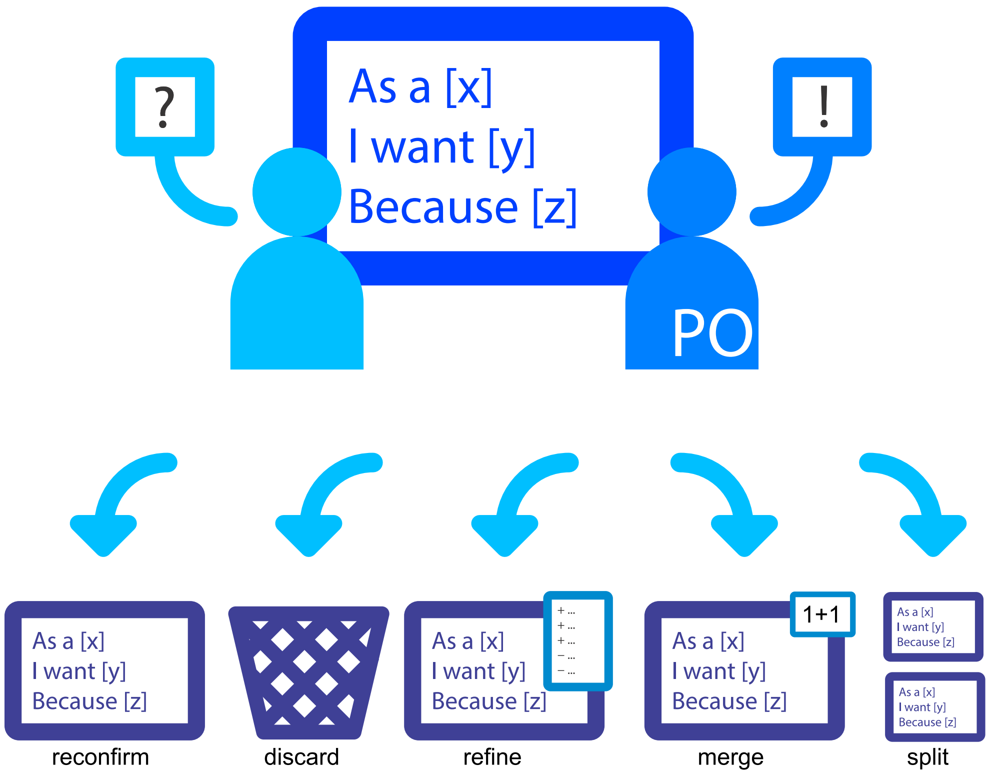
Transparency in actions on requirements

Distributed Agile Requirements Engineering



Xander Verheij

Transparency in actions on requirements

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Xander Verheij
born in Rotterdam, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Transparency in actions on requirements

Author:	Xander Verheij
Student id:	1217739
Email:	`xander@xrv.nl`

Abstract

The combination between agile software engineering and distributed engineering is gaining a growing interest. Combining these however creates an interesting paradox. Where agile clearly states that documentation is not the most important thing, from the field of globally distributed engineering a higher focus on documentation is observed. In this thesis the field of requirements engineering is also taken into the mix. Combining these three a user-story model is defined to take advantage of the, on first sight, downside of working distributed. The fact that face-to-face communication is not possible means that all the communication has to be done using technological support and thus that this communication can be saved. This premise gives an interesting opportunity to keep track of the actions that are taken on user-stories as a result of a conversation. By saving the conversation and coupling the action to parts of that conversation.

Dedicated to my father
Gert Verheij
1955 - 2009

Preface

First of all I would like to dedicate this thesis to the one person I wish could read it, but unfortunately is not able to do so. My father, who died suddenly in December 2009, is the person who learned me to program while I was still a little boy and thus the reason I chose this field of study. Dad, thanks!

My graduation did not went as smooth as can be hoped for, first a lot of effort has been put in a internship overseas which was unfortunately cancelled the last minute. After this a four month internship a little bit close to home. But this also went wrong due to some legal issues. However, no matter the difficulties Prof. Dr. ir. D.M. van Solingen (Rini) has always made sure that my graduation was going to happen, a little later then originally planned. Rini, thanks!

After the initial start-up problems I started internally at the TU Delft under two PhD-students Ben and Kevin, they have helped throughout my graduation by finding the right scope and subject and giving a nudge in the right direction when needed. Next to that they also introduced me to IHomer, the company which has been the scope of my research and where I have been working almost every Tuesday with a lot of enthusiasm. Ben and Kevin, thanks!

Next to giving a pleasant working experience every Tuesday, IHomer also provided the context for conducting my studies and even more important the people that where willing to participate in these studies. All the people at IHomer, thanks!

I also can not forget my mother and sister, not being able to show my dad that I was finally going to graduate has been hard at times, but thanks to them I have always been able to push through.

Not only my mother and sister helped me finish my graduation but my girlfriend Maaike has also been there for me when I needed it the most. Next to that she is also responsible for a drastic reduction of spellings mistakes and the visual aids used in this thesis.

Xander Verheij
Delft, the Netherlands
November 11, 2012

Contents

List of Figures

Chapter 1

Introduction

This chapter will give an introduction considering this Master thesis. In the following sections subsequently some insights will be given about the research context, problem definition, goal and set-up of this thesis.

1.1 Research Context

The research conducted for this thesis is based around a number of fields which will be introduced shortly here and explained in more detail in the chapters to come.

1.1.1 Agile Software Engineering

Agile software engineering is an iterative and evolutionary approach to the engineering of software which relies on frequent feedback and close collaboration with the customer.

Agile methodologies find there basis in the principles from the agile manifesto:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

More about agile and some agile methodologies can be found in chapter 3.

1.1.2 Requirements Engineering

Requirements Engineering (RE) is the identifying, modelling, communicating and documenting of the requirements for a system [83]. Traditional Requirements Engineering states that the requirements need to be gathered and extensively documented before the beginning of the actual implementation of the software. This changes when incorporating agile methodologies, more about requirements engineering in an agile setting can be found in chapter 4.

1.1.3 Global Software Engineering

Global Software Engineering (GSE) is a term used for engineering of software, distributed among different developers that are distributed by different distances. These distances are defined by Carmel [22] as can be seen in figure 1.1. According to Carmel and Agarwal

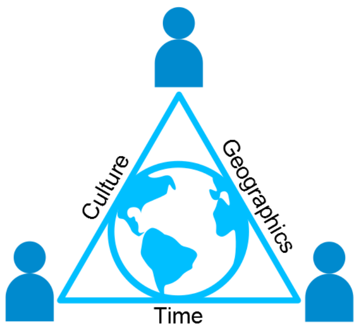


Figure 1.1: A visual representation of the three dimensions of distance.

[23] these distances have an effect on communication, coordination and control. Where communication also has an effect on coordination and control. Next to these challenges concerning distributed development there are also some claimed advantages like reduced development costs, access to large skilled labor pool and closer proximity to market and customer.

1.2 Problem definition

The problem on which this thesis will focus is based upon the three pieces of the context defined above and the ṕaradox t́hat is created when combining the three in the same environment.

- Agile software engineering is based on face-to-face communication where the documentation is more used as a promise for a conversation [70].
- The aim of traditional requirements engineering is to know what to build and document these requirements in large detail before the implementation starts. [83]
- With distributed development while trying to be agile, there is an urge to create more explicit requirements. [49, 30]

Putting these three statements together we can deduct that it seems that distributed development comes in the way of handling your requirements in an agile way, among others because of the lack of face-to-face communication. So the problem can be summarized to:

The reliance on face-to-face communication and documenting as few as possible of agile, goes in against the lack of face-to-face communication and higher documentation dependency of distributed development.

1.3 Goal

The goal of the research conducted in these thesis, comes from the further research into the three different aspects that make up the paradox that has been defined in the problem definition. From the research it became apparent that the most often used method for requirements in agile software engineering is the user-story. This will be confirmed in chapter 4 and further substantiated in section 5.5 These user-stories have a strong basis in the fact that you are supposed to talk about them, this communication is at the center of the paradox created by combining Agile with GSE and RE. Using the research conducted a model identifying the possible actions that can be taken on a requirement following such a conversation has been developed, and because of the distributed aspect of this research these actions can be administrated. The goal of the research conducted in this thesis is thus:

Determine to which extent transparency concerning actions taken on requirements enhances the distributed development process.

1.4 Thesis set-up

To determine how to make the actions taken on requirements transparent some background information is needed. This background information will be given in the first 'theoretical research' part. Here agile, agile requirements engineering and finally distributed agile requirements engineering will be explained.

The second part is about the practical part of this thesis, here the knowledge gained from the theoretical research will be used to actually develop the support for making the actions taken on requirements transparent and validate the created solution.

The third and final part will give the conclusions that can be drawn from the results of the validation done on the implementation created.

Part I

Theoretical Research

Chapter 2

Introduction

The theoretical part of this thesis is set-up in a number of different sections. First the groundwork will be given by explaining the basis of agile development, in this chapter the two most commonly used agile methodologies will be introduced. The next chapter will discuss agile requirements engineering, in this section a structured literature survey is used to find the best way to represent agile requirements according to the literature. The third chapter gives an introduction into global software development and the benefits and challenges that come with it. The fourth and final chapter of this part will go into distributed agile requirement engineering and will conclude with the theoretical model that is used as a basis for the practical research which will be covered in the next part of this thesis.

Chapter 3

What is Agile Software Engineering?

Before agile requirements engineering and distributed agile requirements engineering will be explained first some background knowledge about agile it self is needed. This background information will be presented in this chapter.

The term agile software development was first used in 2001. Most of the agile methods however were already developed during the 1990s. The people who developed these methods came together and during this meeting discovered a common ground, now better known as the Agile Manifesto. This chapter will first give the 'Manifesto for Agile Software Development' and the principles that come with it, after that two of the most common agile methods , Scrum and eXtreme Programming will be discussed in this chapter. The choice for Scrum and XP has been made based on the agile survey from version 1. This survey says that the most used agile methods are Scrum and a Scrum and eXtreme Programming (XP) hybrid.

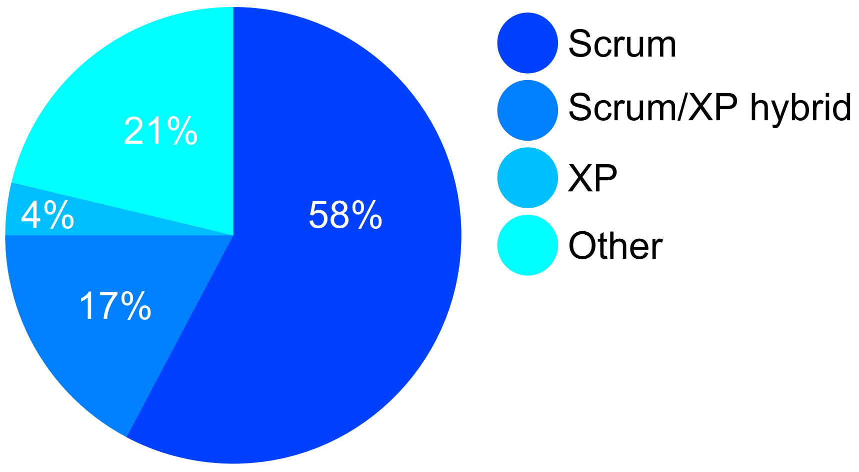


Figure 3.1: Adapted from [116]

3.1 Manifesto for Agile Software Development

The manifesto for agile software development is the following.

> “ We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - Customer collaboration over contract negotiation
> - Responding to change over following a plan
>
> That is, while there is value in the items on
> the right, we value the items on the left more. ”

Next to the manifesto 12 principles were also developed:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time-scale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity –the art of maximizing the amount of work not done– is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviour accordingly.

All of the lightweight methods developed in the mid-1990s are now more commonly knows as agile methods.

3.2 Agile methodologies

3.2.1 Scrum

Scrum is a software development approach that has been developed in the early 1990s by Jeff Sutherland and Ken Schwaber. Its roots lie with empirical process control and complex adaptive systems theory and has been inspired by a Harvard Business Review article (New new product development game [108]).

Scrum is build up out of a number of 3 artefacts, 3 meetings and 3 roles which will be explained later in this section, first an overview of the scrum process is given.

The process

Scrum is an iterative approach, which means that the work is done in small iterations which deliver a working increment of software at the end. An overview of the scrum process can be seen in figure 3.2

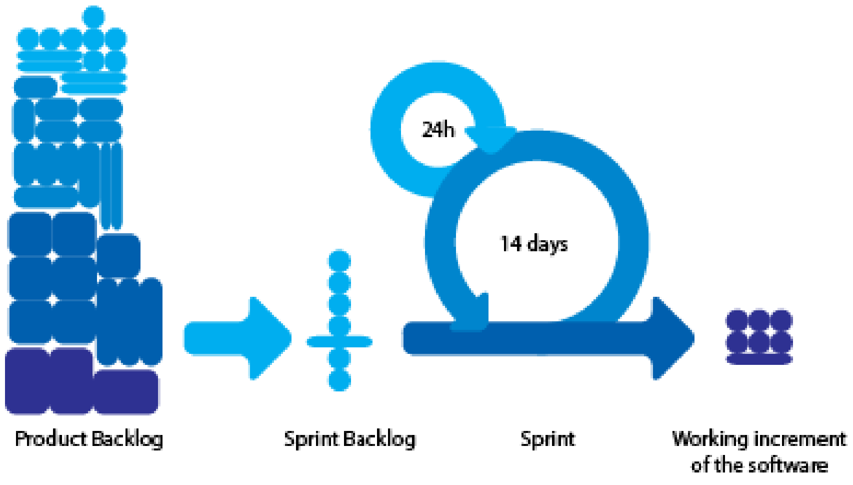


Figure 3.2: Overview of the Scrum process

This figure shows one iteration or in Scrum terms sprint. From taking items from the product backlog to delivering working software. When repeating this process over and over again the software will become a complete product as depitect in figure 3.3.

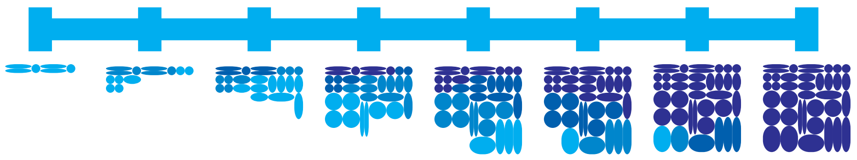


Figure 3.3: A visual representation of Sprints resulting in a complete product

Artefacts

Scrum is made up out of three artefacts which will be explained in the following three paragraphs.

Product Backlog The product backlog is an ordered list of small deliverables. These deliverables can be anything that the business wants but is not yet in the product. The order in which these backlog items appear on the backlog are based upon the value that a completed backlog item would bring to the business. Figure 3.4 represents how the idea for a product is split-up and ordered into backlog-items. How these backlog items are represented will be discussed in detail in the next chapter.

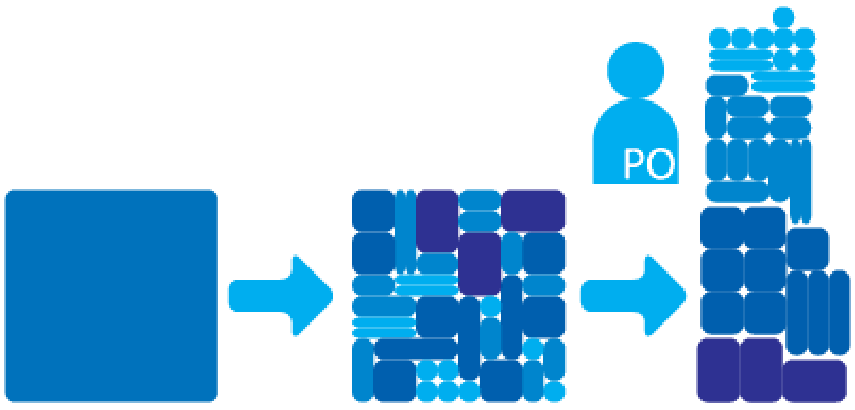


Figure 3.4: A visual representation of a Product Backlog

Sprint Backlog The Sprint backlog consists of the product backlog items selected to be completed in the current time-boxed iteration. Based upon the priorities of the Product Owner and what the team thinks it can accomplish in the coming Sprint.

Burn-down Chart The third and final artefact that is described when using Scrum is the Burn-down chart. This chart is used to keep track of the velocity during a Sprint. In other words, how much work has been done and how much work is still to be done. By estimating the time needed to complete backlog items before hand this chart can be used to see if a team is on schedule on finishing all the items on the sprint backlog before the end of the sprint. Thus the term burn-down comes from the fact that you are burning done the items to be done.

Roles

Scrum describes three different roles which all come together in a so called Scrum-team. A Scrum team is made up of generally 7+-2 people which are cross-functional. Every expertise necessary for finishing the software should be present in the team. Within these teams three roles are present. These roles are:

Product Owner The product owner is responsible for the prioritization of items on the backlog. He or she preferably is someone from the business itself so that the backlog items are ordered by the value they represent to the business.

Scrum Master A Scrum master is responsible for taking away any impediments that the team might encounter during development, and thus to aid the team to reach their maximum velocity and add value to the business as fast as possible. Contrary to what the title suggests the scrum master is not the boss of the team. He is responsible for keeping the process going smoothly and keeping other people out of the way of the team. Not to enforce rules upon the team.

Team member The rest of a Scrum team consists of team members that are cross-functional, which means they can be developers, testers, user interface designers or technical writers. However, that a tester has a speciality in testing does not mean that this should be his only activity. This is why there is no distinction between any other roles. The team is responsible for delivering complete and working software. Thus the software must be designed, developed, tested, etc. all these steps are the responsibility of everybody in the team.

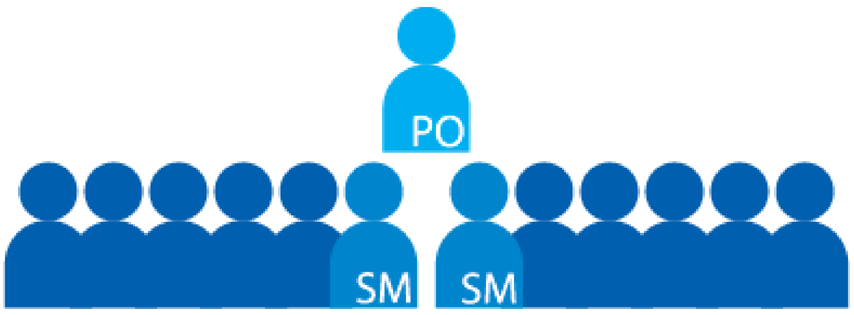


Figure 3.5: A visual representation of the Cross Functional Teams

Events

There are four main events when using Scrum one, daily and two that take place once every sprint. These meetings are the following:

Daily Scrum (Stand-up) Every day there is a 15 minute time-boxed meeting in which the current state of the development is discussed. The name comes from the fact that all the team-members are supposed to be standing. Because of this people remain more active then when sitting and thus helps to keep the meeting short. To streamline the meeting everybody is asked the following three questions:

- What did you do yesterday?
- What will you do today?
- What is in your way (impediments)?

