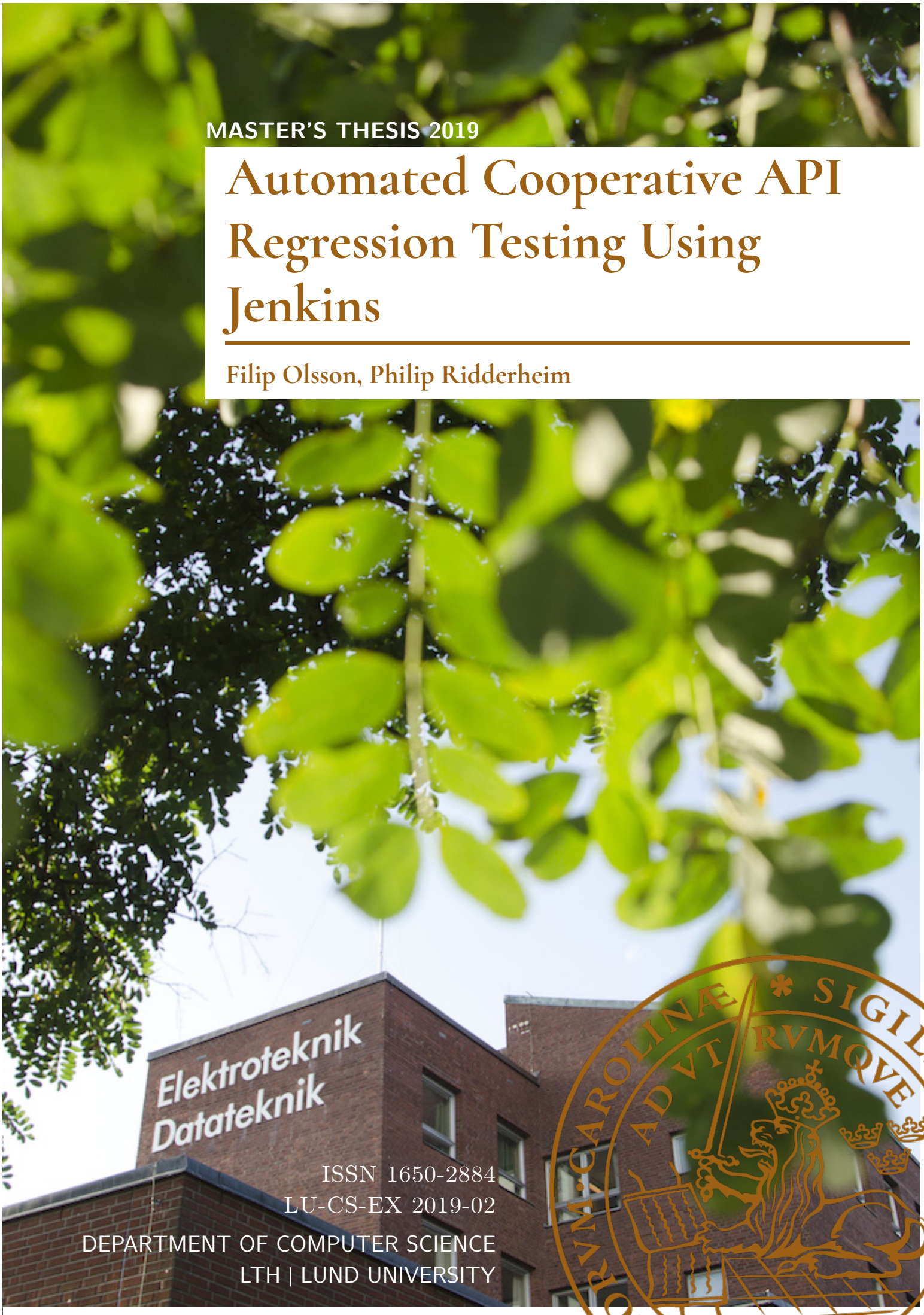

MASTER'S THESIS 2019

Automated Cooperative API Regression Testing Using Jenkins

Filip Olsson, Philip Ridderheim

ISSN 1650-2884
LU-CS-EX 2019-02

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2019-02

Automated Cooperative API Regression Testing Using Jenkins

Filip Olsson, Philip Ridderheim

Automated Cooperative API Regression Testing Using Jenkins

(A Design Science Study at Axis Communications)

Filip Olsson
dic11fol@student.lu.se

Philip Ridderheim
ada07pri@student.lu.se

May 9, 2019

Master's thesis work carried out at Axis Communications.

Supervisors: Per Runeson, Per.Runeson@cs.lth.se
David Vagnell, David.Vagnell@axis.com

Examiner: Emelie Engström, Emelie.Engstrom@cs.lth.se

Abstract

With the increasing focus on short development cycles and rapid release of software updates the need to avoid wasting time has become a high priority. However, dependencies between different software applications, e.g. the dependency between an API and the consumer application, might pose a greater challenge when it comes to validating a new software release. This master thesis project employed a design science research approach to develop a method for minimizing wait time for validating a new update of an API. Two development teams at Axis took part in a proof of concept implementation of a workflow aimed at decreasing the validation time of new API releases. The decrease in validation time is achieved by automating the triggering of the consumers' tests upon deployment of a new API version to the stage environment. By using the resulting meta-data from the consumers' API tests, greater confidence in the correctness of the new API version can be achieved. The meta-data also enables an easier approach to identify if a bug is API-side or consumer-side.

Keywords: regression testing, test artifact, test selection, Cooperative Regression Testing

Acknowledgements

We would like to thank our supervisors Professor Per Runeson for his academic guidance and David Vagnell for his help and guidance at Axis. We would also like to thank Emin Gigovic and Ola Söder for being a part of the early development of this master thesis. And we would like to extend our gratitude to Baldvin Gislason Bern for his input on how to expand on the topics of this thesis in future work. We also would like to thank Emelie Engström for taking on the role of our examiner. And finally we would like to thank Axis Communications AB for providing the working environment and to extend our gratitude to its development teams that have taken part and with a special thank you to the TST and IPA/PIA teams at Axis for their expertise and cooperation in our studies.

Contents

1 Introduction **7**
- 1.1 Background . . . 7
- 1.2 Case Description . . . 7
- 1.3 Limitations . . . 8
- 1.4 Outline . . . 8

2 Theory **9**
- 2.1 Introduction . . . 9
- 2.2 Testing and Regression Testing . . . 9
- 2.3 Agile Software Development . . . 10
- 2.4 Configuration Management . . . 11
- 2.5 API - Application Programming Interface . . . 12
- 2.6 Related Work . . . 12

3 Research Method **15**
- 3.1 Case Company . . . 15
- 3.2 Problem Statement and Objectives . . . 15
- 3.3 Research Questions . . . 16
- 3.4 Research Method . . . 16
- 3.5 Exploring Problem Areas . . . 19
  - 3.5.1 Interviews . . . 19
  - 3.5.2 Focus Group . . . 20
- 3.6 Designing an Artifact . . . 20
- 3.7 Artifact Evaluation . . . 21

4 Identification of Problem Areas **23**
- 4.1 Interviews . . . 23
  - 4.1.1 Participants . . . 23
  - 4.1.2 Observations . . . 25
  - 4.1.3 Conclusions . . . 26

4.1.4 Technical Variables and Limitations . . . 27
4.1.5 Generalization of the Current Workflow Regarding API Releases . 28
4.2 Focus group . . . 29
4.2.1 Participants . . . 29
4.2.2 Discussion . . . 29
4.2.3 Validation of the Focus Group Findings . . . 30
4.2.4 Technical Variables and Limitations . . . 30

5 Design **31**
5.1 Possible Solutions . . . 31
5.1.1 Requested Consumer Blackbox API Regression Testing Against Stage Environment . . . 31
5.1.2 Triggered Consumer Blackbox API Regression Testing Against Stage Environment . . . 32
5.1.3 Inter-group Time Limited Exchange of Testers . . . 34
5.2 Summary . . . 34

6 Implementation **37**
6.1 Technical Environment . . . 37
6.1.1 Jenkins . . . 37
6.1.2 Version Control . . . 38
6.1.3 Programming Language . . . 38
6.2 Workflow . . . 38
6.3 First Implementation Design . . . 39
6.4 Second Implementation Design . . . 39

7 Results and Evaluation **41**
7.1 Evaluation Process . . . 41
7.2 Evaluation Meeting . . . 41
7.2.1 Participants . . . 41
7.2.2 Results and Validation . . . 41
7.3 Questionnaire . . . 42
7.3.1 Participants . . . 42
7.3.2 Results and Validity . . . 42

8 Conclusions **45**

9 Future Work **47**

Bibliography **49**

Appendix A Interview Protocol **55**

Chapter 1
Introduction

1.1 Background

Software development can often be very complex and may include dependencies between different software modules, e.g. an application having a dependency on an API. Therefore it is important to ensure the quality of the software before it is released. However, when releasing software a trade-off must be made between how much time to spend on testing the software and how quick the software should be released.

It is not enough to only ensure that newly added features work correctly since there is also the risk of regressions appearing in the software. Software regressions are bugs that appear in the previously working software after a change has been made. To protect against this it is common to employ regression testing to ensure that e.g. new features has not introduced new bugs in other parts of the software.

One solution to be able to test the software for regressions as well as save time during testing before a release is to utilize automated regression testing. In this project, we explore the possibility of cooperation between different development teams regarding automated regression testing of interdependent code to enable faster delivery time and less test overlap.

1.2 Case Description

This thesis work was conducted at Axis Communication, also referred to as Axis for short. The company was founded in 1984 and started out by developing protocol converters which enabled PC printers to connect to the IBM mainframe [1]. In 1996 Axis developed the first network camera and is now one of the market leaders in network cameras and video encoders.

Axis was an early adopter of agile development methods, driven by both their quickly expanding product portfolio and the high interdependency of its products. Agile allows Axis to more quickly respond to changes affecting multiple development teams. The fact

that different development teams often are responsible for different aspects of a product or service also adds another layer of complexity. One solution for assuring the quality of new software releases is the usage of regression testing. This thesis was initiated in a desire to examine if regression testing of interdependent code between different teams can be tested in a cooperative manner in order to streamline the workflow.

1.3 Limitations

Since this master thesis project was conducted and centered around Axis Communications limitations regarding the feasibility of the solution for other companies could exist.

1.4 Outline

The report starts with the theoretical background in chapter 2, detailing the key areas in this thesis. It also contains the related work that is considered relevant to this thesis. Chapter 3 outlines the research questions for this thesis and the research method used. Chapter 4 describes how the possible solutions were elicited at Axis. Chapter 5 outlines the three possible solutions that were the result of the elicitation process described in chapter 4. Chapter 6 goes through the chosen solution and how it is implemented in the context of an API development team and a consumer team. Chapter 7 contains the results and their evaluation. Chapter 8 outlines the conclusions that are drawn in this thesis. Lastly, possible future work is described in chapter 9.

