Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

Testing and Gherkin in agile projects

by

Magnus Härlin

LIU-IDA/LITH-EX-A--16/008--SE

2016-02-29

Linköpings universitet

Linköpings universitet
SE-581 83 Linköping, Sweden

Linköpings universitet
581 83 Linköping

Linköping University
Department of Computer and Information Science

Final Thesis

Testing and Gherkin in agile projects

by

Magnus Härlin

LIU-IDA/LITH-EX-A--16/008--SE

2016-02-29

Supervisor: Ola Leifler
Examiner: Bernhard Thiele

Abstract

Testing in agile software development is important to ensure that the right product is being developed. Is it possible to include everyone in agile software development by using a business readable DSL and also create test cases based directly on that DSL?

Observations, interviews, a study of literature, third degree collected artifacts and an implementation has been performed to analyse the process of introducing Gherkin as a tool in agile software development projects. The process of performing and conducting tests has been examined at Accedo to understand how Gherkin together with CucumberJS can be used in projects, with the purpose of increasing collaboration between different roles and create a ubiquitous way of referring to the same piece of software without the need to specifying implementation details.

To include the entire project team in the whole process of developing software is essential for a usage of Gherkin to be successful. Since the purpose is that everyone should be able to contribute as well as understand the progress of development in projects and share an agreement on what is being developed. A business readable DSL provides a uniform format to specifying tasks causing the internal communication to be improved in projects.

Keywords : Agile testing, Gherkin, Business readable DSL

Glossary and abbreviations

Term	**Description**
BDD	Behaviour Driven Development or Behaviour Driven Design
TDD	Test Driven Development
DSL	Domain Specific Language
Gherkin	A specific DSL used for Cucumber
Cucumber	A software tool that runs automated tests, interprets Gherkin
QA	Quality Assurance
Tech lead	A role in projects at Accedo, responsible for the technical depth knowledge in projects
Project Manager (PM)	Project manager, responsible for managing projects
User acceptance testing (UAT)	Testing to assure user acceptance criteria
Content management system (CMS)	System for managing and providing content
UX	User experience

Acknowledgements

This report is a master thesis at the program Master of Science in Computer science at Linköping university. The study was performed at Accedo Broadband AB in Stockholm.

I would like to thank my supervisor at Accedo, Thomas Johansson, for his enthusiasm and guidance through the entire process. At the department of computer and information science I would like to thank my supervisor Ola Leifler for his quick responses with constructive feedback on how to improve my thesis study. Also I wish to thank my examiner Bernhard Thiele for his time and feedback.

Contents

Chapter 1

Introduction

In this chapter some background on why the study is made together with an aim of the study is presented. First an introduction to the topic is given followed by some background of the company that the study is performed at.

1.1 Motivation

Testing in agile software development is necessary to establish that the right product has been built but what defines what should be tested and that the tests conducted are testing the right thing? How is it guaranteed that the tests conducted show that the stakeholder requirements are validated? By using a business readable domain specific language such as Gherkin, specifications can be written in plain English while it is still precise enough to drive development and testing and produce documentation that can be understood by everyone involved in a software development project.

The Agile Manifesto states that working software should be prioritized over comprehensive documentation [1]. The main artifact produced in agile software development projects is code, however it is often difficult to discuss requirements with project managers in terms of code. There exists a need that the specifications of a software development project can be discussed at a higher level without going into implementation details of the code [2].

1.1.1 Accedo

Accedo is a company providing applications, tools and services to media companies, consumer electronics and TV operators globally, to help them deliver the next-generation TV experience. Accedo's mission is to solve the technology challenges of delivering an attractive and successful experience across all connected video devices.

All projects performed in Accedo is performed in a more or less agile manner, where sprints are planned and performed in iterations. There are different approaches on how testing and QA is done in projects. It is important that the communication of what should be tested is clear and unambiguous.

Every project at Accedo has a limited QA budget, this makes it important that the hours spent on QA are well planned and not spent on work that could have been solved in an earlier stage. The idea of QA is not to tell the developers what mistakes they have made but rather ensure that what they have done keeps the desired quality.

Accedo desires higher software quality in their projects and is always eager to improve the way to develop software. Accedo performs their projects according to agile methods. That testing is a part of agile methodology is certain but testing can be performed in many ways. In this study an investigative study will be performed on how testing can be performed in agile methods.

1.2 Aim

The purpose of this study is to investigate if Gherkin can be used to improve the way of communicating software requirements in software development projects and to investigate and understand what Gherkin and CucumberJS can and should be used for in agile software development projects. The long term aim is that the code quality should become higher and ensuring that the right product is being developed, tested and delivered and that all roles in projects can agree on what is being developed.

1.3 Research questions

- Can Gherkin as a tool be used to improve communication between the project members in an agile software development project?

Chapter 2

Theory

This chapter presents some background theory on different development methodologies, communications in agile software development, agile testing that are needed to understand the topic and further describe the main points of this study. Then a description of Gherkin and Cucumber is presented and the theory chapter concludes with research methodology.

2.1 Communications in agile software development

Agile software development works with volatile requirements and business environments, this makes communication in agile projects a crucial factor for a successful project. Requirements that constantly change can easily introduce ambiguity especially when there is insufficient personal communication. Agile methods put more emphasis on verbal communication which can help the detection of defects earlier in the project and direct contact with the customer results in less distortion in the information. Effective communication is an important factor to establish requirements, get feedback and ensure that the right product is delivered. The communication in development projects aims to reduce the project development time and cost of change of software [3, 4].

Communication in traditional, non-agile, development methods travels through a long chain of people causing distortion and loss of information in comparison with agile methods that embrace personal communication thus causing less loss of information and reduces the risk of information being distorted along the way. Even though verbal communication is efficient it is not always as well thought out as formal communication, and conversations can be forgotten over time when not documented [3].

In agile projects where direct communication between developer and customer is encouraged it is important that customer and developer have a common basis of understanding. Direct communication i.e. face-to-face provides a richer type of communication compared to communication through documents. Face-to-face communication provides body language, emotions, voice inflection and instant feedback to a conversation, this makes it possible for faster changes and to deliver the right product faster to the customer. Conversations is not just about facts which documentation often is. Conversations bring a new depth to the context and can help to create common knowledge out of an experience [5].

Communication in agile software development can be both internal and external, where the internal communication focuses on the communication within the project team and the external communication is the communication between the project team and other stakeholders [6].

2.1.1 Formal and informal communication

Communications in software development projects can be divided into formal and informal communication. The formal communication refers to explicit communication in a project such as specification documents, product backlogs and review meetings. While the informal communication refers to the conversations between people in a project. Informal communication in form of face-to-face communication is often presented as the best way to build trust among the people participating in a project, increasing collaboration and causing good productivity in software development projects. The formal communication on the other hand provides an explicit dependency between features and requirements in the project and often provide documentation or other data that can be used in specifying the project [6].

Agile software development methods emphasize direct communication over comprehensive documentation and that internal documentation is of no or little direct use to the end customer. Agile scholars state that documentations should be kept to a minimum and practitioners of agile methods seem to think that documentation is important but they do not want to produce the documentation needed, because they want to focus on the working software according to the Agile Manifesto [1, 7]

2.1.2 Agile communication cycle

Communication in agile processes can be divided into three sub phases. First the primary level which mostly concerns the information gathering and understanding the requirements, when the requirements is established between customer and project team. The second phase is the mid level, where requirements might contain ambiguity and may not be as well defined when it comes to how something actually should be implemented. In the second phase there is more focus on internal communication on a more technical level between team members, during this level the communication between customer and project team is decreased and only regards doubts and ambiguities in the requirements. The internal communication in the second phase can be more technical with the purpose of increasing code quality. The third phase is the end level when it is important with more feedback from the customer, during this phase the customer gives feedback to working software delivered by the team [4].

2.2 Domain Specific Language

A domain specific language (DSL) is used to describe a specific domain problem, a DSL can be either a programming language or an executable specification language. DSLs are used to abstract away the implementation details and express the specifications in a specific domain. Some benefits that can be gained from using a DSL is that they can be used to express the problems in a way that is understandable by domain experts, they allow validation at a domain level instead of at an implementation level. DSLs focus on domain knowledge, thus increasing reusability at a domain level. DSLs also increase readability, maintainability and modificability by all participants in a software development team [8].

Downsides with DSLs is that users of the DSL need to learn the DSL and that everyone affected by the DSL must understand how to use it and use it in a uniform way to bring the most value. There is a risk that it only introduces another way of expressing problems and solutions instead of solving the problem of ambiguous documentation [8].

When considering business readable DSLs the question whether the DSL should be written by business people or developers is often raised. According to Martin Fowler the true value of this types of DSLs is when they are made business readable rather than business writeable. When using a business readable DSL programmers can write the code while the business people still can follow the development by understanding the DSL written for that project [9].