This keeps other team-members the opportunity to help where necessary and gives the Scrum master an overview of the impediments that are in the way of a higher velocity.

Sprint planning meeting At the beginning of a Sprint there is a usually time-boxed meeting in which items are pulled of the product backlog. The amount of items that is taken of the backlog depends on the velocity that the team is able to reach. The items that are taken of the product backlog are put on the Sprint backlog.

Sprint review During the meeting at the end of a sprint there are a couple of things that are done. First a working increment of software is presented to the product owner and possible other stakeholders.

Sprint retrospective The sprint retrospective is used to determine how well the Sprint went. What went good, what could go better and off course what should be done to make sure the next Sprint is going even better.

3.2.2 eXtreme Programming

Extreme Programming (XP) was created by Kent Beck [13] and it has earned its name by taking “best practices” to extreme levels. XP has evolved in parallel with Scrum and shares some common ground. Scrum can be seen as a wrapper around XP. Where Scrum focuses on structural issues and external communication, XP focuses more on internal team engineering practices figure 3.6 makes this more clear.

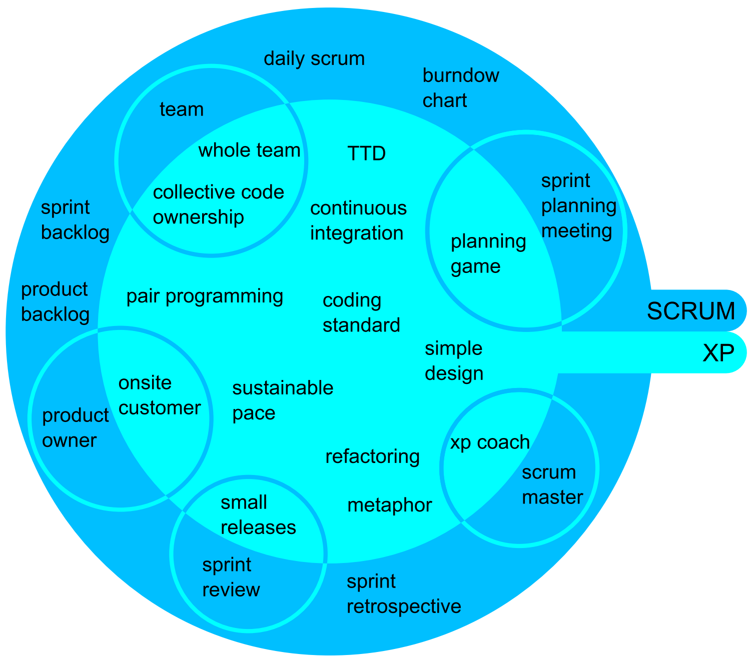


Figure 3.6: The relations between XP and Scrum practices adapted from [66]

Extreme programming is based upon 5 core principles and a number of practices. [15]

Principles

The following five principles have been taken from the website about XP [1].

Simplicity: We will do what is needed and asked for, but no more. This will maximize the value created for the investment made to date. We will take small simple steps to our goal and mitigate failures as they happen. We will create something we are proud of and maintain it long term for reasonable costs.

Communication: Everyone is part of the team and we communicate face to face daily. We will work together on everything from requirements to code. We will create the best solution to our problem that we can together.

[1]http://extremeprogramming.org

Feedback: We will take every iteration commitment seriously by delivering working software. We demonstrate our software early and often then listen carefully and make any changes needed. We will talk about the project and adapt our process to it, not the other way around.

Respect: Everyone gives and feels the respect they deserve as a valued team member. Everyone contributes value even if it's simply enthusiasm. Developers respect the expertise of the customers and vice versa. Management respects our right to accept responsibility and receive authority over our own work.

Courage: We will tell the truth about progress and estimates. We don't document excuses for failure because we plan to succeed. We don't fear anything because no one ever works alone. We will adapt to changes when ever they happen.

Practices

The 13 primary technical practices which encompass eXtreme Programming are the following (as described in [15])

Sit together the whole team develops in one open space.

Whole team utilize a cross-functional team of all those necessary for the product to succeed.

Informative workspace place visible wall graphs around the workspace so that team members (or other interested observers) can get a general idea of how the project is going.

Energized work XP teams do not work excessive overtime for long periods of time. The motivation behind this practice is to keep the code of high quality (tired programmers inject more defects) and the programmers happy (to reduce employee turnover). Tom DeMarco contends that, Extended overtime is a productivity- reducing technique. [31]

Pair programming [115], refers to the practice whereby two programmers work together at one computer, collaborating on the same design, algorithm, code, or test.

Stories the team write short statements of customer-visible functionality desired in the product. The developers estimate the story; the customer prioritizes the story.

Weekly cycle at the beginning of each week a meeting is held to review progress to date, have the customer pick a weeks worth of stories to implement that week (based upon developer estimates and their own priority), and to break the stories into tasks to be completed that week. By the end of the week, acceptance test cases for the chosen stories should be running for demonstration to the customer to drive the next weekly cycle.

Quarterly cycle the whole team should pick a theme or themes of stories for a quarters worth of stories. Themes help the team reflect on the bigger picture. At the end of the quarter, deliver this business value.

Slack in every iteration, plan some lower-priority tasks that can be dropped if the team gets behind such that the customer will still be delivered their most important functionality.

Ten-minute build structure the project and its associated tests such that the whole system can be built and all the tests can be run in ten minutes so that the system will be built and the tests will be run often.

Test-first programming , all stories have at least one acceptance test, preferably automated. When the acceptance test(s) for a user story all pass, the story is considered to be fulfilled. Additionally, automated unit tests are incrementally written using the test-driven development (TDD) [12] practice in which code and automated unit tests are alternately and incrementally written on a minute-by-minute basis.

Continuous integration programmers check in to the code base completed code and its associated tests several times a day. Code may only be checked in if all its associated unit tests and all of unit tests of the entire code base pass.

Incremental design , rather than develop an anticipatory detailed design prior to implementation, invest in the design of the system every day in light of the experience of the past. The viability and prudence of anticipatory design has changed dramatically in our volatile business environment [56]. Refactoring [45] to improve the design of previously-written code is essential. Teams with robust unit tests can safely experiment with refactorings because a safety net is in place.

Next to these 13 primary practices there are also the following 11 corollary technical practices (also taken from [15])

Real customer involvement the customer is available to clarify requirements questions, is a subject matter expert, and is empowered to make decisions about the requirements and their priority. Additionally, the customer writes the acceptance tests.

Incremental deployment gradually deploy functionality in a live environment to reduce the risk of a big deployment.

Team continuity keep effective teams together.

Shrinking team as a team grows in capacity (due to experience), keep their workload constant but gradually reduce the size of the team.

Root cause analysis examine the cause of a discovered defect by writing acceptance test(s) and unit test(s) to reveal the defect. Subsequently, examine why the defects was created but not caught in the development process.

Shared code once code and its associated tests are checked into the code base, the code can be altered by any team member. This collective code ownership provides each team member with the feeling of owning the whole code base and prevents bottlenecks that might have been caused if the owner of a component was not available to make a necessary change.

Code and tests maintain only the code and tests as permanent artifacts. Rely on social mechanisms to keep alive the important history of the project.

Daily deployment put new code into production every night.

Negotiated scope contract fix the time, cost, and required quality of a project but call for an on-going negotiation of the scope of the project.

Pay-per-use charge the user every time the system is used to obtain their feedback by their usage patterns.

A lot of these practices build upon each other. Continuous integration is not possible without test-driven development. And good test-drive development is best achieves by using pair programming.

Artefacts

In contrast to Scrum, eXtreme Programming does not have any artefacts defined. This because XP relies on information sharing using oral communication, the actual code and tacit knowledge transfer. But while oral communication might work when working in small groups, when working in larger groups or with high-risk systems these might not be enough. In these cases there are a number of ”tools” that might be used:

- User story cards;
- Task list;
- CRC cards [114].

[15] This representation of requirements will be discussed in more detail in the next chapter.

Roles

The roles defined in XP will be explained in short: [13] [15]

Manager owns the team and its problems. He or she forms the team, obtain resources, manage people and problems, and interfaces with external groups.

Coach teaches team members about the XP process as necessary, intervenes in case of issues; monitors whether the XP process is being followed. The coach is typically a programmer and not a manager.

Tracker regularly collects user story and acceptance test case progress from the developers to create the visible wall graphs. The tracker is a programmer, not a manager or customer.

Programmer writes tests, design, and code; refactors; identifies and estimates tasks and stories (this person may also be a tester)

Tester helps customers write and develop tests (this person may also be a programmer)

Customer writes stories and acceptance tests; picks stories for a release and for an iteration. A common misconception is that the role of the customer must be played by one individual from the customer organization. Conversely, a group of customers can be involved or a customer representative can be chosen from within the development organization (but external to the development team).

Process

The XP process is made up out of six different phases these phases are the following:

Exploration During the exploration phase the customer specifies what they want to be done in the first iteration. At the same time the project team familiarizes themselves with the tools, technology and practices they will be needing during the project.

Planning The planning phase sets the priority for the requirements is determined and a set of requirements to be completed after the first iteration is agreed upon.

Iterations to release During this phase multiple iteration will take place to get to a first production ready release.

Productionizing The final stage before declaring the product finished; in this phase it is checked if the software is ready to go in production and eventual issues are dealt with or postponed to later releases.

Maintenance When the first release is taken into production the maintenance phases commences. During this phase issues are likely to arise from the users that use the software, and these issues need to be dealt with.

Death When the software is complete and the customer does not want any changes any more. The process is completed. However in most cases this phase is reached because of discontinued use of the software and not because the software is finished.

3.3 Why is agile explained?

Before agile requirements engineering and distributed agile requirements engineering can be explained first some background knowledge about agile is needed. The basis of agile is the frequent and short feedback loops and more communication instead of comprehensive documentation.

Chapter 4

What is Agile Requirements Engineering?

Before distributed agile requirements engineering can be explained some background information is required about agile requirements engineering.

This chapter will go deeper into requirements in an agile development environment. The most used way to represent requirements in agile development can be considered user-stories. User-stories are used in the explanation of eXtreme Programming and also for example Mike Cohn sees them as the de facto standard for agile requirements engineering [27]. Chapter 5 will also use a structured literature survey to further substantiate the choice voor user-stories.

4.1 User-stories

This section will go deeper into what a user-story actually is and of what elements it consists.

The most well known and visible part of a user-story is the card. These cards are frequently put-up onto whiteboard for everybody to see and have a distinct form in which the user-story is described. However this is only the first of three elements of a user-story which all will be described in the following sections.

4.1.1 The three C's

Card

The visible part of the user-story can best be explained by means of the most common layout of these user-story-cards:
This layout for a user-story-card emphasizes the user centric thinking that is common in agile development methods and forces developers to think from a user perspective.

For example the following user-story:

As a user *I want* a MySQL back-end *So that* I can retrieve my Notes.

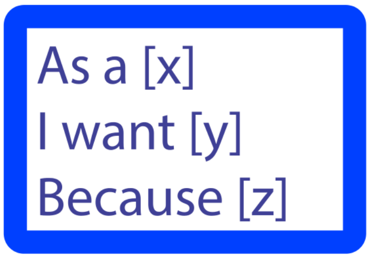


Figure 4.1: A visual representation of a user-story

Is not a good user-story because a user is not interested in what kind of technology is used. He or she simple wants to retrieves the notes, thus a better user-story would be the following.

> *As a* user *I want* to be able to retrieve my notes *So that* I know what to do next.

But whatever is scribbled or typed down on the card part of a user-story this is never the entire story. It is only a promise, a promise for a conversation.

Conversation

The conversation promised by the card is where the actual details become clear. In conversations between the developers and the customer/users the details of what should be done become more clear.

Confirmation

When is the user-story finished? This is determined in the confirmation. Often this is in the form of acceptance tests written by the customer. By giving a scenario that must pass or give a certain output it can be checked if the user-story is completely implemented.

4.1.2 Invest in good user-stories

A common used acronym for user-stories (among others by Mike Cohn [27]) INVEST. This acronym stand for a number of items for which it is beneficial if a user-story conforms to them. The items that INVEST stands for, are the following:

- Independent:
  To aid in planning, prioritization and estimation user stories should be as independent as possible. Interdependencies between stories could result in a story X with a lower

priority than an other story Y to be taken on first. Because story Y depends on X. Which in turn makes the prioritization not purely on business value. More or less the same goes for estimation the estimate for story Y is dependent on story X and thus harder to do.

- Negotiable:
  Stories should not be a fixed contract about how the functionality has to be implemented. Instead the story should only contain the essence of what the software has to do. If all the details are already in the story it seems that there is no more need to be talking about the story to figure out what the exact idea is. Also too many details can give a false idea of precision before hand.

- Valuable:
  Every story should contain value for the customer, this can either be the user of the software developed or the purchaser of the software. So also developer concerns should if possible be expressed in a manner that the value to the customer is described.

- Estimable:
  It should be possible to estimate a story. Not exactly, but a ballpark figure should be possible. To be able to estimate a story, the story needs to be understandable something that relates closely to the negotiable requirement in the INVEST acronym. Also how well a story can be estimated depends on the size; the smaller the story the easier to estimate.

- Small:
  Stories should be small, because: Large stories are hard to estimate, hard to plan – even harder if the story does not fit into one iteration. There are two kinds of big stories. The compound story and the complex story. The first one is a story that comprises multiple smaller stories and can thus be split up, the latter on the other hand is inherently complex and cannot be easily split up into multiple smaller stories.

- Testable:
  A story should be testable, if a story is not testable how do you know it is finished? When a story can not be tested this can mean multiple things: the story isn't clear enough, or the story does not represent anything of value to the customer. In line with this requirement for stories it is often a good idea to write tests before hand, making it completely clear what the acceptance criteria will be for this specific story. This technique is called Acceptance Test Driven Development (ATDD).

This list off-course is not the holy grail, but can be used as a guideline to make it easier to create good user-stories.

4.1.3 User-story representation

As stated before user-stories are usually represented in the form of a card. Often this card is for example an post-it note. Next to the representation of the stand-alone user-stories the representation of the whole is a part of agile requirements engineering as well. This section will give some more insights into the way the set of user-stories is represented in a Scrum environment. The representation of user-stories is a way of managing them and can thus be seen as a part of the requirements management phase of requirements engineering. Some detail about requirements management and agile is given in the next section.

Scrum: Scrumboard

A scrumboard is used to display the state of work during a sprint and usually consists of three parts.

- To-Do;
- In progress (doing);
- Done.

These three parts – usually columns – are pretty self explanatory, but for completeness:

To-do The first column represent the user-stories that are on the sprint backlog, but on which work has not yet started.

In progress The second column represents the user-stories that one or multiple team-members are currently working on.

Done The final column represents the user-stories that are finished. Thus went from to-do to in-progress to done.

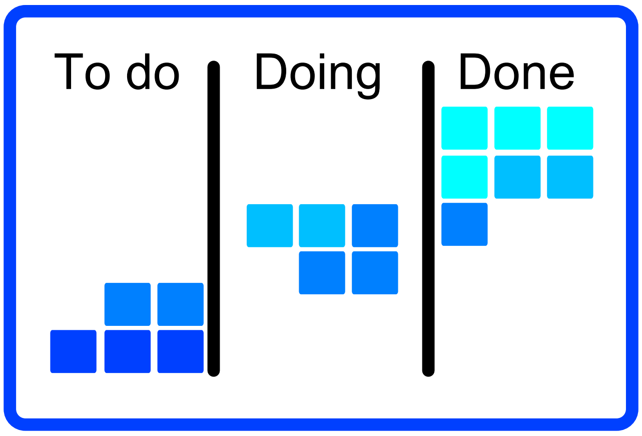


Figure 4.2: A visual representation of a Scrumboard

4.2 Agile Requirements Management

A part of Requirements Engineering (RE) is Requirements Management (RM). The goal of RE is to capture, store, disseminate and manage information. This includes all information concerning: [37]

- change & version control;
- requirements tracing;
- requirements status tracking.

Tracing requirements means keeping track of the relationships and changes that are made in requirements.

According to the agile manifesto which states that working software is more important than comprehensive documentation, this management and thus also the tracing of requirements is not the main priority.

Thus Agile requirements management is traditionally taken care of by putting up the user-story cards on a scrum-board. Which are then moved through the development process to be removed from the scrum-board once they are finished. Changes to user-stories as a result of conversations are added to the card, but when and why information is added is information that is often discarded. This while for example Eberlein says that traceability throughout the life-time of a user-story can be an important or even necessary addition to provide traceability from customers intentions to actual implementation [37].

4.3 Why is agile requirements engineering explained?

Before explaining the distributed version of agile requirements engineering the ‘normal’ version needs to be explained. Just as with agile there is a high reliance on face-to-face communication and less on documentation. Which is also represented by the three parts out of which a user-story consists.

Chapter 5

What is Distributed Agile Requirement Engineering?

What happens when we take agile requirement engineering and thus user-stories and try to use it in an distributed fashion. This chapter will first discuss some problems that arise from this combination, next to this some opportunities that arise from these problems are discussed. Finally this chapter will present a user-story model to take further benefit of the opportunities that arise.

5.1 Problems

When the benefits and challenges of the field of global software engineering and thus distributed engineering are held against the ideas of agile software engineering and thus agile requirements engineering, a number of problems arise. The two main problems are the following:

- Agile relies heavily on face-to-face communication as can be found in the agile manifesto: 'The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.', while this rich form of communication is harder to achieve when incorporating Global Software Engineering (GSE) [34].

- GSE and distributed development rely more heavily on documentation then when working in a co-located setting [49] while agile or more specific user-stories rely more on inferred knowledge and try to minimize the documentation as stated in the agile manifesto: 'Working software over comprehensive documentation'.

The following two sections will go deeper in to these two issues and how they may be resolved.

5.1.1 Lack of face-to-face communication

Agile has a large dependency on face-to-face communication, when we focus on agile requirements engineering and thus user-stories this dependency only grows bigger. As explained before a user-story is made up out of three parts, the three C's.

- Card
- Conversation
- Confirmation

Where the card is only a promise for a conversation, a conversation which is supposed to be face-to-face. This can be seen as a problem when working dislocated. Off-course there are technological tools that can help in overcoming this problem. But the richness of the communication can not reach that of face-to-face communication.

5.1.2 Documentation dependency

According to Delone and Gumm [30, 49] a larger documentation dependency is present when working distributed. This is a direct contradiction to what the agile manifesto says:

> Working software over comprehensive documentation

Or from the agile principles:

> The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Next to this agile methods like Scrum and XP have a stronger focus on communication rather then documentation, something that also returns in the way user-stories are build up.

5.2 Opportunities

Next to the problems that are introduced when combining agile with distributed, there are also some opportunities that can surface from the combination of agile and GSE. Some of these opportunities may lie in a better independence of team and task modularization. The opportunity which will be focused on in the remainder of this thesis follows from the lack of face-to-face communication and is explained in the next section.

5.2.1 Communication saved

While the fact that face-to-face communication is not possible when working distributed this means that all the communication takes place using technological support. This technological support can mean that the communication is saved when communicating, something that is not possible without added effort in a collocated environment. This can give a number of benefits already stated in the previous chapter. An other possible benefit is that communication and thus conversations about user-stories can be saved. When all communication, formal or informal, takes place using technological support it is possible to save and possibly connect it to user-stories. The next part of this chapter discusses a model about what can happen during the conversation about a user-story.

