the logic was encapsulated in the scripts, the size of the activity was very small and the reusability of the service is very high.

Except the above mentioned problems, the integration of new technologies was possible without major complications. Thanks to the micro-service architecture, the development was only limited to one service at a time and no compatibility problems between services occurred. In general the answer to the research question **o1.R2** is as follows. The technology integration concept of JARVIS is suitable for productive environments if certain rules (see description above) are adhered to. By combining the findings from **o1.R1 and o1.R2**, the prerequisite for the next research question is present. After all necessary technologies and activities were introduced, the next relevant aspect is pipeline modelling. This investigation is performed in the next section.

5.5. Findings o1.R3

After the basic building blocks for the reference pipeline were developed and integrated into JARVIS, the next step is modelling the pipeline and answering research question **o1.R3**. As a basis for this evaluation, a more concrete definition of the research question is needed. Hereinafter, the criteria, which are under examination during this section, are defined. They are directly derived from requirements of the reference project towards the current CD solution (in this case Jenkins CI).

c1:Artifact Versioning The current pipeline utilizes a complex versioning (and naming) scheme. This was already addressed in section 5.3 during the implementation of new activities. The versioning is used to provide artifacts for different pipelines and stages. Artifacts are not transferred directly between those, but resolved by an unique version identifier that is calculated every time the pipeline is executed. Since this is also a pipeline modelling concern, an investigation regarding JARVIS abilities to provide this function is needed.

c2:Knowledge Needed This aspect is concerned with the knowledge needed, to model the pipeline. This includes knowledge about the CD system, as well as process and implementation knowledge. Among other, the following questions will be answered. Is it possible to model the reference project with the same (less/more) amount of knowledge, compared to the original pipeline? Additionally, the modelling support by the CD system is compared between the original and JARVIS.

c3:Modelling Support After discussing the general knowledge needed to model the delivery process, this criteria deals with the support through the delivery system itself. This includes functions which reduce the process knowledge and overall helps to create the model for the reference project.

c4:Model Size An important aspect, when modelling pipelines, is the maintainability and complexity of their underlying delivery models. This is directly influenced by the size of the model itself. Hence the overall expressiveness for JARVIS models needs to be accessed and put into context. In this instance this is done by comparing lines of code (loc) between the JobDSL model and the new JARVIS model.

c5:Completeness As described in section 2.3, the delivery model configures the delivery system. This is done through a modelling language or other concept. In the following the modelling concept of JARVIS is evaluated regarding its functional completeness. Can the building blocks, developed during section 5.3, be used to produce the desired artifacts? Is it possible to model required dependencies between steps and stages?

c6:Versioning of models In the current reference project CD solution, the models are encoded in a custom JobDSL developed by the KISTERS AG. These JobDSL-files are located in the project repositories and are under git versioning control. This is done to ensure traceability of changes and allow eventual rollbacks. Since this is an essential requirement in the KISTERS AG environment, this function is also investigated in the JARVIS representation of the reference project pipeline.

In the next part, the refined research questions are investigated and answered in the context of JARVIS and this study. These findings were generated during the modelling process of the KISTERS reference project CD solution.

c1 In the original pipeline, the versioning was needed to address artifacts in the nexus and transfer them between building steps. Additionally the versioning was used for naming the releases. In JARVIS the versioning could be modelled as an individual activity in the beginning of the pipeline. This activity calculated the required information and provided those as an artifact to the rest of the pipeline. The remaining activities can then use this information to name the artifacts accordingly. The addressing and resolving functionality was not needed in JARVIS. In JARVIS each activity declares its output artifacts and these can be referenced by other activities directly through the model. Furthermore, the whole reference project pipeline was modelled in one pipeline and not, as originally done, in three individual pipelines. By that, no artifacts needed to be transferred between pipelines. All in all JARVIS was able to model the versioning scheme and provide the required functionality.

c2 This question investigates the knowledge needed, to encode the delivery model. In the original pipeline, the user needs to know which operations he wants to perform and additionally the sequence of those. Since the JobDSL is used, each operation can be configured as needed for the scenario. Furthermore, every command in the JobDSL encodes multiple Jenkins CI actions. For example the commit command encapsulates the maven compile and the dependency-check command. The user does not know what is included in a JobDSL command. Only the predefined composition commands need to be known. In JARVIS this was not the case. Because no composition or template mechanism exists, all actions need to be known by the user to create the model. A possible solution for this problem is presented in section 5.8. Due to the existence of the smart planner, the user was not expected to know about the command sequence itself. In the next criteria the smart planning function is explored in more detail.

c3 As hinted in **c2**, the smart planner was used to model the reference project pipeline in JARVIS. It can resolve step dependencies on its own and deduce the correct sequence of activities. The user only needs to specify the activities which should be executed and the command graph is created by the system. In theory this feature would thereby significantly reduce the knowledge needed by the user. Unfortunately the smart planning is flawed with a conceptual problem. The current implementation is only able to function on models where no activity is used twice. Since the reference project pipeline reuses multiple activities (e.g. git checkout activities), the smart planner was not able to plan the pipeline automatically. Why the smart planning is not able to work with reused activities and how this can be avoided is explained in chapter 7.4. Because the smart planning was not working as intended, the support of the CD system while modelling the pipeline was very limited.

c4 To keep the complexity low and the maintainability high, a small pipeline model is beneficial. While modelling the reference project pipeline, the JARVIS model was 357 lines of code long. Compared with the original 145 lines of code JobDSL file for Jenkins, the discrepancy is significant. This is mainly due to the lacking template mechanism in JARVIS. One command in the JobDSL can encode unlimited Jenkins actions with very few lines of code. This limitation in JARVIS was already discussed in **c2** and a concept to solve this is presented in section 5.8. In the current state, the JARVIS models are, compared with existing solutions, very verbose.

c5 While the last criteria was concerned with the pure size of the JARVIS model, now the overall expressiveness of is under investigation. For the most part, all activities needed, could be modelled and connected into a pipeline, which produces the desired artifacts. Nevertheless, two problems were made obvious during modelling. On principle, JARVIS expects an artifact flow between activities. Dependencies between activities without artifacts and only based on timing are not considered. This resulted in a workaround to model time dependent steps. Dummy artifacts needed to be introduced and modelled as input dependencies for the dependant activity. These dummy artifacts have no function other than allowing to execute two activities in a defined sequence. The second problem originated while modelling quality gates. It was not possible to model stages that consists only of quality gates. This again was caused by the missing artifact flow between the quality gates and the fact that JARVIS automatically integrates quality gates in the pipeline. It was not possible for JARVIS to determine the correct position in the activity graph without artifacts between the quality gates. To solve this problem the modelling was adapted. Quality gates had to be integrated into stages with transformation activities, which enabled the artifact flow.

c6 JARVIS utilizes an abstraction between internal and external model. Because of this, it is possible to model pipelines in manifold ways. In the reference architecture, modelling is possible in declarative text form (YAML files with custom DSL). This enables the possibility to put the models under version control and enable the benefits accompanied with this (e.g. traceability).

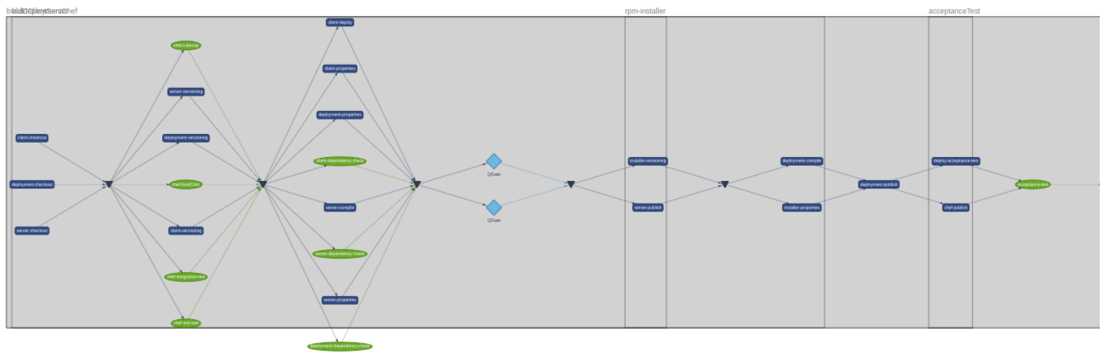


Figure 5.5.: KISTERS reference project pipeline in JARVIS

In this section, the findings related to research question **o1.R3** are presented. All in all the modelling of the reference pipeline exposed some design oversights in the JARVIS modelling concept. Mostly due to the missing template mechanism and the smart planner problems. Despite these difficulties, it was possible to model the reference project pipeline and produce a functional equivalent artifact in JARVIS (equivalent to the original pipeline artifact). In the next section the research question **o1.R4** compares JARVIS with Jenkins CI.

5.6. Findings o1.R4

The last research question closes the first case study by drawing a comparison between JARVIS and an established CD system. In this case, Jenkins CI is chosen, since it is the delivery system for the KISTERS reference project. As handled with research question **o1.R3**, the current question needs to be further specified. The following aspects will be compared.

Performance Speed and performance are key characteristics for a delivery system. Unfortunately comparing two different systems with regard to these traits is not trivial. Not only does the modelling of the pipelines influence the speed, the machines used to host the system do as well. Because of this, a fair comparison between Jenkins CI and JARVIS, regarding build and run times, would be very hard to accomplish, if not impossible. Nevertheless the two solutions can be compared regarding aspects that enable them to increase their performance and achieve better build speeds. Jenkins CI, as well as JARVIS, do offer mechanisms to scale their system horizontally. In Jenkins CI additional build slaves can be added to increase the number of parallel pipeline runs. In JARVIS, a similar feature is present. Instead of adding general purpose capacity, JARVIS lets you scale up individual activities. This is achieved through deploying a second instance of the corresponding activity-service. By that, the distribution of resources is more granular and targeted. Another aspect, influencing performance, is the pipeline modelling. Especially the degree of tasks executed in parallel can decrease the runtime of a pipeline greatly. After fixing the problem, described in section 5.5, the smart planner was able to plan and optimize the reference project pipeline. This resulted in a highly optimized pipeline which utilizes a high degree of parallel activities. Jenkins CI on the other hand does not provide this kind of optimization function. Therefore the original pipeline relied more on sequential steps. The modelled reference project pipeline is shown in figure 5.5. In summary, the performance aspects of JARVIS are as good, if not better, than those of the compared CD system. Especially the automated optimization feature for pipelines, provides hugh benefits.

Information Needs The display of information is important for most software systems. This also holds true for continuous delivery systems. In the following, the information provided by Jenkins CI and JARVIS is compared and the information needs satisfied, evaluated. Jenkins CI does offer the user a wide range of information, including a graphical representation of the pipeline and build trends. Additionally, the pipelines can be modified via GUI. In JARVIS the pipelines are displayed as an activity graph. Furthermore, statistics for each pipeline are calculated and presented. Modifying or creating pipelines is currently only possible by directly working with the model file. A current shortcoming for JARVIS is the presentation of test results. While Jenkins CI, thanks to external plugins, displays test results in a very comprehensive way, the visualization in JARVIS is reduced only to the number of passed/failed tests. No graphical aid (like time graph or bar chart) is used to help the user understand the information more easily. One advantage of JARVIS, regarding the display of information, is again the underlying micro-service architecture. Here especially the view-service, which collects and published all generated information of the infrastructure. By leveraging this fact, adding new information to the UI component is as simple as developing a new REST-API. The operator has full control which information should be published and provided to the user.

During the work on research question **o1.R1-o1.R4** two findings were discovered, which resulted in a further investigation during this case study. These findings, as well as the developed concepts, are presented in the next sections.

5.7. Classification of activity services

As discovered in section 5.3, during the development of new activities, many activities/services function in a similar way. Hereby three classes of activity-services were identified. To reduce the implementation effort and increase the overall quality, design patterns [FF06] for these three types were developed. In this section, the service types are detailed and the underlying design pattern presented.

The first identified service type is named "remote-service". Its purpose is to provide activities with the help of one or multiple external system(s). As shown in figure 5.6, the service is split into two sub-services, which are located on different machines. The activity-service itself acts as the interface to the JARVIS instance and realizes JARVIS activities by calling an external agent to execute this action. The response of the agent is then translated back into an artifact and published by the activity-service as result. For the user, as well as for other infrastructure components, the use of the agent is transparent. Only the activity-service itself knows about the agents existence and interfaces. This type of service is needed, to provide activities like the msi-installer (see 5.3.3), where the specific execution environment is needed and can not be integrated into JARVIS. Additionally, this pattern can be used to outsource long running and work intensive tasks to dedicated external machines. This is useful in case the system, running the JARVIS instance, has some kind of resource limitations. Furthermore a strong technology decoupling is inherit to this service type. The agent can be implemented (and use) any technologies available, without having concerns regarding an integration into the JARVIS infrastructure. One downside of this service type is the increased implementation effort and the higher complexity. By essentially developing two individual applications, the coding and maintenance effort is higher, than any other service type. Furthermore the complexity is increased by orchestrating the service-agent communication. Because of these drawbacks, the use of this service type is only advised in situations where the former mentioned use cases are present. It is not a general purpose activity-service design for JARVIS.

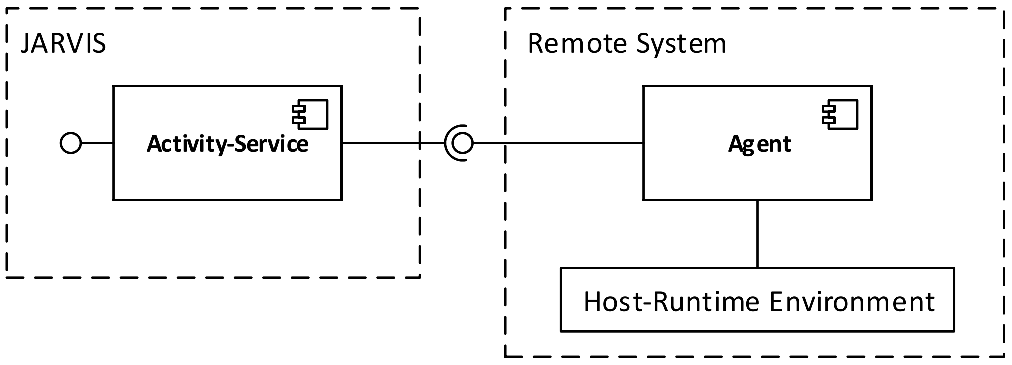


Figure 5.6.: Architecture remote-activity service

The second type of service is designated as one of two general purpose service designs. It is called "library-service" and the architecture is presented in figure 5.7. This service provides JARVIS activities by utilizing an external library, which is integrated into the service itself. An example for such a service would be the git-service, included in the JARVIS reference implementation. It offers its functionality through the Eclipse JGit [JGi] library, that was integrated into the spring service. The benefits of using such a service design are as follows. Developing a library-service is on average the least amount of work, since much functionality is outsourced to external libraries. This also effects the overall maintenance effort for the service, because there is less code to maintain. On the other hand, this design makes the service very dependant to an external source, which can not always be controlled/influenced. If the development of the library goes in an unwanted directions, or is canceled all together, major refactoring is needed. Also fixing bugs or security vulnerabilities may not be possible through the JARVIS operator himself. The use of this service design is also limited to the existence of a suitable library to begin with. If no library exists, in a needed technology, this service-type can not be used.

The last service type is called "shell-service" and used as the second general purpose service design. An architecture sketch is displayed in figure 5.8. As shown, the service leverages the host environment it runs in. For this, a shell executor was developed, which can be used to execute arbitrary bash commands on the the host and receive the output. This type of service generates moderate development effort and complexity. An example would be the maven service used to realize the KISTERS reference project pipeline. It uses the maven environment of the docker-container, the service is shipped with. One reason behind this decision was the incompatibility between the Apache Maven library and the JARVIS Java version (explained in section 5.4). In general, the shell-service is

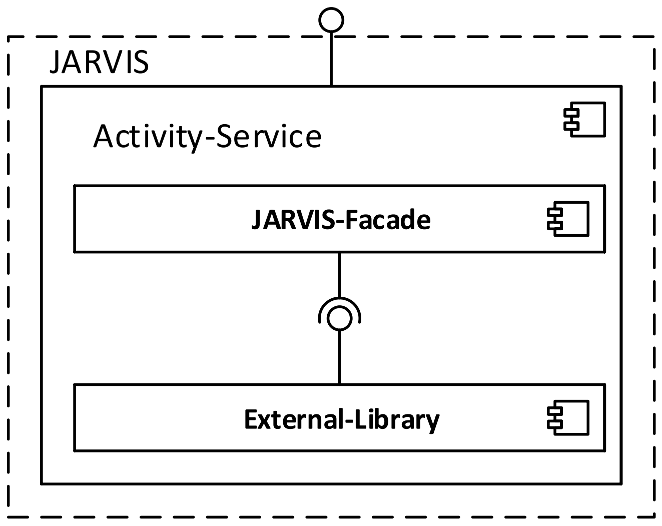


Figure 5.7.: Architecture library-activity service

a service type that can be used if no library exists or many functions for a command line tool are needed to realize the activity. The drawback of this design though is the unstructured access to the host shell. No format is set for the commands or results from the shell. Additionally the user has to know how the command line tool works, without for example IDE support.

The goal of defining the described service types and their architectures, is to make the development of activity-services easier, faster and more structured. To really gain the benefit of this classification, the next step should be to develop code-templates for the respective services. By that, the improvements made during one iteration are channeled into the next version of services and hopefully reduces the failure rate and increases overall quality. Unfortunately developing these templates was out of the scope of this case study. But since it is a promising improvement for the JARVIS eco-system, a future realization is probably. In the next section a concept for the composition of activities is presented and explained.

5.8. Composition of activities

While modelling the reference project pipeline in JARVIS, the need for composed activities was made obvious. Many build steps consist of sub-steps, which are also reused in other composed activities. The main benefit of allowing the composition of activities are as follows. Firstly, the model size can be reduced significantly, since one command block in the model encodes multiple sub-commands. In the case of the reference project modelling

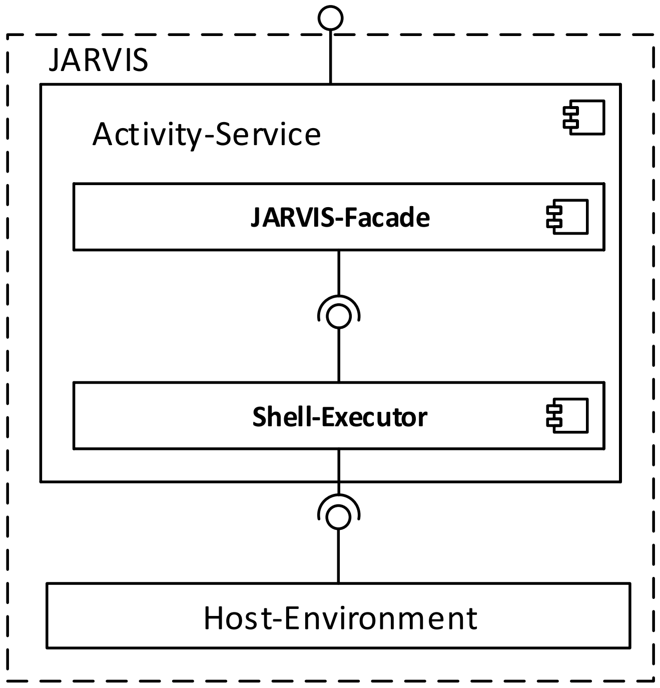


Figure 5.8.: Architecture shell-activity service

the commit stage resulted in six commands being executed (see section 5.3.1). Another benefit is the possibility to enforce certain command combinations. In the mentioned example this would be the rule to always execute the dependency-check command if code gets compiled. As mentioned, JARVIS does not provide support for these composed activities. To that end, a concept was developed during this thesis, which specifies the integration of this mechanism in the JARVIS infrastructure.