2.3 Test driven development

Test driven development (TDD) is about writing automated test cases first and then only write the code that is necessary to pass those tests. The process of TDD is to write tests first, then write the code to pass those tests and then refactor the code. The process can be summed up in five steps.

- Write a new test case
- Run all the test cases and see how the new one fails
- Write just enough code to make the new test pass
- Re-run the test cases and see them all pass
- Refactor code to remove duplication

One of the goals with TDD is to produce just the code that is needed and not any extra code that might or might not be used, since all code that is written is written to just pass the tests that are written in advance of the code there should not be any unnecessary code written [10].

2.3.1 Acceptance test driven development

Acceptance test driven development (ATDD) is a type of TDD where acceptance tests are used to drive the development. ATDD is used to transform requirements into test cases and verifying the functionality of a system [11].

2.4 Behaviour driven development

Behaviour driven development (BDD) also referred to as behaviour driven design is an outside-in development methodology derived from TDD and ATDD. One goal of BDD is to focus development based on business value, by using a common language between developers, business and testers referred to as a ubiquitous language. One of the main points of BDD is that business and technology should be able to refer to the system under development in the same way [11, 12, 13].

The foundation of BDD is user stories written in the format of As a role, I want a feature, So that some benefit is gained. Which are also used for testing. The tests ensure that the desired functionality and behaviour of the system meets the stakeholder's criteria. Scenarios for functional tests in BDD can be written in Gherkin, and then a framework such as Cucumber can be used to execute automated tests [14].

In BDD software requirements are written using scenarios and are written in a DSL that is readable by business people. The DSL describes the software's behaviour and is used to abstract away the implementation details of the software in the specification. The scenarios written are then used as a communication tool between software developers, testers and business analysts to create better quality in the software produced and to include everyone in the project in the entire process of developing software [15].

BDD is just a clarification of how to do TDD, since TDD is often mistaken as a testing technique and not a design technique, Dan North rephrased TDD into BDD with the ambition of shifting the developers mindset away from the testing to the behaviour of the system [13, 14].

2.5 Agile testing

Agile testers are persons in an agile software development project that are open to change, and collaborate well with developers and technical people as well as with business people. They are part of the development team and are integrated well into the project process. They need to understand the developers point of view as well as understand what is desired from the customer [16]. In agile testing all roles in a development project collaborate in ensuring the desired quality of the software and provide the desired business value [17].

An agile tester is someone who participates in the entire process of software development, and is not afraid to give input to increase testability of the code and the overall quality of the code during planning phases. An agile tester is active during the project and not just when actual tests are written or executed. Agile testers aim to deliver value to the customer, encourage communication internally in projects as well as externally, they respond to change and provide continuous feedback [16].

2.6 Gherkin

Gherkin is a business readable DSL that is interpretable by the framework Cucumber among others. Gherkin allows the project team to describe a software system's behaviour without details on how that functionality is implemented. The purpose of using Gherkin is documentation and to automate tests, and is often used together with BDD [18].

When working with software it is always difficult to find out exactly what the stakeholders actually want. Better communication between developers and business people is a key factor to avoid writing code that never will be used. The usage of concrete examples is one technique that helps the communication between business people and developers [19].

The usage of real-world examples is a common ground that makes sense to business people as well as for developers. When describing functionality with the help of a real-world example business people can imagine themselves using the system and provide useful feedback and ideas before any actual code is written. Below is an example of an acceptance criteria for a credit card payment system

> ”Customers should be prevented from entering invalid credit card details” ([19], p.26).

An acceptance criteria in this manner is quite common and useful, but can be interpreted ambiguously and misunderstood by the reader. There is a lack of precision in what invalid credentials are and what should happen if the wrong credentials are entered. Compare that with the concrete example below.

> ”If a customer enters a credit card number that isn’t exactly 16 digits long, when they try to submit the form, it should be redisplayed with an error message advising them of the correct number of digits” ([19], p.26)

Which is a lot more specific, an acceptance criteria written like this makes it possible for a developer to read almost everything needed to start implementing and business people have a clear idea of what is actually being developed. By using a concrete example in this way an acceptance criteria can be used directly as an acceptance test, that is unambiguous enough to test the behaviour of the system. This is where Gherkin is used in software development and can provide documentation that is easily understandable by business people, developers and by Cucumber. The main purpose of Gherkin is the human readability [13, 19].

Gherkin is written in files with the extension ”feature” where each feature have a independent file, one feature file can consist of several scenarios. Each scenario is described with a title and with the given, when, then format. Below is an example of how a feature file is written.

> Feature : Title, Descriptive text of what is desired and the business value of this feature. That identifies the role, feature and benefit from the feature.
>
> > Scenario : Title describing a business situation
> >
> > > Given some precondition
> > >
> > > And some other precondition

When some action by the role

Then some expected testable outcome

Scenario : Another scenario

Continue with more scenarios in the same format.

[18]

A scenario in Gherkin consists of a list of steps and is an example that illustrates a business value. A step starts with one of the keywords: "Given", "When" and "Then", "And" and "But" can be used to write multiple steps underneath each of the keywords. The scenarios should however be kept simple and unambiguous. Cucumber does not make any difference between the keywords in Gherkin, however it is important that the author of the Gherkin make a separation since that is what makes the Gherkin readable. Cucumber then executes the steps defined in a test file [20].

Cucumber is a testing tool to help software development teams to execute the Gherkin feature files and the test files. The feature file is supposed to be business readable as well as precise enough so that it can be used to drive testing and development and be used as documentation in the project [20].

2.7 Cucumber & CucumberJS

Cucumber is a testing framework that executes .feature files containing executable specifications written in Gherkin together with step definitions written in code. Cucumber does not take the name or description of a feature into consideration, that is just treated as plain text, for users to care about. Cucumber executes the test cases written in code and returns whether the tests pass or not. Cucumber needs step definitions to execute actual tests, each step definitions is a piece of code that is associated with a scenario in the Gherkin feature file. The step definition is then the test executed by Cucumber [20].

Tags in Cucumber can be used to group scenarios together. The tags are written with "@" as a prefix and can be placed before features or scenarios. Each feature or scenario can have unlimited amount of tags. The tags can be used to tell Cucumber which scenarios to run or not to run when executed [20].

Cucumber is a tool used in BDD and was first developed to work with Ruby, but with increased popularity in cucumber, it was implemented to support several other languages. CucumberJS is the adaptation to Javascript for Cucumber [21].

2.8 Research methodology

A case study is an empirical method that aims to investigate contemporary phenomena in their context. There are three other major research methodologies that are related to case studies: Survey, experiment and action research. Survey is a collection of standardized information from a specific population. Experiments are characterized by measuring effects on a variable by manipulating another variable. Action research aims to change some aspect of the focus of the research, differences between action research and a case study is that a case study is observational while action research is involved in the change process. A case study often contains influences from other research methods, a study of literature is often done prior to a case study. In software engineering a case study is a suitable method since it studies a contemporary phenomenon in its natural context [22].

When performing research on real world issues a trade-off must be made between the degree of realism and the level of control. By increasing the level of control the degree of realism is often decreased, case studies are per definition performed in real world settings [22].

The research process can be defined as fixed or flexible, where in a fixed process everything is defined when the study begins. In a flexible process the parameters defining the study can be changed through out the conduction of the study. Case studies are mostly performed with a flexible process [22].

2.8.1 Case study

A case study research consists of mainly five steps. The five steps are: case study design, preparations for data collection, collection of evidence, analysis of collected data and reporting of the performed study. Since a case study is performed in a flexible process the steps above are often iterated through several times during the conduction of the study [22].

The objective of a case study is for example one of the below listed:

- Exploratory, to investigate what is actually happening
- Descriptive, to describe a phenomenon
- Explanatory, to find an explanation of a phenomenon
- Improving, to introduce an improvement and observe the effects.

The objective of a case study is initially expressed as a focus point and evolves through the conduction of the study. In software engineering the case can be a software development project [22].

2.8.2 Data collection

Data collected in empirical studies may be quantitative or qualitative. Quantitative data consists of numbers and classes while qualitative data consists of words, descriptions, pictures and so on. Quantitative data is analysed using statistics while qualitative data is analysed through categorisation and sorting [22].

Data collection techniques can be divided into three levels, which are referred to as first degree, second degree and third degree. In the first degree level, data is collected in real time and the researcher is in direct contact with subjects, for example interviews, focus groups, and observations. In the second degree, data is collected without interaction from the researcher, for example when usage of software engineering tools are automatically monitored. Data collected in the third degree is an independent analysis of work artifacts that are already available. For example when documentation produced in software development projects are analysed. First degree data collection requires more effort from both the researcher and the subjects, however both first degree and second degree data collection allows the researcher to control what data is collected. Third degree methods require less effort but do not allow the researcher to control the data collected or the quality of the collected data. Data collected with third degree methods are created for different purposes than the research study [22].