Chapter 2
Theory

2.1 Introduction

This chapter introduces the key concepts in this thesis and their position within the area of software development. The most important concept being regression testing and how it relates to regular software testing. Followed by agile software development, its main focus and why it is a concept rather than a specific workflow.

As the thesis focuses on the interaction between development teams with regards to regression testing, configuration management is an important aspect and is, therefore, explained how it is approached in an agile software development scenario. This is followed by the detailing of the API and its function.

The chapter is concluded with an overview of related work within the field.

2.2 Testing and Regression Testing

IEEE defines software testing as [2]:

> *The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.*

While regression testing is defined by IEEE as [2]:

> *Selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements*

Many interpret regression testing to merely include running the regular test suites for all the features additional times. This fails to capture the intended function of regression testing as its main goals are as Leung and White notes [25]:

- Preserve the quality of the software; the modified software should be at least as reliable as its previous version; this may be achieved in many ways; one possible method is to insist that the same structural coverage is achieved by both versions of the software;
- Ensure the continued operation of the software; this is an important goal because some users may become dependent on the software product, and software developers have a responsibility to continue to provide the same service to users.

Due to the wide range of development methodologies applied across different organizational structures, the effort of regression testing is conducted at varying frequencies depending on the methodology of the particular organizational structure [27][14]. As regression testing serves a verificational purpose for the expected functionality, it is implemented as re-testing of previously tested code and can be implemented on different levels ranging from unit to acceptance testing. Making regression testing not denotable as an independent level of software testing in itself as it can be implemented on all levels of testing. Rather, regression testing is a variant of the testing conducted at each specific level. [27].

Testing and regression testing have certain commonalities in both aim and application. Leung and White define two such shared aims [25]:

- Increase one's confidence in the correctness of a program.
- Locate errors in a program.

They both serve to verify the delivered functionality against the specified functionality. Though with the clear distinction that testing focuses on the most current functionality and regression testing focuses on regressions i.e. that previously established functionality has not been adversely effected [25][5].

Testing and regression testing share a lot in their application, especially relating to the testing environment, which often is the same for the two. This makes test selection strategies aimed at suite minimization a high priority for regression testing as it is often seen as a time-consuming and costly effort [34][30].

2.3 Agile Software Development

IEEE defines agile development as [7]:

> *1. software development approach based on iterative development, frequent inspection and adaptation, and incremental deliveries, in which requirements and solutions evolve through collaboration in cross-functional teams and through continuous stakeholder feedback.*

The core concept of agile development, adaptability, has its roots in the effort to streamline the development process be able to quickly deliver a working solution. A side effect of this is a lower overall cost of each project. By working to minimize efforts that relate to tertiary aspects of the deliverable, such as documentation, more time is spent on the core aspects of the deliverable. Agile also focuses on close customer involvement as to minimize the reaction time to changing requirements and as such long term planning is not considered a part of

agile development. The fact that close customer involvement is premiered over long term planning, interactions and individuals are focused on more than processes and tools needing planning and consideration, as such tertiary efforts are further minimized in agile [8][19].

While there is a general concept of agile development there is no central methodology paired with its inception [19]. Coupled with the fact that each development organizations and its constraints require adaptation to optimize productivity very few organizations use a "best practice" implementation of an agile methodology [29].

As agile focuses on changes rather than following a plan the effort of regression testing needs adapting. In a conventional development organization, regression testing is merely performed at the end of development allowing for considerably more time planning than in an agile organization. In agile, however, regression testing becomes very important as changing requirements necessitate the need for a concise effort to verify that expected functionality is still present and hasn't been lost [21]. This is often achieved by automating the regression testing process to minimize the usage of manpower to complete the task. In turn, freeing up the developers to focus on maintaining rather than executing the tests [6][35].

In an agile organization, the function of regular testing is usually performed by the developers themselves as to remain able to quickly adjust to changing requirements while regression testing is a distinct and separate effort as its test selection process requires more attention to be effective [12][24].

2.4 Configuration Management

IEEE defines Configuration management as [3]:

> *Configuration Management is a discipline applying technical and administrative direction and surveillance to identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.*

A concerted configuration management effort adds value to the project deliverables by enabling the tracking of the development process and enabling multiple developers too more effectively work on the same project with a lowered overhead. Configuration management is vital when it comes to versioning and traceability within a project. By utilizing the traceability aspect of configuration management multiple development branches can be pursued at the same time. The need for bug tracking and fixing necessitates a system for tracking and associating the effort to correct it to the code in question in order to be able to trace its origin, this is provided by configuration management [32][4].

The agile approach to development puts a higher demand on the quality of the configuration management effort than conventional approaches. This is mainly due to the increased volatility of the requirements making traceability relating to design decisions more important [32]. When adding the dimension of a distributed development team, configuration management becomes imperative to be able to effectively leverage all team members [15].

2.5 API - Application Programming Interface

API or Application Programming Interface, is defined as [36]:

> *A set of subroutine definitions, protocols, and tools for building software. In general terms, it is a set of clearly defined methods of communication between various components.*

The need for different programs to be able to communicate without having access to the others source code necessitates the need for an interface to enable the exchange of information, the API enables this exchange to take place. As the interface does not give access to the code of the program, the documentation that is provided with the API is as important as the API itself. The quality of the API is of utmost importance so as to avoid unintended behavior when the API is used [22][26].

2.6 Related Work

Searching for literature regarding inter-team testing generate few relevant results as of the writing of this thesis. This is, however, a reasonable result as it is a point of intersection between testing, requirements, development methodology, and organizational management. This conclusion is further supported when examining literature studies and reviews [20]. The main identified related areas of interest are:

- Duplicate test elimination
- API regression test selection and suite minimization
- Release validation time minimization

For each area of interest, a search was conducted to locate relating works. The search was done using the service LUBSearch that includes a large number of databases of academic works and industry white papers [10]. The search terms used for each area where the entire area name and in addition, parts of the area name were used to further broaden the results. From the results, three works were deemed to be tangential to this thesis work.

The area of duplicate test elimination is given little focus in current academia and is more considered to be solved by suite minimization. Sharma and Singh note that [33]:

> *As the software program is updated and modified, test cases in a test suite can become duplicative. Thus, it would be advantageous to identify and eliminate redundant test cases in a test suite, allowing reduction in the test suite size which, in turn, can decrease testing time and contribute to optimize maintenance effort for the test suite.*

The approach of limiting duplicative tests also works to counteract the slippage of system-wide understanding by the developers as they are made aware of the overlap in test cases in different parts of the system. This inevitably leads to a reduced development time as fewer unnecessary tests are executed.

Testing externally developed software components is usually a difficult task as the information about the component and its construction might be lacking. Chengying and Yangsheng note that [16]:

> *System tester generally can't gain good comprehension to the internal structure of component, so it will be fairly hard for them to perform testing on the system built by externally-provided components. Particularly, it is extremely difficult in the situation of regression testing, and component users don't know the details about change in component, so they aren't able to select the proper test cases to retest the modified system.*

Focusing on the information exchange between the supplier and consumer of a component, as relating to test cases rather than API-documentation, a greater understanding of the component can be achieved by the consumer's test organization and as such provide a higher level of quality in the delivered end product. Especially in the case of regression testing as API-documentation usually do not include expected functionality as previously established. By looking at the test documentation also enables the consumers to more effectively target their regression testing efforts with regards to external components.

The difficulty of achieving a high level of quality with regards to external components is also noted by Alagoz et al. [13]:

> *Since vendors of commercial off-the-shelf (COTS) software components do not release their source code, detecting all fault states of black box systems becomes a much more challenging task for the car manufacturer.*

The issue of regression testing of external components is a large concern as it often leads to the assumption that the supplier of the component has done all the testing needed which can lead to unforeseen and hard to find bugs.

The area of release validation time minimization generated a large number of results but they were either from a requirements standpoint and theoretical or practical and focused on prototyping validation. None was found to be exploring validation time minimization within an organization and from a regression testing or software testing in general, viewpoint. Tests as Requirements as an area would be the closest but with the viewpoint of validation time reduction seems to be unexplored within software testing. Especially, considering validation from outside the development team and automated using consumer tests none the less.

Chapter 3
Research Method

3.1 Case Company

Axis uses agile development methods with small development teams. This enables a more flexible development strategy to be able to faster respond to changing requirements and desires for new features. Therefore this master's thesis project has an agile software development process as the foundation when developing and evaluating its proposed improvements.

3.2 Problem Statement and Objectives

The services and products provided by Axis are often the results of multiple development teams that each are responsible for a different aspect of the product or service. These interdependencies create longer validation times since multiple development teams may have to be part of the validation process. One additional aspect is that there is a chance that some parts of the product or service is tested multiple times but by different development teams. Having this kind of test overlap could be a redundant use of time and resources.

The objective of this thesis project was to explore the possibility of cooperation regarding regression testing to enable less test overlap and shorter validation times. The goals of this thesis project summarized as the following:

1. Explore the current regression testing practices at Axis.

2. Create a proof of concept implementation of a cooperative regression testing workflow.

3. Evaluate the proof of concept implementation.

3.3 Research Questions

The concept of cooperative regression testing had not had its relevance analyzed. Therefore in order to analyze and evaluate if a cooperative regression testing efforts could be applied in a larger context at Axis an initial study regarding the current regression testing climate at Axis had to be conducted. Thus exploring the current regression testing practices (RQ1) and perceived challenges (RQ2) at Axis was deemed the first milestones. The last milestone and main research question of this thesis is to investigate if a cooperative effort regarding regression testing could result in decreasing the validation times of API releases (RQ3).

RQ1 How and by whom is regression testing performed today?

Research into how regression testing is performed at Axis and by whom needed to be conducted to gain a better understanding regarding current regression testing practices. This enables a better understanding of the interdependencies in products and services and how current practices impact testing and validation.

RQ2 What are the main challenges for regression testing activities in large scale agile software development?

To evaluate the potential benefits that cooperative regression testing can provide the main challenges regarding regression testing at Axis needed to be investigated. The perceived regression testing challenges are taken into consideration in the solution design and provide a point of reference to which the implemented proof of concept can be evaluated.

RQ3 How can cooperative regression testing be used in API releases to decrease regression test overlap and shorten validation time?

To investigate how cooperative regression testing could be utilized in API releases a proof of concept design was implemented. This enabled the design to be tested and evaluated in order to improve the solution design.

3.4 Research Method

The methodology used is based on Hevner's guidelines for design science in information system research, see table 3.1 [23]. Design science is an iterative approach and therefore the research method employed must reflect that aspect [23]. The objectives and methods used can be seen in figure 3.1.