5.3 User-story Model

As stated in the previous section an opportunity that arises when working distributed is that communication and thus conversations can be saved. This saved communication could be coupled to user-stories. To make this coupling more transparent a model that defines the different actions that can be taken as an outcome from a conversation about a user-story could be benificial. This model will be defined and explained in the remainder of this section. In the next part of this thesis, this model will be used to implement support for making actions taken on user-stories transparent for users.

For this model the focus will lie on the conversation part of the user-story. As stated in the previous section the fact that the conversation is taking place distributed gives us the chance to record it and link it to the user-story to which it belongs. To get further benefits from this linking like for example tracing what kind of actions have been taking on a user-story a model that contains the different kinds of input, actions and output is needed.

5.3.1 The model

The model consists of three parts, input, action, and output. Thus in short on the input an action is taken which produces the output.

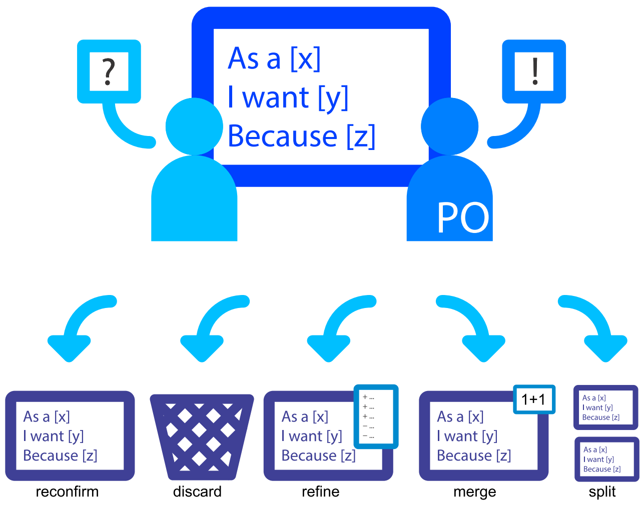


Figure 5.1: A visual representation of the user-story model.

Action

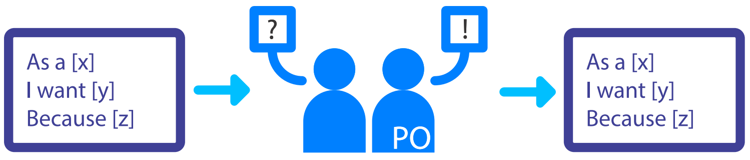
During the conversation about a set of user-stories multiple actions are possible that lead to different outputs. The possible actions are:

- Reconfirm (A_1)

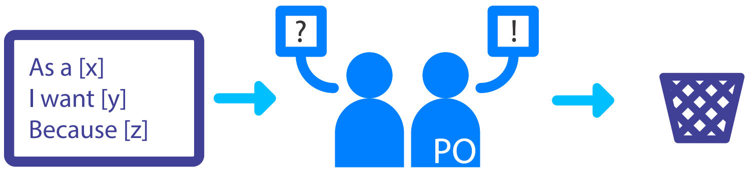
- Discard (A_2)
- Refine (A_3)
- Merge (A_4)
- Split-up (A_5)

First a more elaborate explanation of the different actions that can be taken upon a user-story.

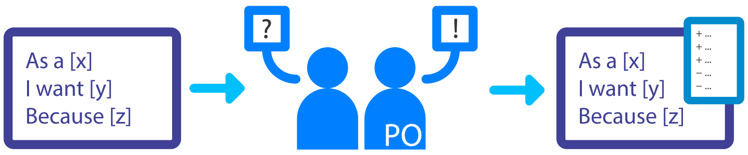
Reconfirm When changing nothing, the user-story simply passes through the conversation and is thus reconfirmed. Thus no other action is performed. Because of the fact that the conversation is part of this and actually did take place it can be argued that simply by having a conversation about a user-story alters the user-story. So when has nothing happened to a user-story? For the sake of this model when no extra information about the user-story has been added it will be considered as if nothing happened. This however does not mean that no information can be gathered from this classification. For example when nothing happens to a user-story apparently the user-story is good as it is now and at the moment does not need further refinement thus it is reconfirmed.



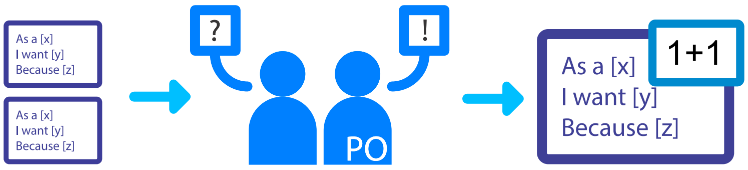
Discard When discarding a user-story this can have a number of reasons. The most positive being the user-story has been completed and is thus no longer needed. In this situation the conversation about the user-story will most likely be in the form of a demo of the added functionality to demonstrate that the user-story has been completed. The conclusion of this conversation will then be that the user-story is no longer needed and will thus be discarded. The alternative to this positive version of discarding a user-story is the situation in which a user-story is no longer needed while not yet developed or not yet completed. Next to that in this case a conversation about such an user-story will result in discarding the user-story.



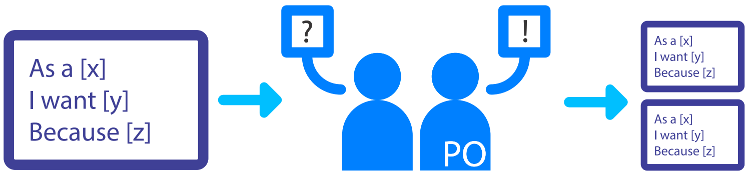
Refine As already stated in the paragraph considering the action 'reconfirm' the conversation itself adds information to a user-story and thus refines it. This refining is a very broad term and can range from actually changing the short description of the user-story to the addition of test-cases or detailed insights about the way it should be implemented.



Merge In the case that multiple user-stories have a lot of commonalities and are too small to be considered separately the conclusion of a conversation about these user-stories can be that they should be merged together to form one larger user-story.



Split-up On the other side of the spectrum, is the case that a user-story can be too large and can be split in to multiple user-stories that all add some kind of value to the product. The conclusion of a conversation about a user-story can be to split it up and create multiple smaller user-stories based upon the larger one.



None of the actions can be taken at the same time.

None of these actions can be taken at the same time, for completeness the list of why each action excludes the other actions.

Reconfirm As the name of the action describes when reconfirming a user-story. Thus doing nothing with a user-story after having a conversation about this user-story. None of the other actions can be taken.

Discard When a user-story is discarded, the user-story is not there any more and no other actions can be taken at the same time.

Refine When information is added to a user-story, the actions 'reconfirm' and 'discard' can not be the case. It can be discussed that merging two user-stories adds information to the user-story however the user-story that exists after merging is considered a new user-story. The same is the case when splitting-up a user-story, this results in multiple user-stories. So these actions do not co inside with refining either.

Merge As stated earlier merging results will create a new user-story and no other actions are possible.

Split-up When a user-story is split up none of the other actions can be taken concurrently.

For clarity, the following table gives the relationship between the different actions. None of the actions can take place at the same time.

	A_1	A_2	A_3	A_4	A_5
A_1	x	-	-	-	-
A_2	-	x	-	-	-
A_3	-	-	x	-	-
A_4	-	-	-	x	-
A_5	-	-	-	-	x

Input

The input is a user-story or in the case of a merge a set of user-stories. For the sake of this model an idea is also considered as a user-story. This because the only difference between a user-story and an idea would be the lack of representation for the user-story in case of an idea. Going from an idea to a an actual user-story would thus mean incorporating the action $refine$ which in turn gives the user-story a representation. To clarify the table:

- US^+ stands for multiple user-stories;
- US stands for a single user-story.

	US^+	US
Reconfirm	-	+
Discard	-	+
Refine	-	+
Merge	+	-
Split-up	-	+

Output

The following table shows what kinds of output are possible after the respective action taken.

- US^+ stands for multiple user-stories;
- US stands for a single user-story;
- $-$ stands for nothing, thus the input is discarded.

	US^+	US	-
Reconfirm	-	+	-
Discard	-	-	+
Refine	-	+	-
Merge	-	+	-
Split-up	+	-	-

5.3.2 Possible flows

The following table sums-up the previous tables and show the different possible flows that can be taken through this user-story model.

Input	Action	Output
US	Reconfirm	US
US	Discard	-
US	Refine	US
US	Split-up	US^+
US^+	Merge	US

5.4 Tool Requirements

Earlier this chapter the opportunity concerning the fact that the communication can be saved when incorporating distributed agile requirements engineering was presented.

As this model has not been defined before it can be assumed that there is not yet any tool support that adheres to this model. There for the following subsection will introduce the requirements a tool should adhere too to support this model.

5.4.1 Requirements

Which requirements should be met to support the user-story model defined in the previous section are the following

- R_1 The tool should be able to show the current/pending/active/open requirements;
- R_2 The tool should be able to visualize the connection between current and historical requirements;
- R_3 The tool should be able to couple fragments of a conversation to a performed action;
- R_4 The tool should support the action: Reconfirm;
- R_5 The tool should support the action: Discard;
- R_6 The tool should support the action: Refine;
- R_7 The tool should support the action: Split-up;

- R_8 The tool should support the action: Merge;

To further support the choice for these requirements a structured literature research has been performed. Next to supporting this requirements the choice for user-stories as the best-practice for agile requirements engineering is also further substantiated. This research and the results can be found in the next section.

5.5 Literature Review

This literature review is performed to further substantiated the requirements selected in the previous section. Next to this also the choice for user-stories will be checked using this literature review.

The structure of this literature review is based upon the procedures for a structured literature review as is described in [65]. On these procedures some exceptions have been made; namely:

- According to Kitchenham [65] the selected papers should be verified by an other person, because writing this thesis is an individual assignment this was not possible.

5.5.1 Review Goal

The goal of this literature review is twofold:

- Determining which technique to use when it comes to requirements in an agile development environment to further support the choice for user-stories;
- Support the selected requirements that have been determined in the previous section.

5.5.2 Review Methods

This section will discuss the review methods used.

Data sources and search strategy The primary data source used is http://scholar.google.com. Using this search engine papers from a multitude of different sources can be found. Using this search engine possible relevant papers have been located using the following search phrases:

- “Agile Requirements engineering” 247 results
- “Agile Requirements elicitation” 14 results
- “Agile Requirements analysis” 13 results
- “Agile Requirements specification” 4 results
- “Agile Requirements validation” 1 result
- “Agile Requirements management” 26 results

Study selection The study selection is subdivided into three steps. The first selection is a practical one. Only papers that are freely available using the TU-Delft network and that are in English are considered for this literature study. Other papers are discarded The papers that survived this first rudimentary selection will undergo a second selection round based on the title and the abstract of the paper. Papers in which agile is mentioned in combination with requirements in the title and/or the abstract are added to the preliminary accepted group. The third and final selection round is based on a more thorough investigation of the introduction and conclusion of the paper. Using the information gathered from these introductions and conclusions the final decision to include or exclude the papers from the study is made.

5.5.3 Included and Excluded studies

After the first selection a total of 179 papers was available to be considered. These are the papers that where found using the stated search phrases, are freely available using the TU-Delft network and are in English.

Included studies

The 179 papers that remained from the first selection are used as input for the second round of selecting. Now the selection is based upon the title and abstract of the paper. After a careful consideration of the 179 titles and abstracts, 64 items where accepted to be used as input for the third and final selection round. This selection based upon the title and abstract resulted in a selected list of papers depicted in table A.1 in appendix A.

Using this list of 64 initially selected papers the third and final round of selecting was done. This is done by determining if the paper is suitable to be added by reading the introduction and conclusion. Some reasons for excluding papers from the final selection was a different focus than the requirements itself like a focus on:

- The prioritization of the requirements. [97] [86] [43]
- A focus on security requirements [95]
- Choosing a requirements engineering methodology [4]

This final selection resulted in the list of papers depicted in table A.2 in appendix A.

5.5.4 Results

The results are like the goal of this literature review twofold, first the results concerning the choice of requirements representation will be given after which the results concerning the requirements will be presented.

Requirements representation

After a careful consideration of the 36 remaining papers from the three selection rounds it became clear that the most common way to represent requirements in an agile development environment is user-stories. From the 36 papers 26 papers mentioned this method as a best-practice or simply as a given fact. These 26 papers are the following: [1] [14] [16] [18] [20] [21] [26] [37] [62] [68] [69] [72] [76] [81] [83] [85] [86] [87] [90] [95] [98] [104] [110] [117]

In the 10 papers remaining, only 4 mention a possible alternative way of representing requirements. These are:

- Agile Requirements abstraction model. [106]
- Business process modelling notation. [71]
- Goal-Responsibility model. [17]
- Scenarios/Aspects. [7]

The remaining papers do not go into the representation of the requirements or do not offer an alternative for the representation of requirements.

Requirement substantiation

For each of the eight requirements defined the final selection of papers has been used to see if the requirement is mentioned or even recommended in the paper. First the eight requirements will be repeated, after that the results will be presented:

- R_1 The tool should be able to show the current/pending/active/open requirements;
- R_2 The tool should be able to visualize the connection between current and historical requirements;
- R_3 The tool should be able to couple fragments of a conversation to a performed action;
- R_4 The tool should support the action: Reconfirm;
- R_5 The tool should support the action: Discard;
- R_6 The tool should support the action: Refine;
- R_7 The tool should support the action: Split-up;
- R_8 The tool should support the action: Merge;

Table 5.1: Requirements substantiation results

R_1	**Recommends**	[104] [69] [67]
	Mentions	[70] [81] [76] [117] [74]
R_2	**Recommends**	[67] [98] [18] [69]
	Mentions	[70] [83] [37] [81] [17] [16]
R_3	**Recommends**	[69]
	Mentions	[70]
R_4	**Recommends**	[]
	Mentions	[]
R_5	**Recommends**	[67]
	Mentions	[95] [21] [76]
R_6	**Recommends**	[104] [69] [70] [67] [81] [85] [26] [62]
	Mentions	[14] [83] [95] [21] [1] [72] [76] [16] [68] [117] [110] [98] [86] [87] [106] [61] [17] [74]
R_7	**Recommends**	[69] [70] [67] [81] [85] [26] [104]
	Mentions	[95] [21] [1] [16] [68] [117] [110] [98] [86] [87] [106] [17] [74]
R_8	**Recommends**	[104] [85] [26]
	Mentions	[90] [106]

5.5.5 Discussion

The most important threat to the overall validity of this literature review is that his literature study has been performed by one person and thus the selections are not validated by a second party. This is a valid point, but is hard to be avoided because of the context in which this literature review is conducted.

Concerning the results of the twofold goal of this literature study there are also some points of discussion, first the point concerning the requirements representation:
The traditional and well known method of representing requirements, use-cases, is not present.
This can be explained by the fact that the selection procedure was focused on the combination of agile and requirements. Use-cases are a requirements representation most seen in a more traditional non-agile development environment.

About the second goal: substantiating the requirements defined it became clear that one of the requirements (R_4) can not be found in the literature, and one requirement (R_3) is only found in two papers. This can be explained by the fact that R_4, standing reconfirming a user-story and thus for doing nothing with a requirement, is not commonly made concrete. And in this case is added to make the user-story model complete. R_3 is about the coupling of the conversation to the requirement, in the context of this thesis possibilities have been identified in the combination between using technological support and user-stories. This can be seen as a positive point, not a lot of research has been done in this field.

5.5.6 Conclusions

Seeing that 2/3 of the papers use user-stories as the representation of requirements and that the alternatives mentioned in the other papers are not widespread in the context of this literature survey, the conclusion can be drawn that user-stories are the most wide spread best-practice when it comes to agile requirements engineering. Thus this is extra support for the choice of user-stories as basis for the representation of requirements.

Concerning the requirements that have been defined, these requirements, with the exception of two, can be substantiated from the literature. The fact that two requirements are hardly found, is because this is what is new in the context of this research.

To determine if a tool that supports the defined model and thus the transparency of actions taken on user-stories some research is required. This research is documented in the next part of this thesis. This part consists of the development of the tool and some limited empirical research into the usefulness of this support.

5.6 Why is distributed agile requirements engineering explained?

To determine what needs to be implemented to support the transparency into actions taken on user stories. An opportunity arose from the fact that all the communication can be saved because communication takes place using technological support. To take further advantage of this opportunity a model has been defined that has been used to determine requirements to which an application which support this transparency should adhere to.

Part II

Practical Research

Chapter 6

Introduction

For the practical part of this thesis we will continue with the theoretical model described at the end of the preceding part. The goal of the practical part of this thesis will be to research if transparency into the actions taken on user-stories is beneficial to the development process. In a traditional co-located agile set-up changes to user-stories simply appear after having a conversation that cannot be retrieved because it has passed. The distributed setting in which the company IHomer is operating gives a good opportunity to research if this added piece of information considering user-stories is beneficial, because of the fact that distributed conversations do not necessarily cease to exist at the end of the conversation.

The rest of this part is set up as follows: first the context for this practical research will be discussed, the next chapter is about how the process is build up and will give a short recap of the theoretical model. After this the actual implementation of this described process is discussed. This part will conclude with the findings from the practical research which will come from an experiment conducted IHomer.

6.1 Context

As stated earlier this practical research is conducted at the company IHomer. Next to this the already existing Iris tool from the Aspic research group will be used as a basis for this research. The company IHomer, the Aspic research group and Iris will be shortly introduced in the following three subsections

6.1.1 IHomer

IHomer is a small Dutch Software Engineering company which was founded in August of 2008. Its main principles are: individuality of its employees, mutual trust between its customers and its employees, transparency of the development process and continuity, both with respect to customer relations and the products it develops. One of the prime driving forces behind supporting the principles of IHomer is working from home. IHomers work from home as much as possible and collaborate by using collaborative technologies.