Triangulation is a way of increasing the precision of collected data in empirical research. The idea of triangulation is to take different angles of the studied objects. Triangulation is important when relying on mainly qualitative data to bring more precision to the study. Different types of triangulation can be applied to a research method [22].

Interviews

Data collection through interviews is an important contribution to case studies, during interviews the researcher asks a series of questions to a set of subjects about the topic of interest in the case study. Interview questions are based on the research questions formulated in the study and aims to provide the subjects opinion on the matter [22].

Interviews can be categorized into unstructured, semi-structured and fully structured. Questions in unstructured interviews are mainly general concerns and interests and the interviews are allowed to develop based on the interests of the subject and the researcher. A fully structured interview is quite the opposite where exact questions and an order of the questions is prepared prior to the interview by the researcher. In a semi-structured interview questions are prepared prior to the interview but the order which the questions are asked is decided by the development of the interview. The purpose of semi-structured interviews is to allow improvisation and exploration of the subject while still assuring that all questions are answered through the interview. In case studies semi-structured interviews are common [22].

Interviews are mostly performed with one subject at the time however group interviews can be conducted as well. Interviews should be recorded to allow the researcher to focus on the interview rather than taking notes on what is being said, after the interviews it is recommended that the records are transcribed into text and that the subject is allowed to read through the text and allowed to alter their statements during the interview if desired. Interview subjects should be chosen with the goal of achieving different results from each interview rather than trying to replicate similarities [22].

Observations

To investigate how a certain task is performed by a group of subjects, observations can be conducted. There are different approaches for observations, one approach is observation of meetings where subjects interact with each other thus generating data for the study. During observations the researcher can be either highly interactive or lowly interactive and the subjects can have a high or a low awareness of being observed. Observations can be useful in case studies to provide a deep understanding of the studied phenomena [22].

Chapter 3

Method

A case study together with a study of literature was made in this study. The case study to investigate a phenomena in a specific context. The literature study to get a further theoretical understanding of the phenomena. Focus was on two different projects which will be referred to as project A and project B.

3.1 Case study

During the study the focus lied on one project (Project B) at Accedo. In that project different meetings were attended, artifacts from the project where analysed, a demonstration implementation was made and interviews with the participants in the project were conducted. In project A the QA kickoff meeting was observed to further understand how the process of planning and assigning QA resources is made in projects.

In project B a Gherkin feature file was written and automated Cucumber test cases was implemented to show how they could be used in that project. The implementation consisted of taking one feature of that project and write that feature in Gherkin and implement CucumberJS tests in Javascript based on that Gherkin document. The purpose of implementing one feature in this way was to show the entire process chain from writing specification in the Gherkin syntax to implementing CucumberJS test cases. This example was demonstrated and compared with the traditional way of specifying, developing and testing software. The demonstration implementation was used during interviews to give the interviewed subjects an introduction to Gherkin. The participants in that project were then allowed to give their feedback on the methodology through interviews and what effects it could have in that project.

3.2 Literature study

The literature study was made during the first five weeks and was used as a base of knowledge for the study and for the theoretical framework. The literature has been used to get a wider understanding of the problem area and possible methods. The purpose of the literature study was to get enough data on what Gherkin is used for and what Gherkin can help with related to communication and specification in development projects and different development methods that are suitable to apply to achieve clearer specifications containing less ambiguities and misunderstandings.

To find relevant literature first the titles were considered, and those articles and papers that were considered to have relevant titles the abstracts were read to sift out the articles and papers with the most relevance. When the abstracts were read the introductions and conclusions on each were read to further decide their relevance for the study. The remaining articles and papers were read entirely as a last way of deciding whether they should be included or not.

Search words that have been used are mainly: Agile, communication, testing, software development, behaviour driven development, domain specific language, test driven development in various orders and combinations. Search engines and databases used are Google Scholar, IEEE Xplore, ACM Digital library, Springer Link.

3.3 Data collection

In this study a qualitative data collection method was chosen, to collect data through interviews and observations on different occasions during the study. During the study artifacts created in project B was analysed.

Project B was observed to see how Gherkin can be introduced to an agile software development project. An example of how Gherkin can be used was implemented in this project to show the concept to participants in that project. The purpose of observing one project was to get a wider understanding of the entire process of performing projects at Accedo.

The participants in project B were asked what the effects of using Gherkin together with CucumberJS have and in what way it would bring the most value in a software development project.

The documentation produced in project B was also studied, to understand how documents are created and what purpose they fulfil. The existing documentation was studied to understand what parts of the documentation could be replaced by documentation written in the Gherkin syntax and if Gherkin could help in creating corresponding documentation or new documentation, both internally and externally.

3.3.1 Artifacts in project B

The main artifacts studied in project B were the tickets produced. The tickets were created for each task that was to be developed in the project. Each ticket associated with a specific task was assigned to different persons in the project, when a developer had implemented that specific task the ticket was labelled as ready to test, and then QA tested the functionality corresponding to that ticket. Depending on the outcome of the tests performed by QA the ticket was either labelled as done or a new ticket was created labelled as a bug, and then assigned to a developer in the project.

Five task tickets and five bug tickets were chosen to represent how tasks and bug reports are formulated in project B. The task tickets were chosen because they illustrate problems in terms of clear, unambiguous specifications. The first task ticket corresponds close to the Gherkin feature implemented in the study and was therefore chosen as an example to show differences in specifying tasks.

The task tickets were analysed and compared to the Gherkin format to show the difference between the task tickets and tasks written with a Gherkin syntax.

The bug tickets chosen to represent how bug reports are handled in project B were chosen to represent the format of bug tickets in project B. All bug tickets are on the same format describing steps to reproduce the bugs, the actual results and the expected results. The bug tickets were chosen to be investigated further since they represent how testing actually is performed as well. The bug tickets produced in project B is the documentation produced in terms of test results from the exploratory testing performed in project B. The bug tickets were sorted based on the reason they occurred and mapped to a Gherkin format to show how the information in the bug tickets could have been used in a Gherkin syntax.

3.3.2 Observations

In the two different projects the QA kickoff meetings were attended. The purpose of those meetings was to decide how the QA hours was going to be distributed on the projects. How many hours that should be spent on different tasks, what types of testing that is desired and required, and in what way the quality in the software delivered is assured. In these QA kickoff meetings everyone participating in that project attended. In the QA kickoff meeting in project A there was a project manager, tech lead, QA lead and in the project B meeting there was a project manager, tech lead, developer, QA lead and a designer.

The meetings were observed to get an understanding of what problems exist in terms of communicating specifications and requirements in projects at Accedo. During the meetings I observed and took notes on how the discussions went forward and how decisions were made. In project A I was observing and taking notes thus causing a high awareness of being observed among the participants.

The QA start off meeting in project B had a quite different setup, where discussion groups were formed and given different areas to focus on so that everyone in the project could contribute with their knowledge. In this project I was taking part in the discussions thus causing a lower awareness of being observed among the participants and getting more insight in the way the discussions were performed.

3.3.3 Interviews

Interviews were conducted to get input from different roles in project B and understand what effects an introduction of Gherkin as a DSL together with automated test cases with CucumberJS has in a project. The interviews also aimed to understand how testing is done in relation to projects and what relation the different roles have to testing and quality in agile software development projects.

The interviews conducted were performed in a semi-structured manner where questions were prepared before the interviews and were aimed to be answered while the interviewees were still allowed to speak freely around the questions and on related topics. All interviews were recorded to make it easier to focus on the interview rather than collecting data, the recordings were listened through after the interviews.

All interviews started with a demonstration of how Gherkin and CucumberJS could be implemented in project B, with the Gherkin feature file and test cases that was written for project B. The implementation was used to show the concept of Gherkin and CucumberJS and how the tools could be applied. First the interviewees was allowed to give some general thoughts and ask questions on the demonstration before moving on to the questions that was prepared. The entire interviews were transcribed into text. The interviews conducted lasted between 30-60 minutes.

One interview was performed with the director of QA at Accedo, he was chosen as an interview subject because of his experience from the QA department at Accedo and because he could provide a QA perspective. He has long experience from working with QA at Accedo and has seen many projects succeed and fail in terms of testing and communication. Another interview was held with the developer in project B, he has been working at Accedo in different projects for less than a year. He was chosen as an interview subject to provide information on how he perceived the testing and QA in project B. The fact that he is quite new to Accedo also made him open to alternative development methodologies and tools such as Gherkin. One group interview was held with a tech lead with several years of experience from the company together with the director of QA.