There are two possible ways to integrate the composition of activities into JARVIS. The first approach would be to encode the logic into one standard activity and make this available to the user. Unfortunately there are multiple drawbacks with this approach. If the whole logic was encoded in a single activity, it would be impossible for the Smart-Planner function to plan the sub-activities. The activity could only be planned as a whole unit and no performance optimization could be applied. Another problem is the reuse of sub-activities. As shown, during the reference project modelling, the sub-activities are reused in multiple compositions. This would not be possible if all of them are encoded as one big activity. Resulting from these drawbacks, the second approach was developed and evaluated. This approach specifies a new activity type which contains only links to other activities. In that case the Smart Planner recognizes the composition-activity and resolves the contained sub-activities. Afterwards the activity-graph can be planned and optimized. Since all sub-activities are in their own activities, they can be reused by different composition-activities. The next step would be to implement the above concept and test it in a productive environment. Unfortunately the implementation of new features was out of the scope of this case study.

5.9. Discussion and Summary

This section is dedicated to summarize and discuss the gathered findings from the first case study. It should also provide an overview about the made observations during the study.

5.9.1. Answers to Research Questions

The first research question is concerned with the basic functionality of the JARVIS reference architecture. To prove the functionality, JARVIS was used to model the continuous delivery pipeline of the KISTERS reference project. To achieve that, the activities in table 5.1 were developed and integrated into JARVIS. As you can see from the table, all required actions, specified in the research question **o1.R1**, are present. Hence **o1.R1** can be answered positively. It is possible to develop and integrate basic continuous delivery process steps in JARVIS.

The next research question deals with the integration of new technologies into a JARVIS based system. Again this was investigated during the modelling of the reference project continuous delivery solution. Three new technologies were introduced into the system and subsequently used in the delivery process. These technologies were Apache Maven, ChefIO and remote deployment to external systems. All three were successfully integrated and provided their functionality. Therefore the answer to research question **o1.R2** is again positive. It is possible to integrate new technologies into JARVIS and use them in a productive scenario.

The third research question examines the ability to model complex pipelines in a JARVIS system. The KISTERS reference project utilizes a delivery pipeline with over 30 individual process steps, including unit-testing, system-testing, integration-testing and deployment to test- and demo-systems. This pipeline was completely modelled in JARVIS. After executing the JARVIS representation of the pipeline, the resulting artifact was compared to the original and no differences could be found. Hereby **o1.R3** is answered again in a positive way. It is possible to model real world continuous delivery scenarios successfully in JARVIS. Some observations made during the process are discussed in the next section.

The last research questions evaluates the ability for a JARVIS based system to replace an existing continuous delivery system (in this case Jenkins). JARVIS in its current state is not suitable for productive use. While the foundations and basic concepts are valid, a lot of functionality is still missing. This is mainly because JARVIS is still undergoing active development and an ongoing research project. Especially regarding convenience and satisfied information needs, more functions are needed. But no feature (present in Jenkins) was found, that could not be integrated in JARVIS without violating the core concepts and ideas. Based on this observation the answer to **o1.R4** is as follows. JARVIS in its current state can not replace an existing productive continuous delivery system. Nevertheless with some additionally development the potential to do so is definitely present.

Service	Activity	Type	Description
Maven-Service	Clean	Transformation	Cleans the workspace of created artifacts
	Compile	Transformation	Compiles the source code
	Deploy	Transformation	Deploys the server to the repository
	DeployClient	Transformation	Deploys the client to the repository
	InstallerInfo	Transformation	Generates the installer meta-information
	OmnibusInfo	Transformation	Generates the omnibus meta-information
	Package	Transformation	Packages the compiled artifact
	projectInfo	Transformation	Generates project meta information
	Verify	Assessment	Executes unit tests for the artifact
	Versioning	Transformation	Increases the version of the artifact
	VersioningRpm	Transformation	Increases the version of the installer
	DependencyCheck	Assessment	Checks for dependency vulnerabilities
Deploy-Service	AcceptanceDeploy	Transformation	Deploy the build artifact to the acceptance system
Chef-Service	FoodCritic	Assessment	Checks Chef cookbook for style errors
	RuboCop	Assessment	Checks the ruby elements of the cookbook
	ChefSpec	Assessment	Executes integration test for cookbook
	ChefTest	Assessment	Executes unit test for cookbook
	buildInstallerRpm	Transformation	Builds the installer as RPM
	chefRelease	Transformation	Creates a release for the chef cookbook
Git-Service	gitCheckout	Transformation	Performs a checkout from version control
	gitInfo	Transformation	Generates git commit information

Table 5.1.: Developed Activities during Case Study 1

5.9.2. Observations

During the work with JARVIS and modelling the KISTERS reference project continuous delivery pipeline, some observations were made. These are presented and discussed in this section.

While developing the activities needed for the pipeline, the similarities between activities and services showed. Based on this observation, three individual service types were identified. Additionally design patterns were developed for those types. These will be used to reduce the effort needed to develop new services in the future and help with best practices. The identified types are of course not exhaustive. It would be beneficial to extend this list in future work with JARVIS. The identified types and their corresponding design patterns are presented in section 5.7.

Another observation was made during the integration of new technologies. It is advantageous to separate the technology of the build artifact from the technology the service is developed in. In the initial implementation of the maven-service this was not the case and it was impossible to build the Java 11 artifacts with a service developed in Java 8. To avoid this, there needs to always be an abstraction between the service technology and the artifact.

The last observations were made during the modelling process of the reference project pipeline. The modelling support through JARVIS smart planner functionality reduced the needed process knowledge significantly. Furthermore the model size was quite extensive in the current JARVIS DSL. For this a possible solution was designed. By composing individual activities and creating new composite-activities, the model size can be reduced. Especially in a scenario like the KISTERS reference project, many activities are always executed after each other. Hence a virtual combination of those activity would not only decrease model size, but also the complexity of the models.

5.9.3. Threads to Validity

If using some form of empirical evaluation, it is important to evaluate possible threads to its validity [FM10]. Therefore the threads for this first case study are discussed in this sections. The main thread to validity during this case study was an eventual existing biased of the researcher. Additionally previously existing knowledge about JARVIS or the case (reference project) could interfere with the results. To avoid both risks, the researcher which modelled the reference pipeline with JARVIS, had no extensive knowledge about both. JARVIS was introduced briefly before the project and absolutely no foreknowledge was present about the reference pipeline.

6. Case Study 2: CD Stakeholder And Requirements

Contents

6.1. Study Design And Execution 49
6.2. Continuous Delivery Stakeholder 50
6.3. Continuous Delivery Requirements 53
6.4. Continuous Delivery Survey 63
6.4.1. Notification by the continuous delivery system 63
6.4.2. Information needs in the continuous delivery system 67
6.4.3. Modelling in the continuous delivery system 71
6.4.4. Multi-Branch support in the continuous delivery system 73
6.4.5. Performance in the continuous delivery system 74
6.5. Discussion and Summary 76
6.5.1. Answers to Research Questions 76
6.5.2. Observations 77
6.5.3. Threads to Validity 77

After evaluating the general capabilities of JARVIS in the first case study, this chapter now presents the results of the second study. The second study is concerned with the identification of continuous delivery stakeholder in general. Additionally the requirements of those stakeholder, regarding continuous delivery, are identified and evaluated against concepts introduced in the JARVIS continuous delivery solution. Of special interest is the identification of potential new stakeholder roles in the continuous delivery environment. This chapter is structured as follows. First the general case study design and the used methodology is introduced. In the next section the identified stakeholders are presented and a short summary for each stakeholder is given. After defining the stakeholders, the found requirements are described and the concepts present in JARVIS, to solve those requirements, explained. The next section is about an executed survey that was used to further explore the found requirements. To finish the chapter, a summary of the gained findings is given.

6.1. Study Design And Execution

To start of this case study, its design is presented and, the methodology used, explained. As initially stated in chapter 3.4, the following research questions form the basis for this case study.

o2.R1 What are the stakeholder of continuous delivery?

o2.R2 What requirements do these stakeholders have regarding continuous delivery?

o2.R3 What concepts does JARVIS incorporate to deal with these requirements?

Analogous to case study one, these research questions are answered in the context of the KISTERS AG (see chapter 3.5) and the underlying case study design is shown in graphic 6.1. To answer research question **o2.R1 - o2.R2**, the chosen method was structured interviews. The selection process of the interview participants is explained in the next section in more detail. Basically three different questionnaires were developed and used to identify potential continuous delivery requirements of the participants. Each questionnaire was hereby targeting a different type of potential stakeholder. This classification is further explained in the next section as well. After the stakeholders and their respective requirements were identified, these requirements compared to the concepts of JARVIS, to find out, how JARVIS can be used to potentially satisfy them. Based on this evaluation the research question **o2.R3** was answered. To further explore some requirements and increase their meaningfulness (e.g. get acceptable time constraints for certain actions), a survey was designed and executed in the whole KISTERS energy business unit. These findings are presented in section 6.4 of this chapter. The survey hereby was designed by the author of this study and ran for three weeks. In the next section the process of identifying the relevant continuous delivery stakeholders and the found roles is described.

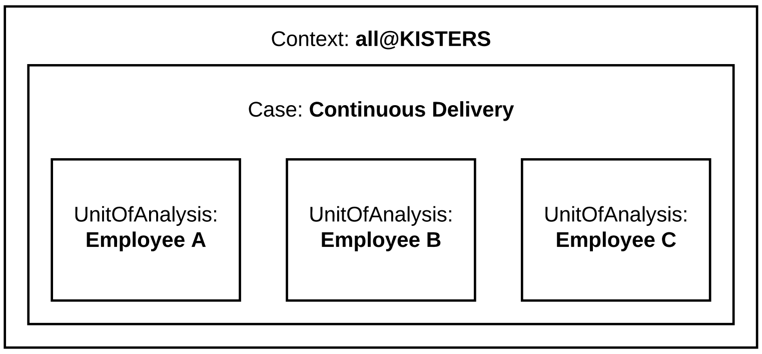

Context: all@KISTERS
Case: Continuous Delivery
UnitOfAnalysis: Employee A
UnitOfAnalysis: Employee B
UnitOfAnalysis: Employee C

Figure 6.1.: Second study, embedded case study on general employee base

6.2. Continuous Delivery Stakeholder

To identify continuous delivery stakeholders, employees of the KISTERS AG were interviewed and questioned regarding their requirements towards CD. For this purpose, three individual questionnaires were designed, each tailored towards a special group of employees. These questionnaires are attached to this thesis in the appendix (see A.5). The first questionnaire was targeted at employees that do not participate directly in the software development process and are potential new CD stakeholder. With the second version, software developers and architects were targeted. The last questionnaire aims to collect requirements from DevOps engineers and persons which directly and very intensively work with the CD system itself. Next the selection process for interview participants needs to be explained. To guarantee the consideration of every person as a potential stakeholder, the organization chart of the KISTERS energy unit was taken as a basis for the selection. This chart is displayed in figure 6.2.

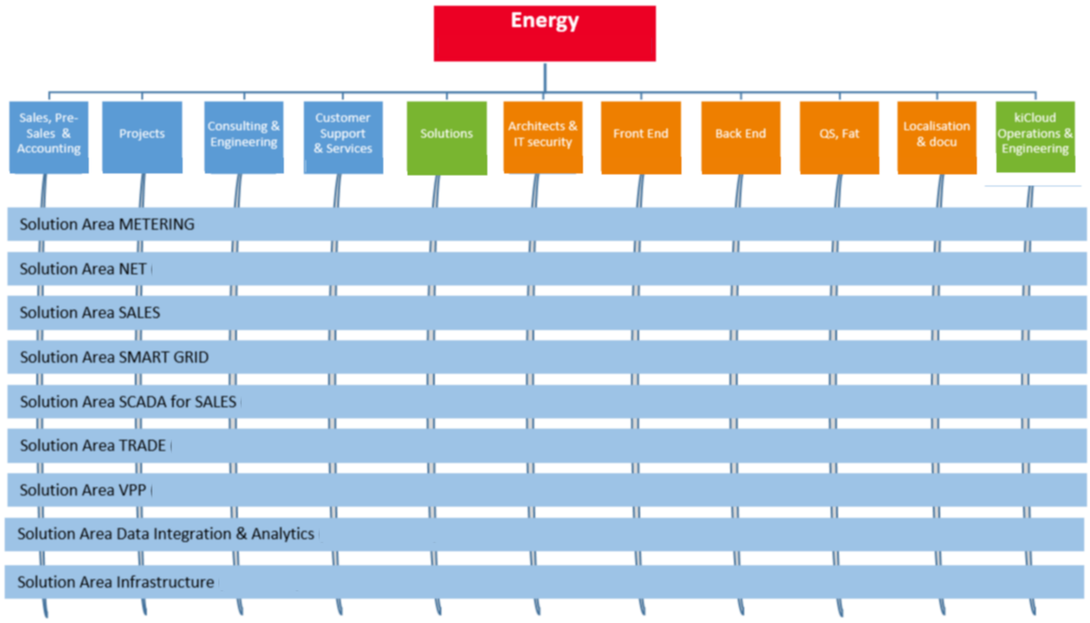


Figure 6.2.: Organization chart of the KISTERS AG BU Energy

Based on this chart, the head of each column was identified as an interview partner, as well as multiple subordinate employees. By choosing this rather neutral selection process, it can be guaranteed not to exclude certain roles based on the researchers biased. This helps to accomplish the goal to identify new stakeholders to the domain of continuous delivery, which were potentially overlooked in other research projects. Furthermore the research question **o2.R1** is answered in general. After explaining the used method and the selection process, the identified stakeholders are presented in the following.

Developer Frontend This role is considered a basic stakeholder for continuous delivery, since the frontend developer directly participates in the software development. The frontend developer is responsible for the design and development of graphical user interfaces and components. As main contact point with CD, the automated building and testing of software components was identified. A special concern for this role is the fast changing technology stack in the field of frontend development. A CD solution must be able to quickly adapt and integrate new technologies. The information needs for this role are build times, quality metrics and test results for their projects.

Backend Developer Analogous to the frontend developer, this role is mainly concerned with the development of corresponding backend applications. In contrast to the frontend, the technology choices are more stable and change less frequent. Especially important for this role is the stability and reliability of the software. This is mainly achieved by big test suites which need to be executed by the continuous delivery system. As with the frontend developer, the main requirement of this role is the automated building and testing of their developed software. The information needs are analogous to those of the frontend developer.

Software Architect This role is responsible for designing the overall software landscape and the interfaces between different components. For this role, CD is important to keep an overview about existing systems/projects and their dependencies between each other. For example dependencies between different components to create another artifact. Additionally the architect has information needs towards the overall technical dept of a project and quality metrics.

DevOps Engineer The DevOps engineer is the closest role to the actual CD system. They often operate the CD system and maintain/configure it. Another major aspect is developing the actual pipelines for the projects (or components). This role needs comprehensive insights into all aspects of the builds and the CD system itself. Because they are the role responsible for creating and maintaining the pipelines, their requirements include the modelling aspect of the CD system (e.g. graphical vs. textual modelling). Their information needs include information like CPU and RAM utilization of the operated CD system as well as all insights into the project pipelines themselves.

Security Officer This role is concerned with the general security aspects of the software products. It tasks include the monitoring of quality guidelines and/or company specific rules. Continuous delivery is important for this role because these guidelines or rules need to be checked constantly. Every time the software changes, the security aspects need to be evaluated as well. Therefore the need for support through the CD system itself is important for this stakeholder.

Cloud Manager The cloud manager is responsible for the cloud concept of a company and operating the developed software as a service in the cloud. The main interest regarding CD for this role is the modelling of cloud systems as continuous delivery pipelines. This includes the provisioning and maintenance of those systems in the CD system itself. The main benefit, this role hopes to gain by using CD, is an increase in general automation and therefore less errors and more performance. Regarding information needs this role is interested in active systems and their current configurations.

Quality Manager The quality manager is responsible for the overall quality of the developed software. This includes testing all functions of the product and ensure compliance to defined quality standards. These two aspects are the main interest of the quality manager regarding continuous delivery and also represent his information needs. The CD system must provide extensive testing support, while on the other hand style guides and other compliance relevant requirements have to be enforced. Since the quality manager is active in multiple projects, a configurable CD system is required to structure the information regarding the already mentioned aspects.

Sales Manager This role does not participate directly in the software development and is therefore an unconventional stakeholder. Its area of responsibility includes selling the software solution of the company and provide general customer support. For this reason the role has extensive information needs regarding features that are currently in development and/or already finished. Additionally the general road map for the development for a certain product is of interest. Since these information can theoretically be derived from each build of a project, this role hopes to gain support in its task by utilizing the CD system.

Project Manager The project manager is responsible for the successful completion and execution of projects. This includes managing the staff used in the project and the overall resources bound. Additionally this role acts as a translator between customer and developers. The project manager takes feedback from the customer, processes and integrates it into the daily work. Furthermore he is tasked with the planning of new features. For this work he needs an overview about the current development state and what features are done or in progress.

6.3. Continuous Delivery Requirements

In the last section the stakeholders for continuous delivery in the KISTERS BU Energy were identified and presented. To answer research question **o2.R2 - o2.R3** this section describes the found continuous delivery requirements of these stakeholders, as well as the concepts/solutions JARVIS offers to deal with them. Before this, a more detailed look into the performed interviews is needed. As mentioned in the last section, three different questionnaires were used to target employee groups more individually. The interviews were conducted as follows. First the participant was emailed the questionnaire in advance to prepare for the interview. Next the participant was ensured that the answers and information provided during the interview were handled confidentially and published in an anonymous way. It was stressed that no negative consequences would arise from participating in these investigation. This process mimics the approach described in [HA05]. The interviews themselves were recorded and later transcribed by the author of this thesis. After the results were extracted and summarized, the protocol was send to the participants to give them a chance to correct eventual misunderstandings. Finally the results were aggregated and formulated into the requirements stated in this section.

Next the found requirements are discussed and presented in the following format. Firstly the requirement is formulated as a statement and described in more detail. This is then proceeded by a list of stakeholders which issued the requirement and their motivation for it. Next the requirement is mapped onto the continuous delivery pipeline components, defined in chapter 2.3 (delivery model / delivery process / delivery system), and possible ways to solve it are put forth. Finally the concepts and/or solutions offered by JARVIS, that can solve this requirement, are discussed.

Req.01 The continuous delivery solution must enable and allow fast releases. This requirement includes multiple aspects. Firstly the individual build steps must be executed quickly and without delay. Secondly it must be possible to scale up the CD system if the need arises. Especially important was the absence of so called "overnight" builds, where developer commit their work and the result is presented to them the next day. This requirement was specified by the developers (frontend and backend), DevOps engineers and the quality manager. All of them aim to increase their productiveness by speeding up the build time and thereby allowing fast releases. Another motivation is the rise of agile development methods, which demand for more frequent releases. Lastly these stakeholders wish to work with experimental features. Such features need to be produced and released and replaced quickly since they only represent ideas and experiments. This requirement concerns the delivery model as well as the system and the process in the same way. The model must be suitable to a fast release cycle and reduce overhead operations which in turn results in a slim delivery process. On the other hand the delivery system must be able to scale up if needed and execute the delivery process as fast as possible. In JARVIS all of the described aspects are present. By using the smart planner the pipeline model is optimized and the resulting process is as fast as possible. Of course the process can be executed by the delivery system (JARVIS) and because of the micro-service architecture, it is possible to scale up each activity individually, if needed.