6.1.2 Aspic

The goal of the Aspic research program is the following (taken from aspic.nl):

The goal of the ASPIC research program is to develop solutions to the problems caused by the difficulties with acquiring and maintaining awareness in a distributed setting. In this research the focus lies on making the sharing of information a more passive activity which in turn will likely lower the effort to share awareness information, cause this information to be more recent and improve the quality of the information as well. To do this we will first identify the information from the context of a project that is important to coordinate and integrate the activities of the members of the development team. Following this we will select those information items for which there is a lot to gain with respect to sharing awareness information in a more passive way and develop supporting technology to valorize the concept. Summarizing, we can define the following three research questions:

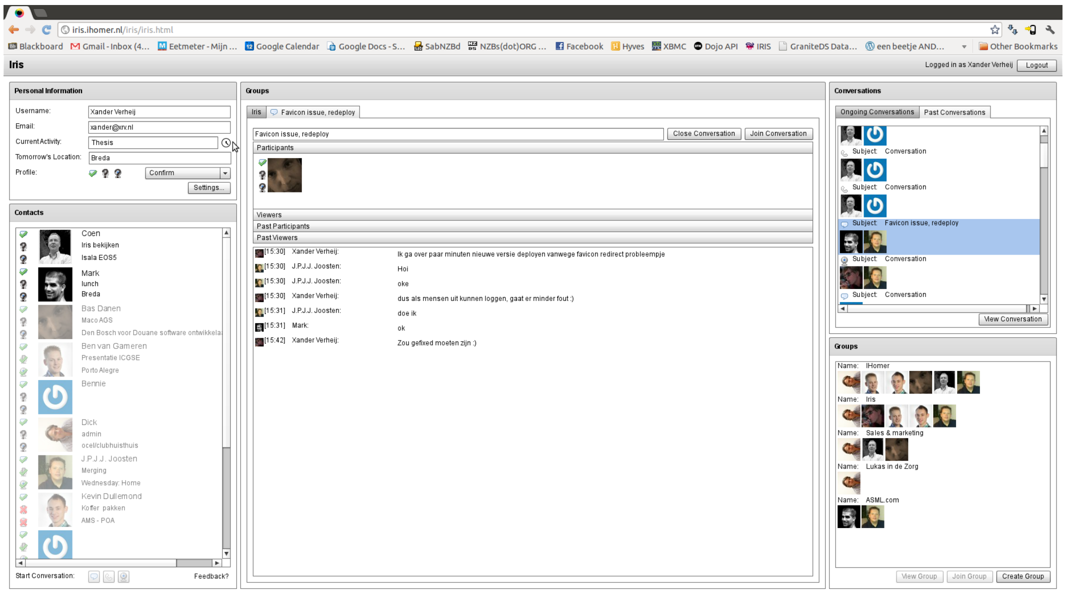
- RQ1: What types of contextual information are important in collaborative software engineering?
- RQ2: For which of those types is there a lot to gain with respect to sharing awareness information in a more passive way in a distributed environment?
- RQ3: How can sharing awareness information in a more passive way be valorized by technological support?

6.1.3 Iris

Iris is the practical combination between the Aspic research and the IHomer way of distributed working.

Figure 6.1: The logo for the Iris software

Iris is a communication tool used to enhance the awareness in distributed teams. Currently it is being developed in close cooperation with the company IHomer. It uses a completely different approach to for example video or audio conversations where the receiving end does not need to accept an incoming conversation. Instead the conversation is started directly, something that more closely resembles the co-located office experience. More about the development process used to develop Iris can be found in the next chapter, some of the technical details about the development of Iris follow in chapter 9.



Username: Xander Verheij
Email: xander@xrv.nl
Current Activity: Thesis
Tomorrow's Location: Breda
Profile: Confirm

Coen	Iris bekijken	Isala EOS5
Mark	lunch	Breda
Bas Danen	Maco AGS	Den Bosch voor Douane software ontwikkela
Ben van Gameren	Presentatie ICGSE	Porto Alegre
Bennie		
Dick	admin	ocel/clubhuisthuis
J.P.J.J. Joosten	Merging	Wednesday: Home
Kevin Dullemond	Koffer pakken	AMS - POA

Favicon issue, redeploy

```
[15:30] Xander Verheij:    Ik ga over paar minuten nieuwe versie deployen vanwege favicon redirect probleempje
[15:30] J.P.J.J. Joosten:  Hoi
[15:30] J.P.J.J. Joosten:  oke
[15:30] Xander Verheij:    dus als mensen uit kunnen loggen, gaat er minder fout :)
[15:31] J.P.J.J. Joosten:  doe ik
[15:31] Mark:              ok
[15:42] Xander Verheij:    Zou gefixed moeten zijn :)
```

Subject: Conversation
Subject: Conversation
Subject: Favicon issue, redeploy
Subject: Conversation
Subject: Conversation

Name: IHomer
Name: Iris
Name: Sales & marketing
Name: Lukas in de Zorg
Name: ASML.com

Figure 6.2: A screenshot of the Iris software

Chapter 7

Process

This chapter will describe the process used to come to an implementation that can aid in the transparency of actions on user-stories. The following section will introduce the process used for the development of Iris in the context of the Aspic research program. The final section of this chapter will show where the development of the tool support fits into this development process [36].

7.1 Process

The process that is used for the addition of the support to Iris will closely follow the process used in the development of Iris itself. Before explaining the process used for the development of Iris first the objectives for the development of Iris and the setting in which it is developed will get some attention.

7.1.1 Objectives

The goal of the Aspic research is to determine how to best support distributed software engineering with technological support for aiding people to acquire sufficient awareness. This main goal can be split in the following two core objectives:

- Support all awareness needs of software developers in a single platform;
- Enable integration of the awareness information from different information sources.

The details about how these objectives where acquired can be found in Dullemond et. al. [36]

7.1.2 Setting

The development and evaluation of the Iris tool is done at the earlier introduced company IHomer. The reason for this choice is the fact that all the people at this company in essence work distributed. This because of the fact that the goal is to work from home as much as possible. The way in which the people are approached within the company is also beneficial for the chosen process. The company does not see the people that work there as employees but as participants. This is reflected in the amount of responsibility every participant has. The experience that the participants have with working distributed is also beneficial for the development of Iris. This advantage is further used by letting the participants actively participate in the development of Iris. Next to this because the participants encounter the issues that Iris are supposed to solve daily, the solutions presented will benefit them directly.

7.1.3 The Iris Process

Based on the characteristics of the Iris project, the setting in which the research is conducted and the experience already gathered earlier in the Aspic program. The process used to develop Iris should fulfil three requirements. Namely:

- It should be able to cope with uncertainty and changing requirements since a genuinely novel product is being created and projects creating genuinely novel products are often face with uncertainty regarding both requirements and implementation technologies.
- It should involve the intended users of the system as strongly as possible because they are quite experienced with working in a distributed setting since this is what they encounter on a daily basis.
- It should stimulate the usage of the platform by all users by providing value as soon as possible.

The importance of the third and final requirement is because earlier research done by the Aspic research group [35] has shown that the value of awareness sharing technology is higher when a larger portion of the team uses this software and that the this is often a problem when introducing such tools in practical settings. To fulfil all three the requirements defined an agile methodology is chosen to realize the Iris platform. Firstly an agile methodology is better able to cope with uncertainty and changing requirements then plan-driven approaches [33]. The way this is accomplished is by gathering rapid feedback from the actual users of the Iris platform by using short iterations, rapid deployment and working closely with the customers. Working closely with the customers is done to acquire feedback but also to discover how value can be added as quickly as possible. This aspects of agile development result in fulfilling the three requirements defined earlier.

For the development of the Iris platform the agile methodology Scrum [102, 101] has been elected. The remainder of this section will go into the project specific way that Scrum has been implemented.

Scrum and Iris

The implementation of Scrum used for the development of the Iris project has chosen for two-week iterations of sprints when using the Scrum terminology. An overview of what these sprints look like can be seen in figure 7.1. The following paragraphs will explain the different meetings that are presented in that picture.

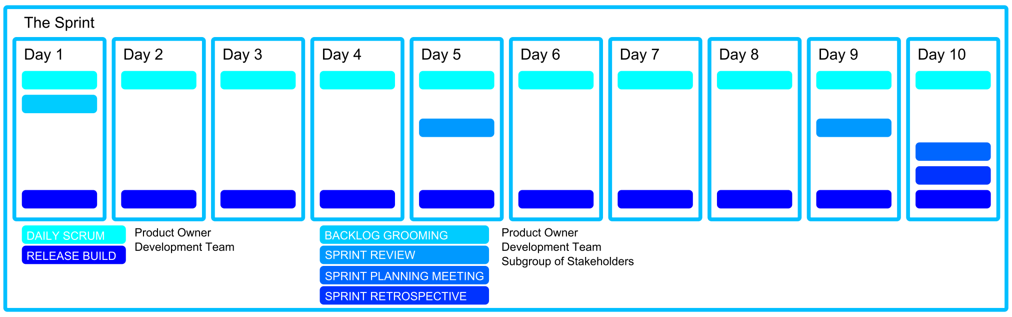


Figure 7.1: A visual representation of the Scrum process used in Iris, adapted from [36]

- Sprint Planning Meeting: Each sprint starts with a sprint planning meeting. In this meeting there is decided what to do in the sprint based on what is currently on the product backlog.

- Sprint Review: At the end of each sprint a sprint review is conducted. The goal of this review is to discuss what has been done in the sprint and compare this to what was agreed upon in the sprint planning meeting at the start of the sprint. The review revolves around reviewing the product and also includes a demonstration of the product.

- Sprint Retrospective: The retrospective is held at the same day as the sprint review for practical reasons – the same people are needed. This sprint retrospective focuses on reviewing the process and is intended to identify how the process can be further enhanced.

- Daily Scrum: During a sprint a daily scrum is held daily. This meeting is used to synchronize activities between the team-members and to create a plan for the next 24 hours.

- Release Build: Also done every day is release build. The reason of doing a build every day is to acquire feedback as soon as possible.

- Backlog Grooming: Twice every sprint backlog grooming takes place. Backlog grooming is the act of adding detail, estimates, and order to items of the on the product backlog.

The Roles Next to the development team which actually develops the Iris platform also the roles of Product Owner and Scrum Master need to be fulfilled. As well the role of Product Owner is fulfilled by one of the participants of IHomer while the latter, the role of Scrum Master is taken care of on a rotation basis by the different team-members of the Iris project.

Insight To provide all the stakeholders of the project insight into the project and the progress being made a Scrum-board is maintained and shared using Trello [1]. Trello is a collaboration tool which allows you to organize projects into boards. On such a board items can be moved along the different stages of the implementation and thus visualize the progress being made. The different stages used for the development of Iris are the following:

- Idea box: This stage is used to allow all the stakeholders to insert ideas or problems they encounter when using the software.
- Product backlog: This is the prioritized list of features for the product.
- Sprint backlog: This is the list of items that are going to be implemented during the current sprint.
- Under development: This list contains the items that are currently under development.
- Deployed in current sprint: This list contains items completed during the current sprint and already deployed in the live system.

In which stage an item is inserted on the board is based on their actual current status. Within the scope of Iris six different types of items are defined on the board:

- Feedback: Reaction to how the system currently functions and explanation of why this is insufficient or suboptimal;
- Idea: A rough idea of an addition or alteration to the product backlog;
- User story: One or more sentences in the everyday or business language of the end user that captures something the user wants to achieve using the system;
- Defect: Report of behaviour of the system that is in contrast with how the system should actually function;
- Task: A unit of work generally between four and sixteen hours which contributes to a backlog item;
- Research task: Because there is also research being done using the Iris platform, these tasks are also placed on the backlog to increase transparency and facilitate sprint planning.

[1] http://www.trello.com

Chapter 8

Feasibility

This chapter will present the questionnaire held at the beginning of the project and the corresponding results to show that the creation of the plug-in is actually useful.

8.1 Questionnaire

To get some insights in how the eventual users of the tool look towards the different actions that can be taken a short questionnaire will be held before the implementation of the support for transparency concerning actions taken on user-stories. The questions in this questionnaire will be based upon the 5 different actions that can be taken. The questions for each action are:

- Q_1 How satisfied are you with the execution of this action?;
- Q_2 How transparent is the execution of this action in hindsight?;
- Q_3 How important is the transparency of the execution of this action?;
- Q_4 How often is this action executed?;
- Q_5 How often is insight into this action required?

This questions will be given in the form of multiple choice questions with 5 options. This gives the following options for the already stated questions:

- Not satisfied ... very satisfied;
- Not transparent ... very transparent;
- Not important ... very important;
- The final two questions have the same 5 options:
  - More than once a day;
  - Once a day;

    - Once a week;
    - Once a month;
    - Once a project.

The questionnaire concludes with three open questions namely:

- On which projects within IHomer are your answers based primarily?;
- How is the transparency concerning actions on user-stories currently handled?;
- How would you like to see the transparency concerning action on user-stories handled?

The first 5 sets of quantitative questions can be directly derived from the 5 actions that the tool should support according to the requirements set-up at the end of chapter 5, namely:

- R_4 The tool should support the action: Reconfirm;
- R_5 The tool should support the action: Discard;
- R_6 The tool should support the action: Refine;
- R_7 The tool should support the action: Split-up;
- R_8 The tool should support the action: Merge.

The final 3 qualitative questions are meant to get some more insight into the current situation and the personal preferences of the participants.

8.2 Results

This section contains the results from the questionnaire held at IHomer. For each of the actions defined a table be presented with the average result for each of the five actions about which questions have been asked. To keep the tables readable the questions have been omitted from the tables them selves. The questions where:

- Q_1 How satisfied are you with the execution of this action?;
- Q_2 How transparent is the execution of this action in hindsight?;
- Q_3 How important is the transparency of the execution of this action?;
- Q_4 How often is this action executed?;
- Q_5 How often is insight into this action required?

Reconfirm

Question	Result
Q_1	3
Q_2	2
Q_3	3
Q_4	3
Q_5	3

Discard

Question	Result
Q_1	2
Q_2	2
Q_3	4
Q_4	2
Q_5	2

Refine

Question	Result
Q_1	4
Q_2	2
Q_3	4
Q_4	3
Q_5	4

Split-up

Question	Result
Q_1	4
Q_2	3
Q_3	4
Q_4	3
Q_5	3

Merge

Question	Result
Q_1	3
Q_2	2
Q_3	3
Q_4	2
Q_5	2

Interesting to see from these results is the fact that for all of the five actions that have been defined in the user-story model, the importance of transparency is ranked higher than the current transparency of the action. To be exact:

- Reconfirm 2 vs. 3
- Discard 2 vs. 4

- Refine 2 vs. 4
- Split-up 3 vs. 4
- Merge 2 vs. 3

From the other questions concerning how the transparency is currently taken care of, the answers can be summarized to:

- Trello [1]
- FogBugz [2]
- Word Documents

The final question about how the participants would like to enhance the transparency yielded some interesting possibilities:

- Time line to display user-stories and actions;
- Coupling of conversations leading to actions to user-stories;
- Coupling of actors to actions;
- Coupling of project to actions;
- Classifying actions using INVEST;
- Visual representation of what happened with a user-story.

The next section will go into the validity of the questionnaire.

8.3 Validity

This section will go into the four tests of validity that are defined in 'Case study research, Design and Methods' by Robert K. Yin [118] and if the research conducted for this thesis complies to the requirements set there. The four tests are:

- Construct validity: establishing correct operational measures for the concepts being studied;
- Internal validity: establishing a causal relationship, whereby certain conditions are shown to lead to other conditions, as distinguished from spurious relationships;
- External validity: establishing the domain to which a studies findings can be generalized;

[1]http://www.trello.com
[2]http://www.fogbugz.com

- Reliability: demonstrating that the operations of a study–such as the data collection procedures can be repeated, with the same results.

Of these four kinds of validity the first two can be claimed because of the following reasons.

Construct validity can be claimed because the questions asked are directly related to the five actions defined in the user-story model. Thus the correct operational measures for the concept are being studied. Next internal validity can be claimed because the questions asked are answered by a large portion of the employees at IHomer.

External validity and reliability can not be claimed, because the questionnaire has only been held at IHomer and thus also has not been repeated, this was due to time constraints.

8.3.1 Threats to validity

Next to the fact that external validity and reliability can not be claimed, there is an other threat to validity, the fact that the sample size is not very large. Approximately 50 % of the participants of IHomer participated but because of the small number of participants this leaves the sample size quite small.

8.4 Conclusion

Taking the results and threats to validity in to account and the suggestions made in the questionnaires, a conclusion can be drawn about the questionnaire that has been held. This conclusion is that it is feasible and that the potential usefulness of a tool that enhances the transparency concerning actions taken on user-stories has been demonstrated.

Thus the plug-in for Iris which will support the requirements defined at the end of chapter 5 and taking into account the suggestions made in the questionnaire will be implemented to research if there actually is an added value. This implementation will get more attention in the next chapter. The added value will be researched in the chapter 10.

Chapter 9

Implementation

Towards the implementation of the support for distributed agile requirements a iterative/agile approach is used. Some groundwork however is still necessary and this basis of the application will first be explained. After this the final tool will be presented including how it works, some technical details and some screenshots.

9.1 The technology

This section will give a short overview of the technology behind the implementation of the tool constructed and thus the technology behind the IRIS tool.

9.1.1 Flex

Apache Flex [1], formerly known as Adobe Flex [2], is a software development kit (SDK) for the creation of cross platform rich internet application (RIA) based on the Adobe Flash platform.

Figure 9.1: Flex logo

[1]http://flex.org
[2]http://adobe.com

9.1.2 RTMFP

RTMFP or Real-Time Media Flow Protocol is a proprietary protocol developed by Adobe Systems for delivering multi-media both using a client-server or peer-to-peer architecture. For the development of Iris an open-source implementation of a RTMFP server is used. This OpenRTMFP server or Cumulus [3] is used for the peer-to-peer audio and video communication supported by Iris.

9.1.3 GraniteDS

Granite Data Services (GDS) [4] is a development and integration solution for Flex and JavaEE. The entire project is opensource and released under the LGPL v2 [5] license.

The main features of GDS are:

- A client development framework (Tide) that brings familiar JavaEE concepts to the Flex side: dependency injection, context management, authentication and secured access, bean validation, etc.
- A comprehensive integration with major JavaEE application servers, frameworks and JPA engines: JBoss, GlassFish, WebLogic, WebSphere, Tomcat and Jetty; Hibernate, EclipseLink, OpenJPA and DataNucleus.
- An efficient real-time module (Gravity), based on Comet implementations, allowing scalable data-push.
- Code generation tools (Gas3) that help in replicating Java entity beans and services into their ActionScript3 equivalent. These tools are available as an Eclipse plug-in or an Ant task.
- Simplified configuration and high performances for critical deployments: most of the configuration is automated by scanning deployment environments and standard libraries are optimized for scalability.

Figure 9.2: GraniteDS Logo

[3] https://github.com/OpenRTMFP/Cumulus

[4] http://graniteds.org

[5] http://www.gnu.org/licenses/lgpl-2.1.html

9.1.4 JPA/Hibernate

To actually persist the data the Java Persistence API with Hibernate as backend is used. These take care of the translation between Java objects and the representation including relations in the database. Thus the act as a so called Object Relation Mapper or ORM. For more information about either JPA or Hibernate [6] please look at there respective websites.

9.2 The groundwork

The basis in which the support is implemented is the Iris software developed by the Aspic research group. This section will give a short overview of the original situation to and where it is now.

9.2.1 Original situation

In the original situation there was no need to save any data thus the Iris software can be depicted as shown in the following high-level picture. Here the server consisted only of

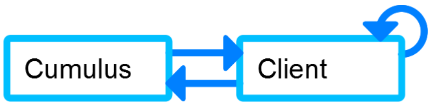


Figure 9.3: High level view of the original situation

the openRTMFP software Cumulus, keeping track of the currently online users and making real-time peer-to-peer communication possible. The Client side is created using the Mate [7] framework for Adobe Flex. When the need arose to save persistent data that would survive the restarting of the server application Iris grew to the intermediate situation shown in the next section.

9.2.2 Intermediate situation

When the need arose to have a more consistent user-list especially when user where not online, some data was being saved on the cumulus server. As can be seen the same set-up as the original situation is used only an database has been used to be accessed from the custom Lua [8] code on the Cumulus server. This situation was however no longer sufficient when a more complex data model was being developed for the persistence of more and more data, there for the next section explains the current situation.