For the group interview no specific questions were prepared, for the other interviews a different set of questions was prepared for the interview with the developer in project B and the director of QA, the questions prepared can be viewed in Appendix A and in Appendix B.

Group interview

A group interview was held with one experienced tech lead and the director of QA at Accedo. In this interview the process of planning and conducting tests at Accedo today was discussed and also how Gherkin and CucumberJS could fit into projects today and how it could be applied and bring the most value to projects. In the group interview it was also discussed what the biggest differences would be by using Gherkin and CucumberJS compared to how the work is done today. The interview was recorded and some parts of the interview was transcribed into text. The reason to not transcribe the entire recording into text was because the meeting sometimes drafted away from the topic and those parts were not relevant for the study. The group interview was performed in an unstructured format, where mainly points of interest where prepared before the interview and discussions during the meeting were allowed to control how the interview developed. The conducted group interview lasted for about 70 minutes.

3.4 Implementation

In project B one functionality was chosen to test out how Gherkin and CucumberJS could be implemented, to describe and test that specific functionality. The functionality was broken down into a business value so that a feature file could be formulated. The formulated feature file was then used to create test cases that could test the specific functionality that was chosen. In project B a client was implemented that was depending on API calls to a Content management system (CMS), called AppGrid, to fetch data for the client and the test cases ensured that data in the correct format were fetched from the CMS. The implemented feature in Gherkin and the test cases written in Javascript as step definitions were used in demonstrations during interviews.

The Gherkin feature file was written based on a specific functionality in the app. The functionality chosen was discussed with the developer and tech lead in project B to make it easier for them to relate to the demonstration. That feature file was then used to create test cases that fulfilled the definitions in that feature file. The step definitions were implemented in Javascript to be able to be executed by CucumberJS.

Chapter 4

Results

In this chapter the results from the study is presented. The chapter presents results from the analysed artifacts, interviews conducted, implementation and observations. The chapter concludes with a section describing some obstacles and challenges with introducing Gherkin and CucumberJS in projects.

4.1 Artifacts project B

The main artifacts produced and used in project B is tickets. Tickets can be of different types, the types used in this project are task tickets and bug tickets. A task corresponds to a desired functionality in the app, which contains a description of what is to be developed. A bug ticket is created when a bug is found by QA and consists of a name, a description on how to replicate the bug, the expected results and the actual result. The different tickets are assigned to different project team members and are used as an internal way of communicating functionality and bugs.

Tickets are handled in a web-based system that all participants in the project has access to and can watch, create and change tickets throughout the project. All tickets have a title and they have information describing the ticket in more detail. They have a type describing whether it is a task or a bug, a priority, which sprint it is assigned to, a status describing the current status and they also contain a description that describes what is to be performed.

4.1.1 Tasks

The tickets in project B were used to describe tasks to be performed to achieve the desired functionality in the project. The description in task tickets is a short text describing what functionality to implement. In the table 4.1 task tickets from project B are shown.

Table 4.1: Task tickets

Title	Description
Task 1: Theme carousel	1. Get themes banner and data as per the appGrid CMS data 2. Theme should be filtered as per device country and language settings 3. Playlistgrid should be populated as per the theme banner selected
Task 2: Exit key on splash	Exit key press on splash screen should be handled and app must exit gracefully
Task 3: Truncate long titles	Truncate titles which are too long and get covered by the return icon in the player overlay
Task 4: Playlist grid, make use of high quality thumbnails	When there is high quality thumbnails available use them and make the css rules based on quality. Different quality thumbnails have different sizes of black border
Task 5: Show theme in same order as in AppGrid	Show the themes in the same order in the app as they are listed in the AppGrid Region lists

Table 4.2: Table showing a comparison between Gherkin and the task tickets

Gherkin	Task 1	Task 2	Task 3	Task 4	Task 5
Feature					
As a	App client	User on the splash screen	-	-	-
In order to	Display playlistgrid populated with correct and filtered content	Leave the app	-	Display highest quality thumbnails available	Display themes in same order as listed
I want	-	To press exit and the app to exit gracefully	-	-	-
Scenario					
Title	Theme carousel	Exit key on splash	Truncate long titles	Playlist grid, make use of high quality thumbnails	Show theme in same order as in AppGrid
Given	-	On splash screen	Titles are too long	There is high quality thumbnails available	-
When	Data is fetched from CMS	Exit key press	A title is displayed	Displaying playlist	Themes are fetched from AppGrid CMS
Then	Get themes banner and data as per the appGrid CMS data **And** Theme should be filtered as per device country and language settings **And** Playlistgrid should be populated as per the theme banner selected	App must exit gracefully	Truncate titles which are too long and get covered by the return icon in the player overlay	Use them and make the css rules based on quality	Show the themes in the same order in the app as they are listed in the AppGrid Region lists

These tickets are created as artifacts in project B. The purpose of these ticket are for the developers to understand what functionality should be implemented. The ticket descriptions in these task tickets could very well have been written with a Gherkin syntax with the purpose of clarifying what to implement and what tests to conduct. Especially a Gherkin syntax would have provided a clear connection from the business value to the test cases. Which is quite hard to follow just by reading these tickets, what to actually do with these tickets require insight in the project and the tickets are not sufficient to provide unambiguous specification on what to develop, deliver and test.

Task ticket 1 in table 4.1 displays several steps that would have been broken down into scenarios in the Gherkin syntax, task 1 in table 4.1 is more or less describing the same functionality as seen in the implemented example in appendix C. The Gherkin feature description is more specific in terms of what is actually expected and provides more details on how the tasks actually should be implemented without dictating how the code should be written.

In table 4.2 a comparison is shown between the task tickets and the Gherkin template. Table 4.2 shows the task descriptions sorted as Gherkin syntax. The light blue cells in table 4.2 show information that is implicit in the description in the tasks while empty cells indicate a lack of information for a Gherkin syntax.

As seen in table 4.2 it is not certain that a usage of Gherkin would have given any extra information compared to how the task tickets are written. However Gherkin would provide a uniform way of expressing tasks so that every task is written in the same format, and the context, action and outcome of each ticket is easily identifiable.

4.1.2 Bug reports

Bug reports have another detail describing the environment the bug was found e.g., the device and operating system the bug was found in. The description in bug tickets contains information about how to reproduce the bug, the actual result and the expected result. In table 4.3 bug tickets that were produced in project B are shown. In bug 5 rewind is mentioned, the functionality of rewind (RWD) is supposed to rewind the video 10 seconds.

Table 4.3: Bug tickets

Bug 1: Player - the banner should disappear only when navigation has stopped and not during navigation	**Steps to reproduce:** 1. Launch the app 2. The home page is displayed with the carousel and playlist grid based on the theme in the carousel 3. Select any item in the playlist grid 4. Playback starts 5. In the player navigate between different player controls on the banner **Actual result:** The banner disappears while navigating between different player controls **Expected result:** The banner should disappear after few seconds when there is no navigation
Bug 2: Splash screen is not loaded correctly when the device language or country is changed	**Steps to reproduce:** 1. Change the country and language 2. Launch the app **Actual result:** There is a glitch before the splash screen loads **Expected result:** The loading of splash screen is smooth
Bug 3: The order of the playlist items is messed up and playback controls are lost	**Steps to reproduce:** 1.Play the last item in the playlist grid 2.Press ”next video” or ”previous video ” buttons **Actual result:** The order of the playlist items are messed up. Sometime the same item is visible twice. Finally the player controls are lost **Expected result:** The order of the playlist is consistent in both the landing page and the player
Bug 4: Player does not remain pause state when FW/RWD	**Steps to reproduce:** 1. Play any content 2. Press pause on the player 3. FWD/RWD on the player **Actual result:** Content starts playing after immediate FWD/RWD **Expected result:** Player should remain in pause state
Bug 5: Cannot rewind in the beginning of playback	**Steps to reproduce:** 1. Launch the app 2.Select any item from the playlist grid and play 3. Playback starts 4. Press Rewind button during first ten seconds of playback **Actual result:** Rewind doesn’t work as expected **Expected result:** The clip should rewind and playback continues

These are examples of the artifacts created in project B in terms of bug reports and is what was passed on from QA to developers on what and how tests failed. In the bug tickets in table 4.3 it is described how the bugs were found together with actual results and expected results. All bug tickets were created in the same format. The bug tickets are what describes how the testing in project B has been performed as well, the testing in project B was exploratory with the goal of finding bugs, the bug tickets are the documentation produced in terms of how the testing is performed. The testing that did not result in finding bugs did not produce any documentation in the project. The bug tickets in table 4.3 occurred for different reasons. In the bugs 1, 4 and 5 there was most likely a miss in communication on what the desired functionality was and in bug 2 and 3 there are faults in the code that caused the bugs to appear. In table 4.4 the bug descriptions are mapped to how Gherkin scenarios could have been written.