Req.02 The continuous delivery solution must guarantee the compliance to certain quality criteria. This requirement ensures that the developed software complies to defined corporate style and security guidelines. The stakeholders for this requirement are the DevOps engineers and security managers especially. To satisfy this requirement the delivery system, as well as the delivery model, are essential. The model must provide the ability to include the quality ensuring steps while the system must execute them and provide appropriate reports. In JARVIS this is accomplished by using so called assessment activities in combination with quality gates. A more detailed explanation of this concept is provided in chapter 2.3.

Req.03 The continuous delivery solution must be able to execute predictive builds. These builds are used to test changes without the risk of breaking the current pipeline, because they are executed in an isolated environment. The relevant stakeholders for this requirement are software developers (frontend/backend) and quality managers.

To satisfy the requirement, a suitable delivery system must be used which can execute these kind of builds. In JARVIS there is no dedicated feature to handle predictive builds. Nevertheless, the concept of isolated build environments is a fundamental feature of JARVIS, since every pipeline run is isolated from each other.

Req.04 The continuous delivery solution must be able to support the use of multiple branches for one project. This includes merging branches automatically during the pipeline run and not having to duplicate the delivery model every time a new branch is created. Relevant stakeholders for this are mainly frontend and backend developers. To satisfy the requirement, firstly it must be able to model the use of multiple branches for a project. Additionally the delivery system must provide the functions to work with branches and perform the required operations. JARVIS has currently no function to realize multi branch support. A possible extension point, to realize this feature, would be the internal model. It has to be extended to support multi branches for one pipeline. Furthermore the smart-planner needs to be modified to plan and optimize pipelines with multiple branches.

Req.05 The continuous delivery solution must be able to define optional quality gates. In certain situation the pipeline requires a quality gate which does not block the entire development process but provides valuable information. For example a dependency check to show outdated external dependencies. It must be possible to model these kind of optional steps. This requirement was motivated by the frontend and backend developers. Analogous to requirement **Req.04** this can be achieved by adapting the model and use a delivery system that can work with this model. In JARVIS the declaration of optional quality gates can be easily achieved. Since the concept for quality gates is already existing, the internal model needs to be extended for an optional flag. Additionally the orchestrator needs to be modified not to stop if a pipeline step is marked optional.

Req.06 The continuous delivery must be extendable with new technologies. Since technologies and libraries, used in software development, change frequently, the delivery solution needs to function in this moving environment. Another important aspect is the extension with new technologies, without affecting the existing pipelines. This requirement was provided by the DevOps engineers, which have to maintain the CD system and realize eventual technology changes. In the concept of the general delivery pipeline, this requirement can be mapped to the delivery system component since it depends solely on its capabilities. One of the key concepts of JARVIS is the seamless integration of new technologies into the system. This is achieved by the use of a micro-service architecture, which gets a new micro-service for each technology. Through separating the different technologies in individual services, no incompatibilities between technology choices can arise.

Req.07 The continuous delivery solution must provide a graphical representation of the delivery model/process. Traditionally, pipelines are modelled as a process and the resulting graphical representation is a command graph. This kind of visualization is useful for the stakeholders to identify individual build steps, get an general overview and track the progress of the current run. The stakeholders which required this feature are developers (frontend/backend), DevOps engineers, architects and quality managers. To satisfy this requirement, the delivery system must be able to provide the needed information and display them accordingly. By utilizing the PipelineViewService (that gathers and provides the needed information) and the custom frontend with activity-graph representation of the pipeline, JARVIS satisfies this requirement.

Req.08 The continuous delivery solution must be expandable by providing hooks for custom scripts. Current CD systems (e.g. Jenkins CI) provide a lot of support via plugins and build in features. In some cases these features do not suffice to model a CD project and developing an own plugin is too time consuming. In such cases the stakeholders must have the ability to execute "general purpose" code to quickly provide functions. This was mainly demanded by the DevOps engineers, which have to create and maintain different CD projects. This requirement can be mapped onto the delivery system component of the pipeline model. In JARVIS this can be achieved by developing a custom activity which takes a shell script as input and executes it.

Req.09 The continuous delivery solution must offer dedicated information regarding the origin of an error. Since delivery systems are complex constructs, the origin of an error is not always easy to deduct. For example if the error is caused by the CD system itself, which often is mistaken for an error in the CD model of the project. The stakeholders which are mostly concerned with this are the DevOps engineers. If errors arise, the DevOps engineers are the stakeholder which have to fix it most of the time. To reduce effort and enable the fix, a clear source of error is needed. This is a problem that can be mapped to the delivery system component. JARVIS tackles this by having extensive logging capabilities and all information can be published by extending the PipelineViewService.

Req.10 The continuous delivery solution must produce traceable and immutable artifacts. To enable traceability and document each pipeline run, each run must produce a dedicated artifact which is stored and versioned. This is especially needed to ensure compliance with ISO certifications or customer requirements. This requirement was issued by the DevOps engineers and the developers, frontend as well as backend and can be mapped on the delivery system component. One of the core concepts of JARVIS is the flow of artifacts between activities. JARVIS thereby exceeds the requirement by providing an immutable artifacts not only per pipeline run, but after each individual activity.

Req.11 The continuous delivery solution must provide the ability to update the system without modifying existing delivery models. In the environment of this case study, hundreds of pipelines are used and update adapt all of the, if the delivery system gets updated, is much effort. To prevent this, the DevOps engineers formulated this requirement. As can be extracted from the requirement itself, this is solved by using a delivery system that has some form of backwards compatibility regarding its models. In the JARVIS system this can be solved by introducing version numbers in the models and releasing different model parser for each version. The backwards compatibility can hereby be as comprehensive as needed.

Req.12 The continuous delivery solution must provide operational information about the delivery system. This information need includes for example RAM and CPU usage of the delivery system as well as a list of all active instances. As operator of the delivery system this requirement was requested by the DevOps engineers to help them manage the system. As **Req.11**, this requirement can be mapped to the delivery system component of the pipeline model. In JARVIS such information are currently not gathered or displayed. A possible solution would be to let activity-services publish their performance information via the data-bus. Analogous to other information needs, the PipelineViewService can collect these information and offer them via an API to the frontend component.

Req.13 The continuous delivery solution must visualize the pipeline in a domain specific way. This includes displaying the component itself and the order in which they are build. Relevant stakeholders for this requirement are the backend developer and the software architects. By using this kind of visualization, dependencies between components can be made clear and a greater overview can be achieved. Furthermore the amount of technical knowledge is reduced by not displaying the pipeline as sequence of technical steps. In JARVIS the current graphical representation is graph that displays the executed activities. If the activities are named accordingly, the required domain specific representation can be achieved. Another way would be to add the possibility to model domain concepts to the modelling capabilities and fully implement a second representation.

Req.14 The continuous delivery solution must provide textual and graphical modelling for the user. Creating and maintaining delivery models can be achieved by writing text files or using some kind of graphical notation. This is dependant on the current scenario and the abilities of the user. Offering both possibilities increases the flexibility and user acceptance of the delivery system. This requirement was specified by the developers (frontend/backend), architects and DevOps engineers. It can be satisfied by using a delivery system that can handle multiple forms of input models and additionally offers tooling to create them. Furthermore the basic concept of the delivery model must allow for varying representations. JARVIS utilizes an internal and external model approach. The external model will always be transformed into the internal and the delivery system itself only works on the internal model. This abstraction allows for almost any external model representation. Currently the user can encode the delivery model in YAML files or create them via BPMN editor.

Req.15 The continuous delivery solution must provide the ability to model system landscapes and manage them through the delivery system. This requirement occurred in the context of cloud based software and was requested by the cloud manager as well as the DevOps engineers. The main benefit, these roles hope to gain by utilizing CD, is an increase in performance and less errors by automating more tasks. Again this requirement can be solved by using a delivery model which can express system landscapes and a delivery system with the needed operations. In JARVIS this feature can be used by developing appropriate activities and maybe adding a more specific external model.

Req.16 The continuous delivery solution must provide immediate feedback if an error occurs. Stakeholders for this requirement are all already presented stakeholders except the sales and security managers. This requirement is solely solvable through the delivery system itself, which needs to notify the user if defined events occur. JARVIS currently has one feature which provides active feedback by the delivery system. This feature was developed as part of the chat ops (continuous delivery controlled via chat messages) expansion. More feedback can be introduced by expanding the JARVIS environment with a new micro-service, dedicated for this task. The new "feedback-service" would be connected to the data-bus and push messages to the relevant recipient.

Req.17 The continuous delivery solution must offer support for multiple testing mechanisms. This includes unit-, integration-, system-tests. The requirement was issued by the developers (frontend and backend) and the quality managers. Both roles need this feature to increase the confidence after each change regarding eventual error or failures. Additionally they want to make sure that features, demanded by the client, work as expected. To realize this features, the delivery system must provide the actions needed to test in multiple ways. Also a graphical representation for test results is needed. By encoding assessments as regular activities, JARVIS does support all kinds of different testing methods. The results of the tests are displayed by clicking on the corresponding quality gate in the activity-graph. Further result graphics can be added by utilizing the PipelineViewService and the client.

Req.18 The continuous delivery system must provide the option to declare pipeline steps "manual". If a step is declared "manual", the pipeline execution is paused until a user triggers manually the defined pipeline step. This feature was required by frontend/backend develops, software architects and DevOps engineers. Not all operations can be decided by machines. For example the deployment to a demo-system may be included in the pipeline as the final step, but not every commit should update the demo system. Like the requirement **Req.05**, this function needs to be intended in the model, as well as the delivery system. It must be possible to model manual steps in the pipeline declaration and execute these in the system. The system also needs to provide some form of interface to trigger the manual step. As with **Req.05**, the requirement is currently not satisfied by JARVIS. It can be achieved by extending the current internal model to include manual steps. In the next step the pipeline orchestrator needs to be modified to stop until the step is triggered manually. Lastly the client needs to be adapted, to allow the user to interact with the pipeline run.

Req.19 The continuous delivery solution must provide interfaces to execute singular pipeline steps outside of a pipeline run. This feature was requested by the frontend/backend developers and the quality managers to be able to test individual pipeline steps without committing any changes to the repository and trigger a pipeline run. To solve this challenge the delivery system must offer extension points or APIs, where the user can upload artifacts. After executing the operation, the system must then provide a method to download the result. JARVIS encodes operation in individual activities. Each activity connects to the JARVIS environment via REST API endpoints. These endpoint can be used to accomplish the required functions.

Req.20 The continuous delivery solution must be able to speed up builds by allowing parallel task execution. The motivation behind this requirement was solely to increase the performance of every CD user. Stakeholders for this requirement are developers, DevOps engineers and architects. To solve it, firstly the model must allow the use of parallel tasks. Secondly the delivery system must have the capabilities to execute these tasks and manage the parallel processing. By using the smart planner component, JARVIS models do not need to consider parallel tasks execution. The smart planner does automatically optimize the model regarding execution speed. This optimized process is then executed and managed by the orchestrator service.

Req.21 The continuous delivery solution must offer individually configurable dashboards to monitor projects. These dashboard must be able include only projects where the user is currently active and it must be able to reduce displayed information (e.g. hide test results for a project). The need for a configurable dashboard was formulated by the developers and the quality managers. They are often active in different projects and need to reduce the received information to gain more focus. To achieve this functionality, the delivery system must enable/allow configurable user views and an user authentication. In the current state, JARVIS has no configurable dashboards. To achieve the requirement, an user authentication method must be included and the client needs to be modified to allow for those dashboards.

Req.22 The continuous delivery solution must visualize dependencies between different process steps. This information is used to simplify the maintenance of delivery models by increasing the overview. Relevant stakeholders for this requirement are developers and DevOps engineers. To satisfy the requirement, the delivery system must offer a suitable graphical representation of the delivery process, defined by the delivery model. JARVIS does this with its current graphical representation as activity graph in the client component.

Req.23 The continuous delivery solution must provide the user with active feedback if defined steps in the delivery process are finished. This feature is used for example to notify stakeholders if the test-system deployment is ready for manual testing or if the demo-system is ready for a presentation. It represents a manual handover between the delivery system and the user. The stakeholders are analogous to **Req.16** and the relevant component is again the delivery system. As with the former requirement, JARVIS does currently not offer this feature. Again it could be satisfied by developing and integrating a new feedback-service in the basic JARVIS infrastructure.

Req.24 The continuous delivery solution must not influence/dictate the project setup. To allow the developers a maximum of freedom during development, the delivery system should work on any project setups with no or very little modification, to accommodate for the system. This requirement was issued by the frontend developers and can be solved by using a suitable delivery system. In JARVIS this characteristic is ensured by using self-programmed activities. These activities can be developed to work on generic project setups without the need for project adaptation.

Req.25 The continuous delivery solution must provide test execution optimization. To increase overall build runtime, test-suites must not always be executed completely or in the same order. The quality managers and developer (frontend/backend) require optimization methods for the partially very large test-suites. This kind of requirement can be resolved by using a delivery-system that performs test execution optimization. During the work of [Nun18] JARVIS was extended by such a regression test optimization function. This work integrated the regression test optimization framework of [Ple15] into JARVIS and allowed assessment activities to leverage the functionality. Currently one optimization strategy is implemented. The strategy tracks the failed tests during a run and changes the execution order for the next run. Tests which have previously failed, are executed first in the consecutive build.

Req.26 The continuous delivery solution must cache pipeline steps and result artifacts. To decrease average build runtime the delivery system must recognize process steps which are the same as already executed steps. This would be especially beneficial if a previous run has failed and the caching can speed up the subsequent run. Uttered was this requirement by the frontend developers, but having this feature would provide benefit for every user of the delivery system. To realize the function, the delivery system must be able to recognize identical steps and have means to save results from previous runs. In JARVIS a caching mechanism is currently under development and will be included in later versions. If the delivery system and model have not changed and furthermore the input is the same, the resulting artifact is directly used as activity result. Thanks to the artifact-oriented approach, described in **Req.10**, this can be achieved by calculating hash numbers for the produced artifacts and storing this information in the artifact store.

Req.27 The continuous delivery solution must remind the user to fix failed pipelines. In the case that the user participates in multiple development projects, it can happen to forget or overlook a broken pipeline. To ensure this state does not persist, the delivery systems continuously reminds the user to fix broken builds. Like the **Req.26** requirement, this was specified by the frontend developers as needed for their daily work. In JARVIS this reminder function is not implemented at the moment, but could be achieved by developing (or expanding) a feedback-service (as described in **Req.23**).

Req.28 The continuous delivery solution must offer delivery models which can be maintained without expert CD knowledge. To increase everyday productivity, multiple stakeholders want to create and maintain their delivery models on their own. This holds true for the frontend/backend developers, architects and quality managers. To solve this requirement, the delivery model must be simple and (best case scenario) domain specific. Furthermore it must provide an abstraction over the technical implementation of the delivery process, since not all stakeholders have a technical background. JARVIS offers an general abstraction between internal and external model, which allows the use of almost any external model suited for the purpose. If chosen appropriately, the external model can be used by any of the former mentioned stakeholders.

Req.29 The continuous delivery solution must allow splitting the model in sub-models and display them graphically. To decrease overall model complexity and be able to assign different parts of the model to different teams, subdividing the pipeline is needed. This was required by the architects, motivated by their top down view on projects and planning interests. To accomplish this requirement, all aspects of the pipeline model need to be modified. The delivery model must define handover points between parts of the model. Futhermore the process must be adapted to consider the breakdown of the model. Finally the delivery system must support this requirement by executing the resulting processes, represent them graphically and support the modelling. In JARVIS, there are no sub-pipelines possible at the moment. Introducing this would require an extension of the internal model and modifying the planner, as well as the orchestrator.

Req.30 The continuous delivery solution must offer versioned models. To trace model changes and revert them eventually, the use of versioned models is required. This was declared by the backend developers and the DevOps engineers. This is a requirement which can be satisfied by a suitable delivery system that saves each version of a delivery model in addition to other meta information (e.g. author or date). For JARVIS this was introduced partially. By having multiple external models, the versioning is only available for textual models. JARVIS loads its textual model generally from a git repository, which in turn guarantees versioning of the delivery model.

Req.31 The continuous delivery solution must show features which are currently under development for a project. Additionally is must be possible to see the current road-map. This requirement was specified by the sales and project manager role in order to gain insights in the current state of a product. These departments needs to know which features are ready to be sold to customers and which are near completion. Since these information are theoretically available in the delivery system, it should be adapted to satisfy the information need. For solving this requirement als three components of the pipeline model must be taken into consideration. In JARVIS this feature is not present and no obvious solution to implement it exist. Therefore some research into this topic is needed.

Req.32 The continuous delivery solution must be able to integrate remote systems into the deliver process. Mainly this is targeted at managing systems at a remote location (e.g. at a clients side) and updating software there. The cloud manager needs this requirement for customers which require (or insist) on an on premise solution. By integrating the remote machines in the general CD environment the maintenance effort can be lowered to a degree, which compares to a local installation. In a best case scenario this requirement only needs the delivery system adapted, without any modification to the model (it is transparent if the operation is executed locally or remote). In JARVIS this can be directly accomplished by developing a suitable activity.

Req.33 The continuous delivery solution must display security information about software components. These includes access rights and currently used external tooling. For the external tooling, known security vulnerabilities are searched, as well as used licenses. This was required by the security managers, that have to ensure the compliance to defined company standards. As with general quality criteria (like required in **Req.02**), the delivery model and system have to be able to realize the requirement. In JARVIS these security concerns can be realized by developing corresponding assessment-activities and integrating them into the delivery model.

Req.34 The continuous delivery solution must be able to be audited. Again this requirement was specified by the security manager who needs this function to ensure ISO compatibility for the company/software. This is solely solvable by a delivery system that keeps track of the necessary information. JARVIS, in its current state, does not track any information related to audit events. By using the central communication bus, to collect information from the whole infrastructure, this requirement can nevertheless be satisfied.

By collecting and analyzing the requirements, the research question **o2.R2** was answered and by that our understanding of continuous delivery was deepen. In the next step, the concepts and ideas included in JARVIS, to cope with those requirements were introduced. This helped answering the last research question of this chapter (see **o2.R3**). In summary JARVIS is able to satisfy almost all identified requirements or at least have a clear concept how to achieve it. This was mainly possible thanks to the modular architecture and use of micro-services. Expansions with new technologies or functions can be achieved without influencing or modifying the existing environment. Additionally the split between external and internal delivery model allowed for a very flexible modelling approach. Of course not all of the mentioned requirements are currently satisfied in the reference implementation. Especially regarding information needs there is a huge potential for improvements and addons. But since the basic concepts of JARVIS support the integration of the required functions, JARVIS can be considered as a suitable replacement CD system for productive use. In the next section, a selection of the found requirements is further concretized to gain more insights and increase the insights into continuous delivery stakeholder requirements.