9.2.3 Current situation

To facilitate the wish for more persistent data about for example groups of users, the support for user-story actions and more, a method for data storage was needed which lead to the

[6]http://www.hibernate.org/
[7]http://mate.asfusion.com/
[8]http://www.lua.org/

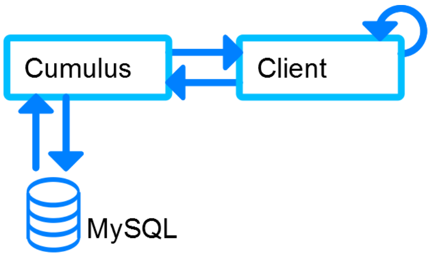


Figure 9.4: High level view of the intermediate situation

following high level picture. This is the situation which is used as the basis for the rest of the process. As this is the situation on which the extra functionality has been build some

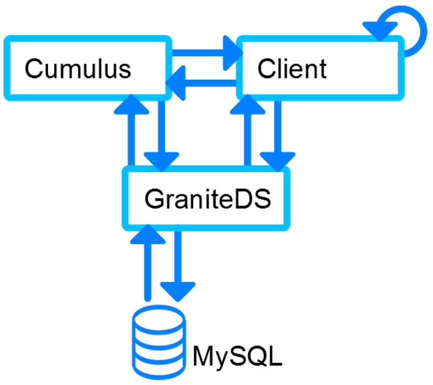


Figure 9.5: High level view of the current situation

more details concerning the state of the Iris software and how it is build will be discussed.

Set-up of Iris

For the current version of Iris the Mate framework has been replaced in favor of GraniteDS, this because of the build in support for data storage and the communication of this data between the server and the client. This also means that the Cumulus server has been stripped of all the data storage functionality that had been added for the intermediate state. GraniteDS gives the ability to create a entity model on the server side which can is automatically generated on the client side, the capabilities of GraniteDS also include easy to build services which allow for transparent communication of the entities from the server to the client and vice versa. The entity model used can be seen in Fig 9.6. This entity model is retrieved from the server using so called services. These services are called by the client-side controllers which then put the results of this calls into models which again are binded to the view to be displayed. The communication back from the view is done using events which some functions in the controllers listen for. Fig 9.7 gives a short overview of how each of the different modules of Iris work.

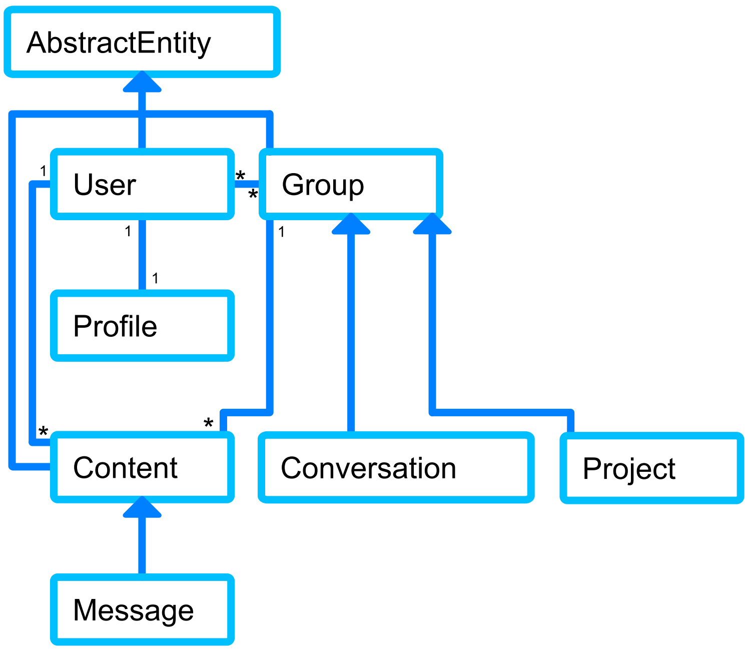


Figure 9.6: Entity model

9.2.4 Screenshots

This subsection contains two screenshots of the most common views on the Iris software. The first one being an ongoing conversation (see Fig 9.8), the second one being an group (see Fig 9.9).

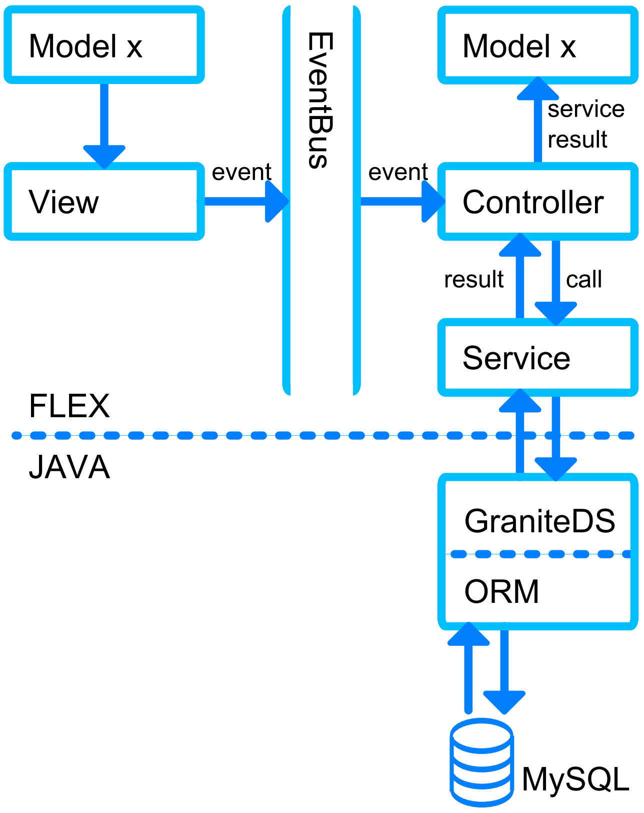
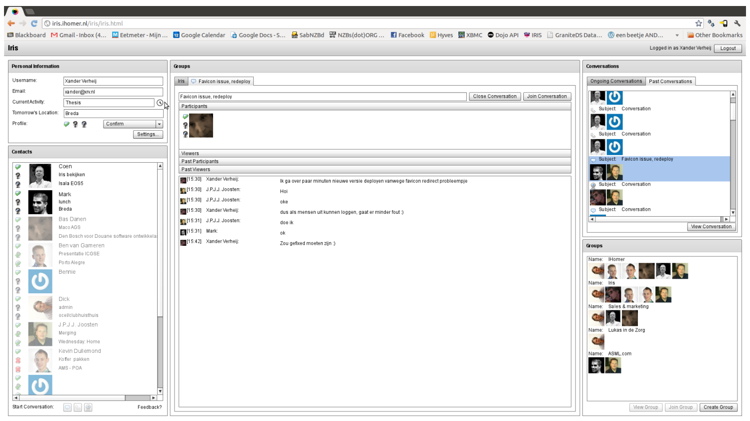


Figure 9.7: Overview of the set-up of Iris.



Username: Xander Verheij
Email: xander@xrv.nl
Current Activity: Thesis
Tomorrow's Location: Breda
Profile:

Coen
Iris bekijken
Isala EOS5

Mark
lunch
Breda

Bas Danen
Maco AGS
Den Bosch voor Douane software ontwikkela

Ben van Gameren
Presentatie ICGSE
Porto Alegre

Bennie

Dick
admin
ocel/clubhuisthuis

J.P.J.J. Joosten
Merging
Wednesday: Home

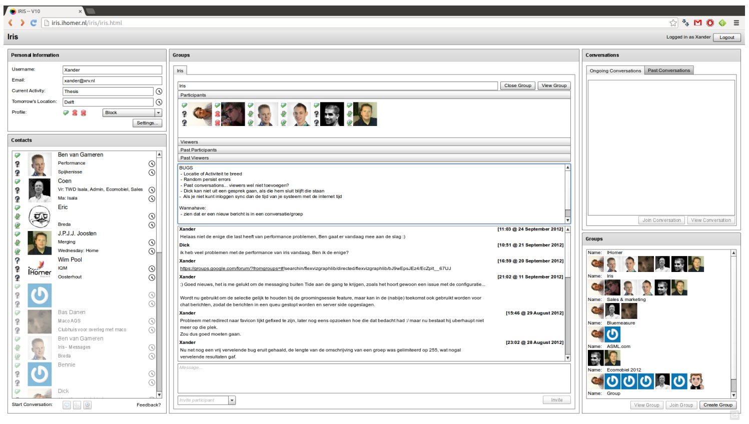
Kevin Dullemond
Koffer pakken
AMS - POA

```
[15:30]  Xander Verheij:    Ik ga over paar minuten nieuwe versie deployen vanwege favicon redirect probleempje
[15:30]  J.P.J.J. Joosten:  Hoi
[15:30]  J.P.J.J. Joosten:  oke
[15:30]  Xander Verheij:    dus als mensen uit kunnen loggen, gaat er minder fout :)
[15:31]  J.P.J.J. Joosten:  doe ik
[15:31]  Mark:              ok
[15:42]  Xander Verheij:    Zou gefixed moeten zijn :)
```

Subject: Conversation
Subject: Conversation
Subject: Favicon issue, redeploy
Subject: Conversation
Subject: Conversation

Name: IHomer
Name: Iris
Name: Sales & marketing
Name: Lukas in de Zorg
Name: ASML.com

Figure 9.8: A screenshot of the IRIS software, showing a conversation.



Username:	Xander
Email:	xander@xrv.nl
Current Activity:	Thesis
Tomorrow's Location:	Delft
Profile:	Block

Ben van Gameren
Performance
Spijkenisse
Coen
Vr: TWD Isala, Admin, Ecomobiel, Sales
Ma: Isala
Eric
Breda
J.P.J.J. Joosten
Merging
Wednesday: Home
Wim Pool
IQM
Oosterhout
Bas Danen
Maco AGS
Clubhuis voor overleg met maco
Ben van Gameren
Iris - Messages
Breda
Bennie
Dick

Iris

BUGS
- Locatie of Activiteit te breed
- Random persist errors
- Past conversations... viewers wel niet toevoegen?
- Dick kan niet uit een gesprek gaan, als die hem sluit blijft die staan
- Als je niet kunt inloggen sync dan de tijd van je systeem met de internet tijd

Wannahave:
- zien dat er een nieuw bericht is in een conversatie/groep

Xander **[11:03 @ 24 September 2012]**
Helaas niet de enige die last heeft van performance problemen, Ben gaat er vandaag mee aan de slag :)

Dick **[10:51 @ 21 September 2012]**
ik heb veel problemen met de performance van iris vandaag. Ben ik de enige?

Xander **[16:59 @ 20 September 2012]**
https://groups.google.com/forum/?fromgroups=#!searchin/flexvizgraphlib/directed/flexvizgraphlib/bJ9wEpsJEz4/EcZplt__67UJ

Xander **[21:02 @ 11 September 2012]**
:) Goed nieuws, het is me gelukt om de messaging buiten Tide aan de gang te krijgen, zoals het hoort gewoon een issue met de configuratie...

Wordt nu gebruikt om de selectie gelijk te houden bij de groomingsessie feature, maar kan in de (nabije) toekomst ook gebruikt worden voor chat berichten, zodat de berichten in een queu gestopt worden en server side opgeslagen.

Xander **[15:46 @ 29 August 2012]**
Probleem met redirect naar favicon lijkt gefixed te zijn, later nog eens opzoeken hoe die dat bedacht had :/ maar nu bestaat hij uberhaupt niet meer op die plek.
Zou dus goed moeten gaan.

Xander **[23:02 @ 28 August 2012]**
Nu net nog een vrij vervelende bug eruit gehaald, de lengte van de omschrijving van een groep was gelimiteerd op 255, wat nogal vervelende resultaten gaf.

Message...

Groups
Name: iHomer
Name: Iris
Name: Sales & marketing
Name: Bluemeasure
Name: ASML.com
Name: Ecomobiel 2012
Name: Group

Figure 9.9: A screenshot of the IRIS software, showing a group.

9.3 Final tool

Now that the global inner workings of Iris have been explained this chapter will continue with a somewhat more detailed view of the added functionality to support transparency into actions taken on user-stories. This will be done by going into the technical details of the final tool and the workflow supported by screenshots of how the tool works.

9.3.1 Technical details

This section will go into the entity model and supporting controllers, services and views created for the added functionality. This functionality will be created as loosely coupled from the base software as much as possible. This because of the need for an plug-in based architecture for the Iris platform. The plug-in will be coupled to Fogbugz [?]. Fogbugz is the issue management system used at IHomer and this coupling will be used to fill the system with initial data.

Entity Model

Figure 9.10 gives the main set-up of the entity model used for the plugin. The Item entity is added here twice for reasons of clarity. Next to this three entitiess also two classes are used to save settings that are needed to make the connection with Fogbugz as well as the project specific settings. Al these relations are only present in the plug-in code, thus not polluting the base code of the Iris project. Some more detail about the different entities:

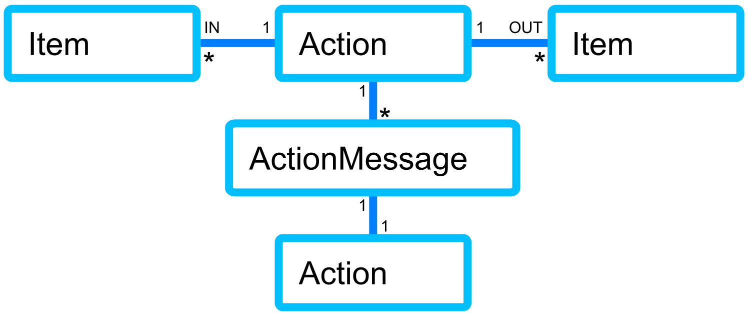


Figure 9.10: Entity model of the plugin.

- Action
  - Represents the action taken, thus containing an Enumeration with the 5 different actions:
    - ∗ Reconfirm
    - ∗ Refine
    - ∗ Discard
    - ∗ Split-up

    * Merge
  - Next to the type of action taken a relationship to the ActionMessage is present to connect the part of the conversation that is responsible for the taken action.
  - One-to-many incoming Items are defined, depending on the taken action this can be one or more items.
    * 1 incoming item; action: Reconfirm, Refine, Discard, Split-up;
    * 1+ incoming item; actions: Merge;
  - Finally one-to-many outgoing Items are defined, depending on the taken action this can be one or more items.
    * 1 outgoing item; action: Reconfirm, Refine, Discard, Merge;
    * 1+ outgoing item; actions: Split-up;
- Item
  - The item is the actual requirement and is coupled to an issue in Fogbugz using the unique id that is present in this system.
- ActionMessage
  - To keep the message entity from the Iris project untouched this entity is used to make the coupling between the action and the messages that lead to that action.

Next to this 3 entities defined to hold the data needed, there are also 2 entities used to keep the settings needed for the coupling with Fogbugz. These are the following:

- Settings
  - This keeps track of the url and token needed to contact the Fogbugz server.
- Project Settings
  - This keeps track of which Fobugzproject is coupled to the Iris project.

Model, Views, Controllers and Services

This section gives an overview of the different models, views, controllers and services that have been defined.

Views The following views have been defined for the plug-in:

- GraphView
- ProjectSettingsView
- SettingsView
- RightConversationView

- A view for each action that can be taken:
  - ReconfirmView
  - DiscardView
  - RefineView
  - MergeView
  - SplitView

Model To keep all the data that is visible on the different views under control one model is defined keeping track of the different items needed for the proper functioning of the plug-in:

- Global settings;
- Project settings: the settings for the different projects;
- Projects: the different projects available from Fogbugz;
- InstanceModels: the models defined per project.

This instanceModel contains all the information needed to have a conversation about the different requirements.

Services Two services have been defined to get all the information needed for the proper functioning of this plug-in one to get the information from the Java backend and one to get the information from FogBugz using the XML webservice. This webservice is rerouted through a local proxy because of the very strict cross-domain rules that are enforced by Flex.

Controllers Only one controller is defined for this plug-in this controller is used to keep track of all the events thrown and update the Model accordingly.

How all of these items come to getter can be seen in figure 9.11:

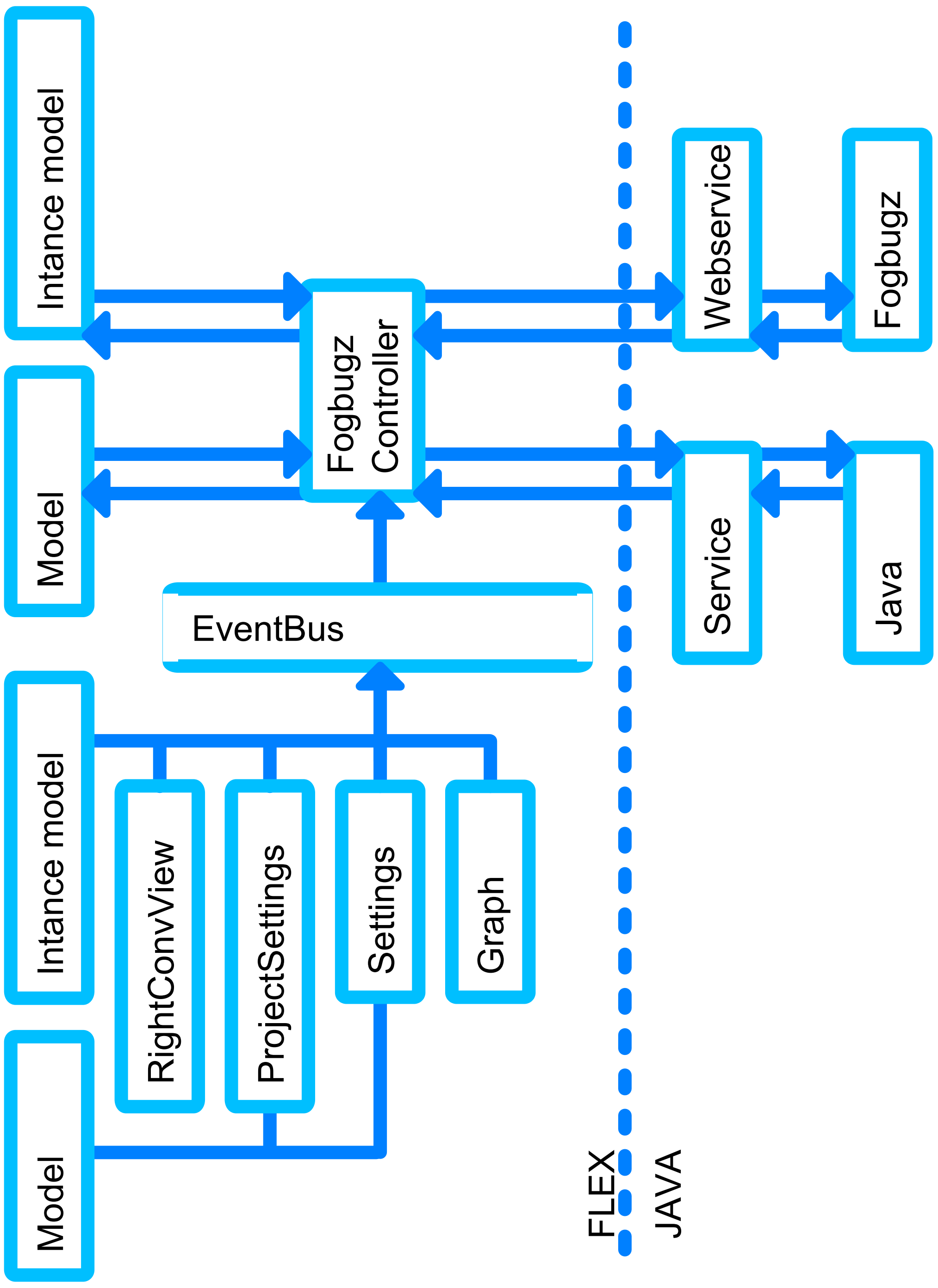
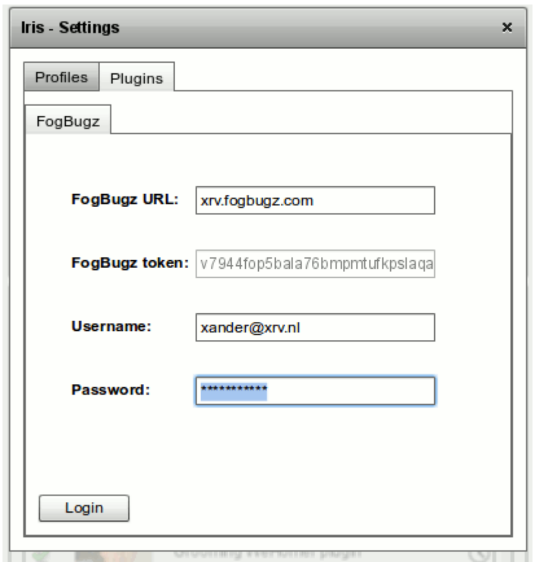


Figure 9.11: Entity model of the plugin.