Table 4.4: Table showing a mapping from bug tickets to Gherkin

Gherkin	Bug 1	Bug 2	Bug 3	Bug 4	Bug 5
Scenario Title	Player - the banner should disappear only when navigation has stopped and not during navigation	Splash screen is not loaded correctly when the device language or country is changed	The order of the playlist items is messed up and playback controls are lost	Player does not remain pause state when FW/RWD	Cannot rewind in the beginning of playback
Given	The home page is displayed with the carousel and playlist grid based on the theme in the carousel	-	Last item in playlist grid is playing	During play of any content	Playback starts
When	Select any item in the playlist grid and navigate between different player controls	Changing country and language and relaunch app	Pressing "next video" or "previous video" buttons	Press pause on the player **And** press FWD/RWD on the player	Press Rewind button during first ten seconds of playback
Then	The banner should disappear after few seconds when there is no navigation	The loading of splash screen is smooth	The order of the playlist is consistent in both the landing page and the player	Player should remain in pause state	The clip should rewind and playback continues

4.2 Implementation

An example of how Gherkin could be applied in project B was implemented. The Gherkin feature file created in the study can be seen in appendix C. When executing CucumberJS without any defined step definitions CucumberJS gives as output what methods to be implemented and what they should test in plain text based on the feature file. Those methods were copied into a Javascript file and step definitions was implemented for each method. The test code written can be viewed in appendix D.

4.3 Interviews

In this section results from the interviews conducted are presented.

4.3.1 Testing in projects today

In projects at Accedo today there is no strict definition on how testing is to be performed. In every project there is a QA kickoff meeting with the purpose of defining how the desired quality in the project is assured. These meetings have no explicit format except that they all have a QA kickoff checklist that is to be filled in prior to the meeting and used as a base for the meeting.

Accedo has one department responsible for new projects and another department for products. The products department has a strict plan on how testing should be performed when adding new functionality. While the projects department sets up new strategies for testing and assuring quality in each project. As the director of QA said in an interview *"Testing in products is always the same, there is a clear definition on how to test, while in projects there is never the same"*.

All projects at Accedo is performed in different ways and this makes it difficult to create one uniform way of how testing should be planned and conducted in every project. The director of QA at Accedo says *"The important thing is that there is an agreement between the different roles on what we are developing"*. He also agrees that the introduction of an business readable DSL is a way to include everyone in the process of creating an agreement on what is being developed and how it should be tested. The Gherkin syntax can provide this in projects but requires that all roles are interested in participating in the formulation of the agreement. Further the director of QA says *"It is important that we share knowledge, all roles in projects must understand how QA works in order to contribute to our work"*. This means that for Gherkin to be effective in projects at Accedo it is needed that everyone is interested in the entire process of developing software.

The collaboration between QA and developers in projects at Accedo is in need to be improved, the QA department know exactly what to test and how to work when it comes to the products department but in projects there is little or no formal definitions on how to work. As the director of QA said in a interview *"When asking developers if it is working they answer ask QA and when QA is asked why it is not working they answer ask the developers"*. This is a vicious circle that cause irritation and misunderstandings between QA and the developers. As the developer in project B says *"Sometimes QA does not at all test the things that we thought we told them to test"*. And these sort of misunderstandings can be very time consuming.

4.3.2 Testing in Project B

The testing performed in project B was mostly manual exploratory testing performed by QA. The developer in project B developed and sent builds over to the QA department to test all new functionality. The developer in project B says in an interview *"I think they mostly click around in the app to test the desired functionality"*. Even though the project had an assigned QA role in the project there was still some ambiguity on what was supposed to be tested. The developer said *"The last days of the project we received a bug report from QA that showed a bug that has been there from day two of the project"*. This makes the work for the developers hard, since the bug has been there the entire project but not noticed until the very last few days of the project. The reason for bugs being found and reported in this way are many but a more detailed test plan with clear unambiguous specifications would cause more structure to the testing and it would be easier to see what parts was tested or not earlier in the project. A specification written in the Gherkin syntax can provide this information, and make it more clear what functionality is developed and tested when directly related to a specific Gherkin scenario or feature. Especially it provides a way for developers and QA to refer to the same piece of software in a ubiquitous way.

The developer in project B stated *"By referring to functionality through a Gherkin syntax, it is possible for PMs, QA and customer to understand what piece of functionality is missing"* in comparison to show a piece of code with the implemented functionality it becomes much more clear at a management level how the development is progressing. The Gherkin feature file can be used as a checklist on what functionality is implemented and what is yet to be implemented. This also brings value for QA that can easier understand what functionality is actually implemented so that they do not find a bug in the app that the developers are quite aware is not yet implemented properly. Since QA is performing manual exploratory testing it is important that they know exactly what functionality is implemented and what is missing. To test functionality that is not yet developed is an common pit fall in cases where they just test based on a statement of work (SoW) or some other vague specification such as the task tickets in table 4.1. With Gherkin it would be easy to refer to specific scenarios that should be tested.

4.3.3 Quality in software development projects

All interviewees agree that the responsibility for code quality should be on the developers writing the code. The developer in project B state that *"Even if QA is responsible for asserting quality, they do not have the technical knowledge to create code quality"*. Therefore the responsibility of code quality must be on the developer. The interviewee further states that he has no direct expectation from the PM regarding code quality. Even if he agrees that the PM should be responsible for the overall quality of the delivered product.

The testing performed by developers today is mostly sanity checks to assert that the functionality actually is there before sending it over to QA for assertion. When discussing quality and testing with the developer in project B he says *"You do not loose as much time as one might think by spending development time on testing, especially regressions in the code can be found much earlier"*. However it is not time he chooses to spend during development since it is not expected.

When discussing whether the introduction of a business readable DSL is a good idea or not for projects at Accedo at the group interview, the tech lead at Accedo said *"If we never challenge the way we are working in projects we will not get anywhere in terms of quality"*. A big issue discussed during the group interview was when there is time to test new tools and methods in projects and the tech lead answered *"We should not be afraid to fail sometimes, at least then we know what is not working out for us"*. This shows that at least there is an openness to new tools and methods. While he still agrees that no one probably want to spend their time in a project and risk failure when they are presented with the choice of working like they are used to or to test out something new that they have never heard of. An introduction to how the new tools and methods should be used is necessary for persons in projects to even consider applying new tools and methods in a project.

4.3.4 Applicability of Gherkin in projects

When discussing with the developer in project B if it would be possible to use Gherkin and CucumberJS in that project he says *"I do not think it would have been possible in this project, since it was unclear what was actually going to be developed from the beginning"*. While he still thinks that automated tests executed by CucumberJS could have been useful for some parts of the project, and that testing in the project would have been more successful with a more detailed test plan.

To be able to use the Gherkin and create test cases based on the feature file there is a need for a test engineer in that project that can create the actual test to be executed.

For Gherkin to really express it's true potential it needs to be written by a cross-functional team. When the developer in project B was asked who should be responsible for writing the Gherkin feature files he answered *"I do not think a developer in a project can write a Gherkin specification alone"*. For the Gherkin feature files to be written specific enough for testing and development and still abstract enough for everyone to understand it should be written in collaboration with one representative from each area of interest. The area to be represented should be development, testing and project management. The developer in project B also said that *"There is a need for at least some technical knowledge to be able to write features in a Gherkin syntax"*. This further emphasizes the need for a common understanding between all roles in a development project and further an interest of contributing to the specifications by all participants in the project.

4.4 Observations

The QA start off meeting in project A consisted of discussions between the project manager, tech lead, QA lead and director of QA. The QA kick off template was filled in prior to the meeting that was further discussed and details to the test plan were decided during the meeting. In this meeting the project manager was in charge of deciding how the QA hours and development hours should be spent in the project. While the tech lead was providing the technical depth knowledge in relation to development, and the two persons from QA provided their experience and knowledge in testing and quality assurance.

The QA start off meeting in project B had quite a different approach where everyone participating in the project attended and could contribute with their specific area of expertise to decide how the QA hours should be spent in that project. The team was divided into three pairs that was assigned different areas of the project do discuss. The tasks assigned to the different groups was to discuss what QA needed to be done in the specific areas. The areas was UAT, integration tests and UX (User experience). Then a discussion in the whole group was held to decide how the QA hours should be spent on the different areas and what QA should focus on.

In both meetings there was an expectation from the project team to create scenarios of the project which later could be translated into test cases by QA. Even though most projects are agile with cross-functional teams the QA process was considered to be external to the actual project by most project members.