The desire at Axis to examine if interdependent code between development teams could be cooperatively tested had not had its relevance in a larger context investigated. Therefore to explore the problem area and its relevance an initial study was conducted with interviews from different development teams, see guideline #2 in table 3.1. This initial study made the foundation for the more focused study regarding cooperative regression testing (RQ 3) which resulted in a proof of concept implementation. An overview of the work conducted can be

#	Guideline	Description
1	Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
2	Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
3	Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
4	Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
5	Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
6	Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
7	Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 3.1: Hevner's Design guidelines

seen in figure 3.2.

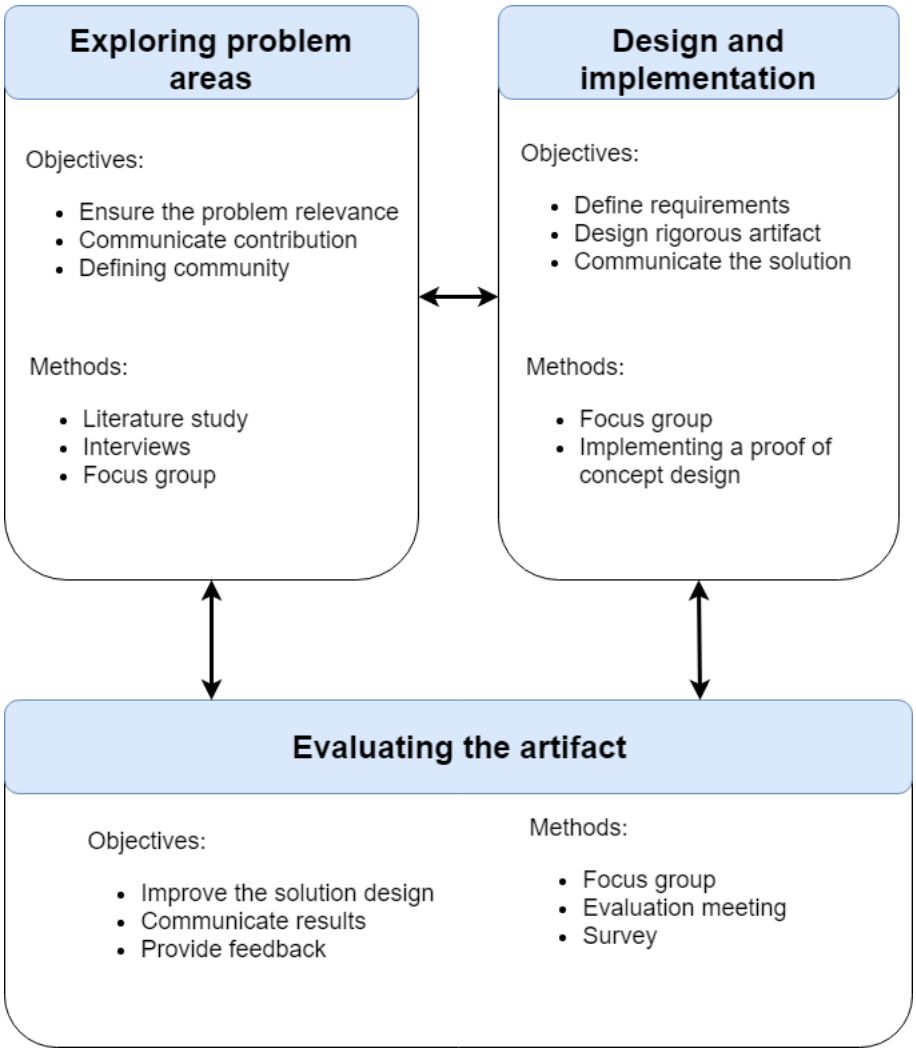

Figure 3.1: Overview of the research method

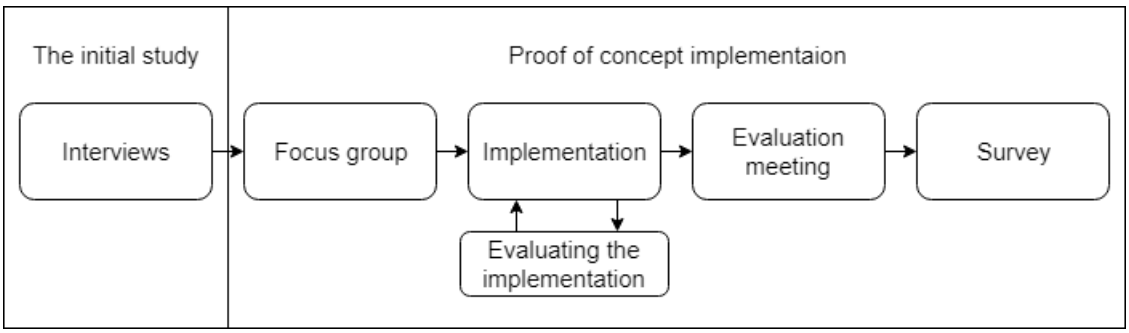

Figure 3.2: Overview of the work conducted

3.5 Exploring Problem Areas

When exploring regression testing challenges the first step is to examine published studies regarding the subject area. However, there is no guarantee that the perceived challenges regarding regression testing are the same at Axis. Thus to ensure the problem relevance a case study needs to be conducted in order to gain insight in what developers at Axis perceive as relevant challenges, see guideline #2 in table 3.1.

The initial study at Axis involved interviews with developers to gain a broad overview of current regression testing practices and challenges at Axis. Thereafter a focus group was used in a more pragmatic approach which resulted in a proof of concept implementation.

3.5.1 Interviews

We chose a semi-structured interview approach to maximize the amount and quality of information that could be collected [28]. When creating the interview protocol the format of who, what, why, how and when was used as a framework for the flow of questions but since the structure chosen is semi-structured the flow was not to be enforced during the interviews [31]. Instead, the format was used as a general idea of the interview progression and as a checklist in order to ensure that every question was answered. In addition to the aforementioned format the interview was structured in a general sense to start with questions about the interviewee and his/her background and then move on to questions about the team and lastly move on to work practices and workflows, see appendix A.

To get a broad overview of regression testing practices at Axis different development teams needed to be represented. The development teams elicited for interviews needed to be somewhat spread out over different development branches in the organization as well as the criteria that the development team utilized regression testing. Due to time constraints and the size of the company, not all relevant development teams could be represented and therefore six different development teams were selected.

After conducting the interviews a thematic analysis was conducted. The analysis was conducted by identifying different problem areas regarding regression testing. The statements made were then compiled after their corresponding theme which was then used to answer RQ2.

The execution of the interviews can be summarized as:

1. Create the consent form and interview protocol
2. Conduct the interviews
3. Transcribe
4. Summarize and explore themes
5. Draw conclusions

3.5.2 Focus Group

The focus group meeting was initiated with a presentation of the purpose of this master thesis as well as the basic concepts of cooperative regression testing. The format of the meeting was an open discussion where possible cooperative regression implementations where discussed. Requirements and perceived improvements and risks of the aforementioned solutions where also discussed.

The focus group was conducted in order to gain a pragmatic view of how improvements can be made to regression testing in the context of an API supplier and its consumers. Thus two appropriate development teams needed to be selected. One to represent the API and its development and one to represent the consumer and its application where the API is used. This is used to give a real-world example of how a solution could be formulated and was then analyzed with the broader overview gained from the interviews on how a more generalizable artifact could be designed.

The execution of the focus group can be summarized as:

1. Compile the material for the focus group
2. Conduct the focus group meeting
3. Transcribe
4. Summarize the result
5. Draw conclusions

3.6 Designing an Artifact

When using design-science the end result needs to be a viable artifact and needs to be able to be represented and communicated effectively, see guideline #1 and #7 in table 3.1. To ensure the viability and to present the final artifact in an effective manner, a proof of concept implementation of a cooperative regression testing workflow was done.

The design process was based upon the knowledge gained from exploring the problem area. The initial requirements were to have less test overlap and decrease validation time. However, after conducting the focus group meeting the relevance of decreasing test overlap was considered to be of less importance and was therefore not considered in the proof of concept implementation. An additional design consideration was to make the design viable in a larger context at Axis so that it can be implemented and used by other development teams.

The artifact constitutes an automated workflow between an API development team and a development team whose application consumes the API. The workflow is defined as the consumers' automated regression test suit being triggered when the API release is deployed to the stage environment. This enables regression testing in the consumer application thus ensuring that no regressions have occurred in the application as a result of the new API version. The automation server Jenkins was used to implement an interpretation of the artifact as a proof of concept in a project involving two interdependent development teams.

3.7 Artifact Evaluation

When using design-science the end result is a useful artifact to a relevant problem, see guideline #1 in table 3.1 [23]. Thus the problem relevance must be carefully studied and the artifact evaluated to ensure that the artifact achieves its purpose, see guideline #2 and #3 in table 3.1.

The evaluation of the artifact was present at every step of the design and implementation process. And the evaluation criteria is based on the exploration of the problem area and on feedback from the development teams elicited for the proof of concept implementation. There where two implementations of the proposed workflow. The second design addressed and improved upon design considerations found in the first design.

After the second proof of concept implementation, an evaluation meeting with the involved development teams was held in order to evaluate the end result. A survey was also conducted to explore the opinions of development teams at Axis regarding the solution.

Chapter 4
Identification of Problem Areas

This chapter presents the case study that was conducted at Axis Communications. The purpose of the case study was to first investigate how regression testing is performed at Axis and the perceived challenges. This was done to gain insight into the different problem areas. The knowledge regarding regression testing practices and challenges was then followed by a focus group that involved two Axis development teams in order to conceptualize possible improvements.

4.1 Interviews

The purpose of the interview series was to establish how regression testing is currently performed at Axis and if it differed between comparable development teams enabling a better understanding of the problem areas.

4.1.1 Participants

In total six participants were interviewed during the course of the study from two departments of Axis. All of the participants were either explicitly employed as testers or they had software testing as part of their work duties. The participants formed a nonuniform group in regards to academic background, amount of time at Axis and if they were employees or consultants, see table 4.1.

The structure of the development groups also varied, such as:

- Developers do the regression testing themselves, without an assigned tester.
- One tester is assigned to the development team that leads the regression testing effort.
- A separate team is assigned to perform regression testing that is not part of the development team.

This variation of a diverse set of viewpoints enabled more generalized conclusions to be drawn as noted by DiCicco-Bloom and Crabtree [17].