6.4. Continuous Delivery Survey

After interviewing the stakeholders, described in 6.2 and collecting their continuous delivery requirements, this section aims to further investigate and concretize some of those requirements. To accomplish this, first the selection criteria for the requirements in the survey need to be defined. Selected were such requirements which are either formulated unclear by the stakeholders or provided room for interpretation. Furthermore, this survey was used to evaluate priorities between requirements. These priorities can be used to help the KISTERS AG improve their continuous delivery solution and form a basis for decision for eventual expansions of JARVIS.

The survey was active for three weeks and had a total of 57 participants. Of those 57 participants, 42 fully completed the survey and are thereby relevant for the result. Potential participants came from the KISTERS BU energy and were selected with the same priorities as the interview partners in 6.3. The selected persons were notified via email and the survey was executed via the online tool LimeSurvey [Lim]. To gain the required insights, five categories of questions were developed with 18 questions in total. The results are presented in the following in the context of their respective question category.

6.4.1. Notification by the continuous delivery system

This question group is concerned with the feedback aspect of continuous delivery and the way the user receives notifications. This was motivated by requirements like **Req.16, Req.23 and Req.27**. Also, like explained in 6.3, JARVIS is currently missing a concrete concept for user feedback. The gained information can be used to develop and integrate such a concept successfully into the JARVIS eco-system. In this category, five different questions were asked, which were focused around preferred feedback methods and acceptable feedback times. Furthermore it was evaluated if the user likes to be notified actively by the delivery system or prefers a passive approach. During the interviews the importance of feedback was mentioned multiple times and various forms of feedback were required. This overall significance was confirmed by the survey, as shown in figure 6.3. Furthermore the survey evaluated the acceptable reaction time of the CD system, in case an error occurs. Three time intervals were given to chose from, representing general reaction times for computer systems. The majority of the participants are satisfied with a reaction time between 3-5 minutes after the event has happened. For the complete distribution see the graph 6.4. Next, the two requirements **Req.23 and Req.27** were investigated. During the interviews it was stated that an active notification by the CD system is required and also a reminder function for failed pipelines. Contrary to the interviews, the survey results showed only an average interest in those features and ranked their perceived importance only at 2.21 and 3.24 out of 5.0. The respective information are visualized in 6.5 and 6.6. After supporting the general importance of feedback by the continuous delivery system, the user were finally asked to rank feedback methods according to their preferences. The result of that ranking is displayed in table 6.1 with notification via email as preferred method.

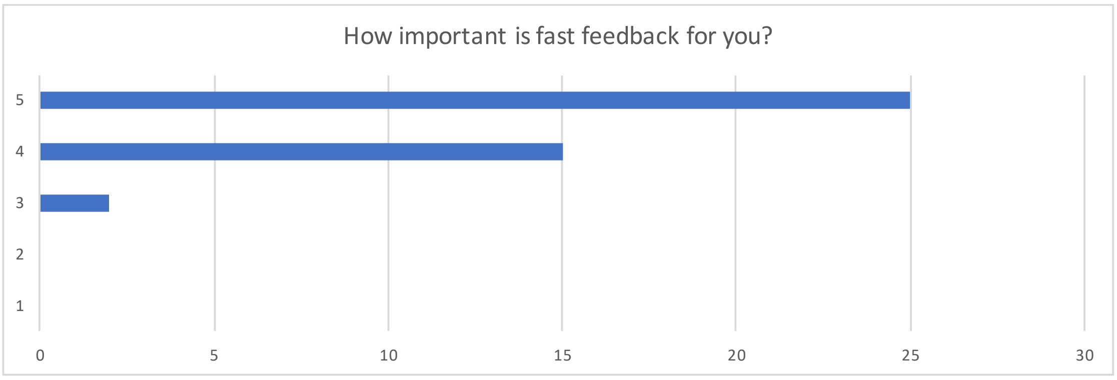


Figure 6.3.: Results How importance is fast feedback for you if a pipelines fails?

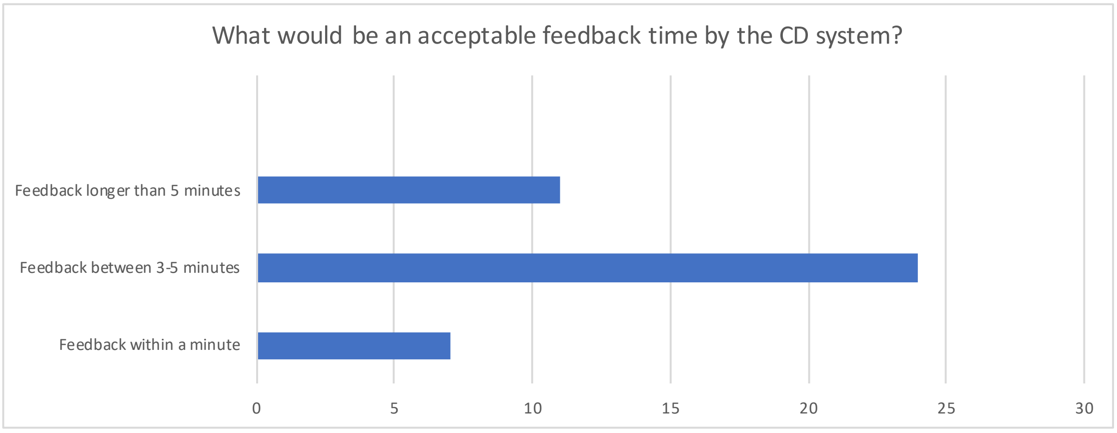


Figure 6.4.: Results What would be an acceptable feedback time by the CD system?

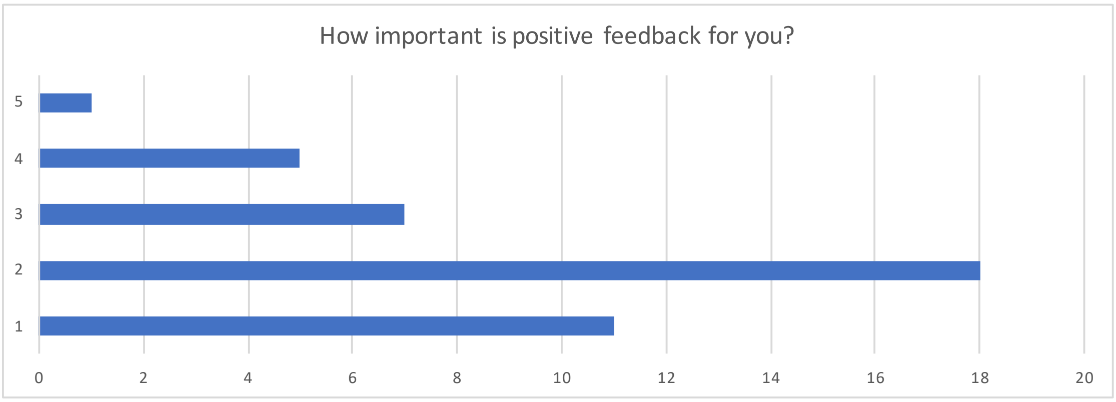


Figure 6.5.: Results How important is positive feedback for you?

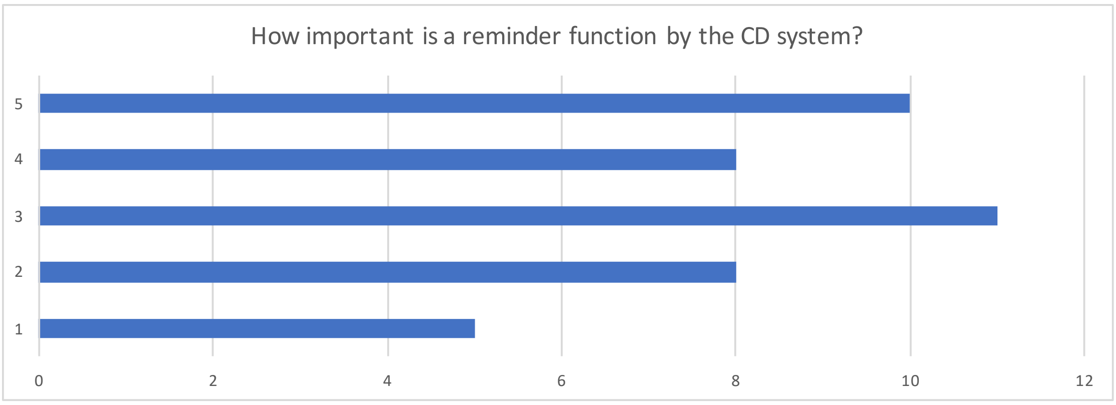


Figure 6.6.: Results How important is a reminder function by the CD system?

Method	Count	Percentage
Notification via Email	20	47.62%
Notification via chat message (e.g. Skype)	7	16.67%
Notification via hardware (e.g. lamp)	7	16.67%
Display with dashboard in the hallway	4	9.52%
Not completed or not displayed	4	9.52%

Table 6.1.: Preferred notification methods

6.4.2. Information needs in the continuous delivery system

After discussing feedback requirements in the last section, this section now tries to concretize information needs the continuous delivery system must satisfy. Requirements related to unsatisfied information needs, or their graphical representation, were issued multiple times and by different stakeholders. Such requirements were **Req.07, Req.09, Req.12, Req.13, Req.21, Req.22, Req.31, Req.33, Req.34**. In this question group, participants were asked to evaluate and rate the importance of those requirements. Furthermore, the priority of configurable dashboards is determined. The charts 6.7, 6.8, 6.9, 6.10 and 6.11 show the perceived importance of each information need up for debate. Having the information which pipeline is dependant to which other pipeline was stated as the most important need (4.1 out of 5). This was followed by the graphical representation of pipelines as sequence of steps, as provided by most established CD systems. Despite being required by multiple stakeholders, the need for a configurable dashboard in the CD system was not rated as important. With 3.46 as the arithmetic mean, it scores worse than suggested by the conducted interviews. The full distribution is displayed in the chart 6.12

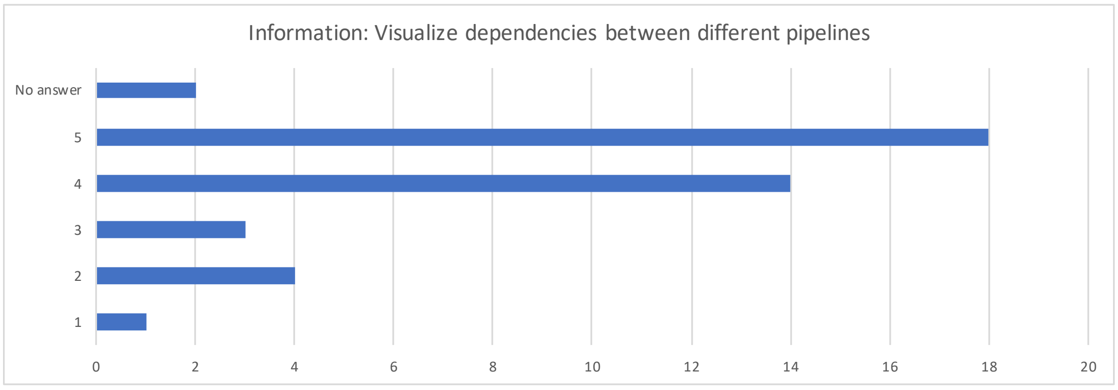


Figure 6.7.: Importance visualize dependencies between different pipelines

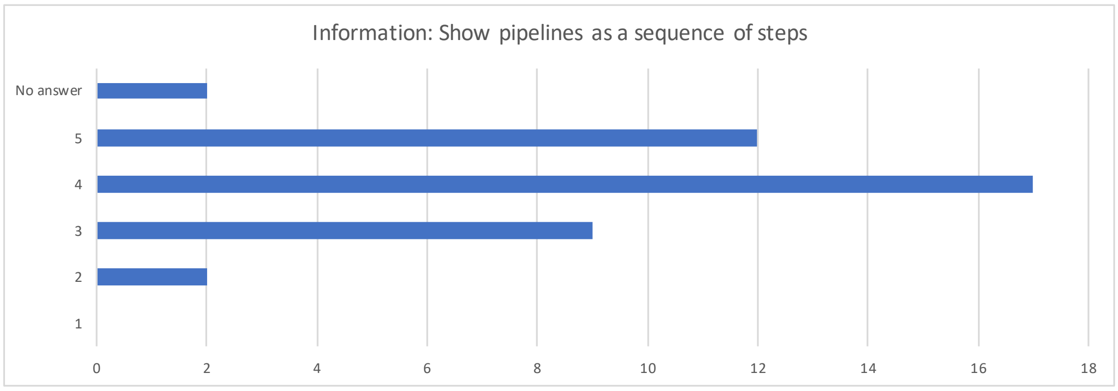


Figure 6.8.: Importance show pipelines as a sequence of steps

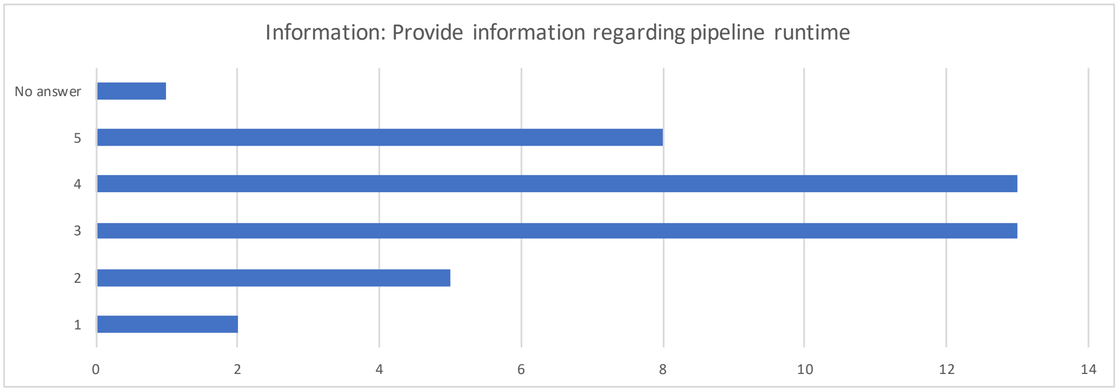


Figure 6.9.: Importance provide information regarding pipeline runtime

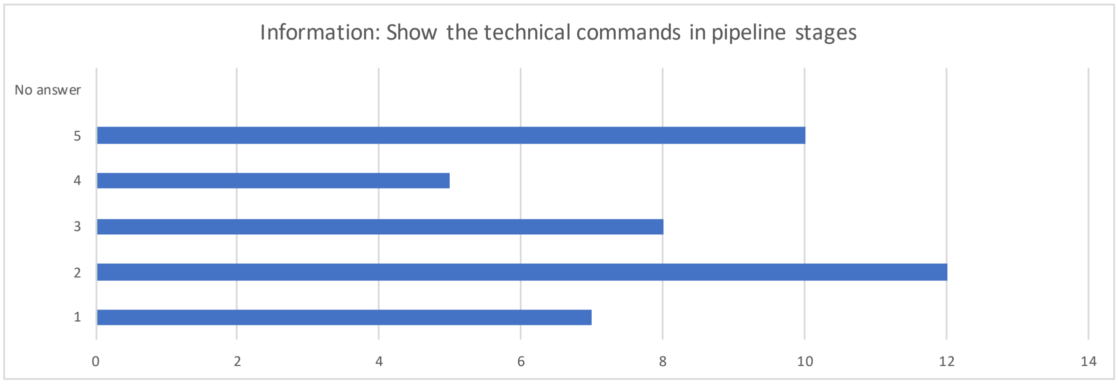


Figure 6.10.: Importance show the technical commands contained in a pipeline

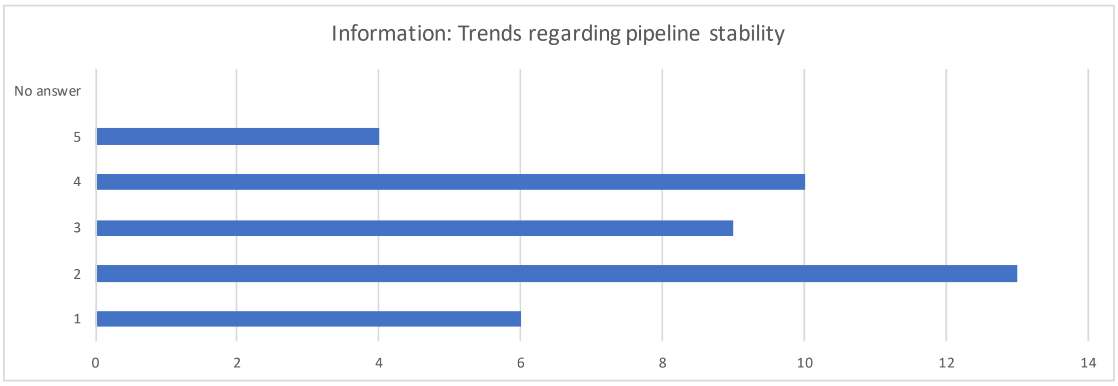


Figure 6.11.: Importance show trends regarding stability

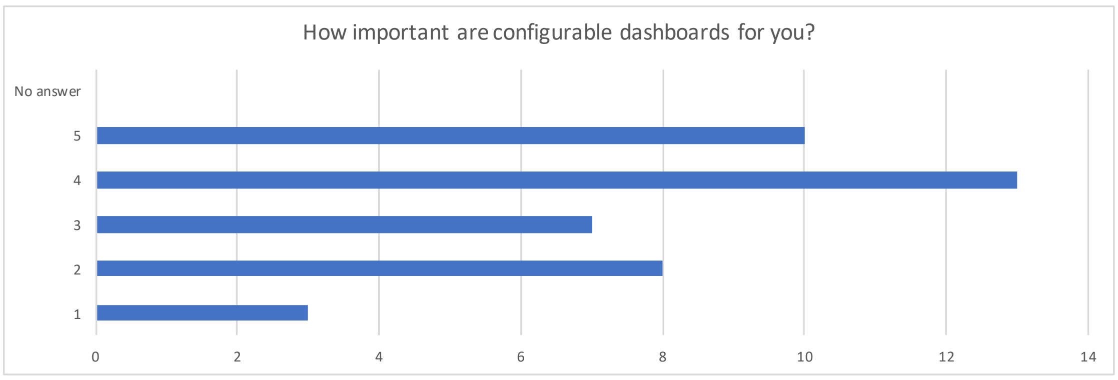


Figure 6.12.: Results How important are configurable dashboards?

6.4.3. Modelling in the continuous delivery system

The next question group is about the modelling aspect of the continuous delivery solution. This was motivated by requirements like **Req.05, Req.14, Req.15 and Req.18**. The first requirement, which was confirmed by the survey, was **Req.15**. Modelling system landscapes and environments in the delivery system was generally perceived as important and useful (see chart 6.13). Next the preferred modelling approach was evaluated. During the interviews, stakeholders were often unsure if they benefit more from graphical or textual modelling. So the survey was used, to get a deeper understanding of this question. As shown in figures 6.14 and 6.15, the textual approach was favored slightly but the differences were rather small (3.08 vs. 3.26). This supports the requirement **Req.14**, which states that a continuous delivery system must offer both approaches. Another requirement, that was supported by the survey, was the importance of self-service for pipelines through the user themselves. Users want to be able to create and maintain their pipelines, without the need to contact the delivery team. The corresponding graph is shown in figure 6.16. Finally, as seen in figure 6.17, the requirement for optional pipeline steps was confirmed.