9.3.2 Workflow

This section will go through the plug-in in the following steps:

- Set the global settings; Fig 9.12
- Set the project specific settings; Fig 9.13
- Start a conversation about the requirements; Fig 9.14
- Execute the five actions;
  - Discard; Fig 9.15
  - Reconfirm; Fig 9.16
  - Refine; Fig 9.17
  - Split-up; Fig 9.18
  - Merge; Fig 9.19
- Show the graph generated. Fig 9.20



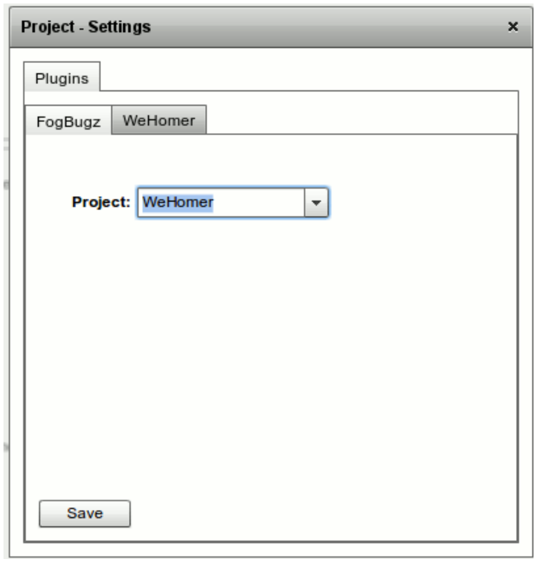
FogBugz URL: xrv.fogbugz.com

FogBugz token: v7944fop5bala76bmpmtufkpslaqa

Username: xander@xrv.nl

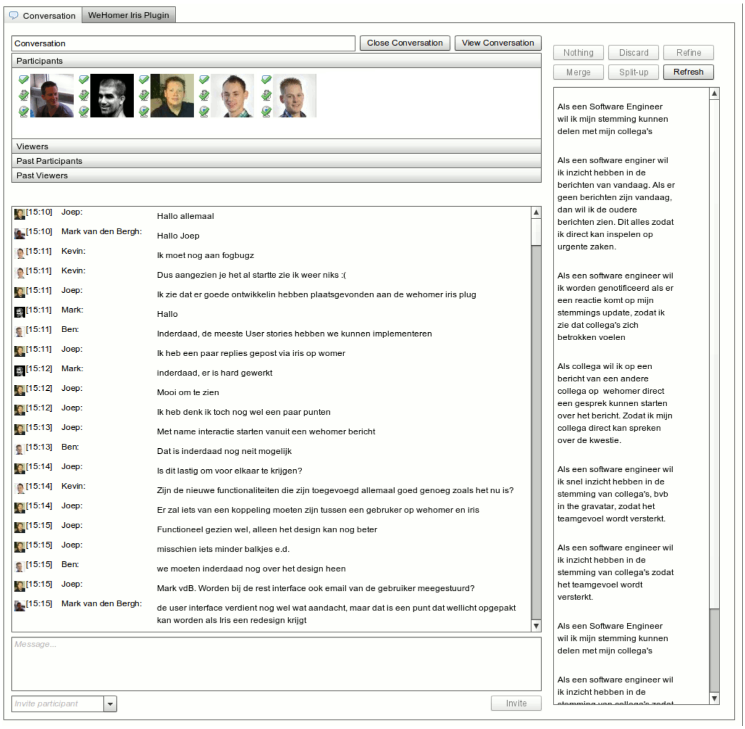
Password: ***********

Figure 9.12: Set the global settings



Project: WeHomer

Figure 9.13: Set the project settings



```
[15:10]  Joep:                Hallo allemaal
[15:10]  Mark van den Bergh:  Hallo Joep
[15:11]  Kevin:               Ik moet nog aan fogbugz
[15:11]  Kevin:               Dus aangezien je het al startte zie ik weer niks :(
[15:11]  Joep:                Ik zie dat er goede ontwikkelin hebben plaatsgevonden aan de wehomer iris plug
[15:11]  Mark:                Hallo
[15:11]  Ben:                 Inderdaad, de meeste User stories hebben we kunnen implementeren
[15:11]  Joep:                Ik heb een paar replies gepost via iris op womer
[15:12]  Mark:                inderdaad, er is hard gewerkt
[15:12]  Joep:                Mooi om te zien
[15:12]  Joep:                Ik heb denk ik toch nog wel een paar punten
[15:13]  Joep:                Met name interactie starten vanuit een wehomer bericht
[15:13]  Ben:                 Dat is inderdaad nog neit mogelijk
[15:14]  Joep:                Is dit lastig om voor elkaar te krijgen?
[15:14]  Kevin:               Zijn de nieuwe functionaliteiten die zijn toegevoegd allemaal goed genoeg zoals het nu is?
[15:14]  Joep:                Er zal iets van een koppeling moeten zijn tussen een gebruker op wehomer en iris
[15:15]  Joep:                Functioneel gezien wel, alleen het design kan nog beter
[15:15]  Joep:                misschien iets minder balkjes e.d.
[15:15]  Ben:                 we moeten inderdaad nog over het design heen
[15:15]  Joep:                Mark vdB. Worden bij de rest interface ook email van de gebruiker meegestuurd?
[15:15]  Mark van den Bergh:  de user interface verdient nog wel wat aandacht, maar dat is een punt dat wellicht opgepakt
                              kan worden als Iris een redesign krijgt
```

Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

Als een software enginer wil ik inzicht hebben in de berichten van vandaag. Als er geen berichten zijn vandaag, dan wil ik de oudere berichten zien. Dit alles zodat ik direct kan inspelen op urgente zaken.

Als een software engineer wil ik worden genotificeerd als er een reactie komt op mijn stemmings update, zodat ik zie dat collega's zich betrokken voelen

Als collega wil ik op een bericht van een andere collega op wehomer direct een gesprek kunnen starten over het bericht. Zodat ik mijn collega direct kan spreken over de kwestie.

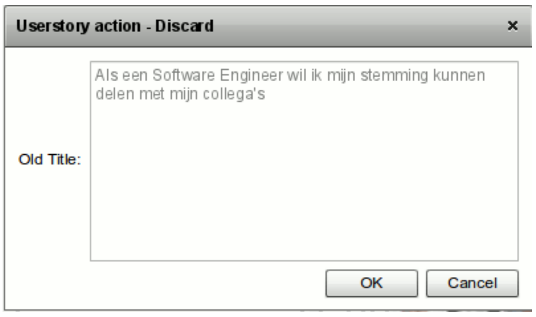
Als een software engineer wil ik snel inzicht hebben in de stemming van collega's, bvb in the gravatar, zodat het teamgevoel wordt versterkt.

Als een software engineer wil ik inzicht hebben in de stemming van collega's zodat het teamgevoel wordt versterkt.

Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

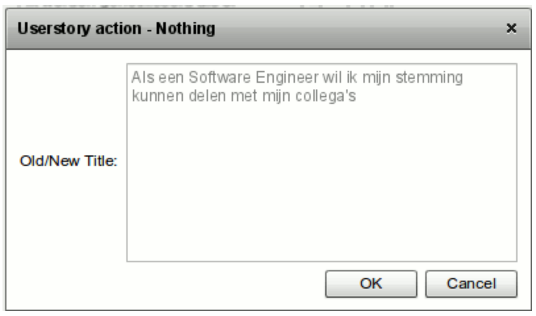
Als een software engineer wil ik inzicht hebben in de

Figure 9.14: A conversation with the requirements on the side



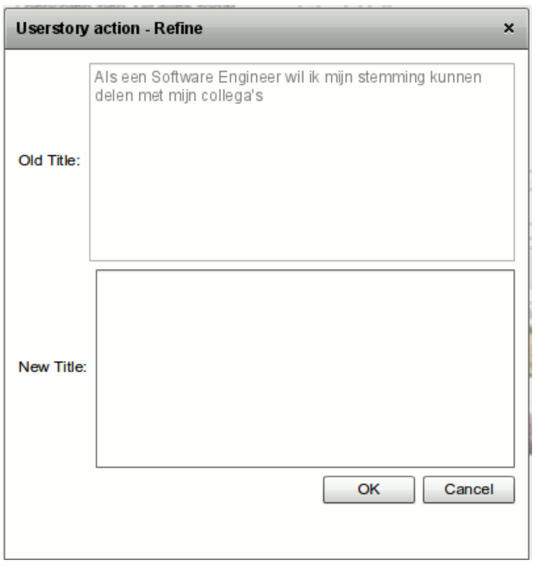
Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

Figure 9.15: Perform the discard action



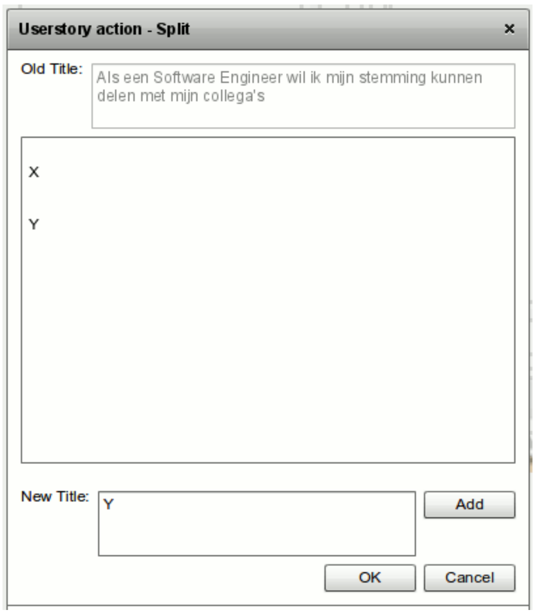
Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

Figure 9.16: Perform the reconfirm action



Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

Figure 9.17: Perform the refine action



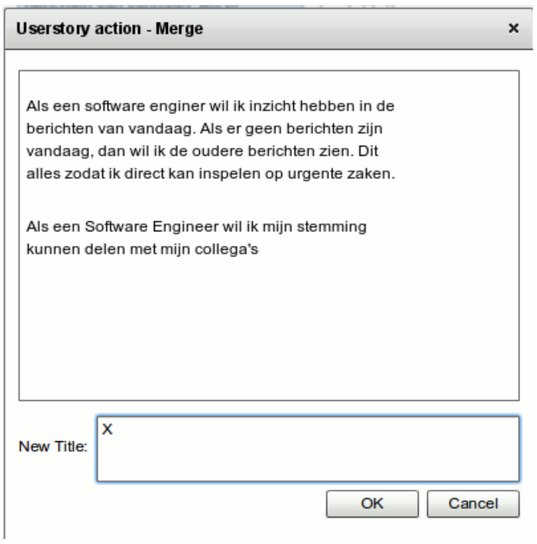
Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

X

Y

Y

Figure 9.18: Perform the split action



Als een software enginer wil ik inzicht hebben in de berichten van vandaag. Als er geen berichten zijn vandaag, dan wil ik de oudere berichten zien. Dit alles zodat ik direct kan inspelen op urgente zaken.

Als een Software Engineer wil ik mijn stemming kunnen delen met mijn collega's

X

Figure 9.19: Perform the merge action

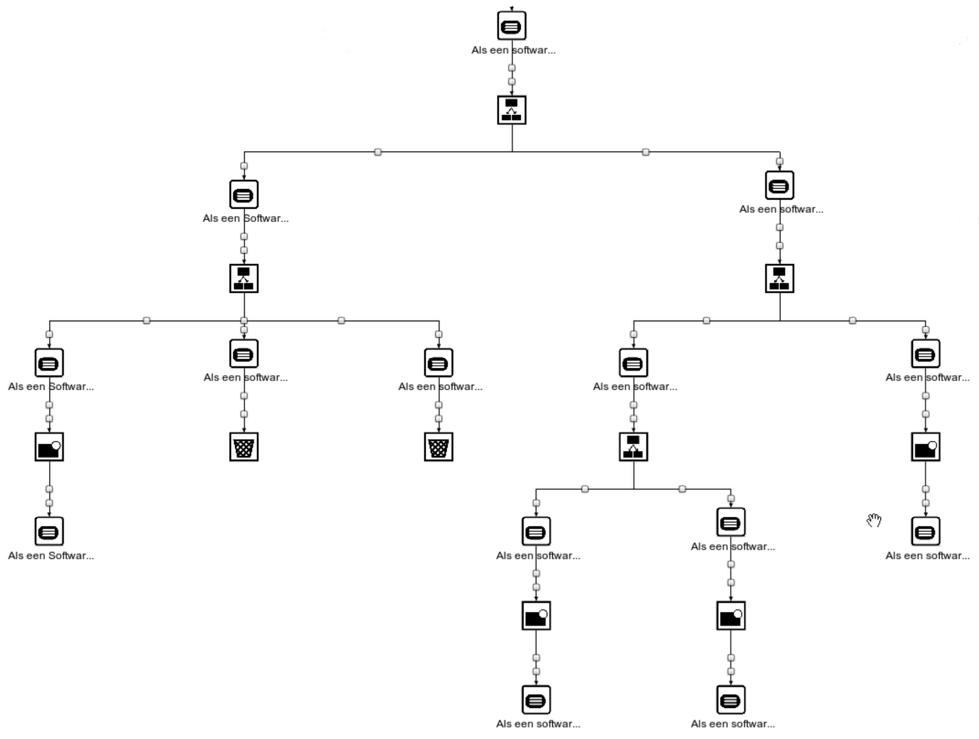


Figure 9.20: A part of the generated graph

Chapter 10

Results

This chapter contains a short description of the experiment conducted after the first questionnaire and also includes the questionnaire held after the experiment that has been conducted when the implementation was complete and the results that came from this questionnaire.

10.1 Experiment

To get insight into the added value of transparency concerning actions taken on user-stories / requirements a small experiment has been conducted at IHomer. This experiment has been conducted around the development of a plug-in for Iris, this plug-in will introduce some new already externally available information to Iris.

10.1.1 Setting

This experiment has been conducted in a largely distributed setting. The first grooming session took place with the people on the same location (IHomer), but here, with the exception of the communication, using the tool has been kept to an absolute minimum. The two other sessions were held with the participants being distributed and working from home.

10.1.2 Set-up

The experiment is set-up around three grooming sessions where user-stories, are revised and discovered concerning the implementation of this plug-in into Iris. These grooming sessions took place over the range of two weeks, and the actual implementation of the plug-in also took place during this period. After the completion of the three grooming sessions the people who have participated in the grooming sessions were given a questionnaire consisting of the same quantitative questions as the initial questionnaire, only in this case the questions are asked specifically about the Iris platform. Next to these quantitative questions some other questions concerning the subjective added value of the insight into actions taken on user-stories as well as the specific implementation created for this experiment are asked. This questionnaire is shown in the following section.

10.1.3 Questionnaire

The questions of the questionnaire are based on the questions asked in the initial questionnaire. This time the questions are scoped to Iris. For clarity first the five actions on which the questions are applicable will be repeated:

- Reconfirm
- Discard
- Refine
- Split-up
- Merge

The questions for each of the five actions are:

- Q_1 How satisfied are you with the execution of this action?;
- Q_2 How transparent is the execution of this action in hindsight?;
- Q_3 How important is the transparency of the execution of this action?

These questions have been asked in the form of multiple choice questions with 5 options. This gives the following options for the already stated questions:

- Not satisfied ... very satisfied;
- Not transparent ... very transparent;
- Not important ... very important;

Next to these identical questions from the initial questionnaire some other qualitative questions with 5 options have been added. The added questions are the following:

- How good is the representation of current requirements in Iris?
  - Answer ranging from: bad to good.
- How would you enhance the representation of the current requirements.
- How good is the visualization of the connection between current and past requirements?;
  - Answer ranging from: bad to good.
- How would you enhance the visualization?
- How good is the coupling between conversation fragments and the requirements?;
  - Answer ranging from: bad to good.

- How would you enhance this coupling?

These questions can again be coupled to the following requirements previously defined in chapter 5:

- R_1 The tool should be able to show the active requirements;
- R_2 The tool should be able to visualize the connection between current and historical requirements;
- R_3 The tool should be able to couple fragments of a conversation to a performed action;

The questions added about the specific implementation created for this experiment are the following:

- What is your opinion about the support within Iris?;
- What would you change about the support within Iris?;
- Did you have an advantage using the support within Iris?

The results of the initial questionnaire and the results from the experiment will be presented in the next section.

10.2 Results

This section contains the results from the initial questionnaire and the questionnaire held at the end of the short experiment held.

- Q_1 How satisfied are you with the execution of this action?;
- Q_2 How transparent is the execution of this action in hindsight?;
- Q_3 How important is the transparency of the execution of this action?

Reconfirm

Question	Initial Result	Result
Q_1	3	4
Q_2	2	4
Q_3	3	2

Discard

Question	Initial Result	Result
Q_1	2	4
Q_2	2	4
Q_3	4	4

Refine

Question	Initial Result	Result
Q_1	4	4
Q_2	2	4
Q_3	4	4

Split-up

Question	Initial Result	Result
Q_1	4	4
Q_2	3	4
Q_3	4	4

Merge

Question	Initial Result	Result
Q_1	3	4
Q_2	2	4
Q_3	3	4

The results from the questions asked about the specific implementation are the following:

- Q_1 How good is the representation of current requirements in Iris.
- Q_1 How good is the visualization of the connection between current and past requirements;
- Q_1 How good is the coupling between conversation fragments and the requirements;

Question	Result
Q_1	3
Q_2	4
Q_3	2

Added to each of the above questions was also the question about how this could be enhanced. Some excerpts from the results are the following.