Interviewee	Role	Years of Experience
A	Experienced Engineer	10+
B	Senior Engineer	10+
C	Engineer	3-5
D	Senior Engineer	5-10
E	Engineer	3-5
F	Test Consultant	5-10

Table 4.1: Interviewees

Code	Area	Affected	Description
C1	Organizational	6/6	
C1.1	Roles	2/6	Lack of explicitly defined roles, relating to regression testing
C1.2	Workflow	4/6	Workflow differences between teams
C1.3	Planning	3/6	The planning within the team
C2	Tools	6/6	
C2.1	Linkage	6/6	Linkage between planning and execution/reporting tools
C2.2	Appropriation	2/6	Appropriation of new tools
C3	Information	6/6	
C3.1	Distribution	6/6	The flow of information between teams or end consumers
C3.2	Documentation	3/6	The level of documentation with regards to traceability of results
C4	API	5/6	Knowledge regarding the level of testing performed outside of the team with regards to APIs

Table 4.2: Identified possible areas of improvement

4.1.2 Observations

During the interviews, different perceived regression testing challenges were found. Such as having differences in production and development environments. Since some products and services are not suitable to be tested in the production environment software regressions resulting from missed aspects when deploying to production can have a big impact. Another stated challenge was difficulty acquiring relevant test data to use in test cases and also knowing when the test cases had covered enough test scenarios. One solution to estimate the required test coverage is to utilize risk assessment when introducing new features. However, missing important aspects or impact areas in the risk assessment were considered to be a challenge. Creating a good workflow between interdependent development teams in regards to the testing and validation was also considered challenging.

After transcribing the interviews a thematic analysis was conducted where themes were identified and possible areas of improvements were categorized after the major themes, see table 4.2. The major themes were organizational, tools, information and API.

The organizational category includes how the development teams are comprised, how workflows are managed and how planning in regards to regression testing is performed. The interviewed development teams have different approaches to regression testing which varied greatly. Examples of different approaches found were:

- Each test case was coded and followed with each test-run being saved including test data.
- The regression testing was done in an exploratory fashion and was entirely at the individual tester's discretion with only a pass or fail being recorded for the entire test-run.

All development teams had, though, with different approaches, a considered, to varying degrees, methodology for their regression testing effort. The general purpose and added value of regression testing were shared among all participants, namely that it enables the developers to validate that previously expected functionality has not been inadvertently broken. The vast majority of the participants considered have a generally well-balanced approach and no drastic changes were required. Planning and execution of the regression tests were by most teams conducted by personnel other than the developers who coded and generated the regular testing cases. The planning of regression testing activities was conducted such that the developers were included in the process.

Tool separation was equal throughout all participants teams i.e. the planning tool was separate from the execution and reporting tool. Most development teams had plans of various levels of maturity regarding the development of a linking function for their respective tools to enhance the traceability in their system. A few teams also had plans to change planning tools but not because of integration concerns but rather because of organizational improvements.

Very few had structures for linking requirements/stories to regression test cases, which the participants not having such linkage conceded was a shortcoming. In almost all of the participants' development teams, a team-wide code review was standard for the regression testing activities, but that most emphases was put on individual expertise for all participants teams. All participants felt the need for better coordination with other development teams and a large number felt that the information structure regarding what teams were using their deliverables was in need of improvement. In contrast, they had a clear picture of who was the product owner and that the product owner was involved in the planning of the project.

All of the participants reported that APIs created by other development teams at Axis were used in their projects. No participant reported that specific API-tests were conducted but rather their correct function was assumed. The lack of API-testing was compounded by the fact that all but one participant reported that the APIs were accessed live and not mocked during the regression testing phase. As the interviewed development teams' overall issues were similar, even though their constellations differed, would indicate that the above observations are sound and to a large extent generalizable.

4.1.3 Conclusions

One major conclusion from the interviews is that a structure exists for all the interviewed development teams such that a cooperating effort regarding regression testing of interdepen-

dent API's could be beneficial. All the interviewed development teams used API's developed by other groups at Axis and in some cases also maintained their own API's that other development teams at Axis are currently using making a cooperative effort a viable way of making regression testing more efficient within the organization. When considering the major perceived regression testing challenges a cooperative regression testing effort could facilitate all stated challenges, to a varying degree. Having a cooperative effort could act as a method of gaining additional relevant test data as well as creating a broader coverage of test scenarios, which in turn minimize the risk of bugs appearing in the final product. Since the consumers know what scenarios the API will be used in, relevant test data and test scenarios in the form of edge cases could be of great benefit to the developers of the API. This cooperation will, in turn, create the need for continuous dialogue between the cooperating development teams. This could result in workflow incompatibilities between the development teams having less of an impact since upcoming events which could have a negative impact could be communicated at an earlier stage with a continuous dialogue. When considering the grouped areas, C*x* in table 4.2, of improvement both the appropriation and linkage of tools were for the most part already in consideration and in some cases as far as in the trial phase. Thus efforts of trying to improve these areas are already underway and therefore of less importance to this thesis when considering which areas could benefit the most of further improvements. The other grouped areas could also benefit from a cooperative regression testing effort thus ensuring the problem relevance of the artifact. With a cooperative effort, continuous communication between the different development teams would improve the flow of information which should have a positive impact on information distribution between development teams. Depending on how the communication is conducted, greater incentive on documentation of changes and future improvements could be seen since it would be of relevance to the continuous dialogue. With a continuous dialogue, an improvement in making the organizational workflows more compatible could be facilitated.

The API area would see the most benefit to a cooperative effort. The cooperative effort would increase knowledge about the usage of API's, provide relevant test data, increase test coverage as well as exchange areas of expertise between the different development teams.

The background to this master thesis included looking at possible improvements to regression testing regarding interdependent API's and with the analysis of the interviews, it is also clear that other areas of improvements and major perceived challenges regarding regression testing could also be facilitated. The end conclusion for the interview series is that there is great potential in improving the efficiency of the regression testing effort at Axis as there is a potential for a cooperative effort to improve feedback time and general awareness between teams.

4.1.4 Technical Variables and Limitations

By recording the interviews in a video and audio format a lot of time spent taking notes was reduced. This meant that a consent form needed to be developed which specified how the collected data was to be handled and outlining the consent given by participating as well as the rights to withdraw consent. The recordings were later transcribed into text in order to be analyzed. Since this is a time-consuming process, time boxing on the interview questions was used to ensure the interview was conducted within a reasonable time frame as well as ensuring that every question had an appropriate allocated time slot. This was in practice hard

to keep track of since the semi-structured format meant that the interview did not follow a strict flow. However, the interviews never exceeded the allocated time frame so there was never a need to stop the interviewee in order to get through all the questions on time.

4.1.5 Generalization of the Current Workflow Regarding API Releases

To enable a discussion for the focus group regarding how a cooperative regression effort could be designed a generalized workflow had to be illustrated.

From the interviews, it was observed that the current workflow within Axis is influenced to a large degree by the agile approach to software development. The strive with an agile approach is that releases should be done often and incrementally compared to the traditional development approaches. Therefore a good and thought out approach to testing, building and deploying is required to avoid introducing bugs into the deliverable. To this end, stage environments are widely used at Axis where testing and building can be done iteratively and the pre-release process can be automated in an effort to minimize overhead.

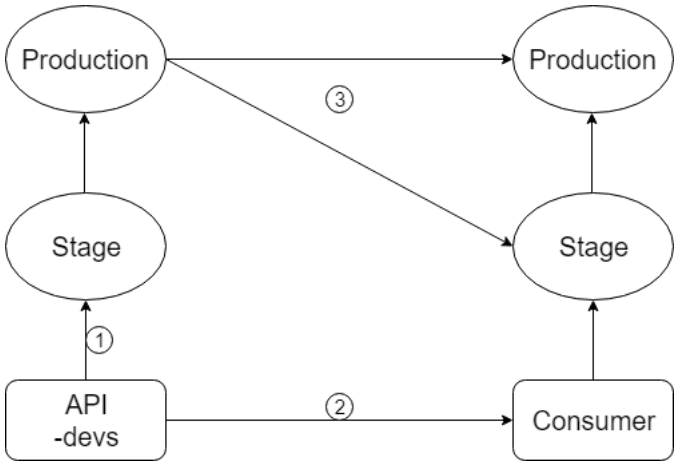

Figure 4.1: A workflow chart of the current workflow within Axis

In figure 4.1 the following steps are included in the release of a new version

1. The API developers push a commit to the stage environment. In the stage environment, all the testing before release is done such as to ensure correct behavior of the software.

2. After the pre-release testing and any additional steps have been completed the new version is pushed to the production environment.

3. The consumers can then access the new version of the deliverable.

4.2 Focus group

The purpose of the focus group was to validate and further the process of conceptualizing solutions based on the conclusions from the interview series.

4.2.1 Participants

In total four participants were elicited for the focus group. Two participants represented the API development team with one being a developer and the other a tester of the chosen API. Two developers represented the development team that consumes the API. The constellation of the development team that consumes the API does not have explicitly delegated responsibilities for the testing effort but is instead delegated to the developers after emerging needs. However, two participants were sick on the day of the focus group meeting and could not attend. The two participants who were present at the focus group meeting were the least experienced (working at Axis) out of the two representatives from each development team. This brought the validity of the findings from the focus group meeting in question. Therefore additional validation for the findings needed to be facilitated, see section 4.2.3.

4.2.2 Discussion

The focus group meeting started with a short presentation about the aim of this thesis work and a workflow example demonstrating a basic situation where a cooperative effort could be performed by two interdependent development teams. An open discussion was held about what the current workflow at the respective development teams are and how possible improvements could be made in regards to achieving a more cooperative approach to regression testing.

Three possible designs on how cooperative regression testing could be implemented emerged throughout the discussions. The designs were then evaluated by the participants on what positive impacts they could have but also risks involved in the designs. The three designs can be described as:

1. The API consumers run their regression tests against a stage environment for the new API release

2. The API consumers tag the regression test cases that includes calls to the specific API and then the API developers can run the tagged regression tests before new releases

3. The developers and consumers of the API exchange testers for a short period of time on a set basis

The possible foreseen improvements for all the different designs included gaining valuable test data, exchange expertise, shorten validation time and gaining broader test coverage. However, the designs were considered to have different risks involved with them. Design 1 was the one considered to have the least entailing risk and was already in a somewhat degree utilized by the API development team. However, design 2 was considered to have a considerate risk of a bad return of invested time and might instead make the testing effort take up more time and resources. Design 3 was considered to be a good system once implemented

however the time and effort setting up the system and redoing the regression tests to include the new functionality would have been significant. The design would also increase the need for the consumers to keep their regression tests up to date regarding and over confidence in a new release of the API might occur if the test cases by the consumers are not sufficient. The last concern of design 3 was that a dilution of responsibility might occur by which the testing effort of the API is pushed onto the consumers.