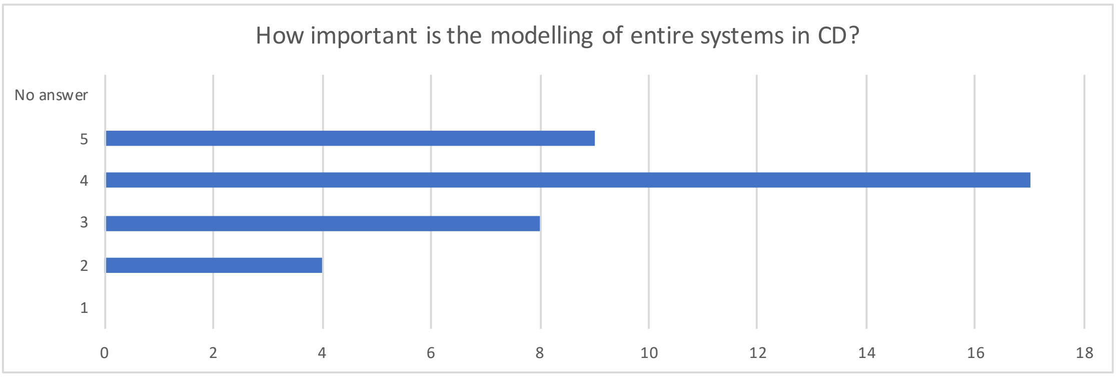


Figure 6.13.: Results How important is modelling system landscapes in CD?

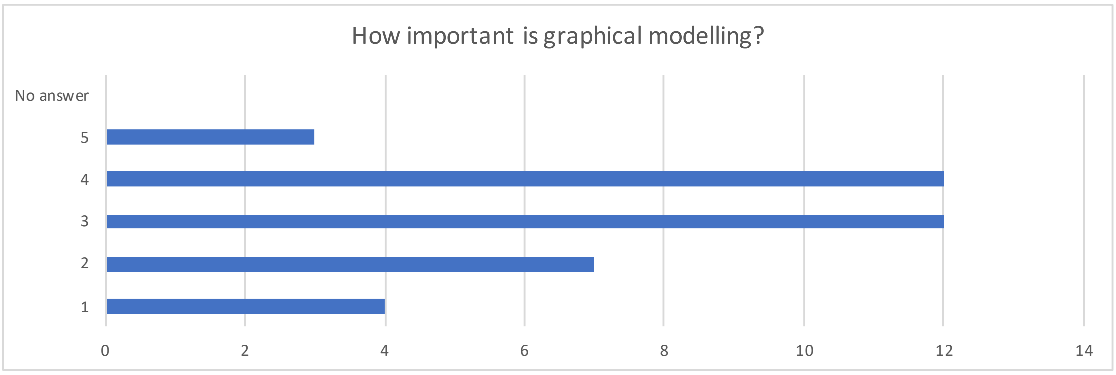


Figure 6.14.: Results How important is graphical modelling?

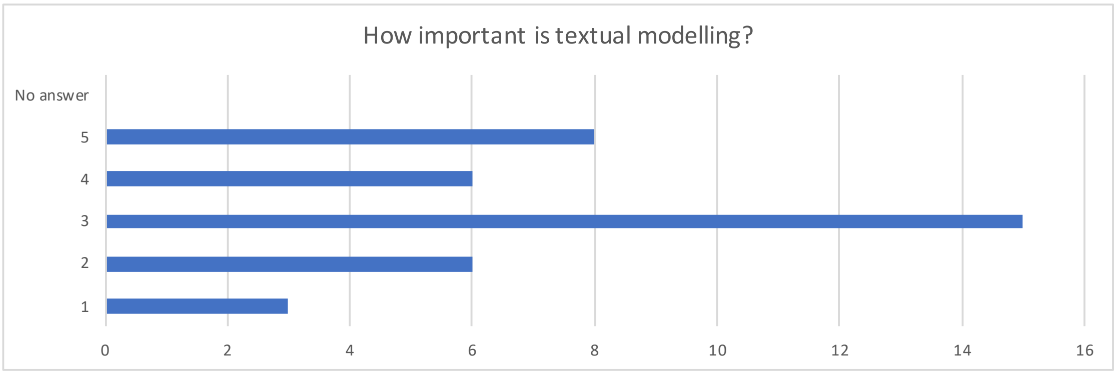


Figure 6.15.: Results How important is textual modelling?

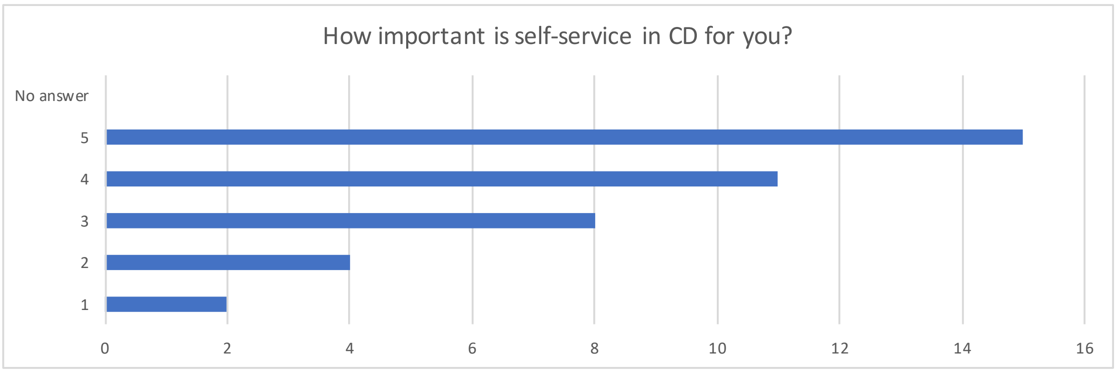


Figure 6.16.: Results How important is self-service in CD for you?

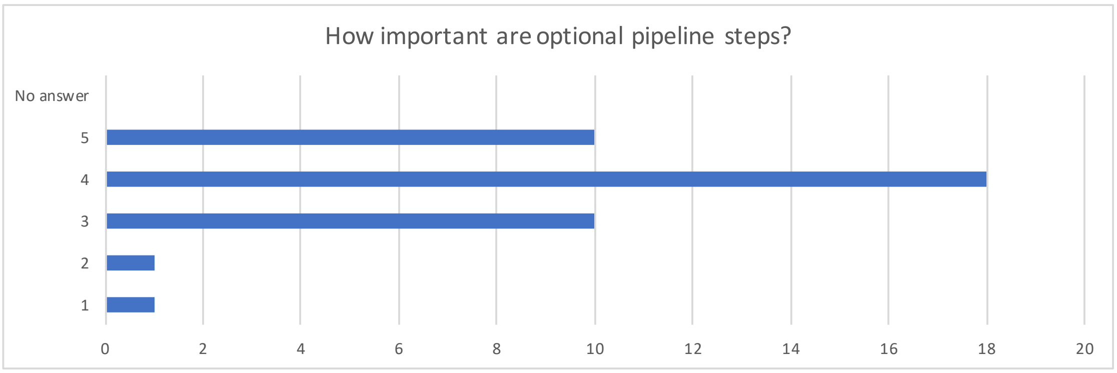


Figure 6.17.: Results How important are optional pipeline steps?

6.4.4. Multi-Branch support in the continuous delivery system

The next requirements which were investigated were **Req.03 and Req.04**. These requirements are concerned with the ability of the continuous delivery system to support multi-branch projects. As see in 6.18 and 6.19, both features are rated very highly regarding their overall importance and perceived usefulness. This strongly supports the first impressions made during the interviews.

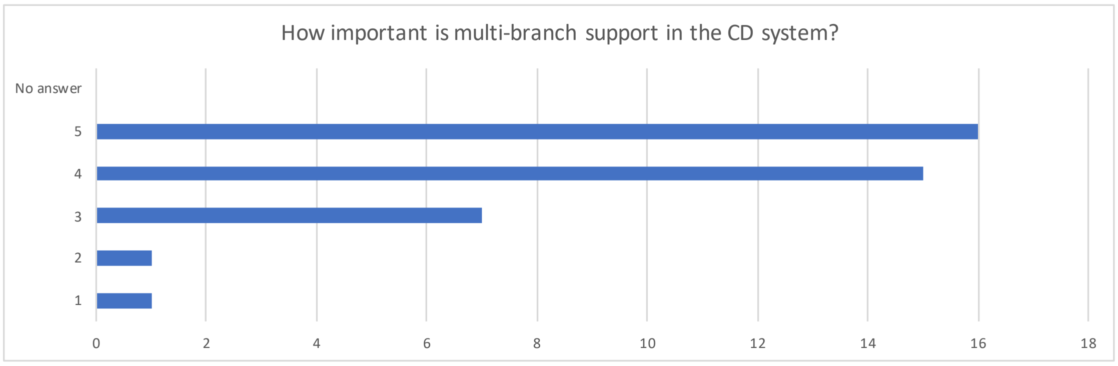


Figure 6.18.: Results How important is multi-branch support through the CD?

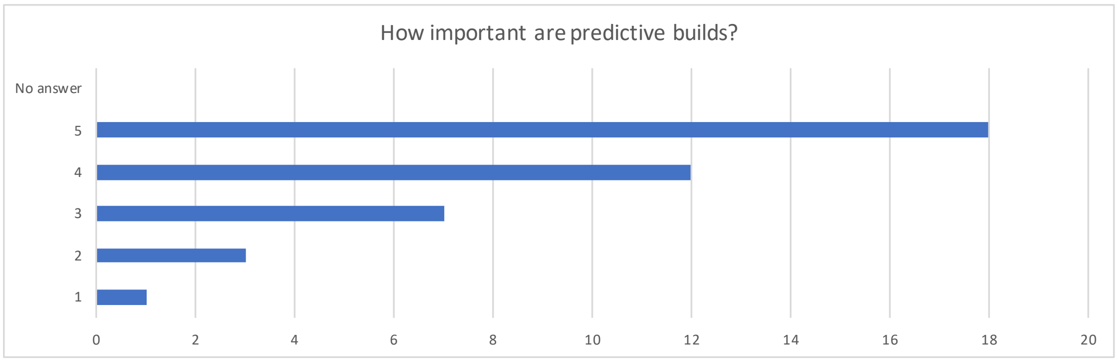


Figure 6.19.: Results How important are predictive builds?

6.4.5. Performance in the continuous delivery system

Finally, the last question group was about the performance of the continuous delivery system. This was due to numerous requirements which demanded the speedup of certain CD activities. To start of, the general importance of speed in the context of continuous delivery was evaluated. As hinted by the interviews, the perceived importance was overwhelming (see 6.20). In the next step, two particular features were put up for discussion. The first feature was specified by the requirement **Req.26** and concerned with caching of pipeline steps. This was rated with an average importance, like displayed in graph 6.21. Requirement **Req.19** on the other hand was prioritized high (see 6.22). It demands the availability of APIs to execute pipeline steps locally to speed up the development process. During the interviews, the stakeholders demanded fast actions but were mostly unable to quantify the desired performance increase. To fill this knowledge gap, the participants were collectively asked, what they would consider an acceptable runtime for certain activities. The results of this are shown in table 6.2 and can be used to improve the delivery solution at the relevant places.

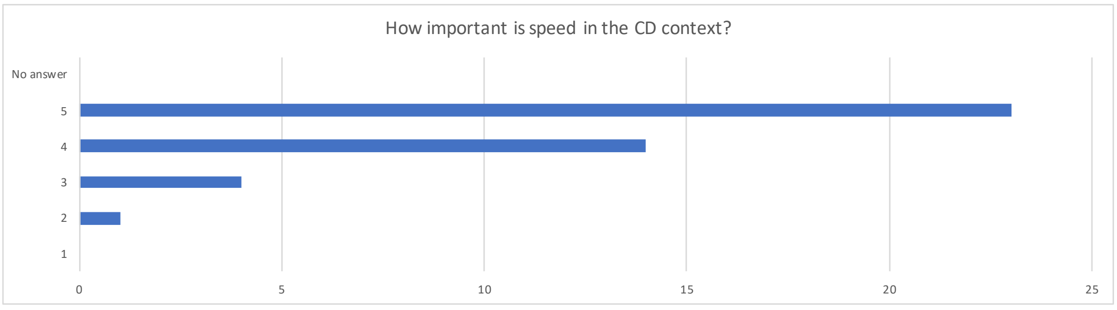


Figure 6.20.: Results How important is speed in the CD context?

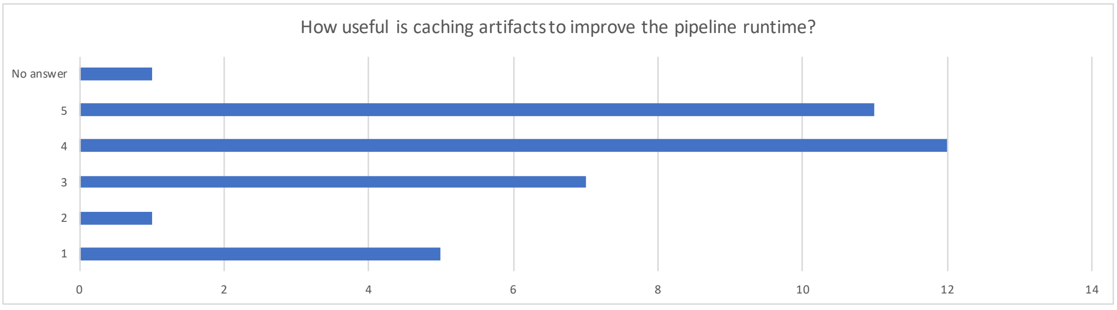


Figure 6.21.: Results How useful is caching artifacts to improve the pipeline runtime?

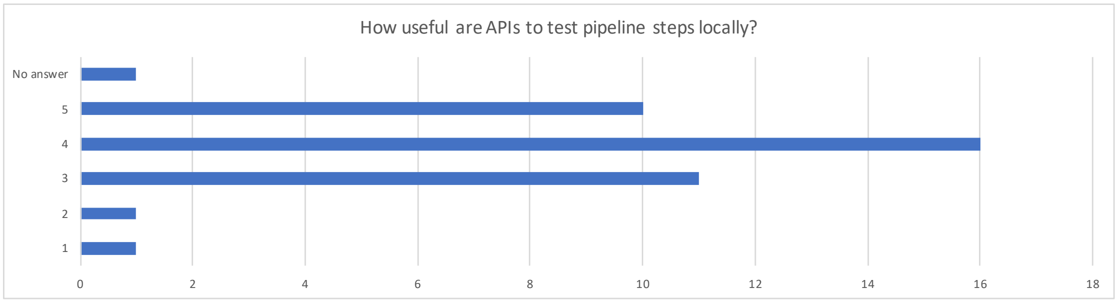


Figure 6.22.: Results How useful are APIs to test pipeline steps locally?

Operation	Count	Min	Max	Avg
Checkout and compile source code	27	0.5	120	10.28
Execute unit tests	27	1	60	7.48
Execute integration tests	27	1	120	18.81
Execute system tests	27	1	240	33.41
Deploy artifact to demo system	27	1	60	10.96
Runtime complete pipeline	30	5	600	75.63

Table 6.2.: Acceptable runtime for certain CD operations

6.5. Discussion and Summary

This section is dedicated to summarize and discuss the gathered findings from the second case study. It includes an evaluation of the collected requirements and the results of the survey.

6.5.1. Answers to Research Questions

The first research question is targeted at the identification of continuous delivery stakeholders. To find these stakeholders, employees of the KISTERS AG were interviewed and questioned regarding their continuous delivery requirements. If requirements exists, the role of the employee was added as a stakeholder. Furthermore the whole business unit was taken into consideration as potential stakeholder. By that, roles with no (known) or limited connection to the software development process were also considered. All in all, nine different stakeholders were identified and their requirements collected. Next to known continuous delivery stakeholders like software developers and DevOps engineers, multiple new roles were added. Firstly the security manager or security department. They would benefit by continuously evaluating the security aspects of the software during each build. For example checking external dependencies for security vulnerabilities. Another unconventional stakeholder is the cloud manager. This role aims to use the continuous delivery solution to enable cloud operation for the developed software. Next to automatic deployment, also the automatic management of cloud systems is required. This includes the provisioning and maintenance of nodes in a cluster. The desired goal is achieving "rapid elasticity" and "measured services" [BWZ15]. With rapid elasticity the capabilities or resources can be scaled up or down as demanded by the situation. Measured services is targeted at optimizing the available resources. Only provide resources (e.g. machines) which are needed to ensure operation. Lastly the sales department and project management were identified as new stakeholders. Booth roles are interested in the current state of the software regarding completed or upcoming features. Since the delivery system reevaluates the software during each build, continuous delivery could be extended to help these roles.

The second research question is concerned with the continuous delivery requirements of the stakeholders. Again these findings were gathered during the interviews (34 requirements were identified). In this section the requirements and stakeholders are investigated in a comprehensive way. For a list containing the individual requirements, see section 6.3. Table 6.3 provides a mapping between requirements and stakeholders. From this overview multiple conclusions can be made. First lets explore the stakeholder correlations. The developer stakeholders are the main target group for continuous delivery (20 requirements for each developer type). They get the most value out of it and use it in their everyday work. As shown in 6.3 there is almost no difference between frontend and backend developer present. The only variation in requirements is related to different graphical representations for information needs and testing methods. The next biggest stakeholder group is the DevOps engineers. Their requirements have more supporting characteristics since they are responsible is keeping the continuous delivery system up and running. The requirements for the software architect are almost all related to information needs and their representation. Most of the new stakeholders (cloud, security, project and sales management) only issued very specific and few requirements. They are not fully involved in the software development process and only need support through the continuous delivery system rarely. But this does not diminish the benefit they would get from using CD. It only states the limited area of applicability. The requirements that were stated the most were **Req.16, Req.23, Req.07 and Req.01**. Two of these requirements were concerned with the feedback functionality of the continuous delivery system, which highlights the importance of feedback for the stakeholders. This is then followed by the need for a graphical representation of the delivery process. Finally the speed of the delivery system and the ability to provide fast releases was demanded.

To support an eventual realization of the requirements, they were mapped with their relevant components from the pipeline delivery model (see 2.2). This mapping is shown in table 6.4. As shown, most of the requirements can be satisfied by adapting the continuous delivery system itself. This mostly includes adding the required function or information. It also suggests, the preference of stakeholders to formulate their needs in a functional way. Another observation can be made regarding the relation between delivery model and process. Requirements that involve the process, also concern the model but not vice versa. In general the delivery process itself is related to the least amount of requirements.

6.5.2. Observations

These observations were made during the survey which was executed after the interviews. With this survey the most prominent requirements were further explored and additional findings gained. Main findings of the survey were the confirmation of feedback and speed as the most important aspects of a delivery system. Contrary to the high number of stakeholders which demanded the requirement **Req.16** during the interviews, the survey showed a reduced need for it. Another important aspect of the survey is the evaluation of acceptable system response times. Many stakeholders stated a speed increase for the continuous delivery system is needed. But they rarely could quantify the amount of the increase. The survey asked the participants to specify acceptable times for certain pipeline steps and activities. These results are shown in 6.2 and can now be used to adapt the existing CD solution accordingly.