The communication process can be seen in figure 4.1. The figure shows the route of communication and in what format the communication is in. The figure shows how internal communication in projects is done today. Figure 4.1 describes that all roles communicate with each other in specific terms and a shared communication between all roles is little or even non-existent. From each box via arrows there is a translation taking place by the role in the box, for example the PM translates requirements into acceptance criteria and user stories, these are then translated into code by developers and into test cases by QA.

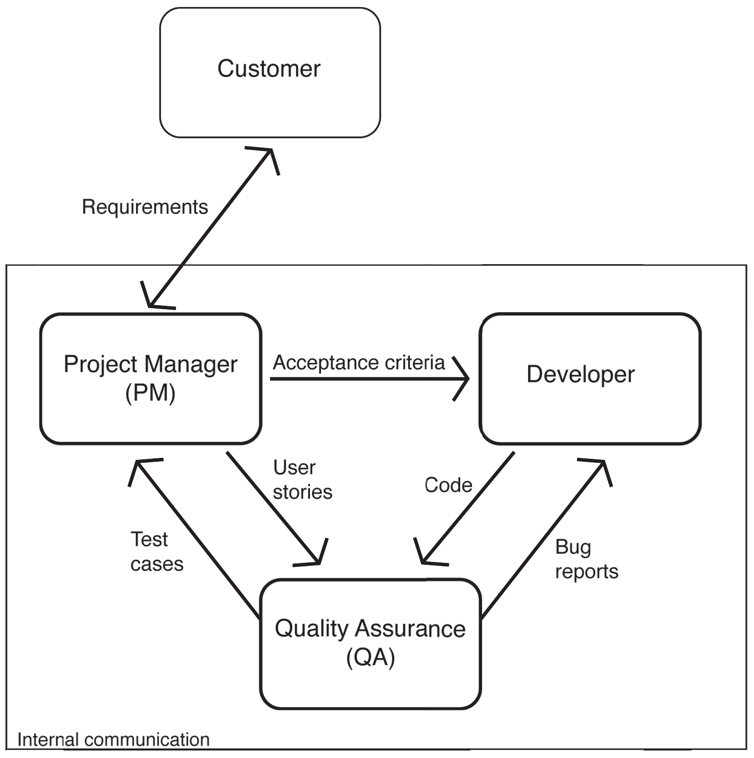

Figure 4.1: Figure of communication without Gherkin

Chapter 5

Discussion

In this chapter the results and the methods used during the study will be discussed. First a discussion of the results is presented, followed by a discussion of the methods used. The discussion concludes with some ethical and societal aspects of the study.

5.1 Artifacts, interviews and observations

The tickets produced in project B show one way of specifying bugs and tasks in a project. The description of these tickets could have been expressed in a Gherkin syntax even though the developer from project B was of the opinion that it was not suitable or even possible.

There might be cases where unclear specifications actually increase informal communication which increases trust among project members [6]. However it is hard to say since the informal communication is not stored in any sense and it is difficult to measure the amount of informal communication taking place in a project.

During the observations it became clear that the project teams are not quite as agile as first thought, the development team together with the PM was agile but when it came to QA and testing a waterfall model was applied, where QA was performed in the end of each sprint after development [1].

The different observations gave a picture of how planning and conduction of tests is performed in projects.

5.1.1 Gherkin as communication

When introducing Gherkin the communication process looks a bit different as can be seen in figure 5.1. The purpose is that all project member should contribute as well as understand a shared specification. The developer in project B was of the opinion that specifying the project in the beginning of project B with a Gherkin syntax would have been difficult or even impossible, since it was not decided what was actually going to be developed before development started. However if the descriptions written in task tickets in project B would have been expressed in a Gherkin syntax it would have provided a uniform format to all descriptions and some formats would have been broken down into more specific scenarios with a clear description of the state of the system and a testable outcome when an action occurs.

With a Gherkin syntax focus should be on the human readability of the desired functionalities [19]. Gherkin's most significant trait is its near natural language syntax, which describes the system from a user's perspective and allowing all roles in a project regardless of their technical background to understand and contribute to the Gherkin specification [13].

In figure 5.1 a hypothetical mean of communication with a Gherkin syntax is shown. All roles provide input to Gherkin and by collaboratively writing Gherkin scenarios different value can be gained for the different roles via Gherkin. In figure 5.1 the PM provides acceptance criteria to Gherkin, developers ensure that enough technical consideration is taken into account in writing the specification and QA make sure that the specification is specific enough for test cases. In the figure communication in terms of code is only for developers to care about. The green arrows in figure 5.1 is there to show what value could be provided directly to a customer with a Gherkin syntax, where they can provide changes in requirements when they can relate to the requirements via Gherkin scenarios. The connection via Gherkin between scenarios and test cases can provide the customer with a result of UAT in terms of scenarios being developed and tested in a format where details on how testing and development is abstracted away from the documentation.

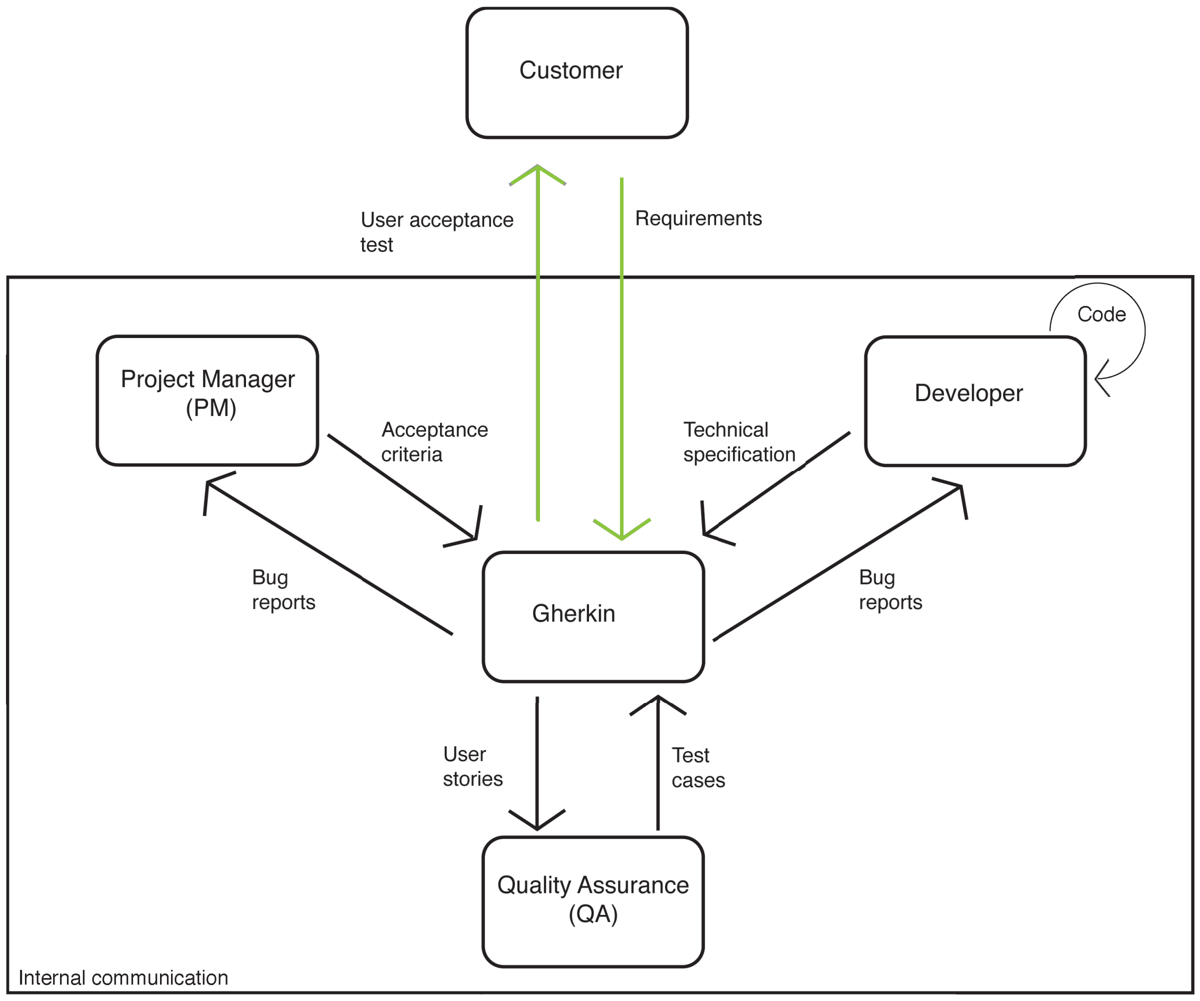

Figure 5.1: Figure of communication with Gherkin

The bug tickets created in project B that are compared in 4.4 are not so different from a Gherkin syntax. The difference is that they are produced after the bug is found and not prior to development. The bugs 1, 4 and 5 in table 4.3 would not have occurred at all if the descriptions of the corresponding functionality had been written in the format shown in table 4.4 before the development because in that case it would have been clear what the outcome should have been in ambiguous cases.