4.2.3 Validation of the Focus Group Findings

The validation of the focus group meeting was done by compiling the findings and ideas from the focus group into a document for the participants who could not attend so that they could review and provide feedback on the findings of the meeting. The document used the same format as the focus group meeting in that it started with a short presentation of the purpose of this master thesis as well as the same workflow example which could incorporate cooperative regression testing between two interdependent development teams. It was done this way to give the readers a frame of reference from which the discussed solutions could be evaluated. After the introduction, the solutions discussed were listed with their respective pros and cons. The document ended with describing what should be included in the feedback, such as their thoughts on the proposed solution as well as if they could think of any additional solutions. The document was also sent to one additional person who was not elicited for the focus group but regularly performs testing tasks for the application development team. The person in question is part of an external QA team.

The feedback received was that a cooperative regression testing effort could be of great value. Design solution 2, in section 4.2.2, was seen as the design with the most potential. An additional feature that was proposed for this solution was to include a response in the form of an automatically generated email after the regression tests have been performed as to inform the involved parties of the result. The external QA tester for the application team also proposed that an additional solution could be for the application team to document what functionality in the API was used by the application as a way of enabling the API developers to better estimate risks assessments with new releases. This was argued to have the drawback of being time-consuming when creating the documentation as well as when utilizing it when performing a risk assessment. It was also deemed as having bad scalability since a lot of consumers would create the need for overwhelming documentation.

An overarching sentiment was that test overlap was not a high priority as it was deemed to be more beneficial to gain additional test coverage even if it resulted in increased test overlap. Due to this fact, it was decided that test overlap minimization as stated in section 3.3 would be set aside for possible future work as it could be beneficial once a cooperative regression testing workflow was put into place.

4.2.4 Technical Variables and Limitations

The focus group was audio recorded so that it could later be transcribed and further analyzed. Thus a consent form had to be used for the same reasons as the interviews. Small changes were made to the consent form which was mostly due to that only audio would be recorded and not video. This decision came after transcribing the interview where it was determined that the captured video had little use and instead was sufficient to capture only the audio.

Chapter 5
Design

5.1 Possible Solutions

From the focus group, three possible solutions could be compiled. The first solution centers on the consumer regression tests being triggered by the consumers at the API developers behest. The second solution removes the consumers from the triggering process and instead has the consumer's test being triggered by the AP developers. The third solution aims at giving the testers better insight into the other team by having an exchange program where testers switch places.

5.1.1 Requested Consumer Blackbox API Regression Testing Against Stage Environment

This solution aims to improve the feedback time from the consumers for new releases of an API. This is achieved by including the consumers' test suites as part of the pre-release process as can be seen in figure 5.1.

In figure 5.1 the following steps are included in the release of a new version

1. The API developers push a release to stage.
2. The consumers are requested to run their tests against the stage environment.
3. The consumers trigger the API tests.
4. The consumers API tests are run against the API stage environment.

By allowing consumer teams access to run API tests against the stage environment API regressions can be mitigated or even avoided before a release. If coupled with better more proactive communication the feedback and validation process before the release will be quicker

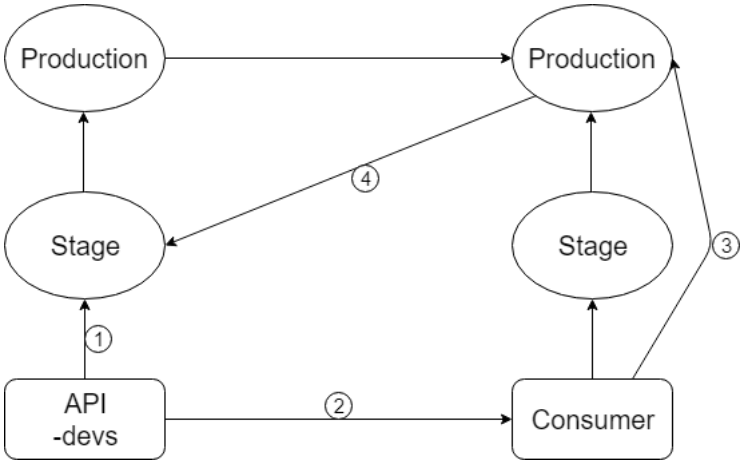

Figure 5.1: Requested consumer blackbox API regression testing.

and more informed. This will enable the API developers to have a greater confidence in the correctness of their releases.

This solution leverages the consumer's tests such that they to a certain extent validate the release prior to it going live. This solution would also enable a better exchange of ideas as a side effect of the more proactive communication between API developer and consumer.

Stakeholders

The main stakeholders in this situation are:

- The API developers
- The consumers that are requested to test the staged API
- The consumers who are not requested to test the staged API

Limitations and Risks

The main limitations in this scenario are that the API developers do not get additional feedback only that it will be attained before the release to production. This solution will also be limited by the response time of the consumers and their planning as they might not have resources enough to complete their API regression testing in the allotted time given by the API developers or at the specified time. This could lead to the API being tested by fewer consumers before release lowering the efficacy of this solution. Another risk is that having the consumers perform their API tests before release adds a level of coordination and planning that might hinder the API developers from maximizing their output.

5.1.2 Triggered Consumer Blackbox API Regression Testing Against Stage Environment

This solution is a variation on the one in section 5.1.1 with the aim to further reduce the feedback time by removing the need for the consumer to trigger their test suite as can be

seen in figure 5.2.

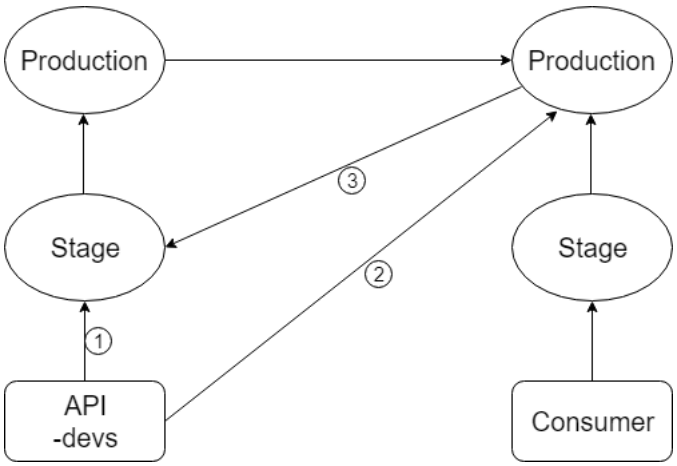

Figure 5.2: Triggered consumer blackbox API regression testing.

In figure 5.2 the following steps are included in the release of a new version

1. The API developers push a release to stage.
2. The API developers trigger the consumer's API tests.
3. The consumer's API tests are run against the API stage environment.

By having the consumers of the API curate a suite of tests that from the consumer perspective covers their access scope for the API. This suite would be part of their own normal set of tests, for their deliverable, but it would be tagged as API tests. This would enable the API developers to trigger the tests to run when a successful deploy to the stage environment has been completed. Upon completion, both teams would be notified of the test results such that failures can be addressed.

This would bypass the need for the consumer to schedule personnel to run the test suite against the stage environment of the API. By having the same test report would ease the communications and coordination efforts between the consumers and API developers.

Stakeholders

The main stakeholders in this situation are:

- The API developers
- The consumers whose tests are run on the staged API.
- The consumers whose tests are not run on the staged API.

Limitations and Risks

The main limitation is that the API developers might over time push the responsibility of testing the API towards the consumers, resulting in the risk of an unsatisfactory test suit

or missed edge cases. Technical knowledge discrepancies relating to differences between the consumers and API developers environments resulting in a large overhead in coordination and education in order to get the scheme into effect. The consumers API tests might stagnate and become outdated resulting in not being a good representation of the consumers API access scope.

5.1.3 Inter-group Time Limited Exchange of Testers

This solution is aimed at increasing the cross-team awareness when it comes to their counterparts regression testing efforts.

By implementing a scheme to exchange testers between the API developers and consumers a greater understanding for the other team's functionality and testing approach can be achieved through direct interaction with their test suites. This can also be done as a series of recurring meetings to exchange ideas and compare current approaches in situations where one team does not have specific testers.

Stakeholders

The main stakeholders in this situation are:

- The API developers
- The consumers whose testers are part of the exchange scheme.
- The consumers whose testers are not part of the exchange scheme.

Limitations and Risks

The main limitation of this solution is that some development teams lack dedicated testers. The duration of the exchange would also greatly affect the value as there would be an initial startup time each exchange cycle. In addition, the exchange would require support from both development teams' managers as it would mean an interruption in the work process. The metrics to gauge the added value of this scheme would be hard to define as it would rely on the subjective opinion of the exchanged testers.

5.2 Summary

The outcome of the interview study and focus group meeting was three different solutions for improving the regression testing efforts with regards to a possible cooperative regression testing effort by the API developer and consumer. Two of these focus on decreasing the validation time of a new API release.

Solution 1 achieves this by having the API developers proactively require the consumers to complete their regression tests against their staged version before it gets released to production. However, there is no guarantee that there is a rapid response from the requested consumers which will result in a longer validation time.

Solution 2 takes this change a step further and has the API developers trigger the consumers' regression test suites. Thus removing the need for the consumers to spend man hours specifically when the API developers need them to. By extension solution 2 creates a greater incentive for the consumers to keep their API regression tests up to date.

Solution 3 focuses on information and knowledge spreading by exchanging testers between the API developer team and the consumer team. This solution enables both teams to gain a greater understanding for the other team's work and thought process with regards to test planning. However, the possible gains were difficult to anticipate from this solution, according to the focus group.

Based on the results from the interviews and the unanimous opinion of the focus group the best solution was deemed to be solution 2.

Chapter 6
Implementation

Based on the results from the interviews and the focus group the best solution was deemed to be the solution from section 5.1.2. The solution comprises of the consumer's test suit being automatically triggered by a successful API deployment to the stage environment. This chapter details the technical environment used to do a proof of concept of the solution and to enable results and conclusions to be drawn on an actual implementation of the design.

6.1 Technical Environment

6.1.1 Jenkins

The main system involved in the evaluation of the chosen solution from section 5.1.2 is the automation server Jenkins, an open source system enabling the automation of key non-human intensive parts of the software development process. Jenkins is an especially helpful automation tool in an organization focused around the continuous integration/delivery methodology [11].

Jenkins is built to be modular, enabling its use with almost any programming or scripting language and version control system. Jenkins is the clear choice in the proof of concept implementation since it is widly used at Axis and is already being used by the two development teams selected for the proof of concept implementation [9].

In the proof of concept implementations, the Jenkins server has jobs that are chained together which creates a workflow like process. Each job has a main task which could be deploying the software or running automated regression tests. The first job in the chain is triggered when a new release is ready for deployment on the stage environment. If the deployment is successful then it will trigger the next job in the chain and so forth. If a job is not successful with its tasks then the metadata from the job can be examined to determine the error. When the error has been handled then the job that failed can be triggered manually to continue the workflow.