6.5.3. Threads to Validity

As with the first case study, the threads to validity need to be investigated. The first thread is a possible bias while selecting the interview participants. Especially if new continuous delivery stakeholder should be identified. To counter this, the selection included the whole business unit as a whole (see 5.1 for selection strategy). It increased the amount of work, but no possibly interesting employee could be overlooked. Another thread was the evaluation and interpretation of the interview results by the researcher. While performing these tasks, the researcher could introduce his opinions and biases into

Req.	Frontend	Backend	Architect	DevOps	Security	Cloud	Quality	Sales	Project	Sum
Req.01	X	X		X			X			4
Req.02				X	X					2
Req.03	X	X					X			3
Req.04	X	X								2
Req.05	X	X								2
Req.06				X						1
Req.07	X	X	X	X			X			5
Req.08				X						1
Req.09				X						1
Req.10	X	X		X						3
Req.11				X						1
Req.12				X						1
Req.13		X	X							2
Req.14	X	X	X	X						4
Req.15				X		X				2
Req.16	X	X	X	X		X	X			6
Req.17	X	X					X			3
Req.18	X	X	X	X						4
Req.19	X	X					X			3
Req.20	X	X	X	X						4
Req.21	X	X					X			3
Req.22	X	X		X						3
Req.23	X	X	X	X		X	X			6
Req.24	X									1
Req.25	X	X					X			3
Req.26	X									1
Req.27	X									1
Req.28	X	X	X				X			4
Req.29		X								1
Req.30		X		X						2
Req.31								X	X	2
Req.32						X				1
Req.33					X					1
Req.34					X					1
Sum	20	20	8	17	3	4	10	1	1	

Table 6.3.: Requirements mapped on the identified stakeholders

the results. To avoid this, the final transcripts were send to the participants and they had to confirm the answers and feedback.

Requirement	Delivery Model	Delivery System	Delivery Process	Sum of Components
Req.01	X	X	X	3
Req.02	X	X		2
Req.03		X		1
Req.04	X	X		2
Req.05	X	X		2
Req.06		X		1
Req.07		X		1
Req.08		X		1
Req.09		X		1
Req.10		X		1
Req.11		X		1
Req.12		X		1
Req.13		X		1
Req.14	X	X		2
Req.15	X			1
Req.16		X		1
Req.17		X		1
Req.18	X	X		2
Req.19		X		1
Req.20	X	X	X	3
Req.21		X		1
Req.22	X		X	2
Req.23		X		1
Req.24		X		1
Req.25		X	X	2
Req.26		X		1
Req.27		X	X	2
Req.28	X			1
Req.29	X		X	2
Req.30		X		1
Req.31	X	X	X	3
Req.32		X		1
Req.33	X	X		2
Req.34		X		1
Sum	13	30	7	

Table 6.4.: Requirements mapped to their Components from the Delivery Pipeline Model

7. Case Study 3: Continuous Delivery Modelling

Contents

7.1. Study Design And Execution 82
7.2. Process-Based modelling with BPMN 83
7.3. Process-Based modelling with DSL 84
7.4. State-Based modelling with DSL 85
7.4.1. State-Based modelling 86
7.4.2. Evaluation State-Based Modelling 88
7.5. Discussion and Summary 89
7.5.1. Answers to Research Questions 89
7.5.2. Threads to Validity 90

To finalize this thesis, the third case study was used to gain more information about pipeline modelling in general and especially the modelling paradigms present in JARVIS. As described in chapter 2.3, process oriented modelling is currently implemented in JARVIS. This paradigm sees the pipeline as a process of steps that are executed in sequence (or parallel if possible). Produced artifacts are used as input for the following steps and the last artifact is considered the result of the pipeline. To create delivery models in this process oriented way, JARVIS offers two different modelling techniques. The first technique is creating the model in text-form via custom DSL. This was the first method implemented in JARVIS and is considered the standard modelling approach. Based on this, the next extension was graphical modelling with BPMN. With this method, models are created by declaring different activities as nodes and connecting them to result in an execution graph. The first method is supported by the smart planning component of JARVIS, which links dependent activities automatically and optimizes the pipeline in regard to runtime. During this chapter, these functions are now investigated and mapped onto real world continuous delivery challenges. Additionally it is evaluated if they provide added value to that challenges. While modelling the reference project pipeline in chapter 5, multiple shortcomings in the smart planning function were discovered. To counter these, a new modelling paradigm was designed. The so called state-based modelling is also evaluated in this study. In the next section the overall case study design is presented, followed by the results for each individual modelling method. Finally a summary is given to conclude the chapter.

7.1. Study Design And Execution

This chapter starts of by describing the overall case study design and the methodology used to generate the findings. This case study is concerned with the evaluation of the the experimental modelling features of JARVIS and their performance in a productive environment. To that end, the following research questions were defined and worked on during this case study.

o3.R1 Can the experimental modelling features be mapped on common problems in continuous delivery?

o3.R2 Do the experimental features solve or improve on those problems?

As with the previous case studies, the chosen context for this was the KISTERS AG. Due to time constraints, not all experimental features of JARVIS were evaluated in this study. Because of their central position in continuous delivery, the modelling paradigms and methods, present in JARVIS, were chosen for evaluation. This includes process based modelling by DSL, as well as process based modelling with the help of the BPMN notation. Furthermore the so called state-based modelling is evaluated as a potential new paradigm in JARVIS. More information about state-based modelling is provided in the following sections. As shown in figure 7.1 this case study was designed as an embedded case study on the case "CD Modelling".

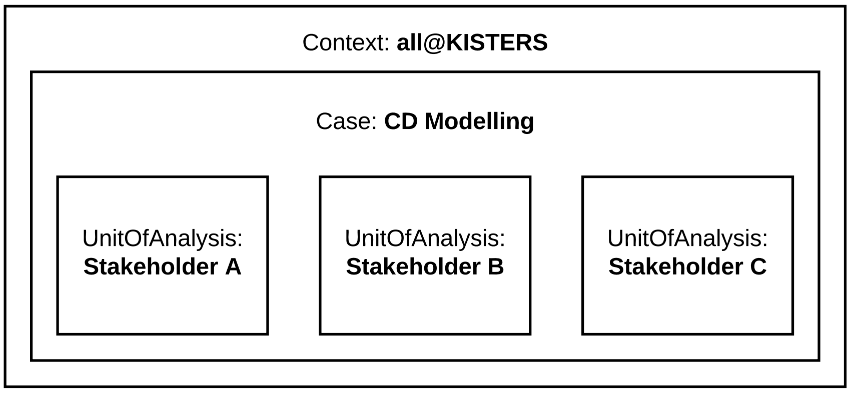

Context: **all@KISTERS**
Case: **CD Modelling**
UnitOfAnalysis: **Stakeholder A**
UnitOfAnalysis: **Stakeholder B**
UnitOfAnalysis: **Stakeholder C**

Figure 7.1.: Third study, embedded case study with CD modelling as case

To evaluate the approaches and answer the research questions **o3.R1 - o3.R2**, the following process was executed. First, suitable stakeholders were selected to perform the evaluation. Out of the stakeholders found during the second case study (see 6), the DevOps engineers were selected to participate in this study. The reasoning behind this was the heavy involvement of these persons in creating and maintaining delivery models. This increased knowledge was needed, to investigate the modelling methods in every detail and especially the ability, to compare them to existing solutions, was helpful. The participants were then asked to model the KISTERS reference project (see 5.2), which they were familiar with, with the help of all three methods. Each approach was then individually evaluated and rated by the participants. This feedback then was collected and processed by the author of this thesis. It represents the answers to the specified research questions.

7.2. Process-Based modelling with BPMN

To start of, the participants had to model the reference project CD process oriented by using the BPMN notation. During the work of [Wil18], JARVIS was extended by the possibility to model pipelines with BPMN. The goal of this work was the reduction of complexity for delivery models, which in turn reduces the maintenance effort for those models. Additionally the common knowledge of BPMN should improve the user acceptance and decrease the initial hurdle to work with the delivery model for new user. To answer the research question **o3.R1**, the participants were initially asked to specify the continuous delivery challenge/task they would try to tackle with this method. In this instance, the DevOps engineers stated that they would use this method to gain more insight and overview regarding their delivery models. Furthermore, they assessed this method would enable more/new stakeholders to customize their models and participate in the CD process. Lastly the resulting activity graph makes it easier to talk about the model. Contrary to a textual representation, the model is directly encoded in a visual form, which is more suitable for discussions or meetings. Resulting of this feedback, the method was mapped on two continuous delivery challenges by the participants. The first problem is the complexity of the delivery models. Representing the models in the BPMN notation was recognized as a way to reduce that complexity and enable easier access for new user. The other problem was a missing graphical representation of the delivery model itself. If process oriented modelling with BPMN improves on those challenges is evaluated in the next section.

Finding out, if BPMN modelling offers any contributions for solving the previously mentioned challenges, required the participants to model the KISTERS reference project pipeline in JARVIS with the help of BPMN. The first finding is related to the general overview, the BPMN models provide. When the model gets larger, the participants reported a significant decrease in clarity of the model. This was mainly caused by the fact that the implementation uses the same shape for each activity. Even with labeled shapes, finding the correct activity was a hard tasks. In general, modelling the whole process was assessed as very time intensive work. Especially configuring every activity with the required parameters is very time consuming. The participants rated the process modelling with BPMN as slower than using a traditional DSL. Another feedback the user reported was regarding the use of the smart planner. If a pipeline is modelled via BPMN the user does not expect the system to modify the resulting process in any way. If BPMN is used, the smart planner should not intervene and optimize the model. This results in the maximum knowledge, required by the user regarding the delivery process of all existing modelling methods. Nevertheless the participants rated the BPMN method as suitable for beginner users, since the notation is well known and no syntax or commands need to be learned. Additionally the graphic models were very suitable for visualizing the models and identifying eventual problems. The first answer to research question **o3.R2** is as follows. Modelling CD with the help of BPMN and in a process oriented way, does reduce the complexity of the delivery models slightly. But since every step of the process has to be included by the user, the resulting graphs can get big and confusing. Also the process knowledge needed, by the user, must be very large, since support through the smart planner is not possible. BPMN models show their strength in smaller projects and when multiple persons are involved. The ability to visualize and then discuss the models was evaluated as very important by the participants. Another improvement could be to use a custom editor for the creation of the models. Currently a generic BPMN editor [Cam] is used, which has a lot of unnecessary features. Reducing this to the essentials and eventually introducing custom shapes for activities could help with the general overview.

7.3. Process-Based modelling with DSL

This section is about the second modelling method, which is process-based modelling with the custom JARVIS DSL. In JARVIS a custom DSL was introduced which utilizes the YAML syntax. This was introduced as the default approach to create delivery models. Listing 26 shows an extract of such a model.

```
planner:
expand: false
stages:
- name: buildClientServer
    transformations:
    # --- Server --- #
    - server-checkout
    - server-versioning
transformations:
### ---- Server ---- ###
- name: server-checkout
    service: git-service
    command: checkout
    tag: "server"
    parameters:
    repositoryUri: https://myRepoUrl.de
    branch: master
    credentials:
        type: plain
        username: myUserName
        password: myPwd
- name: server-versioning
    service: kisters-maven-service
    command: versioning
    tag: "server"
```

Source Code 7.1: Extract from a JARVIS delivery model

Analogous to the previous method, first the research question **o3.R2** is worked on. The participants stated that they would map this method to the general requirement to specify the delivery model. The specific challenges behind this are the resulting size, complexity and comprehensibility of the delivery models. In the next step the participants modelled the reference project pipeline again. This time they created the delivery model with the DSL in a processed based approach. They observed the same problem regarding the size of the model as with BPMN modelling. If every step of the process has to be explicitly stated, the models get larger and thereby harder to understand and maintain. To counter this, a composition functionality for activities was proposed by the participants. A possible extension of JARVIS with such a function was already described in chapter 5.8. Furthermore the need for tooling support was highlighted. Productive modelling with a DSL requires autocomplete and syntax highlighting to increase productivity. This can be achieved through a custom editor or an IDE. Again the process knowledge needed, to work with the delivery models, is significant. But it was assessed as less than with BPMN modelling. This was due to the fact that the smart planner was used. Only the activities needed where specified in the models, but their execution order and dependencies were determined by the smart planner. The comprehensibility was assessed as very high and as an improvement to existing modelling methods. Here the declarative approach, inherit with the use of the YAML syntax, helps to structure the models. No conditional commands or loops are present which also decreases the complexity. All in all the process based modelling with the JARVIS DSL was evaluated as a slight improvement to the specified challenges. By using the YAML syntax and the combination with the smart planner the complexity of delivery models decrease and the comprehensibility increases.

7.4. State-Based modelling with DSL

While executing the first case study (see 5), a problem with the smart planner was made obvious. The automatic chaining of activities based on their input and output types only worked on simple delivery models. If, for example in the KISTERS reference project model, an activity is used twice, the smart planner can not resolve which following activity to connect. Figure 7.2 illustrates this problem. Activity A1 and A2 are the same activity used twice in the delivery model. Both produce an artifact of type A. The receiving activity B1 on the other hand needs an artifact of the type A as input. Normally the smart planner would chain the activities based on their input/output signature to help the user create the optimal delivery process. Because of ambiguity this is not possible in the shown case. Possible solutions would include duplicating the activities and defining new input/output types. This was discarded due to increased model size and development effort to create the activities. To solve this problem a new modelling paradigm was designed. It is described in this section, followed by a theoretical evaluation in the context of this case study.

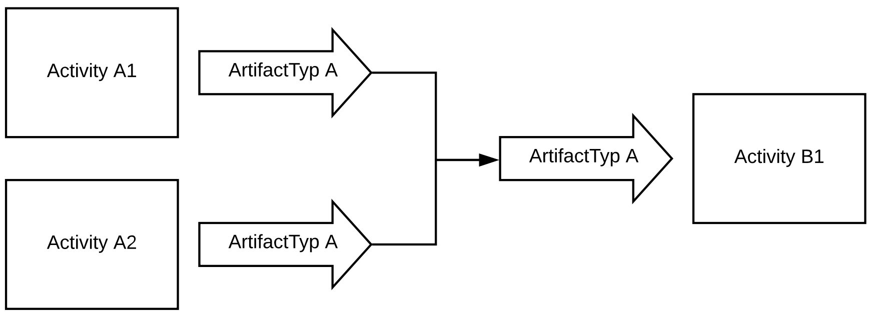


Figure 7.2.: Unsolvable model for the smart planner

7.4.1. State-Based modelling

In this section the idea of state based modelling is introduced and the concept explained. Most continuous delivery models are designed with a process in mind. The user knows what actions should be applied and the order in which they are executed. This is transferred directly into the respective model. State-based modelling tries to shift this focus away from the process and towards the artifact itself. Instead of describing the way to reach the desired result, the user models the desired state of the final artifact and not the process to get to it. The process is not encoded in the model but in the implementation of the JARVIS activities. Since JARVIS works with custom activities anyway, this is no overhead. In the following, the concept is explained by means of a small example. An application consisting of two individual components which are bundled into one final artifact. Both components are JAVA applications which are build by maven with the same JARVIS activities. The component C is considered as the final artifact after the "Maven Package" activity was executed. Figure 7.3 shows a graphical representation of the pipeline that is used to illustrate the state-based approach.

In state-based modelling, the model is divided into components. Each of this component can have multiple states and each state is produced by an activity. In this example the component C can achieve the states "workspace" and "packaged". The state "workspace" is produced by the checkout activity and "packaged" by "Maven Package". To model this pipeline, only the desired final state needs to be specified. Because the state "package" is dependant to "workspace", the user does not need to model the checkout activity explicitly. This is different to process based modelling where the whole process must be outlined. Important for state-based modelling is the use of some kind of planning service, like integrated into JARVIS. This service is responsible for resolving all needed previous states to reach a desired end-state. In listing 31 the whole state-based model

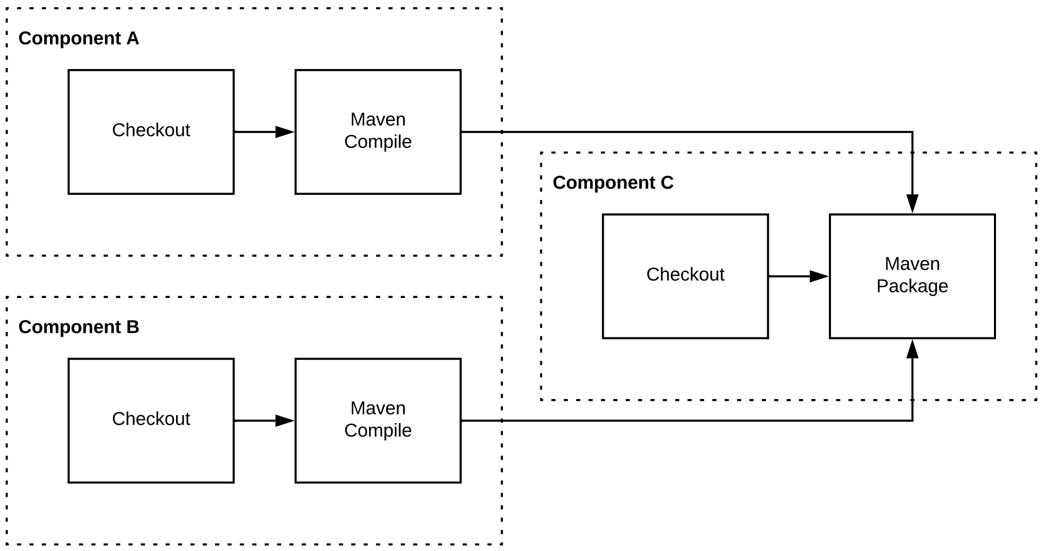


Figure 7.3.: Pipeline for example application

for the pipeline in 7.3 is given. The model includes each component that the pipeline contains. Since component C in state "packaged" is the desired result artifact for this pipeline, it is also stated. As mentioned (and shown in 7.3) C consists of the component A and B. These dependencies must also be declared to help the planner resolve them. In this case the "Maven Package" activity has two input parameters which demand artifacts with the state "classes". This state is produced by the "Maven Compile" activity, which is present in both components A and B. By using the artifact states, the smart planner can now backtrack to determine which activity from the component A (and B) produces the needed state. In this case the "Maven Compile" activity would be chosen, which in turn requires an input artifact with the state "workspace" (produced by the Checkout activity). Following this approach, the smart planner can resolve huge parts of the pipeline automatically, resulting in minimal process knowledge required by the user. Additionally the problem with duplicated activities is solved. The "Checkout" activity was used three and the "Maven Compile" activity two times. But this concept also has some limitations. For example activities can only be used once per component. This was assessed as an acceptable trade against using them only once per pipeline.

```
- name: componentA
source:
    git:
    repositoryUri: https://repoA.de
    branch: master
    credentials:
        username: userName
        password: gitPwd
```

```

- name: componentB
source:
    git:
    repositoryUri: https://repoB.de
    branch: master
    credentials:
        username: userName
        password: gitPwd

- name: componentC
source:
    git:
    repositoryUri: https://repoC.de
    branch: master
    credentials:
        username: userName
        password: gitPwd
state:
    package:
        firstComponent: componentA
        secondComponent: componentB
```

Source Code 7.2: Pipeline Model for example project state-based

7.4.2. Evaluation State-Based Modelling

As previously mentioned, the state-based modelling approach was only designed as a concept during this thesis. Subsequently there was no possibility for the participants to fully test the approach in this case study. Therefore the evaluation was limited to a theoretical modelling session without support through JARVIS. The participants were told what the ideas behind state-based modelling is and how its suppose to work. Based on this information, they were asked to model the reference project in a state-based way. The findings related to the research questions of this study are presented in this section. As with the first two methods, the initial step is investigating research question **o3.R1**. For this, the participants mapped state-based modelling with a DSL onto the following CD challenges. The first challenge was the size and complexity of the delivery models. At first glance it seems the approach could reduce those attributes and make working with delivery models more easy. Another aspect of CD, that could be improved by state-based modelling, is the needed process knowledge. The less internal knowledge the user need to create their delivery models, the more they are incited to work with them. After the targeted CD challenges were identified, the participants started with the theoretical modelling of the KISTERS reference project delivery model. Hereby the following findings were gained, related to research question **o3.R2**. Initially the new modelling method requires rethinking on the users part. Normally user think in processes while creating pipelines. To see the pipeline as a state description for components is a new way to interact with delivery models. If the initial hurdle was passed, the modelling was assessed as less complex by the participants (in comparison to process based modelling). The user does not need to know each step of the process involved to get the artifact in the desired state. It suffices to know "what" to get and in which state. By that, even unexperienced user can be able to use the CD on their own. Another finding was about the success to reduce overall model complexity. As seen in 31, the models are relative small in relation to what they represent. Especially the scalability of those models is very good. If more pipeline-steps are introduced, the model does not necessary get bigger. All steps inserted before the final state can be resolved automatically and need not to be mentioned in the delivery-model. Based on this evaluation, the second research question is considered answered for state-based modelling. Of course this needs further investigation after the real implementation was introduced in the JARVIS eco-system.