How would you enhance the representation of the current requirements?

One issue surfaced here from almost all the responses on the questionnaire. Namely to enhance the ways in which the requirements can be identified. At the moment they are only displayed using the description. Some more information could make them easier recognizable. In the same visual representation line some other remarks where made:

- Better mark the discussed requirements;
- Highlight keywords in the requirements;
- Integration of the graph functionality in the list
  - Show the parents of a requirement;
  - Show the taken actions on a requirement.

How would you enhance the graphical visualization?

The general consensus in the responses is that the graph generated is clear. However some remarks about the ability to identify the different requirements were made. Some possible enhancements mentioned are the following:

- Put the currently active requirements on the same level, to make them better recognizable;
- More color to make the requirements better recognizable;

How would you enhance the coupling between requirements and the conversation

This part of the plug-in got the lowest score and is indeed still in the start-up phase. From the questionnaires it can be concluded that the coupling is appreciated but that there is a lot of room for enhancement. Some possible problems and enhancements mentioned in the results are:

- Conversations don't always go in order, thus before making a decision about one requirements multiple requirements came alone, possible solutions mentioned are:
  - Do not couple all the messages, but be able to make a sub selection;
  - Be able to change the coupled conversations parts after the actions has been taken;
  - Ability to add a summary of the conversation;
  - Add separate remarks to the action;
  - A more elaborate view for the conversation to make it easier to search the conversation and possibly copy from it.
- Ability to see the conversation part coupled to an action in the context of the complete conversation;
- Ability to also couple audio/video conversations to actions.

What is your opinion about the support within Iris?

From the responses on the questionnaire the overall opinion is that the support is reasonable and that the concept is beneficial.

What would you change about the support within Iris?

In the responses on this question, results where in the same category as the earlier responses to the possible enhancement questions. This can be summarized as enhancing the layout, some specific points mentioned:

- Give requirements a more prominent place in the conversation
  - Make it visible which action is taken by whom;
  - Make it clearer which requirements are still pending;
- Ability to create new requirements.

Did you have an advantage using the support within Iris?

The general consensus from the responses can be summarized in the fact that the respondents see the added functionality as an added value. Some reasons mentioned are the following:

- It is beneficial to archive the conversations about multiple items on one place;
- The transparency on actions helps getting an overview of the entire project;
- In comparison to just using FogBugz we go through the user-stories quicker;
- Sometimes it was hard figuring out where user-stories on the backlog came from, with the conversation history this would not happen;
- Making the conversations about the requirements explicit helps keeping the entire team involved;
- Making it easier to track the development for people not directly involved with the project.

10.3 Validity

For this questionnaire the validity has been checked against the four test already mentioned in chapter 8 just as with the initial test.

Construct validity can be claimed because in case of this research the requirements to which the researched tool should adhere to have been defined before hand. These requirements have also been to develop the questions for the questionnaires, thus the correct operational measures for the concept have been established and construct validity can also be claimed.

Internal validity can be claimed because the questions that are being asked are based on the user-story model and requirements for a tool that are defined in chapter 5. The questions about the possible added value are all asked specifically about the tool and after working with the tool thus the causal relationship that the results from this questionnaire are the result of the added functionality can be assumed. This leads to the fact that also internal validity can be claimed.

The latter two, external validity and reliability can not be claimed, because of the following reasons.

External validity as said is about knowing whether a studies findings can be generalized beyond the immediate research. This research has only be conducted at one company and thus no claims can be made about generalizing the results beyond this company. Reliability can not be claimed because the experiment has only been performed once, mainly because of time constraints.

Part III

Conclusions and future work

Chapter 11

Conclusions and Future Work

This chapter will start with the contributions made to the field of distributed agile requirements engineering. After this the conclusions that can be drawn from the results of the experiment conducted will be presented. Subsequently there will be some discussion and reflection concerning the results, conclusions drawn and the process followed. Finally this chapter will end with some pointers for possible future work and future research.

11.1 Contributions

In this study research has been done to determine to which extend transparency in to actions taken on user-stories is beneficial for the distributed development process. The usefulness of this can be found in the fact that more and more development is being done distributed, either globally or on a smaller scale. By combining the three different fields which formed the basis for this study, namely Global Software Engineering, Agile Software Engineering and Requirements Engineering (RE) comes an interesting paradox. Where agile software engineering focuses on less documentation and more communication, global software engineering and thus distributed engineering is said to have more reliance on documentation. Despite this paradox and an ever growing use of both agile and distributed working, there is not a lot of research done in this crossroads between fields. For this reason the research into Distribute Agile Requirements Engineering has been performed.

Before being able to research to which extend the transparency in to actions can be beneficial it was needed to determine how to make this transparency possible. This has been done by giving some background information about agile software engineering, agile requirements engineering and finally distributed agile requirements engineering. The last one of these three resulted in a user-story model which in turn led to a number of requirements to which a tool should adhere. These requirements have been given some extra validation by using a structured literature survey to verify the requirements and next to this validate the choice for user-stories as a basis concerning agile requirement engineering.
Using this model, requirements have been developed to which an application to support the model should adhere to. Using these requirements and the results from an initial question-

naire showed that the potential users see an added value concerning this functionality . A plug-in for the Iris project has been developed.
Using the developed plug-in for the Iris project a small experiment has been conducted around the development of an other plug-in for the Iris project. This to determine if the added functionality actually added value to the development process. This research focused on possible enhancements and the subjective view of the participants into the added value.

11.2 Conclusions

The main question that was attempted to be answered in this thesis is the following:

> ”To which extend does transparency concerning actions taken on requirements enhances the distributed development process?”

Within the scope of this research and using the results of the questionnaire that has been held after a small experiment. This experiment has been conducted using a plug-in for the Iris project which has been developed using the requirements that have been defined using the user-story model as a basis. From the results of this experiment gave a clear view on the matter.

From the results can be concluded that the participants of the experiment see the transparency concerning actions taken on requirements as an added value to the development process, this conclusion can be drawn from the quantitative results from the questionnaire, in which with the exception of one action all the actions there importance are classified with a 4 on a scale from 1 to 5. The same conclusion can be drawn from the results of the qualitative questions. From these responses it came forth that the respondents see the transparency as an added value on a number of different ways. Some of the reasons mentioned are:
It is possible to..

- Keep track of where a user-story comes from.
- Keep track of all the conversations about a certain project on one place;
- Going through all user-stories goes quicker;
- Keeping the team more together because the conversation about user-stories is made explicit.

Taking all of these results together the conclusion can be summarized in:
Yes, transparency concerning actions taken on requirements enhances the distributed development process. However to determine to which extend this enhancement is applicable more research is needed. Some thoughts about future research will be given in the section about recommendations and future work.

11.3 Reflection

The most important limitation of the research performed for this thesis is the limited size of the group where the results are based on. Both the initial questionnaire and the final questionnaire were limited in terms of number of responses, the first mainly because of the size of IHomer, the latter because of the fact that the team that develops Iris is not any bigger. Concerning the team that develops Iris there is an other limitation, for the duration of the creation of this master thesis I have also been involved in this team thus I have been closely involved in the entire creation of the Iris project and off course the people involved in this development, thus these people could have a bias towards the topic researched.

11.4 Recommendations for future work and research

From the results of this research some pointers for future work and research have become apparent. The research has shown that more work in to the plug-in especially on the field of graphical representation and the coupling of the conversation textual or audiovisual to the taken actions could further enhance the experience and perhaps the benefits of the tools. An other interesting suggestion made that also came from the literature is to classify actions using the INVEST acronym and perhaps also check new and existing user-stories against these rules.

To determine to which extend the transparency into actions on user-stories has an effect on the development process more research is required. Preferably using multiple larger projects that have more people working on them and run during a longer time. Also a concurrent same size and type project that does not use a tool to support this transparency would be beneficial to showing to which extend it helps.

Bibliography

[1] N.N.B. Abdullah, S. Honiden, H. Sharp, B. Nuseibeh, and D. Notkin. Communication patterns of agile requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering*, page 1. ACM, 2011.

[2] N.N.B. Abdullah, H. Sharp, and S. Honiden. Communication in context: A stimulus-response account of agile team interactions. *Agile Processes in Software Engineering and Extreme Programming*, pages 166–171, 2010.

[3] P.J. Ågerfalk. Investigating actability dimensions: a language/action perspective on criteria for information systems evaluation. *Interacting with Computers*, 16(5):957–988, 2004.

[4] G. Aiello, M. Alessi, M. Cossentino, A. Urso, and G. Vella. Rtdwd: real-time distributed wideband-delphi for user stories estimation. *Rapid Integration of Software Engineering Techniques*, pages 35–50, 2007.

[5] M.Z. Akbar. Requirements elicitation in agile processes. 2010.

[6] T.J. Allen. Managing the flow of technology: Technology transfer and the dissemination of technological information with the r&d organization. 1977.

[7] J. Araujo and J.C. Ribeiro. Towards an aspect-oriented agile requirements approach. In *Principles of Software Evolution, Eighth International Workshop on*, pages 140–143. IEEE, 2005.

[8] T. Aschauer, G. Dauenhauer, P. Derler, W. Pree, and C. Steindl. Could an agile requirements analysis be automated?lessons learned from the successful overhauling of an industrial automation system. *Innovations for Requirement Analysis. From Stakeholders Needs to Formal Designs*, pages 25–42, 2008.

[9] T. Aschauer, G. Dauenhauer, P. Derler, W. Pree, and C. Steindl. Key factors of a successful agile requirements analysis in the realm of overhauling an industrial automation system. *Innovations for Requirements Engineering*, page 21, 2008.

[10] Z. Bakalova, M. Daneva, A. Herrmann, and R. Wieringa. Agile requirements prioritization: what happens in practice and what is described in literature. *Requirements Engineering: Foundation for Software Quality*, pages 181–195, 2011.

[11] M. Bass and D. Paulish. Global software development process research at siemens. In *Third International Workshop on Global Software Development*, pages 8–11, 2004.

[12] K. Beck. *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[13] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[14] E. Bjarnason, K. Wnuk, and B. Regnell. A case study on benefits and side-effects of agile practices in large-scale requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering*, page 3. ACM, 2011.

[15] B. Boehm. A survey of agile development methodologies. 2007.

[16] K. Boness and R. Harrison. Goal sketching: Towards agile requirements engineering. In *Software Engineering Advances, 2007. ICSEA 2007. International Conference on*, pages 71–71. IEEE, 2007.

[17] K. Boness, R. Harrison, and K. Liu. Goal sketching: An agile approach to clarifying requirements. *International Journal On Advances in Software*, 1(1):1–13, 2009.

[18] S. Bose, M. Kurhekar, and J. Ghoshal. Agile methodology in requirements engineering. *SETLabs Briefings Online*, 2008.

[19] G. Boström, J. Wäyrynen, M. Bodén, K. Beznosov, and P. Kruchten. Extending xp practices to support security requirements engineering. In *Proceedings of the 2006 international workshop on Software engineering for secure systems, ACM New York, NY, USA*, 2006.

[20] P.S. Brockmann and T. Thaumuller. Cultural aspects of global requirements engineering: An empirical chinese-german case study. In *Global Software Engineering, 2009. ICGSE 2009. Fourth IEEE International Conference on*, pages 353–357. IEEE, 2009.

[21] L. Cao and B. Ramesh. Agile requirements engineering practices: An empirical study. *Software, IEEE*, 25(1):60–67, 2008.

[22] E. Carmel. *Global software teams: collaborating across borders and time zones*. Prentice Hall PTR, 1999.

[23] E. Carmel and R. Agarwal. Tactical approaches for alleviating distance in global software development. *Software, IEEE*, 18(2):22–29, 2001.

[24] E. Carmel and P. Tjia. *Offshoring information technology: sourcing and outsourcing to a global workforce*. Cambridge Univ Pr, 2005.

[25] A. Chaudhary, A. Punia, and M. Pujar. Requirements engineerings role in agile development. 2008.

[26] C.P. Chetankumar Patel and M.R. Muthu Ramachandran. Story card based agile software development. *International Journal of Hybrid Information Technology (IJHIT)*, 2(2):125–140, 2009.

[27] M. Cohn. *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[28] D. Damian, F. Lanubile, and H.L. Oppenheimer. Addressing the challenges of software industry globalization: the workshop on global software development. In *Proceedings of the 25th International Conference on Software Engineering*, pages 793–794. IEEE Computer Society, 2003.

[29] D.E. Damian and D. Zowghi. The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 319–328. IEEE, 2002.

[30] W. DeLone, J.A. Espinosa, G. Lee, and E. Carmel. Bridging global boundaries for is project success. In *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pages 48b–48b. IEEE, 2005.

[31] T. DeMarco. *Slack: Getting past burnout, busywork, and the myth of total efficiency*. Broadway, 2002.

[32] P. Dourish and S. Bly. Portholes: Supporting awareness in a distributed work group. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 541–547. ACM, 1992.

[33] K. Dullemond and B. van Gameren. Technological support for distributed agile development. *Department of Software Technology, Delf University of Technology, Delf*, page 223, 2009.

[34] K. Dullemond, B. van Gameren, and R. van Solingen. Virtual open conversation spaces: Towards improved awareness in a gse setting. In *Proceedings of the 2010 International Conference on Global Software Engineering*, pages 247–256, 2010.

[35] K. Dullemond, B. van Gameren, and R. van Solingen. An industrial evaluation of technological support for overhearing conversations in global software engineering. *Delft University of Technology, Tech. Rep*, 2011.

[36] K. Dullemond, B. van Gameren, and R. van Solingen. Supporting distributed software engineering in a fully distributed organization. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*, pages 30–36. IEEE, 2012.

[37] A. Eberlein and J.C.S. do Prado Leite. Agile requirements definition: A view from requirements engineering. In *Proceedings of the International Workshop on Time-Constrained Requirements Engineering (TCRE02)*, pages 4–8, 2002.

[38] C. Ebert and P. De Neve. Surviving global software development. *Software, IEEE*, 18(2):62–69, 2001.

[39] C. Ebert, C.H. Parro, R. Suttels, and H. Kolarczyk. Improving validation activities in a global software development. In *Proceedings of the 23rd international Conference on Software Engineering*, pages 545–554. IEEE Computer Society, 2001.

[40] J.A. Espinosa and E. Carmel. The effect of time separation on coordination costs in global software teams: A dyad model. In *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on*, pages 10–pp. IEEE, 2004.

[41] K. Feng, M. Lempert, Y. Tang, K. Tian, K. Cooper, and X. Franch. Defining project scenarios for the agile requirements engineering product-line development questionnaire. Technical report, Technical report, co-published as UTDCS-21-07 (UTD) and LSI-07-14-R (UPC), 2007.

[42] K. Feng, M. Lempert, Y. Tang, K. Tian, K. Cooper, and X. Franch. Developing a survey to collect expertise in agile product line requirements engineering. In *Inaugural International Research-in-Progress Workshop on Agile Software Engineering, Washington DC*. Citeseer, 2007.

[43] K. Feng, M. Lempert, Y. Tang, K. Tian, K. Cooper, and X. Franch. Developing a survey to collect expertise in agile product line requirements engineering processes. Technical report, Technical report, copublished as UTDCS-18-07 (UTD) and LSI-07-15-R (UPC), 2007.

[44] J. Fogarty, S.E. Hudson, C.G. Atkeson, D. Avrahami, J. Forlizzi, S. Kiesler, J.C. Lee, and J. Yang. Predicting human interruptibility with sensors. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 12(1):119–146, 2005.

[45] M. Fowler and K. Beck. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[46] J.E. Galegher, R.E. Kraut, and C.E. Egido. *Intellectual teamwork: Social and technological foundations of cooperative work*. Lawrence Erlbaum Associates, Inc, 1990.

[47] N. Ganesh and S. Thangasamy. Issues identified in the software process due to barriers found during eliciting requirements on agile software projects: Insights from india. *International Journal of Computer Applications*, 16(5):7–12, 2011.

[48] R.E. Grinter, J.D. Herbsleb, and D.E. Perry. The geography of coordination: dealing with distance in r&d work. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work*, pages 306–315. ACM, 1999.

[49] D.C. Gumm. Distribution dimensions in software development projects: a taxonomy. *Software, IEEE*, 23(5):45–51, 2006.

[50] C. Gutwin and S. Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)*, 11(3):411–446, 2002.

[51] C. Gutwin, R. Penner, and K. Schneider. Group awareness in distributed software development. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 72–81. ACM, 2004.

[52] J.D. Herbsleb and R.E. Grinter. Architectures, coordination, and distance: Conway's law and beyond. *Software, IEEE*, 16(5):63–70, 1999.

[53] J.D. Herbsleb and R.E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the 21st international conference on Software engineering*, pages 85–95. ACM, 1999.

[54] J.D. Herbsleb and A. Mockus. An empirical study of speed and communication in globally distributed software development. *Software Engineering, IEEE Transactions on*, 29(6):481–494, 2003.

[55] J.D. Herbsleb and D. Moitra. Global software development. *Software, IEEE*, 18(2):16–20, 2001.

[56] J. Highsmith. *Agile software development ecosystems*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[57] C.W. Ho, M.J. Johnson, L. Williams, and E.M. Maximilien. On agile performance requirements specification and testing. In *Agile Conference, 2006*, pages 6–pp. IEEE, 2006.

[58] E. Hochmüller. The requirements engineer as a liaison officer in agile software development. In *Proceedings of the 1st Workshop on Agile Requirements Engineering*, page 2. ACM, 2011.

[59] G. Hofstede and M.H. Bond. Hofstede's culture dimensions. *Journal of cross-cultural psychology*, 15(4):417–433, 1984.

[60] H. Holmstrom, E.O. Conchuir, J. Agerfalk, and B. Fitzgerald. Global software development challenges: A case study on temporal, geographical and socio-cultural distance. In *Global Software Engineering, 2006. ICGSE'06. International Conference on*, pages 3–11. Ieee, 2006.

[61] L. Jun, W. Qiuzhen, and G. Lin. Application of agile requirement engineering in modest-sized information systems development. In *Software Engineering (WCSE), 2010 Second World Congress on*, volume 2, pages 207–210. IEEE, 2010.

[62] S. Kelly. Towards an evolutionary framework for agile requirements elicitation. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 349–352. ACM, 2010.

[63] S. Kelly and F. Keenan. Investigating the suitability of agile methods for requirements development of home care systems. *Software Engineering and Applications*, page 890, 2010.

[64] L. Kiel. Experiences in distributed development: a case study. In *International Workshop on Global Software Development at ICSE*, pages 44–47. Citeseer, 2003.

[65] B. Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33:2004, 2004.

[66] H. Kniberg. Lean from the trenches. *The Pragmatic Bookshelf*, 2011.

[67] K. Kolehmainen. Agile requirements engineering: Building tool support for xp, 2003.

[68] B. Kovitz. Hidden skills that support phased and agile requirements engineering. *Requirements Engineering*, 8(2):135–141, 2003.