6.1.2 Version Control

A key part in the scheme is a working approach to version control of the development projects involved. Though the specific version control tool is not central to the scheme as Jenkins is able to integrate with a wide array of different version control systems.

For the chosen projects, at Axis, the version control systems were git for both the API and consumer side. Git is one of the most common version control systems on the market and open source enabling easier integration with 3rd party systems.

6.1.3 Programming Language

The two projects used in the proof of concept implementation were both written in Java. The Java applications were built using the build automation tool Maven. However Jenkins supports a wide array of different automation build tools making a similar workflow implementation possible for projects in other programming languages.

6.2 Workflow

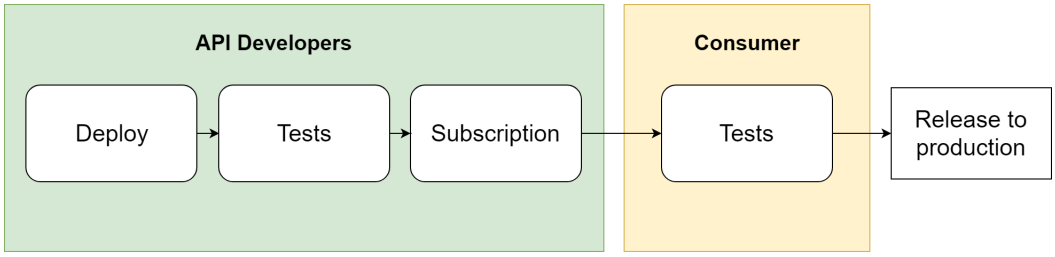

Figure 6.1: Visualization of the cooperative regression testing workflow.

The chosen implementation is applied to Jenkins using a workflow that can be seen in figure 6.1. As the developers of the API push a new version to the stage environment the Jenkins server triggers a series of regression and integration tests created by the API developers, see the API developers deploy and tests tasks in figure 6.1. If those tests are completed without any test case failures a downstream job called subscription is triggered, see the subscription task in figure 6.1. The subscription job serves the function of decoupling the API developers own test job and the trigger which will start the consumers' test suits. This enables an easier grasp of the purpose for each of the chained Jenkins job. In addition, the subscription job clarifies for the consumers of the API which job should trigger their API test suite job. This enables the developers to compartmentalize information such that the handling of consumer regression tests are separate from the tests conducted by the developers of the API.

As the subscription job is triggered, it, in turn, triggers a list of consumer-supplied and curated test suites, see the consumer test task in figure 6.1. By comparing the metadata from these results, conclusions can be made regarding the risk of a regression having occurred. By automating the process of triggering the consumers' API test suits runs the results are almost immediate which enables a faster correction of any emerged regressions.

6.3 First Implementation Design

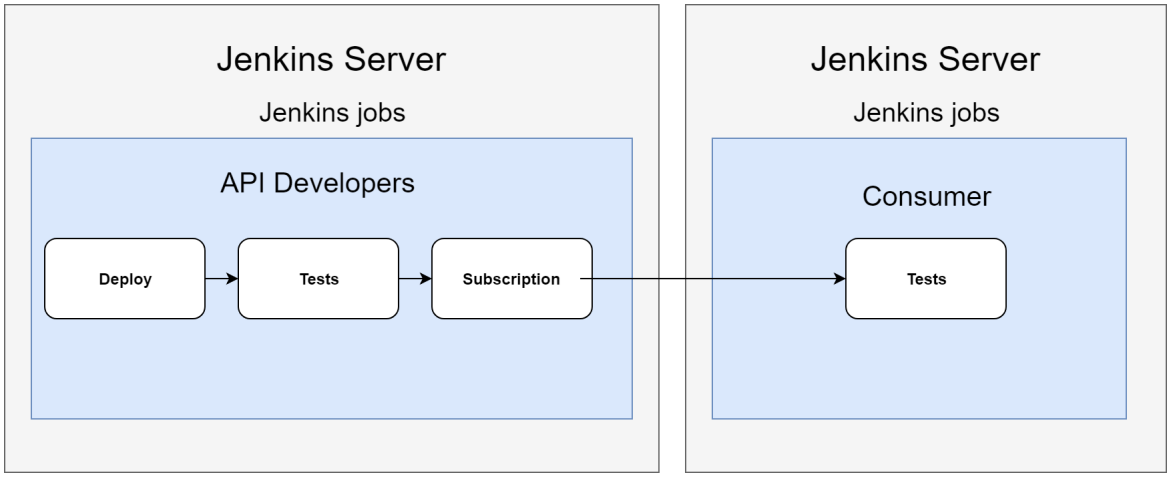

Figure 6.2: Visualization of the first implementation design.

The first implementation included making minor changes to already implemented Jenkins jobs for the respective development teams, see figure 6.2. However different Jenkins servers were utilized by the development teams. To enable the adaptation of the above workflow to a scenario with separated Jenkins instances for the API development and the consumers would need certain alternations to be made. A connection between the two servers had to be designed in order to enable the subscription job to trigger the consumer API tests. This was done by having a dedicated job that was triggered by the subscription job and which sent a network message which triggered the consumer API tests. Two main issues for this implementation would be upstream pipeline status notification and authentication.

The API and consumer projects being hosted on separate Jenkins instances would negate the possibility of leveraging the Jenkins integrated downstream status notification as the function is only available within the same Jenkins instance and no plugin to solve this issue is widely available such that the notification information is bidirectional over separate Jenkins instances.

The separation of Jenkins instances poses a need to authenticate the subscriptions job as to be able to trigger the consumer API tests, this could either be done by using username and password credentials or by way of tokens. The implemented solution used tokens in an unencrypted network message which is not secure as the message could potentially be sniffed and therefore a better solution to this problem would be preferred.

6.4 Second Implementation Design

The second implementation of the workflow involved only one Jenkins server and mirrored the flow in figure 5.2. The second implementation design can be seen in figure 6.3 and was implemented due to the issues faced with the first implementation as well as the fact that the consumer jobs were already planned to be moved to the same Jenkins instance as the API. The consumer test suit was therefore implemented as a Jenkins job on the same Jenkins server.

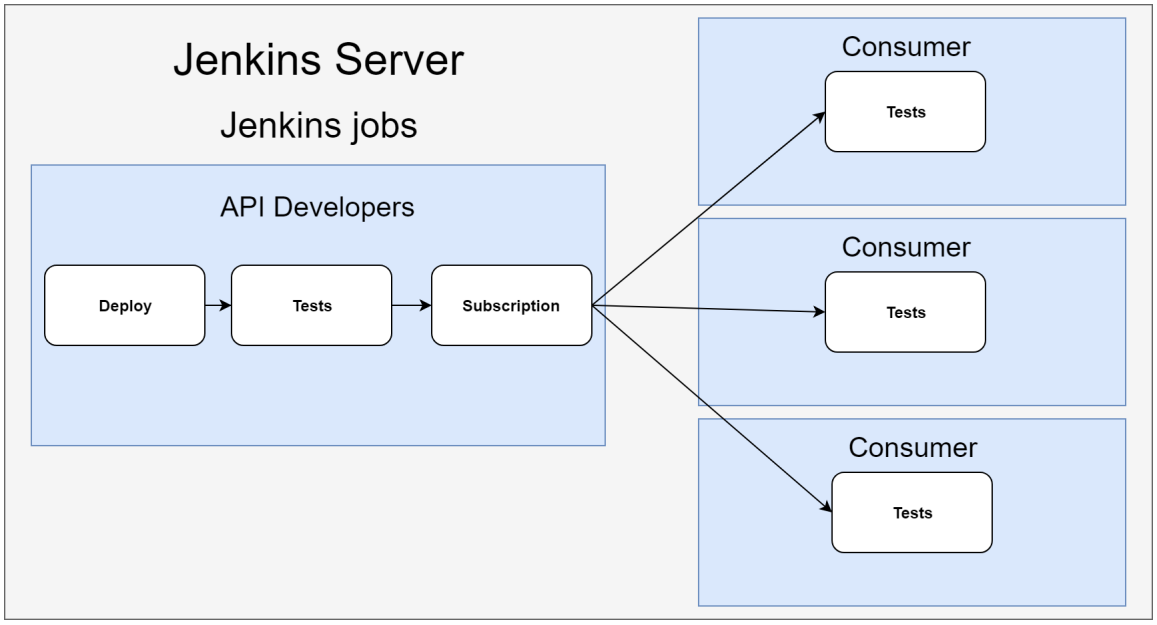

Figure 6.3: Visualization of the second implementation design.

This implementation also served to simplifying and improving this project's implementation as well as remove extraneous factors from the evaluation process.

Another adaptation was the change from running the consumer test suites against the production environment to the stage environment as the regression test suites were not version controlled such that the specific set of regression tests run for the current production release was not easily accessible over time. The impact however of running the regression tests against the consumer stage environments was deemed as having a low impact on the results.

In this proof of concept implementation, it was decided in assent with the API development team that the Jenkins integrated downstream status notification would handle the immediate feedback from the results of the consumer API tests. This is done by Jenkins reporting the results of the consumer API testing job in three different colors. Green being everything passed, yellow meaning there were some warnings or tests that failed and red representing critical errors. This would give the API development team a quick overview of the results but any failures of the regression tests would generate an e-mail report sent to the developers of the application which would then have a deeper look and then report the findings to the API developers.

The status notification can be easily expanded upon such as integrating it with dashboards and code review tools among others. However, the simplistic notification design of the implementation is in this project deemed to be sufficient as the focus in the project is the workflow and not the specific notification format.

Chapter 7
Results and Evaluation

7.1 Evaluation Process

A process of evaluation was established to both ensure that the system fulfills the needs of Axis and to be able to collect relevant feedback from an academic perspective. Due to sprint lengths and planning limitations, the main evaluation process was conducted using an evaluation meeting consisting of relevant experts close to the involved projects at Axis. The evaluation meeting was then followed up by a questionnaire to further catch relevant feedback and reflections from teams that were not involved in the implementation.

7.2 Evaluation Meeting

This meeting was used to evaluate the result of the proof of concept implementation. The main point of the meeting was to analyze how the two development teams elicited in the proof of concept implementation perceived the end result.

7.2.1 Participants

In total five participants were elicited for the evaluation meeting. Three participants represented the API development team with one being a developer, a tester and the product owner of the chosen API. Two developers represented the development team that consumes the API.

7.2.2 Results and Validation

From the API-team, the implementation and enacting of the workflow had been perceived as simple and straight forward. It was seen as a clear improvement as it put focus on the

need for the consumers to verify that they actually perform adequate tests against the API. By running the consumers' tests it gave the API-team a greater confidence that the release fulfilled the de facto requirements set forth by the consumers.