7.5. Discussion and Summary

This section is dedicated to summarize and discuss the gathered findings from the third case study. Since the answer to the research question was largely the result of an observation, no dedicated section for the observations is present.

7.5.1. Answers to Research Questions

Both research questions are concerned with the experimental features of JARVIS and how they perform in a real world scenario. Because of time limitations, only the modelling approaches, present in JARVIS, were evaluated. This included the process oriented modelling with BPMN and custom DSL as well as a new paradigm. The new paradigm is called state based modelling and was designed during this thesis. To evaluate the paradigms, they were first mapped on common continuous delivery problems/challenges by experts. These experts then used the approach to model the reference project pipeline from the first case study and give feedback regarding the viability of these methods. The process based modelling with BPMN was assessed as suitable solution for small to medium sized delivery models. Furthermore it was rated as time intensive and required a lot of process knowledge. The advantages of this method were a graphical representation of the model which enables the user to talk about and discuss it. This can be especially useful if multiple persons are involved in maintaining the delivery model.

The next method was the process based modelling with a custom JARVIS DSL. This was evaluated as a suitable general purpose way to work with delivery models. It was faster than BPMN and the provided help of the smart planner reduced the needed process knowledge significantly. One identified drawback of this methods was again the size and complexity of the delivery models. For large models (like the reference project) the textual representation has limits. After reaching a certain size the model becomes hard to maintain and understand. All in all the approach was assessed as suitable for most scenarios if enough tooling support is present.

The new paradigm is the state-based modelling. The core idea is not to model the delivery process itself but the desired end-state of an artifact. The needed process to reach this artifact, is resolved by a planning component and only needs human input for dependencies between components. By using this method the required process knowledge was reduced significantly and also the model size was decreased. Nevertheless a new way of thinking continuous delivery is required by the user. User traditionally think in process steps and using a state-based approach took some time getting use to. But after the initial hurdle was taken, the user were able to work faster with the state-based method than with the process-based counterparts. Therefore the initial evaluation of state-based modelling was a success and further research into this topics is required.

7.5.2. Threads to Validity

There are two main threads to validity in this case study. The first ist the choice regarding the experts performing the study. Like in the second study, the participants were chosen on a non personal basis. Persons that were identified as DevOps engineers were chosen to perform this study. Additionally they had no previous knowledge about JARVIS and therefore went into the study unbiased. The next thread was the introduction of JARVIS by the researcher. It had to be as neutral as possible and only the necessary information about the modelling were given. Still the chance to have given too much (or little) background information is not zero. But this was accepted for this study because it was only considered initial research into the experimental features of JARVIS.

8. Summary and Future Work

Contents

8.1. Summary 91
8.2. Future Work 93

This chapter is used to provide a summary about the performed case study and the gained results. Additionally is present ideas for future work in the field of the JARVIS continuous delivery reference architecture.

8.1. Summary

During the thesis the continuous delivery reference architecture JARVIS was evaluated regarding its productive performance and usability. To achieve this, a case study was designed to investigate JARVIS. Following the generals rules [RH09] for case study design, three objectives for this study were specified. These objectives are listed below. Furthermore the case study was executed in collaboration with an industrial partner of the SWC. The KISTERS AG (see 3) provided the productive environment and context for this work.

o1 Make a qualitative assessment regarding JARVIS as substitution for established continuous delivery solutions

o2 Make a qualitative assessment regarding the usability of JARVIS on a comprehensive list on stakeholders

o3 Make a qualitative assessment if the experimental features of JARVIS solve common continuous delivery problems

To further structure the thesis, the overall case study was divided into three smaller case studies. Each targeting one individual objective. The first case study was investigating the basic functionality of JARVIS and ultimately the ability to replace an existing delivery system with one following the JARVIS architecture. The way to determine this ability was to take a fully functional delivery solution and reconstruct it in a JARVIS based system. For this the KISTERS reference project was chosen as a suitable test subject since it represents all technology and design choices made by the company. It current continuous delivery solution (currently managed by Jenkins CI) was then modelled in JARVIS. During the process multiple relevant aspects for a delivery system were evaluated to answer the designated objective. Firstly the capability of JARVIS to provide basic delivery process steps (activities) and integrate new technology into the infrastructure. All needed activities (for the reference project) were developed and integrated into JARVIS in addition to the required technologies. Therefore this requirement was confirmed. The next requirement JARVIS needed to satisfy was the ability to handle complex delivery models and provide ways to work with them. As shown in 5.5 the full reference project delivery pipeline was modelled in JARVIS and produced the same final artifact than the original. Hence this requirement was also satisfied. Lastly the answer towards the ability to use JARVIS as a productive delivery system was worked on. Because of the current development state of JARVIS an unconditional use in a productive scenario can not be suggested. All the concepts needed to achieve the core functions (and more) of a system like Jenkins CI are present and working as expected. But many convenience functionality for every day work, like displayed information and user management, are missing. But from a conceptual point of view, a JARVIS based system provides all necessary features/functions to be used productively. Only the current reference implementation needs to be extended. During this first case study two major observations regarding working with JARVIS were made. The first is the identification of service classes. Three service types/classes were identified and design patterns for this services were designed. This should improve the overall quality of activity-service implementation. The next observation was related to the modelling. Because the reference project delivery-models are extensive, the need for activity composition was found. Future implementations of JARVIS should enable the user to compose individual activities into new activities. This reduces model size and provides huge usability benefits.

While the first case study investigated if the current solution could be represented in JARVIS, the next study was about possible improvements to that solution. But before the JARVIS system could be evaluated against these improvements it was necessary to know what there was to improve. To answer this, the second case study firstly identified continuous delivery stakeholder in a productive context like the KISTERS AG. This was done by interviewing key personnel from the business unit and gathering their continuous delivery requirements. These interviews were then evaluated and processed into a comprehensive list of stakeholders and requirements (see 6). Following this, the requirements were evaluated against JARVIS to identify concepts in which a JARVIS based system would solve those. For most of the requirements a concept or solution was present to satisfy them. So the overall assessment of JARVIS usability was positive. To further increase the knowledge regarding these requirements a survey was designed and executed to gain even more information. This survey was used to specify acceptable response time for a continuous delivery system and evaluate the value of certain requirements, identified during the interviews.

The last case study looked into one experimental feature of JARVIS. This feature was the modelling of delivery processes in JARVIS. Currently two approaches are present in a JARVIS based delivery system. These approaches are process based modelling with BPMN notation and custom JARVIS DSL. Both approaches were evaluated by experts regarding their real world applicability and usability. These experts modelled the reference project pipeline with those methods and then gave feedback. Both methods were found suitable but not without flaw. The handling of large delivery models was confusing and complex also the process knowledge required by the user was significant. To improve on these problems, a new paradigm for modelling delivery processes was developed. It is called "state-based" modelling and was also evaluated in the same scenario. In state based modelling a pipeline is not described as a process. So no sequence of steps is modelled or declared. A state based delivery model consists of components and only the desired state of the final artifact is declared (in addition to component state dependencies). The underlying process to bring the artifact in this state is automatically resolved by the delivery system. The result of that evaluation was very positive. The knowledge decrease to work with the models was noticeable and additionally the size of the models was also reduced. State based modelling was assessed as a suitable alternative to the already existing solutions and offers potential which has to be explored in future work.

8.2. Future Work

Because of the scope of this thesis and the time available, not all concepts or ideas could have been implemented during this thesis. This section provides suggestions and inspiration for future work building on the results of this case study.

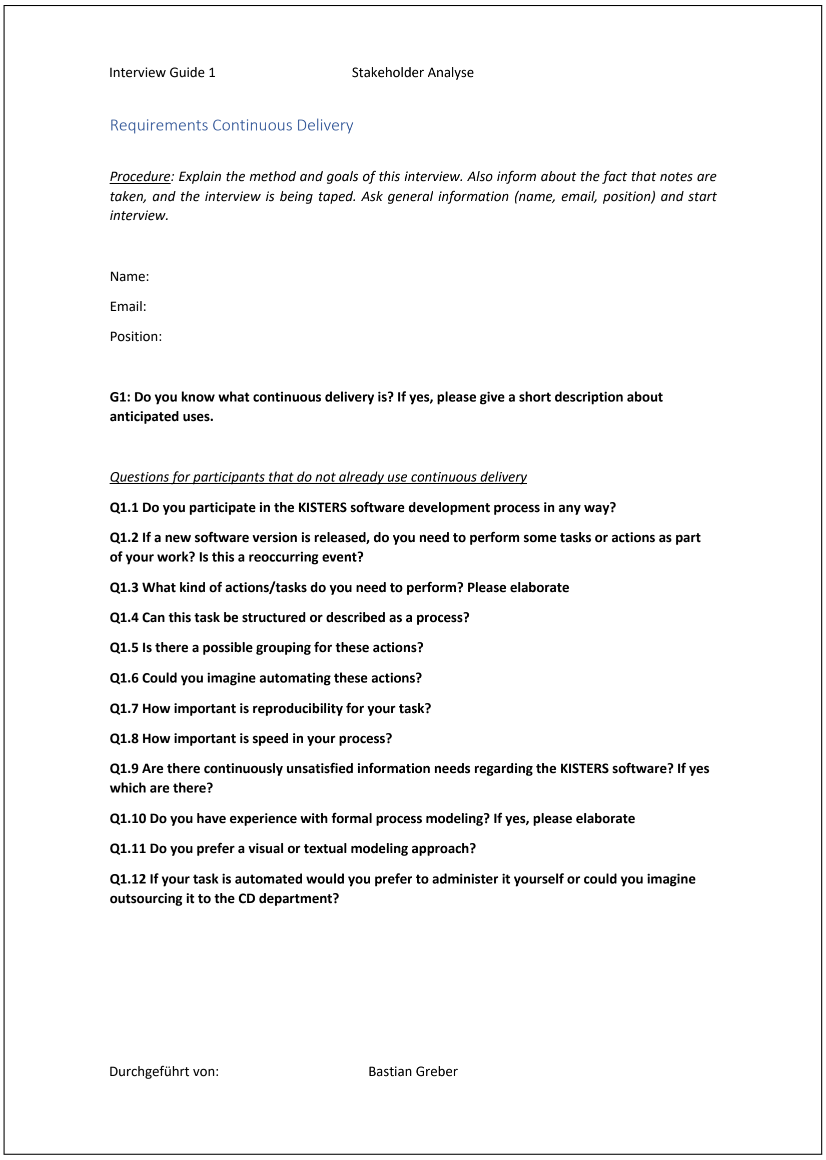
Service Starter Based on the identified activity-service classes in 5.7 and the designed patterns for them, a service starter would further decrease the effort to develop activity-services. Because the current reference implementation of JARVIS is developed in Spring Boot, a Spring Boot Starter could be developed for each service type. If the implementation is changed to a different base technology the another template-solution to quickly develop new services based on the found patterns should be introduced.

Composite Activities As stated in 5.8 the composition of activities would reduce the size of the delivery models and also decrease the complexity. A possible concept for integrating this functionality was designed during this work. The next step would be the development and integration of this feature.

Confirm Requirements The found requirements (during the study and interviews) were collected in one productive environment. To further increase the quality of the research and identify eventual patterns or contradictions, the study should be repeated in a different company. These second study would act as a control group and maybe produce further requirements and stakeholders.

State Based Modelling The idea of state-based modelling was only introduced during this thesis. So more research into this topic is needed to grasp the full benefits and identify eventual limitations. During the evaluation in chapter 7.4.1 no concrete implementation was present in JARVIS. The modelling was performed without executing the resulting process at the end. The next step should be the implementation of state-based modelling into JARVIS. After this is done, more research into its capabilities with other projects should be conducted.

A. Interview Guides


Interview Guide 1 Stakeholder Analyse

Requirements Continuous Delivery

Procedure: Explain the method and goals of this interview. Also inform about the fact that notes are taken, and the interview is being taped. Ask general information (name, email, position) and start interview.

Name:

Email:

Position:

G1: Do you know what continuous delivery is? If yes, please give a short description about anticipated uses.

Questions for participants that do not already use continuous delivery

Q1.1 Do you participate in the KISTERS software development process in any way?

Q1.2 If a new software version is released, do you need to perform some tasks or actions as part of your work? Is this a reoccurring event?

Q1.3 What kind of actions/tasks do you need to perform? Please elaborate

Q1.4 Can this task be structured or described as a process?

Q1.5 Is there a possible grouping for these actions?

Q1.6 Could you imagine automating these actions?

Q1.7 How important is reproducibility for your task?

Q1.8 How important is speed in your process?

Q1.9 Are there continuously unsatisfied information needs regarding the KISTERS software? If yes which are there?

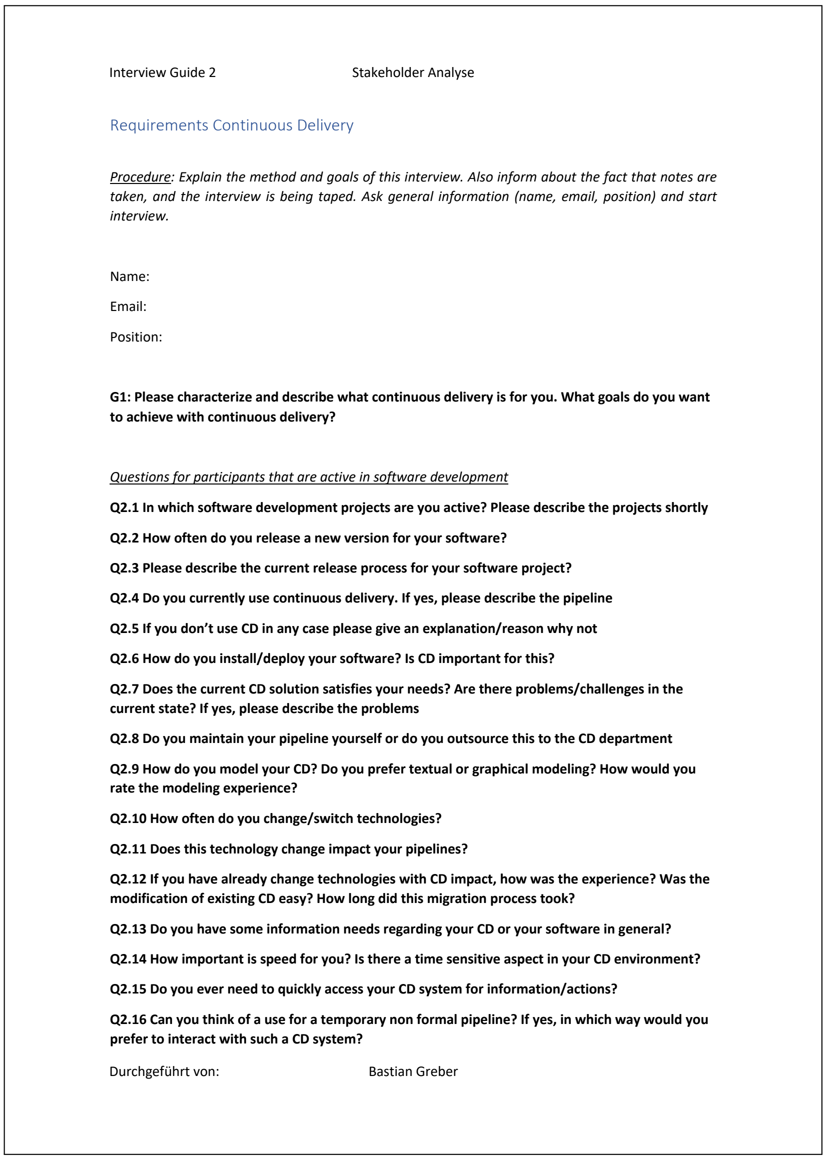
Q1.10 Do you have experience with formal process modeling? If yes, please elaborate

Q1.11 Do you prefer a visual or textual modeling approach?

Q1.12 If your task is automated would you prefer to administer it yourself or could you imagine outsourcing it to the CD department?

Durchgeführt von: Bastian Greber

Figure A.1.: Interview Guides page 1 of 5



Interview Guide 2 Stakeholder Analyse

Requirements Continuous Delivery

<u>Procedure</u>: Explain the method and goals of this interview. Also inform about the fact that notes are taken, and the interview is being taped. Ask general information (name, email, position) and start interview.

Name:

Email:

Position:

G1: Please characterize and describe what continuous delivery is for you. What goals do you want to achieve with continuous delivery?

<u>Questions for participants that are active in software development</u>

Q2.1 In which software development projects are you active? Please describe the projects shortly

Q2.2 How often do you release a new version for your software?

Q2.3 Please describe the current release process for your software project?

Q2.4 Do you currently use continuous delivery. If yes, please describe the pipeline

Q2.5 If you don't use CD in any case please give an explanation/reason why not

Q2.6 How do you install/deploy your software? Is CD important for this?

Q2.7 Does the current CD solution satisfies your needs? Are there problems/challenges in the current state? If yes, please describe the problems

Q2.8 Do you maintain your pipeline yourself or do you outsource this to the CD department

Q2.9 How do you model your CD? Do you prefer textual or graphical modeling? How would you rate the modeling experience?

Q2.10 How often do you change/switch technologies?

Q2.11 Does this technology change impact your pipelines?

Q2.12 If you have already change technologies with CD impact, how was the experience? Was the modification of existing CD easy? How long did this migration process took?

Q2.13 Do you have some information needs regarding your CD or your software in general?