5.2 Business readable DSL

Applying Gherkin in software projects can provide a ubiquitous way of referring to the same piece of software for everyone participating in a software development project. By creating a document in the Gherkin syntax, a developer can refer to a specific scenario that is implemented and QA can test that specific scenario based on the Gherkin feature file. The PM in that project can then look at the test results and see exactly what scenarios and features have passed the tests, without having to know how they are tested but just see that the specific scenario or feature has been implemented and tested. This would provide an uniform way of communicating between the different roles in software development projects [13].

When creating the Gherkin feature file and implementing test cases based on that feature file, it became apparent that there is no guarantee that the specifications becomes more clear just because of the usage of Gherkin. To use a Gherkin feature file for development, testing and delivery a lot of time and work needs to be spent on creating and maintaining the feature files and it takes effort from the entire project team to contribute in creating and maintaining the features written in Gherkin [23].

The usage of Gherkin still needs guidelines and rules on how to be used. Just because the Gherkin syntax is used it does not result in specifications being unambiguous and clearly written. For a business readable DSL to be effective it is important that one dedicated person is responsible for keeping the Gherkin features consistent and avoiding duplication. How to use the Gherkin syntax also has to be clearly defined, so everyone in a project understands what the different notions actually can and should be used for, it is also important that there is room for applying the Gherkin and using it in the development process [23].

The usage of a DSL to establish that the right product is being developed is used to better connect the desired business value with what is being developed. The purpose of a DSL is also that it should be understandable and writeable by all roles participating in a software development project [17, 13].

5.3 Development methodology

By doing projects in a TDD or BDD manner a Gherkin feature file can be used as a common document to drive development, testing and assuring that the right product has been developed. Gherkin is a useful development methodology for its near natural language syntax. The near natural language syntax is easy to understand and write regardless of the technical knowledge of the user. Gherkin needs the help of TDD or BDD to reach its full potential. The specification of what to develop, test and deliver is so much more important from the beginning in BDD and TDD. With a Gherkin syntax user acceptance tests can be used as executable specifications rather than just test descriptions [13].

The Gherkin feature can be used as a means of communication by referring to Gherkin scenarios instead of chunks of code. The positive effects of Gherkin is depending more on how and by whom the feature files are written, and the most important factor to increase both communication and quality in agile software development projects is that all roles in a project are able to discuss the same functionality without misunderstandings and can agree on what is being developed, tested and delivered. That business people and developers must work together to create working software is stated as one of twelve principles in the agile manifesto [1, 17].

5.4 Method

When choosing task tickets and bug tickets it would have been good to find bug tickets that were clearly an outcome from a task ticket, so that a connection from the tickets in table 4.1 would have been easy to see in the tickets in table 4.3. However the task tickets found were not that precise and not all bugs were the outcome of task tickets. It can still be seen from some of the bugs in table 4.3 that they are the result of two roles interpreting the same functionality in different ways.

It is difficult to control an interview in a semi-structured manner, where there is a desire that the interviewee speaks freely on the subject while still keeping focus on the prepared questions and keeping the interviews at a reasonable length. Questions without specific order puts more effort into preparing questions that can be discussed in any order while still being precise enough to provide any relevant results and might provide different results depending on which order they are asked. This is also the purpose of semi-structured interviews where no questions are forgotten while any extra information from the interview subject is encouraged and provide a wider coverage than a fully structured interview.

The interviews was conducted based on how the interviewees perceive a usage of Gherkin and automated test cases written based on that feature file. It would have been interesting to ask an experienced test engineer questions about how they would have interpreted the Gherkin feature file created and see if different persons would have written the same test cases for the same Gherkin feature file, to determine whether the created Gherkin feature file was clear and unambiguous enough to decide what tests to implement.

Since Gherkin and CucumberJS test cases were implemented and then shown to the interviewees, they do not have any real experience from working with the tools. If the interview subjects would have been given tasks to experiment with the tools they might have answered differently in the interviews. Since the demonstrations were the first introduction the interviewees got to Gherkin and CucumberJS it was easy to capture their first impression of the tools.

5.4.1 Validity

During the interviews it is important that the interview questions are interpreted in the same way by the researcher and the interviewees. This was done by an introduction to the interviews. Each interview started with a demonstration of the implementation of Gherkin and CucumberJS to introduce the interviewees to the topic. The interviewees were also given an introduction of the aim of the study prior to the interviews to ensure that they understood what the purpose of the prepared questions were [22].

The collection of third degree data from one project provided data that are close to a real world context even if the researcher has little or no control of the collected data. The collected data was produced for different reasons than the research and the requirements of data validity and completeness is not necessary the same as for the research study [22].

Interviews, third degree data collection and observations were different methods used to collect data in the study. Data triangulation is achieved by collecting data from different data sources or collecting the same data at different occasions. The observations conducted were in two different projects at different times and different settings, providing a broader picture and higher validity in the data collected [22].

Reliability

Reliability of a study determines the likelihood of another researcher performing the same study would result in the same results. To achieve reliability the method used during the study is documented in a step by step manner. However the findings in this study are mainly based on an interpretation of the data made by the researcher. A researcher with a different background and different experience could have interpreted the same data in different ways [22].

5.5 The work in a wider context

During the interviews the interviewee was asked for their consent to record the interview on an audio file. All interviewees were informed how the audio file was going to be used and treated and when transcribed they were allowed to read through and confirm what was said during the interviews. The interviewees has been made anonymous in text, except for the director of QA at Accedo which is quite obvious who it is, he was informed of how he is mentioned in the text and was allowed to read through the report and give his consent.

In the code shown in appendix D values of some of the variables and url:s was chosen to be changed into variable names instead of writing the real value used. This decisions was made since those values do not affect the interpretation of the test cases and the point of showing the test cases in the appendix was to show how the Gherkin feature file could be interpreted into test cases [22].

Chapter 6

Conclusion

In this chapter the conclusions from the results and discussions of the study are presented.

6.1 Conclusion

By writing specifications with the Gherkin syntax, task descriptions can be made more clear in terms of separating tasks with scenarios and provide a uniform way of specifying tasks, because in every task description the state of the system is clearly defined, together with an action that is to be performed to achieve a testable outcome. The usage of Gherkin forces every task description to clearly define the pre-condition with the keyword ”given”, the action to be performed with the keyword ”when” and a testable outcome with the keyword ”then”. Giving a clear overview of what state the functionality is desired, the functionality and what the functionality achieves in the format of a testable outcome. However Gherkin does not necessarily provide extra information in all cases but enforce descriptions of tasks to be written in the same way, thus interpretation of the descriptions is more clear to all roles in a development project.

In terms of bug reports Gherkin can provide a short phrase in plain English on what scenario is failing and that scenario is directly linked to an action to be performed thus giving the developer enough information to fix the bug while a less technical person can still understand in what piece of functionality the bug appears.

One important factor in agile development projects is a shared understanding by all participants in the project of what is being developed, tested and delivered. It is also important that all roles in a project understand each others work to the extent that they can work together in specifying the project and Gherkin is one way of creating a shared specification in a language that is domain specific yet precise enough to describe the desired features to be implemented. Especially can Gherkin be used as a tool to introduce a more collaborative mindset where all roles in projects contribute early in the project process, thus avoiding a vicious circle of developers not knowing what QA is testing and QA not knowing what is being developed.

Features written in a DSL such as Gherkin do not necessarily mean that there are more information communicated through documents, however it will provide a uniform format of describing specifications and abstract away the implementation details in descriptions. This gives the internal communication a uniform structure that improves the existing communication. One of the most important factors for a usage of a DSL to be effective is the collaborative working with the specifications. The technical knowledge from a developer is necessary yet not enough to write clear unambiguous specifications in a DSL.

Bibliography

[1] Kent Beck et al. Manifesto for agile software development. http://agilemanifesto.org/. Accessed: 2015-10-26.

[2] Robert Chatley, John Ayres, and Tom White. Lift: Driving development using a business-readable DSL for web testing. In *Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on*, pages 460–468. IEEE, 2010.

[3] Mikko Korkala, Pekka Abrahamsson, and Pekka Kyllönen. A case study on the impact of customer communication on defects in agile software development. In *Agile Conference, 2006*, pages 11–pp. IEEE, 2006.

[4] S Bhalerao and M Ingle. Analyzing the modes of communication in agile practices. In *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, volume 3, pages 391–395. IEEE, 2010.

[5] Grigori Melnik and Frank Maurer. Direct verbal communication as a catalyst of agile knowledge sharing. In *Agile Development Conference, 2004*, pages 21–31. IEEE, 2004.