By automating the triggering of the consumers' API tests a lot of waiting and communications overhead can be avoided. By combining the results of multiple consumers, conclusions can be made with regards to possible faults locations and severity further improving the confidence level regarding the correctness of the release. Another benefit of automating this process is that a shorter validation time can be achieved.

Regarding feedback, it was considered that the API-team should receive only Pass/Fail/Warning status reports on the completed test runs through upstream/downstream notification on the Jenkins dashboard. The consumer should, however, receive an email when a test case fail and then be tasked with following up what has caused the failure and then contact the API-team. This conclusion was made based on what type of information each team would be interested in.

One decision taken during the evaluation meeting was that a questionnaire should be sent out to internal API consumers that hadn't been involved in the proof of concept implementation. This would also ease the consumers of that API into adopting the solution presented in this thesis. The benefit of using consumers that haven't been involved up to this point is that they would be more impartial and unbiased in their responses and would give insight into the opinions and willingness of API consumers to implement the suggested workflow using Jenkins.

7.3 Questionnaire

To enable a broader and more impartial viewpoint on the implemented solution other internal consumers of the API's developed by the API-team were elicited. This group involves a large number of varied consumers using different types of programming languages, testing approaches, and team constellations.

With the questionnaire, a short description of the purpose of the survey and a description of the proposed workflow was included. This was done so that the reader could gain an understanding about the implementation and its purpose before answering questions regarding the subject.

7.3.1 Participants

From the questionnaire, a total of 9 responses were sent in over the course of a week. With the distribution of different team roles as can be seen in figure 7.1.

7.3.2 Results and Validity

The majority of the responses, 6 out of 9, answered that their team was currently using Jenkins and 7 out of 9 responded that they were a part of their teams' regression testing effort. The majority had test cases that involved API-calls that were not mocked and only one answered that they had no test cases that involved API-calls, see figure 7.2

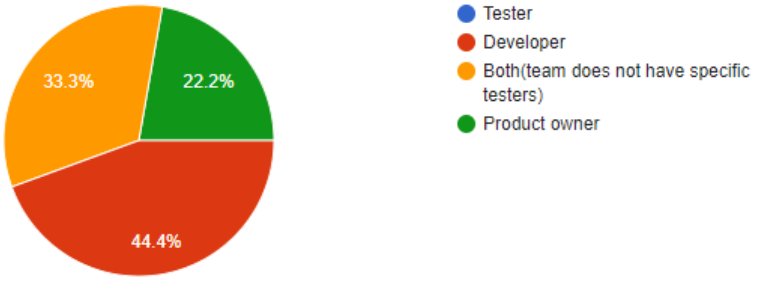

Figure 7.1: Respondents roles in their respective teams.

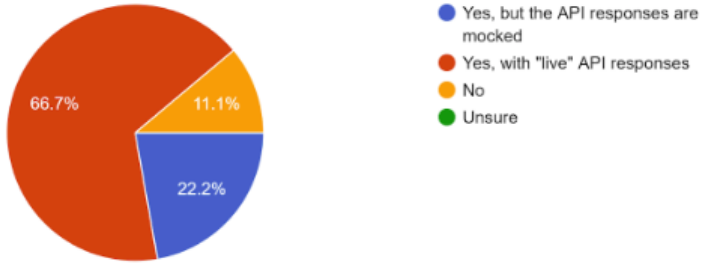

Figure 7.2: Question: Do you have test cases that involve API-calls?

The problem relevance of regression testing including API-calls is clear since almost everyone answered that they had experienced that an API update had resulted in bugs, see figure 7.3. This sentiment could also be reflected in that the majority of the participants felt that implementing the proposed workflow would benefit their work since the average points given was 4.1 on a scale from 1 to 5, see figure 7.4. In contrast when questioned about if the proposed workflow would improve the communication with the development team of the API only an average score of 3.3 points were given.

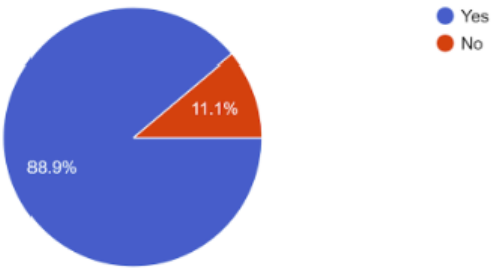

Figure 7.3: Question: Has an API update resulted in bugs/broken your product?

Regarding how the feedback structure should be handled there was some split opinions. Almost half of the participants would like to have the same structure as the proof of concept implementation where the consumers receive an email if a test case fails. The API-team would only get feedback from watching the upstream/downstream notification on the Jenk-

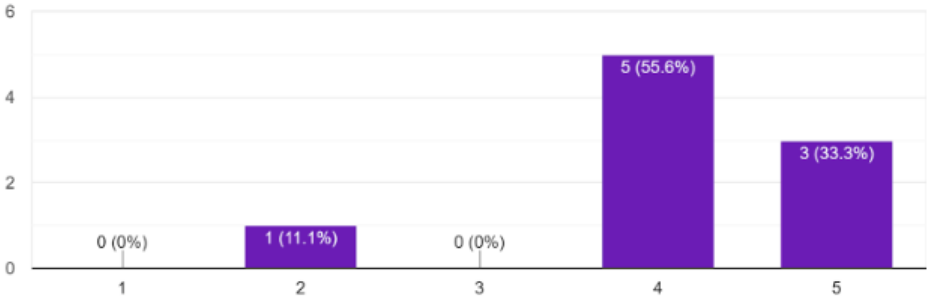

Figure 7.4: Question: Do you feel that the proposed workflow would benefit your work?

ins dashboard. When asked to estimate how long it would take to implement the proposed workflow there was no unified estimation. However, when asked about being interested in this kind of workflow for their team the vast majority expressed an interest, see figure 7.5.

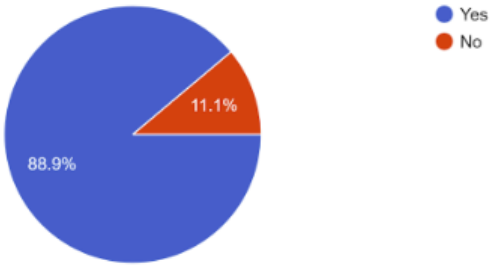

Figure 7.5: Question: Would you be interested in this kind of workflow for your team?

The validity of this survey is questionable since some participants were unsure about the implications and difficulty of implementing said workflow. This could be the result of having limited experience with Jenkins and/or that the information regarding the implementation given not being adequate. However, the opinions and perception of the proposed workflow are mostly positive with it being a perceived improvement to the current practices.

Chapter 8
Conclusions

The purpose of this thesis has been to design a cooperative regression testing effort that can be implemented to shorten validation time for API-releases in cooperation with consumers within the same company. The effort has been focused on producing a workflow that is lightweight enough to be easily implemented and adjustable to be applicable to as many development scenarios as possible.

The answer to **RQ1** is that it is not a uniform group performing the regression testing at Axis. Instead, the practitioners were of different academic background, experience, work titles, and employment forms. How the regression testing was performed also differed greatly with some development teams employing automated regression test suites while others performed regression testing in an exploratory fashion.

The main regression testing challenges stated when researching **RQ2** varied from:

- Having relevant test data
- Differences in production and development environment
- Knowing when you have covered enough test scenarios
- Creating a good workflow between inter-dependent teams
- Missing important aspects or impact areas in the risk assessment

Decreasing the regression test overlap in **RQ3** was deemed to be of lower importance and was thus decided to not be an integral part of the proof of concept implementation. It was instead set aside as possible future work. The shortened validation time was however deemed to be of high importance. A workflow comprising of a cooperative regression testing effort to shorten the validation time before an API release was designed and validated using a proof of concept implementation.

The analysis of exploring the problem areas shows that many of the main regression testing challenges that developers face today could be eased with the use of a cooperative regression testing effort. To facilitate this, an IT artifact has been designed which consists of a

workflow involving interdependent development teams and a technical implementation using Jenkins. A proof of concept of this design was implemented and later evaluated to ensure that it achieved the desired end result.

In order to implement such a regression testing effort, there are however two prerequisites that heavily impact the feasibility of the implementation for the given software project. The first and most crucial part is that there exists a dependency between software projects such that this can be leveraged when performing regression testing. The second prerequisite is the utilization of automated regression testing that includes the functionality of the aforementioned dependency in its test cases.

The observed benefits of implementing this workflow were a decrease in validation times before release as well as increase the confidence in that the release does not break any expected functionality from the consumers perspective. By automating the regression testing trigger, the response time is improved if a regression is found. This enables the API development team to correct the error faster and thus minimizing the risk of missing expected deadlines. The fact that the API team will be able to trigger and validate their release against the functionality the consumers expect, expressed as API-tests, doubles as a validation of the software requirements.

If the consumers already have a well maintained automated regression test suite the change over to the proposed workflow, as described in chapter 6, should be quick and easy to implement. However, if the development teams are not familiar with using Jenkins this would increase the time required to enact the change. The implementation does require a good communication effort between the teams and it is crucial that responsibilities are agreed upon so that the API developers and the consumers have a clear picture of how test case failures are reported and followed up. If the consumers' test suite includes many test cases that do not test the functionality of the API it can be favorable to use test selection. This can be done to e.g. reduce the run time of the regression tests.

The interview and focus group study gave a good overview of current work practices at Axis. The study was a stepping stone, **RQ1**, that was necessary to be able to identify the challenges as perceived by the developers and consumers of APIs within Axis. The challenges, **RQ2**, was to a large extent focused on the communications efforts between teams and less often on technical aspects. As a result of the varied team constellations a workflow that would both improve communications and also to a lesser extent improve the technical coordination between teams, **RQ3**. The workflow was well received both by the teams involved in the implementation and also by consumers of another API that had not been involved in the thesis process giving credence to the conclusion that the workflow was sound.

Chapter 9
Future Work

This chapter includes topics regarding further research into the subject and also possible features for the implementation which could benefit certain software projects.

- **Design validation using a larger set of consumers**: In this thesis, only one consumer was used when implementing the proof of concept. However, the end goal is to use as many consumers as possible in order to get better test coverage so that a higher correctness confidence can be achieved. To better validate the design an implementation of the workflow where more consumers are involved could bring forth new obstacles, such as increased communication overhead.

- **Determining consumer test coverage**: It would be beneficial to enable a way of measuring test coverage based on the consumers' test suites. If this is possible it could have a huge impact on the ability to analyze the results of a test run. The information gathered could be used as metadata for the decision making process regarding when to push to production.

- **Including or using external consumers**: When changing the scope of the implementation to include external consumers the implementation becomes more complex. Since the workflow is not completely in-house, aspects regarding security and communications become more uncertain. One of the main aspects being confidentiality and at what level information sharing would be appropriate with regards to different types of partnership agreements. However, it would be valuable to further research this subject since the possible benefits could be worth-while.