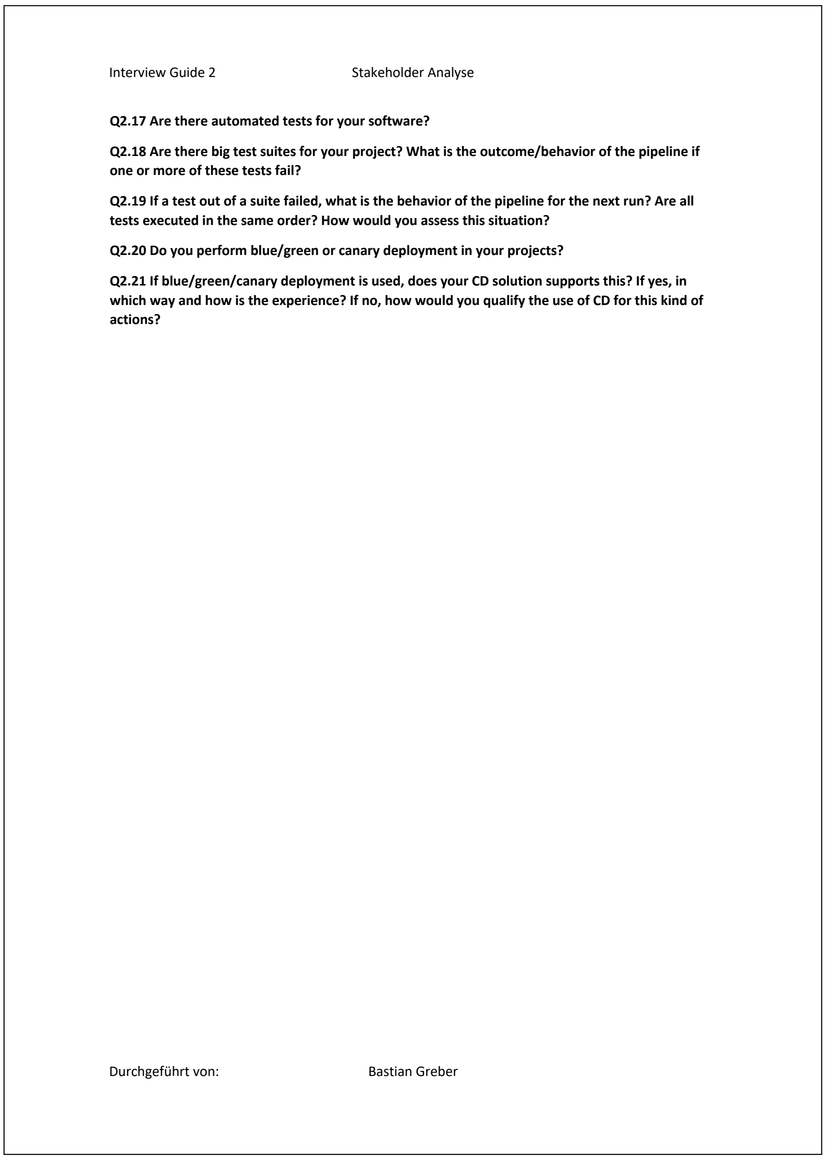
Q2.14 How important is speed for you? Is there a time sensitive aspect in your CD environment?

Q2.15 Do you ever need to quickly access your CD system for information/actions?

Q2.16 Can you think of a use for a temporary non formal pipeline? If yes, in which way would you prefer to interact with such a CD system?

Durchgeführt von: Bastian Greber

Figure A.2.: Interview Guides page 2 of 5


Interview Guide 2 Stakeholder Analyse

Q2.17 Are there automated tests for your software?

Q2.18 Are there big test suites for your project? What is the outcome/behavior of the pipeline if one or more of these tests fail?

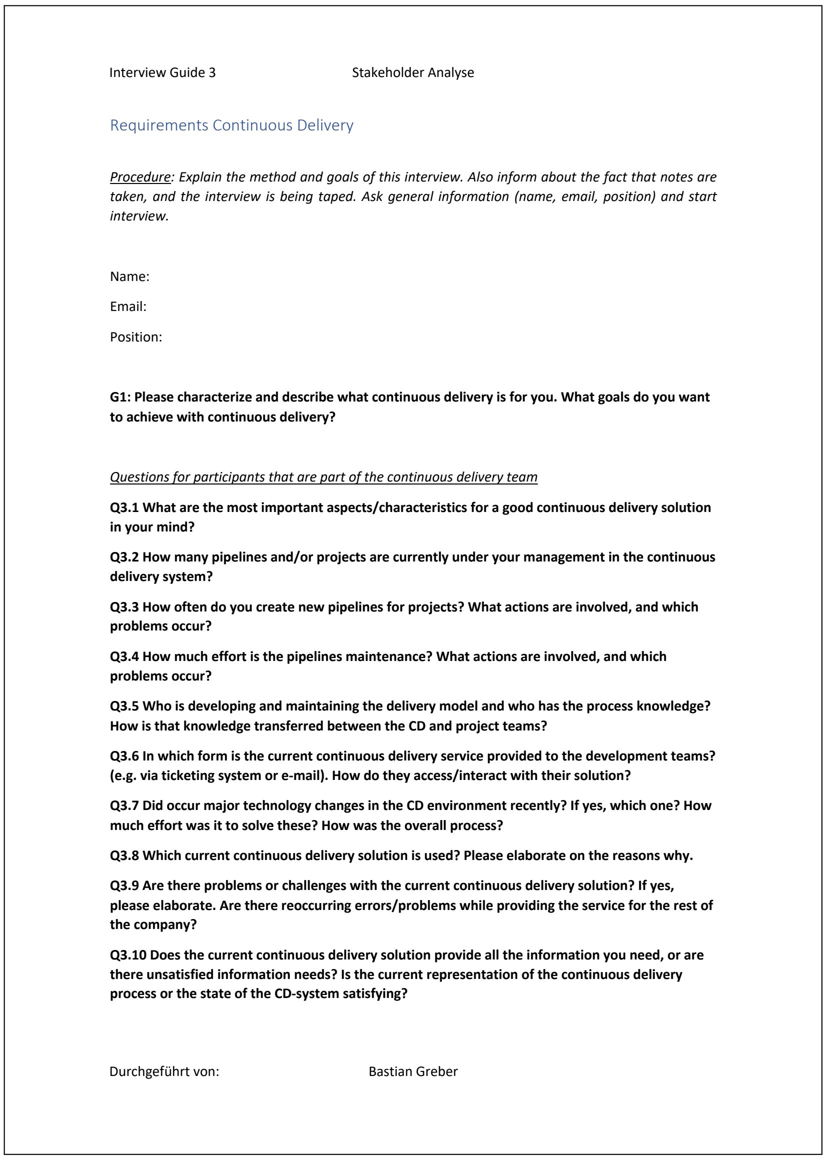
Q2.19 If a test out of a suite failed, what is the behavior of the pipeline for the next run? Are all tests executed in the same order? How would you assess this situation?

Q2.20 Do you perform blue/green or canary deployment in your projects?

Q2.21 If blue/green/canary deployment is used, does your CD solution supports this? If yes, in which way and how is the experience? If no, how would you qualify the use of CD for this kind of actions?

Durchgeführt von: Bastian Greber

Figure A.3.: Interview Guides page 3 of 5



Interview Guide 3 Stakeholder Analyse

Requirements Continuous Delivery

<u>Procedure</u>: Explain the method and goals of this interview. Also inform about the fact that notes are taken, and the interview is being taped. Ask general information (name, email, position) and start interview.

Name:

Email:

Position:

G1: Please characterize and describe what continuous delivery is for you. What goals do you want to achieve with continuous delivery?

<u>Questions for participants that are part of the continuous delivery team</u>

Q3.1 What are the most important aspects/characteristics for a good continuous delivery solution in your mind?

Q3.2 How many pipelines and/or projects are currently under your management in the continuous delivery system?

Q3.3 How often do you create new pipelines for projects? What actions are involved, and which problems occur?

Q3.4 How much effort is the pipelines maintenance? What actions are involved, and which problems occur?

Q3.5 Who is developing and maintaining the delivery model and who has the process knowledge? How is that knowledge transferred between the CD and project teams?

Q3.6 In which form is the current continuous delivery service provided to the development teams? (e.g. via ticketing system or e-mail). How do they access/interact with their solution?

Q3.7 Did occur major technology changes in the CD environment recently? If yes, which one? How much effort was it to solve these? How was the overall process?

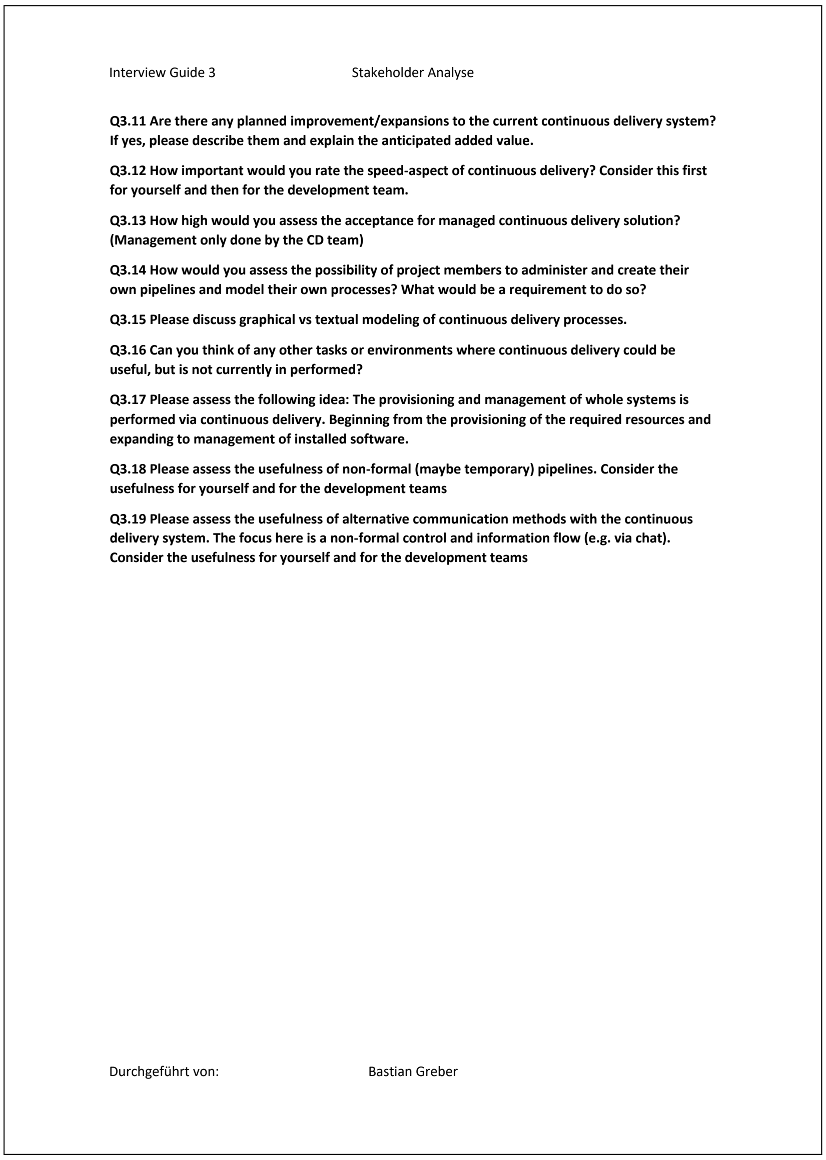
Q3.8 Which current continuous delivery solution is used? Please elaborate on the reasons why.

Q3.9 Are there problems or challenges with the current continuous delivery solution? If yes, please elaborate. Are there reoccurring errors/problems while providing the service for the rest of the company?

Q3.10 Does the current continuous delivery solution provide all the information you need, or are there unsatisfied information needs? Is the current representation of the continuous delivery process or the state of the CD-system satisfying?

Durchgeführt von: Bastian Greber

Figure A.4.: Interview Guides page 4 of 5



Interview Guide 3 Stakeholder Analyse

Q3.11 Are there any planned improvement/expansions to the current continuous delivery system? If yes, please describe them and explain the anticipated added value.

Q3.12 How important would you rate the speed-aspect of continuous delivery? Consider this first for yourself and then for the development team.

Q3.13 How high would you assess the acceptance for managed continuous delivery solution? (Management only done by the CD team)

Q3.14 How would you assess the possibility of project members to administer and create their own pipelines and model their own processes? What would be a requirement to do so?

Q3.15 Please discuss graphical vs textual modeling of continuous delivery processes.

Q3.16 Can you think of any other tasks or environments where continuous delivery could be useful, but is not currently in performed?

Q3.17 Please assess the following idea: The provisioning and management of whole systems is performed via continuous delivery. Beginning from the provisioning of the required resources and expanding to management of installed software.

Q3.18 Please assess the usefulness of non-formal (maybe temporary) pipelines. Consider the usefulness for yourself and for the development teams

Q3.19 Please assess the usefulness of alternative communication methods with the continuous delivery system. The focus here is a non-formal control and information flow (e.g. via chat). Consider the usefulness for yourself and for the development teams

Durchgeführt von: Bastian Greber

Figure A.5.: Interview Guides page 5 of 5

Bibliography

[BWZ15] L. Bass, I. Weber, and L. Zhu. *DevOps: A Software Architect's Perspective*. SEI Series in Software Engineering. New York: Addison-Wesley, 2015. ISBN: 978-0-13-404984-7. URL: `http://my.safaribooksonline.com/9780134049847` (cited on page 76).

[Cam] Camunda. *Camunda*. URL: `https://camunda.com/de/` (cited on page 84).

[Cap] Capgemini. *Capgemini*. URL: `https://www.capgemini.com` (cited on page 23).

[Che] ChefIO. *ChefIO*. URL: `https://www.chef.io/` (visited on 10/02/2019) (cited on page 30).

[Che15] L. Chen. “Continuous delivery: Huge benefits, but challenges too”. In: *IEEE Software* 32.2 (2015), pp. 50–54. ISSN: 07407459. DOI: `10.1109/MS.2015.27`. arXiv: `arXiv:1011.1669v3` (cited on pages 3, 4, 13).

[DMG07] P. Duvall, S. M. Matyas, and A. Glover. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley Signature Series. Upper Saddle River, NJ: Addison-Wesley, 2007. ISBN: 978-0-321-33638-5. URL: `http://my.safaribooksonline.com/9780321336385` (cited on page 4).

[Dör17] J. S. Döring. “An Architecture for Self-Organizing Continuous Delivery Pipelines”. PhD thesis. 2017 (cited on pages 1, 6, 14, 15, 19).

[Eri] Ericsson. *Ericsson*. URL: `https://www.ericsson.com` (cited on page 24).

[FF06] E. Freeman and E. Freeman. *Entwurfsmuster von Kopf bis Fuß*. Köln: O'Reilly, 2006. ISBN: 3-89721-421-0 (cited on page 41).

[Fly06] B. Flyvbjerg. “When I first became interested in in-depth case-study research”. In: *Qualitative Inquiry* 12.2 (2006), pp. 219–245. URL: `http://flyvbjerg.plan.aau.dk/MSFiveMis9.0SageASPUBL.pdf` (cited on page 9).

[FM10] R. Feldt and A. Magazinius. “Validity threats in empirical software engineering research - An initial survey”. In: *SEKE 2010 - Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering* (2010), pp. 374–379 (cited on page 47).

[HA05] S. E. Hove and B. Anda. “Experiences from conducting semi-structured interviews in empirical software engineering research”. In: *Proceedings - International Software Metrics Symposium* 2005.October 2005 (2005), pp. 203–212. ISSN: 15301435. DOI: `10.1109/METRICS.2005.24` (cited on pages 10, 11, 53).

[HF10] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* 2010, p. 497. ISBN: 978-0-321-60191-9. DOI: `10.1007/s13398-014-0173-7.2`. arXiv: `arXiv:1011.1669v3` (cited on pages 3, 4).

[Jen] Jenkins. *Jenkins CI.* URL: `https://jenkins.io/` (visited on 10/02/2019) (cited on pages 14, 30).

[JGi] JGit. *JGit.* URL: `https://www.eclipse.org/jgit/` (cited on page 42).

[JH07] Jochen Ludewig and Horst Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken.* Dpunkt.Verlag GmbH, 2007 (cited on page 3).

[Kle+15] S. Klepper et al. "Introducing Continuous Delivery of Mobile Apps in a Corporate Environment: A Case Study". In: *Proceedings - 2nd International Workshop on Rapid Continuous Software Engineering, RCoSE 2015* (2015), pp. 5–11. DOI: `10.1109/RCoSE.2015.9` (cited on page 23).

[LIL17] E. Laukkanen, J. Itkonen, and C. Lassenius. "Problems, causes and solutions when adopting continuous delivery—A systematic literature review". In: *Information and Software Technology* 82 (2017), pp. 55–79. ISSN: 09505849. DOI: `10.1016/j.infsof.2016.10.001` (cited on pages 1, 3, 4, 14).

[Lim] LimeSurvey. *LimeSurvey.* URL: `https://www.limesurvey.org/` (cited on page 63).

[LPA15] E. Laukkanen, M. Paasivaara, and T. Arvonen. "Stakeholder Perceptions of the Adoption of Continuous Integration-A Case Study". In: *Proceedings - 2015 Agile Conference, Agile 2015* Ci (2015), pp. 11–20. DOI: `10.1109/Agile.2015.15` (cited on pages 13, 24, 25).

[Mav] Maven. *Maven.* URL: `https://maven.apache.org/` (cited on page 30).

[Nun18] T. Nuntapramote. "Adapting Regression Test Optimization for Continuous Delivery". PhD thesis. 2018 (cited on pages 1, 6, 60).

[Ple15] C. Plewnia. "A Framework for Regression Test Prioritization and Selection Ein Framework für die Priorisierung und Selektion von Regressionstests". PhD thesis. 2015 (cited on page 60).

[RD03] R. W. Root and S. Draper. "Questionnaires as a software evaluation tool". In: December (2003), pp. 83–87. DOI: `10.1145/800045.801586` (cited on page 11).

[RH09] P. Runeson and M. Höst. "Guidelines for conducting and reporting case study research in software engineering". In: *Empirical Software Engineering* 14.2 (2009), pp. 131–164. ISSN: 1382-3256. DOI: `10.1007/s10664-008-9102-8`. URL: `http://link.springer.com/10.1007/s10664-008-9102-8` (cited on pages 9–11, 15–17, 19, 91).

[Rob02] C. Robson. "Real World Research, 2th Edition". In: 2002 (cited on page 9).

[SLD18] A. Steffens, H. Lichter, and J. S. Döring. “Designing a next-generation continuous software delivery system”. In: (2018), pp. 1–7. ISSN: 02705257. DOI: 10.1145/3194760.3194768 (cited on pages 5, 8).

[SWC] SWC. *SWC Aachen*. URL: https://www.swc.rwth-aachen.de/ (visited on 10/04/2019) (cited on page 6).

[Vir15] M. Virmani. “Understanding DevOps & bridging the gap from continuous integration to continuous delivery”. In: *5th International Conference on Innovative Computing Technology, INTECH 2015* Intech (2015), pp. 78–82. DOI: 10.1109/INTECH.2015.7173368 (cited on page 3).

[Wil18] N. Willig. “Implementierung von Continuous Delivery mittels BPMN Implementation of Continuous Delivery through BPMN”. PhD thesis. RWTH Aachen, 2018 (cited on pages 1, 6, 14, 83).

[Yin94] R. K. Yin. “CASE STUDY RESEARCH Design and Methods Thousand Oaks: Sage Publications (second edition)”. In: (1994). URL: http://www.madeira-edu.pt/LinkClick.aspx?fileticket=Fgm4GJWVTRs{\%}3D{\&}tabid=3004 (cited on pages 10, 15, 18, 19).

[Zsu00] R. B. Zsuzsa Varvasovszky. “How to do (or not to do) A Stakeholder Analysis”. In: *Health Policy and Planning* (2000) (cited on page 10).