[6] Minna Pikkarainen, Jukka Haikara, Outi Salo, Pekka Abrahamsson, and Jari Still. The impact of agile practices on communication in software development. *Empirical Software Engineering*, 13(3):303–337, 2008.

[7] Christoph Johann Stettina and Werner Heijstek. Necessary and neglected?: an empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM international conference on Design of communication*, pages 159–166. ACM, 2011.

[8] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36, 2000.

[9] Martin Fowler. Business readable DSL. http://martinfowler.com/bliki/BusinessReadableDSL.html, December 2008. Accessed: 2015-10-16.

[10] Susan Hammond and David Umphress. Test driven development: the state of the practice. In *Proceedings of the 50th Annual Southeast Regional Conference*, pages 158–163. ACM, 2012.

[11] Carlos Solís and Xiaofeng Wang. A study of the characteristics of behaviour driven development. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 383–387. IEEE, 2011.

[12] Dan North. Behaviour driven development. http://behaviourdriven.org. Accessed: 2015-10-05.

[13] Marc Hesenius, Tobias Griebe, and Volker Gruhn. Towards a behavior-oriented specification and testing language for multimodal applications. In *Proceedings of the 2014 ACM SIGCHI symposium on Engineering interactive computing systems*, pages 117–122. ACM, 2014.

[14] Mathias Landhäußer and Adrian Genaid. Connecting user stories and code for test development. In *Proceedings of the Third International Workshop on Recommendation Systems for Software Engineering*, pages 33–37. IEEE Press, 2012.

[15] Mazedur Rahman and Jerry Gao. A reusable automated acceptance testing architecture for microservices in behavior-driven development.

In *Service-Oriented System Engineering (SOSE), 2015 IEEE Symposium on*, pages 321–325. IEEE, 2015.

[16] Nitin Bharti. Ten principles for agile testers. https://dzone.com/articles/agile-testing-principles, July 2010. Accessed: 2016-01-11.

[17] Tariq M King, Gabriel Nunez, Dionny Santiago, Adam Cando, and Cody Mack. Legend: an agile DSL toolset for web acceptance testing. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 409–412. ACM, 2014.

[18] Gherkin. https://github.com/cucumber/cucumber/wiki/Gherkin, February 2015. Accessed: 2015-10-16.

[19] Matt Wynne and Aslak Hellesoy. *Cucumber Book*. Pragmatic Bookshelf, 2012.

[20] Reference. https://cucumber.io/docs/reference. Accessed: 2015-10-19.

[21] Omar Gonzalez. Cucumber and js: Getting started with cucumber.js. http://transitioning.to/2012/01/cucumber-and-js-getting-started-with-cucumber-js/. Accessed: 2015-10-30.

[22] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[23] Mark Micallef and Christian Colombo. Lessons learnt from using DSLs for automated software testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*, pages 1–6. IEEE, 2015.

Appendix A

Interview developer project B

Below are the questions that were prepared prior to the interview with the developer in project B.

- Is there a need for communication via a DSL?

  Do you think Gherkin would be useful for that?

- Who do you think is responsible for the code quality?

- What kind of tests do you as a developer perform?

- What role do you think QA should have in a project?

  How did you perceive QAs role in project B?

- What decided what was to be developed in project B?

  Would a formulation in the Gherkin syntax have made it more clear?

- What are your anticipations regarding code quality, on PM? on QA?

Appendix B

Interview director of QA

Below are the questions that were prepared prior to the interview with the director of QA.

- Do you think Gherkin can be used to reduce ambiguities in requirements at an earlier stage of projects?
- What value do you think Gherkin can bring to projects?
- What do you think are the biggest challenges with applying Gherkin to projects?
- What are the main differences with Gherkin compared to how you plan and conduct tests today?
- What is most important to consider when defining scenarios and test cases?
- Who do you consider is responsible for code quality in projects?

Appendix C

Gherkin feature file

Below is the Gherkin feature file showing what was implemented during the study.

```
@Feature1
Feature: A request to AppGrid to fetch themes from AppGrid to Client
As an AppGrid API Client
In order to fetch playlists from themes
I want to retrieve themes in the correct format

@Scen1
        Scenario: Establish connection and receive region code
                Given A connection to AppGrid for the app is
                  established
                When A request is sent to get region data
                Then A Region code in 24 characters should be
                  returned
@Scen2
        Scenario: Request and receive entries in correct format
                Given A region code is fetched
                When A request to get
                the entries for that region code is sent
                Then Each entry should contain
                themes, regionCode, regionName, id and typeID
@Scen3
        Scenario: Receive theme ID
                Given Entries for a region code is fetched
                When All themes from AppGrid is available
                Then Each theme ID should contain 24 characters
```

Appendix D

Test cases

The test cases implemented in javascript, which was executed by CucumberJS.

```
var XMLHttpRequest = require('xmlhttprequest').XMLHttpRequest;
var assert = require('assert');

module.exports = function() {

var mySessionKey;
var myRegion = null;
var myEntries = null;
var noRegions;
var themesArr = [];
var xhttp = new XMLHttpRequest;

//key, uuid and url_n have other values in real code.
key = "Access-key"; //Key is a key for accessing content
uuid = "unique-user-id"; //uuid is an identifier for a specific device
url_n = "url"; //The specific url to send request to, n = 1,2,3

//@Scenario1
this.Given(/^A connection to AppGrid for the app is established$/,
 function (callback) {
  // Write code here that turns the phrase above into concrete actions
  xhttp.open("GET", url_1, false);
  xhttp.setRequestHeader("X-Application-Key", key);
  xhttp.setRequestHeader("X-User-ID", uuid);
  xhttp.send();
  if(xhttp.status == 200) {
  mySessionKey = JSON.parse(xhttp.responseText).sessionKey;
  };
  callback();
});

this.When(/^A request is sent to get region data$/, function (callback) {
  xhttp.open("GET", url_2, false);
  xhttp.setRequestHeader("X-Session", mySessionKey);
  xhttp.send();
  if(xhttp.status == 200) {
  myRegion = JSON.parse(xhttp.responseText).region;
  }

  callback();
});

this.Then(/^A Region code in (\d+) characters should be returned$/,
        function (arg1, callback) {
  // Write code here that turns the phrase above into concrete actions
  assert.equal(myRegion.length, arg1,
   "Region code should be " + arg1 + " characters");
  callback();
});

//@Scenario2
this.Given(/^A region code is fetched$/, function (callback) {
  // Write code here that turns the phrase above into concrete actions
  assert.notEqual(myRegion, null, "No regionCode is fetched");
  callback();
});

this.When(/^A request to get the entries for that region code is sent$/,
        function (callback) {
  // Write code here that turns the phrase above into concrete actions
 var url = url3 + myRegion +"&preview=true";
  xhttp.open("GET", url_3, false);
  xhttp.send();
     if(xhttp.status == 200) {
  myEntries = JSON.parse(xhttp.responseText); //response JSON parsed
  noRegions = myEntries.pagination.total;
  }
  callback();
});

this.Then(/^Each entry should contain themes, regionCode, regionName,
 id and typeID$/,
        function (callback) {
  // Write code here that turns the phrase above into concrete actions
  var i;
  for(i = 0; i < noRegions; i++){
    assert.notEqual(myEntries.entries[i].themes, null, "Themes is null");
    assert.notEqual(myEntries.entries[i].regionCode, null,
    "regionCode is null");
    assert.notEqual(myEntries.entries[i].regionName, null,
    "regionName is null");
    assert.notEqual(myEntries.entries[i]._meta.id, null, "id is null");
    assert.notEqual(myEntries.entries[i]._meta.typeId, null,
     "typeId is null");
  }
  callback();
});

//@Scenario3
this.Given(/^Entries for a region code is fetched$/, function (callback) {
  // Write code here that turns the phrase above into concrete actions
  assert.notEqual(myEntries, null, "No Entries is fetched");
  callback();
});

this.When(/^All themes from AppGrid is available/, function (callback) {
  // Write code here that turns the phrase above into concrete actions
  var i;
  var myEntryRegions;
  var themesList;
  var url;

  for(i = 0; i < noRegions; i++) {
    myEntryRegions = myEntries.entries[i];
    themesList = myEntryRegions.themes;
     themesArr = themesArr.concat(themesList);
  }
callback();
```

```
});

this.Then(/^Each theme ID should contain (\d+) characters$/,
        function (arg1, callback) {
  // Write code here that turns the phrase above into concrete actions
  var i;
  for(i = 0; i < themesArr.length; i++) {
  assert.equal(themesArr[i].length, arg1, "Theme Id:" + themesArr[i] +
  " is not " + arg1 +" characters");
}
  callback();
});

}
```