- **Examine the suitability of different kinds of tests and dependency constellations**: In this thesis functional test cases have been the focus, software testing however also involves non-functional tests such as testing the software under heavy load. Using different test types could be evaluated in the context of cooperative regression testing. Other dependencies besides API and API consumers could be tested and evaluated.

These constellations could include e.g. hardware platforms and the firmware running on it.

- **Code Review and Tests as Requirement tools**: It would be beneficial to investigate integrating the implementation with a code review tool such as Gerrit to enable better tracking and validation of the test results. It would also be valuable to look into or design a tool integration to support the effort of tests as requirements. This would give the API developers even better confidence in the release correctness [18].

Bibliography

[1] The Axis story. `https://www.axis.com/about-axis/history`. [Online; accessed 20-Dec-2018].

[2] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, Dec 1990.

[3] IEEE Standard Glossary of Computer Languages. *IEEE Std 610.13-1993*, 1993.

[4] IEEE Standard for Configuration Management in Systems and Software Engineering. *IEEE Std 828-2012 (Revision of IEEE Std 828-2005)*, pages 1–71, March 2012.

[5] ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing –Part 3: Test documentation. *ISO/IEC/IEEE 29119-3:2013(E)*, pages 1–138, Sept 2013.

[6] How to Optimize Regression Testing in Agile Development. `https://www.scnsoft.com/blog/regression-testing-in-agile-development`, 2017. [Online; accessed 20-Dec-2018].

[7] ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541, Aug 2017.

[8] 12 principles behind the agile manifesto, 2018. https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/.

[9] About - Git. `https://git-scm.com/about`, 2018. [Online; accessed 5-December-2018].

[10] Basic Search: Lund University Libraries. `http://lubsearch.lub.lu.se/`, 2018. [Online; accessed 10-June-2018].

[11] Jenkins User Documentation. `https://jenkins.io/doc/`, 2018. [Online; accessed 10-November-2018].

[12] A. and N. Chauhan. A regression test selection technique by optimizing user stories in an Agile environment. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 1454–1458, Feb 2014.

[13] I. Alagöz, T. Herpel, and R. German. A Selection Method for Black Box Regression Testing with a Statistically Defined Quality Level. In *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 114–125, March 2017.

[14] E. Alégroth, R. Feldt, and H. H. Olsson. Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pages 56–65, March 2013.

[15] L. Bendix and C. Pendleton. Configuration Management: Mother's Little Helper for Global Agile Projects? In *2012 IEEE Seventh International Conference on Global Software Engineering Workshops*, pages 28–32, Aug 2012.

[16] M. Chengying and L. Yansheng. Regression testing for component-based software systems by enhancing change information. In *12th Asia-Pacific Software Engineering Conference (APSEC'05)*, pages 8 pp.–, Dec 2005.

[17] B. DiCicco-Bloom and B. Crabtree. The qualitative research interview. *Medical Education*, 40(4):314–321, 2006.

[18] E. Engström M. Borg E. Bjarnason, M. Unterkalmsteiner. An industrial case study on test cases as requirements. In T. Dingsøyr C. Lassenius and M. Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 27–39. Springer International Publishing, 2015.

[19] Beck et. al. The Agile Development Manifesto. `http://agilemanifesto.org`, 2001. [Online; accessed 20-Dec-2018].

[20] Vahid Garousi and Mika V. Mäntylä. A systematic literature review of literature reviews in software testing. *Inf. Softw. Technol.*, 80(C):195–216, December 2016.

[21] T. D. Hellmann, A. Sharma, J. Ferreira, and F. Maurer. Agile Testing: Past, Present, and Future – Charting a Systematic Map of Testing in Agile Software Development. In *2012 Agile Conference*, pages 55–63, Aug 2012.

[22] M. Henning. API: Design Matters. *ACM Queue*, 5(4), 2018.

[23] A. Hevner, S. March, J Park, and S Ram. Design science in information systems research. *MIS Quarterly*, 28(1):75 – 105, 2004.

[24] P. Kandil, S. Moussa, and N. Badr. A methodology for regression testing reduction and prioritization of agile releases. In *2015 5th International Conference on Information Communication Technology and Accessibility (ICTA)*, pages 1–6, Dec 2015.

[25] H. K. N. Leung and L. White. Insights into regression testing (software testing). In *Proceedings. Conference on Software Maintenance - 1989*, pages 60–69, Oct 1989.

[26] B. A. Myers. Human-Centered Methods for Improving API Usability. In *2017 IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI)*, pages 2–2, May 2017.

[27] G. J. Myers, T. Badgett, T. M. Thomas, and C. Sandler. *The art of software testing*. 2 edition, 2004.

[28] B. Regnell P. Runeson, M. Höst. *Att genomföra examensarbete*. 2006.

[29] A. Rauf and M. AlGhafees. Gap Analysis between State of Practice and State of Art Practices in Agile Software Development. In *2015 Agile Conference*, pages 102–106, Aug 2015.

[30] R. H. Rosero, O. S. Gómez, and G. Rodríguez. An approach for regression testing of database applications in incremental development settings. In *2017 6th International Conference on Software Process Improvement (CIMPS)*, pages 1–4, Oct 2017.

[31] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131, Dec 2008.

[32] S. Saito, Y. Iimura, A. K. Massey, and A. I. Antón. How Much Undocumented Knowledge is there in Agile Software Development?: Case Study on Industrial Project Using Issue Tracking System and Version Control System. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 194–203, Sept 2017.

[33] C. Sharma and S. Singh. Mechanism for identification of duplicate test cases. In *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, pages 1–5, May 2014.

[34] H. Suganuma, K. Nakamura, and T. Syomura. Test operation-driven approach on building regression testing environment. In *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*, pages 323–328, Oct 2001.

[35] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky. Agile software testing in a large-scale project. *IEEE Software*, 23(4):30–37, July 2006.

[36] Wikipedia. Application programming interface — Wikipedia, The Free Encyclopedia. `http://en.wikipedia.org/w/index.php?title=Application%20programming%20interface&oldid=850975267`, 2018. [Online; accessed 24-July-2018].

Appendices

Appendix A
Interview Protocol

The interviews were carried out both in English and Swedish. In the case of Swedish the questions were translated of hand. Subitems represent sought information rather than actual questions.

1. What is your name and age?
2. What is you background?
    2.1. Academic
    2.2. Work, type and not necessarily where.
3. Where in the company do you work?
    3.1. Employee or consultant?
    3.2. Previous positions within the company?
    3.3. Work tasks/duties/responsibilities?
4. How is your team comprised?
    4.1. How is the staffing level on your team?
    4.2. What roles do your team members have?
    4.3. What are their levels of experience? Senior/junior
5. What projects do you work on?
    5.1. Who are involved in your projects?
    5.2. Are all the members of your team involved in the same projects?
    5.3. What parts of the development structure do you work on?

6. Do your applications have any dependencies on software (e.g. web based api) that a different team in axis is the developers of?
7. How would you describe the purpose of regression testing?
    7.1. What benefits does it bring your development team?
8. How is regression testing planned in your projects?
    8.1. How is the workflow setup, does it differ between projects?
    8.2. Who is involved in the testing and regression testing activities?
    8.3. Are the requirements linked to the test selection?
9. Are the dependencies a part of the test context? And if so how are the results handled?
10. How is responsibility distributed?
    10.1. Is it explicitly delegated or implicitly?
11. Is it planned alongside other testing activities?
    11.1. Is the same person responsible for planning the other testing as well?
12. How does quality assurance factor in to the regression testing workflow?
    12.1. Expertise, code review, explicit measures?
    12.2. Is the quality assurance verified by someone else than the one responsible for planning?
13. What are the challenges you have encountered with regression testing?
    13.1. What is going well and what is in need of improving?
14. When is software testing performed in your projects?
15. When is regression testing performed in your projects?
    15.1. How is it iterated?
    15.2. How is regression testing activities followed up during the course of the development plan?
16. What tools are used during regression testing activities?
    16.1. Are there separate tools for planning, exec and reporting or are they a suite?
    16.2. How are the results reported/used?
17. Is there anything you want to add?

DEPARTMENT FOR COMPUTER SCIENCE | LUND UNIVERSITY | PRESENTED 2019-02-07

MASTERS THESIS Automated Cooperative API Regression Testing Using Jenkins
A Design Science Study at Axis Communications
STUDENTS Filip Olsson, Philip Ridderheim
SUPERVISORS Per Runeson (LTH), David Vagnell (Axis Communications)
EXAMINER Emelie Engström (LTH)

Using cooperative regression testing to shorten validation times in API development

POPULAR SCIENCE PAPER **Filip Olsson, Philip Ridderheim**

In application development the usage of APIs have become commonplace. This dependency can be detrimental if a new API version introduces unexpected behaviour in the applications consuming it. This work describes how to enable consumer validation of API releases using a workflow aimed at reducing the validation time.

Regression testing is a type of testing that is used to find unintended changes to already established functionality, so called regressions. In agile development regression testing is important as requirements change in a more rapid pace than in traditional development. Adding APIs that might have different development cycles creates the necessity for a thought through workflow. One that enables both the developers of the API and its consumers to streamline the regression testing process while still enabling communication and feedback between them to work.

After researching the current regression testing practices at Axis Communications a workflow was designed to shorten the validation time for API releases. The designed workflow was put to the test with a proof of concept implementation using the automation server Jenkins, see figure 1. Jenkins enables jobs (tasks) to be chained together in order to automate processes.

The purpose of the workflow is to leverage regression tests already implemented for the consumers applications when testing for regressions in the API. The process of having the consumers also test for regressions serves as a validation that the new version of the API has not caused any regressions in the consumers' applications. The process is automating this process shortens the validation time.

The development teams involved in the proof of concept implementation considered that the implementation had achieved its goal.

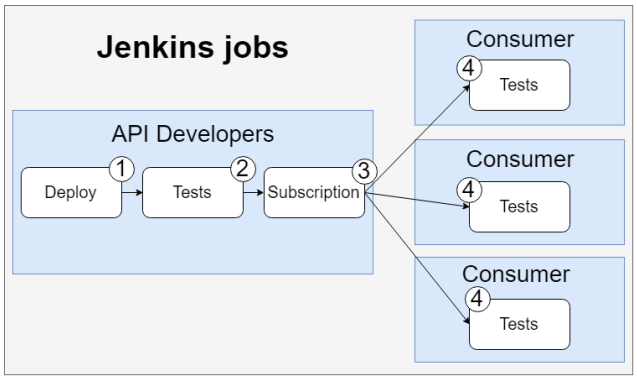

Figure 1: The workflow implemented using Jenkins jobs.