[69] C. Lee and L. Guadagno. Fluid: Echo–agile requirements authoring and traceability. In *Proceedings of the 2003 Midwest Software Engineering Conference*, pages 50–61. Citeseer, 2003.

[70] C. Lee, L. Guadagno, and X. Jia. An agile approach to capturing requirements and traceability. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE 2003)*, 2003.

[71] R. Macasaet, L. Chung, J.L. Garrido, M. Noguera, and M.L. Rodríguez. An agile requirements elicitation approach based on nfrs and business process models for micro-businesses. In *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, pages 50–56. ACM, 2011.

[72] N. Maiden. Exactly how are requirements written? *Software, IEEE*, 29(1):26–27, 2012.

[73] N. Maiden and S. Jones. Agile requirements can we have our cake and eat it too? *Software, IEEE*, 27(3):87–88, 2010.

[74] D. Mishra and A. Mishra. Managing requirements in market-driven software project: agile methods view. *Tehnicki Vjesnik*, 17(2):223–229, 2010.

[75] MFF Nasution and H.R. Weistroffer. Documentation in systems development: A significant criterion for project success. In *System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on*, pages 1–9. IEEE, 2009.

[76] J. Nawrocki, M. Jasiñski, B. Walter, and A. Wojciechowski. Extreme programming modified: embrace requirements engineering practices. In *Requirements Engineering, 2002. Proceedings. IEEE Joint International Conference on*, pages 303–310. IEEE, 2002.

[77] C.J. Neill and P.A. Laplante. Requirements engineering: the state of the practice. *Software, IEEE*, 20(6):40–45, 2003.

[78] J. Nowicki. Challenges of transitioning from traditional to agile requirements engineering: A case study. 2011.

[79] E. Ó Conchúir, H. Holmström Olsson, P.J. Ågerfalk, and B. Fitzgerald. Benefits of global software development: exploring the unexplored. *Software Process: Improvement and Practice*, 14(4):201–212, 2009.

[80] J.S. Olson and S. Teasley. Groupware in the wild: lessons learned from a year of virtual collocation. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 419–427. ACM, 1996.

[81] P. Onions and C. Patel. Enterprise soba: large-scale implementation of acceptance test driven story cards. In *Information Reuse & Integration, 2009. IRI'09. IEEE International Conference on*, pages 105–109. IEEE, 2009.

[82] K. Orr. Agile requirements: opportunity or oxymoron? *Software, IEEE*, 21(3):71–73, 2004.

[83] F. Paetsch, A. Eberlein, and F. Maurer. Requirements engineering and agile software development. In *Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003. WET ICE 2003. Proceedings. Twelfth IEEE International Workshops on*, pages 308–313. IEEE, 2003.

[84] D.L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[85] C. Patel and M. Ramachandran. Insert: An improved story card based requirement engineering practice for extreme programming. 2008.

[86] C. Patel and M. Ramachandran. Story card maturity model (smm): A process improvement framework for agile requirements engineering practices. *Journal of Software*, 4(5):422–435, 2009.

[87] C. Patel and M. Ramachandran. Story card process improvement framework for agile requirements. *Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization*, page 61, 2009.

[88] J. Peeters. Agile security requirements engineering. In *Symposium on Requirements Engineering for Information Security*, 2005.

[89] M. Pichler, H. Rumetshofer, and W. Wahler. Agile requirements engineering for a social insurance for occupational risks organization: A case study. In *Requirements Engineering, 14th IEEE International Conference*, pages 251–256. IEEE, 2006.

[90] S. Powell, F. Keenan, and K. McDaid. Enhancing agile requirements elicitation with personas. *IADIS Int. Journal on Computer Science and Information Systems*, 2(1):82–95, 2007.

[91] Z. Racheva and M. Daneva. How do real options concepts fit in agile requirements engineering? In *Software Engineering Research, Management and Applications (SERA), 2010 Eighth ACIS International Conference on*, pages 231–238. IEEE, 2010.

[92] Z. Racheva, M. Daneva, and L. Buglione. Supporting the dynamic reprioritization of requirements in agile development of software products. In *Software Product Management, 2008. IWSPM'08. Second International Workshop on*, pages 49–58. IEEE, 2008.

[93] Z. Racheva, M. Daneva, and A. Herrmann. A conceptual model of client-driven agile requirements prioritization: results of a case study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, page 39. ACM, 2010.

[94] Z. Racheva, M. Daneva, K. Sikkel, R. Wieringa, and A. Herrmann. Do we know enough about requirements prioritization in agile projects: Insights from a case study. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 147–156. IEEE, 2010.

[95] B. Ramesh, L. Cao, and R. Baskerville. Agile requirements engineering practices and challenges: an empirical study. *Information Systems Journal*, 20(5):449–480, 2010.

[96] A. Read, N. Arreola, and R.O. Briggs. The role of the story master: A case study of the cognitive load of story management tasks. In *System Sciences (HICSS), 2011 44th Hawaii International Conference on*, pages 1–10. IEEE, 2011.

[97] K. Reed, E. Damiani, G. Gianini, and A. Colombo. Agile management of uncertain requirements via generalizations: a case study. In *Proceedings of the 2004 workshop on Quantitative techniques for software agile process*, pages 40–45. ACM, 2004.

[98] P. Rodríguez, A. Yagüe, P.P. Alarcón, and J. Garbajosa. Some findings concerning requirements in agile methodologies. *Product-Focused Software Process Improvement*, pages 171–184, 2009.

[99] S. Sahay. Global software alliances: the challenge of standardization. *Scandinavian Journal of Information Systems*, 15(1):3–21, 2003.

[100] K. Schmidt. The problem withawareness': Introductory remarks onawareness in cscw'. *Computer Supported Cooperative Work (CSCW)*, 11(3):285–298, 2002.

[101] K. Schwaber et al. Scrum development process. In *Proceedings of the Workshop on Business Object Design and Implementation at the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95)*, 1995.

[102] K. Schwaber and J. Sutherland. Scrum guide. *Scrum Alliance*, 2011.

[103] D. Seminar. Science of design: High-impact requirements for software-intensive systems. 2009.

[104] S. Soundararajan and J.D. Arthur. A soft-structured agile framework for larger scale systems development. In *Engineering of Computer Based Systems, 2009. ECBS 2009. 16th Annual IEEE International Conference and Workshop on the*, pages 187–195. IEEE, 2009.

[105] J. Suzuki and Y. Yamamoto. Leveraging distributed software development. *Computer*, 32(9):59–65, 1999.

[106] R.B. Svensson, S.Ö. Birgisson, C. Hedin, and B. Regnell. Agile requirements abstraction model–requirements engineering in a scrum environment. *Department of Computer Science, Lund University*, 2008.

[107] A. Syri. Tailoring cooperation support through mediators. In *ECSCW*, volume 97, pages 157–172, 1997.

[108] H. Takeuchi and I. Nonaka. The new new product development game. *Harvard business review*, 64(1):137–146, 1986.

[109] J.E. Tomayko. Engineering of unstable requirements using agile methods. In *International Workshop on Time-Constrained Requirements Engineering (TCRE02). Essen*, 2002.

[110] T.H. Um, N.H. Kim, D.H. Lee, and H.P. In. Requirements elicitation in the agile process with social network services, 2010.

[111] J. Van Maanen and A. Laurent. The flow of culture: some notes on globalization and the multinational corporation. *Organization theory and the multinational corporation*, pages 275–312, 1993.

[112] K. Vlaanderen, S. Brinkkemper, T. Cheng, and S. Jansen. Case study report: Agile product management at planon. 2009.

[113] T. Weakland. 2005 global it outsourcing study, 2005.

[114] N.M. Wilkinson. *Using CRC cards: an informal approach to object-oriented development*. Number 6. Cambridge University Press, 1998.

[115] L. Williams and R.R. Kessler. *Pair programming illuminated*. Addison-Wesley Professional, 2003.

[116] www.versionone.com. State of agile survey, 2010.

[117] A. Yagüe, P. Rodríguez, and J. Garbajosa. Optimizing agile processes by early identification of hidden requirements. *Agile Processes in Software Engineering and Extreme Programming*, pages 180–185, 2009.

[118] R.K. Yin. *Case study research: Design and methods*, volume 5. Sage Publications, Incorporated, 1994.

Appendix A

Literature study

Table A.1: Round 1: Selected papers

Title	Year	Reference
A conceptual model of client-driven agile requirements prioritization: results of a case study	2010	[93]
Agile management of uncertain requirements via generalizations: a case study	2004	[97]
Agile Requirements Can We Have Our Cake and Eat It Too?	2010	[73]
Agile requirements engineering for a social insurance for occupational risks organization: A case study	2006	[89]
Agile requirements prioritization: what happens in practice and what is described in literature	2011	[10]
Agile security requirements engineering	2005	[88]
Case Study Report: Agile Product Management at Planon	2009	[112]
Challenges of Transitioning from Traditional to Agile Requirements Engineering: A Case Study	2011	[78]
Communication in Context: A Stimulus-Response Account of Agile Team Interactions	2010	[2]
Could an agile requirements analysis be automated?Lessons learned from the successful overhauling of an industrial automation system	2008	[8]
Defining Project Scenarios for the Agile Requirements Engineering Product-line Development Questionnaire	2007	[41]
Developing a survey to collect expertise in agile product line requirements engineering	2007	[42]
Developing a Survey to Collect Expertise in Agile Product Line Requirements Engineering Processes	2007	[43]
Do We Know Enough about Requirements Prioritization in Agile Projects: Insights from a Case Study	2010	[94]
Documentation in systems development: A significant Criterion for Project Success	2009	[75]
Extending XP practices to support security requirements engineering	2006	[19]
How Do Real Options Concepts Fit in Agile Requirements Engineering?	2010	[91]
Investigating the Suitability of Agile Methods for Requirements Development of Home Care Systems	2010	[63]
Issues identified in the software process due to barriers found during eliciting requirements on agile software projects: Insights from India	2011	[47]
Key factors of a successful agile requirements analysis in the realm of overhauling an industrial automation system	2008	[9]
Management challenges to implementing agile processes in traditional development organizations	2005	[9]
On agile performance requirements specification and testing	2006	[57]
Requirements Elicitation in Agile Processes	2010	[5]
Requirements Engineerings Role in Agile Development	2008	[25]
RTDWD: real-time distributed wideband-delphi for user stories estimation	2007	[4]
Science of Design: High-Impact Requirements for Software-Intensive Systems	2009	[103]
Supporting the Dynamic Reprioritization of Requirements in Agile Development of Software Products	2008	[92]
The Role of the Story Master: A Case Study of the Cognitive Load of Story Management Tasks	2011	[96]
A case study on benefits and side-effects of agile practices in large-scale requirements engineering	2011	[14]
A soft-structured agile framework for larger scale systems development	2009	[104]
Agile Methodology in Requirements Engineering	2008	[18]
Agile Requirements Abstraction Model–Requirements Engineering in a Scrum Environment	2008	[106]
Agile requirements definition: A view from requirements engineering	2002	[37]
Agile requirements engineering practices and challenges: an empirical study	2010	[95]
Agile requirements engineering practices: An empirical study	2008	[21]
Agile Requirements Engineering: Building tool support for XP	2003	[67]
Agile requirements: opportunity or oxymoron?	2004	[82]
An agile approach to capturing requirements and traceability	2003	[70]
An agile requirements elicitation approach based on NFRs and business process models for micro-businesses	2011	[71]
Application of Agile Requirement Engineering in Modest-Sized Information Systems Development	2010	[61]
Communication patterns of agile requirements engineering	2011	[1]
Cultural Aspects of Global Requirements Engineering: An Empirical Chinese-German Case Study	2009	[20]
Engineering of unstable requirements using agile methods	2002	[109]
Enhancing Agile Requirements Elicitation with Personas	2007	[90]
Enterprise SoBA: large-scale implementation of acceptance test driven story cards	2009	[81]
Exactly How Are Requirements Written?	2012	[72]
Extreme programming modified: embrace requirements engineering practices	2002	[76]
FLUID: Echo–Agile Requirements Authoring and Traceability	2003	[69]
Goal sketching: An Agile Approach to Clarifying Requirements	2009	[17]
Goal sketching: Towards agile requirements engineering	2007	[16]
Hidden skills that support phased and agile requirements engineering	2003	[68]
INSERT: An Improved Story card Based Requirement Engineering Practice for Extreme Programming	2008	[85]
Managing requirements in market-driven software Project: agile methods view	2010	[74]
Optimizing Agile Processes by Early Identification of Hidden Requirements	2009	[117]
Requirements Elicitation in the Agile Process with Social Network Services	2010	[110]
Requirements engineering and agile software development	2003	[83]
Requirements engineering: the state of the practice	2003	[77]
Some findings concerning requirements in Agile methodologies	2009	[98]
Story Card Based Agile Software Development	2009	[26]
Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices	2009	[86]
Story Card Process Improvement Framework for Agile Requirements	2009	[87]
The requirements engineer as a liaison officer in agile software development	2011	[58]
Towards an aspect-oriented agile requirements approach	2005	[7]
Towards an evolutionary framework for agile requirements elicitation	2010	[62]

Table A.2: Round 2: Selected papers

Title	Year	Reference
A case study on benefits and side-effects of agile practices in large-scale requirements engineering	2011	[14]
A soft-structured agile framework for larger scale systems development	2009	[104]
Agile Methodology in Requirements Engineering	2008	[18]
Agile Requirements Abstraction Model–Requirements Engineering in a Scrum Environment	2008	[106]
Agile requirements definition: A view from requirements engineering	2002	[37]
Agile requirements engineering practices and challenges: an empirical study	2010	[95]
Agile requirements engineering practices: An empirical study	2008	[21]
Agile Requirements Engineering: Building tool support for XP	2003	[67]
Agile requirements: opportunity or oxymoron?	2004	[82]
An agile approach to capturing requirements and traceability	2003	[70]
An agile requirements elicitation approach based on NFRs and business process models for micro-businesses	2011	[71]
Application of Agile Requirement Engineering in Modest-Sized Information Systems Development	2010	[61]
Communication patterns of agile requirements engineering	2011	[1]
Cultural Aspects of Global Requirements Engineering: An Empirical Chinese-German Case Study	2009	[20]
Engineering of unstable requirements using agile methods	2002	[109]
Enhancing Agile Requirements Elicitation with Personas	2007	[90]
Enterprise SoBA: large-scale implementation of acceptance test driven story cards	2009	[81]
Exactly How Are Requirements Written?	2012	[72]
Extreme programming modified: embrace requirements engineering practices	2002	[76]
FLUID: Echo–Agile Requirements Authoring and Traceability	2003	[69]
Goal sketching: An Agile Approach to Clarifying Requirements	2009	[17]
Goal sketching: Towards agile requirements engineering	2007	[16]
Hidden skills that support phased and agile requirements engineering	2003	[68]
INSERT: An Improved Story card Based Requirement Engineering Practice for Extreme Programming	2008	[85]
Managing requirements in market-driven software Project: agile methods view	2010	[74]
Optimizing Agile Processes by Early Identification of Hidden Requirements	2009	[117]
Requirements Elicitation in the Agile Process with Social Network Services	2010	[110]
Requirements engineering and agile software development	2003	[83]
Requirements engineering: the state of the practice	2003	[77]
Some findings concerning requirements in Agile methodologies	2009	[98]
Story Card Based Agile Software Development	2009	[26]
Story Card Maturity Model (SMM): A Process Improvement Framework for Agile Requirements Engineering Practices	2009	[86]
Story Card Process Improvement Framework for Agile Requirements	2009	[87]
The requirements engineer as a liaison officer in agile software development	2011	[58]
Towards an aspect-oriented agile requirements approach	2005	[7]
Towards an evolutionary framework for agile requirements elicitation	2010	[62]

Appendix B

Questionnaire results

This appendix contains the complete results from the two questionnaires held. The first section contains the results from the first questionnaire held before implementation started. The second section contains the results from the second questionnaire. These questionnaires has been held in Dutch thus the results are also in Dutch.

B.1 Questionnaire 1

Table B.1: Questionnaire results – Discard

Discard				
Satisfaction	Transparency	Importance	Times executed	Times needed
3	3	3	once a project	once a project
4	3	4	once a day	once a week
4	1	5	once a week	once a week
1	1	4	once a project	once a project
1	1	3	once a week	once a week
1	1	3	once a month	once a month
2	2	4	once a project	once a project

Table B.2: Questionnaire results – Refine

Refine				
Satisfaction	Transparency	Importance	Times executed	Times needed
4	3	5	once a project	once a month
4	1	4	once a day	once a day
4	1	5	once a day	once a day
4	2	5	once a day	once a day
4	1	2	once a month	once a month
5	5	5	more than once a day	more than once a day
4	2	4	once a week	once a day

Table B.3: Questionnaire results – Split-up

Split-up				
Satisfaction	Transparency	Importance	Times executed	Times needed
4	4	4	once a project	once a month
4	1	4	once a week	once a week
4	1	5	once a day	once a day
4	3	3	once a day	once a day
5	5	5	once a month	once a week
4	3	3	once a week	once a week
4	3	4	once a week	once a day

Table B.4: Questionnaire results – Merge

Merge				
Satisfaction	Transparency	Importance	Times executed	Times needed
3	2	4	once a project	once a project
4	1	4	once a month	once a month
4	1	4	once a month	once a month
2	3	3	once a week	once a week
5	4	2	once a month	once a month
2	3	3	once a project	once a project
2	3	4	once a week	once a week

Table B.5: Questionnaire results – Reconfirm

Reconfirm				
Satisfaction	Transparency	Importance	Times executed	Times needed
2	3	3	once a project	once a week
5	1	1	once a project	once a project
4	1	4	more than once a day	once a day
2	1	4	once a day	once a day
4	1	2	once a week	once a month
2	1	4	once a month	once a month
2	3	3	once a week	once a day

On which projects within IHomer are your answers based primarily?

- Lukas, Stretchwrapp;
- Iris;
- Iris;
- QIPC;
- Iris;
- iris :).

How is the transparency concerning actions on user-stories currently handled?

- Withtin Fogbugz, division into area's / Kanban board;
- Set-up of Trello;
- In Trello you can see changes to all user stories in order. I use this to find out what happened;
- Word documents;
- Trello features a not so transparent way to show and oversee these actions. They are hidden under several mouse clicks;
- Trello.

How would you like to see the transparency concerning actions on user-stories handled?

- I am currently happy with the way user stories are defined, but within the project I work not enough cases are described as a user story, because it is more work (laziness of the programmers - including myself ;))

- Timeline on which the user stories and actions can be displayed
  - Conversations that lead to an action link to the corresponding user story;
  - Actors that work on a user story work link to the corresponding user story;
  - Projects link to the corresponding user story.
- Actions link to what the direct cause / reason of this action is;
  - Creating user stories can be supported with advice
  - Actions on user stories can be classified using for example INVEST
  - Creating, editing and searching user stories and the subsequent changes must be easily accessible
  - Possibility to be notified about changes to certain user stories or changes of a particular type (within a specific project)
- -
- Visually on the board with user stories. At a high level immediately see that a user story is added, removed (e.g. trash), or edit / merge / split.
- In combination with Iris

B.2 Questionnaire 2

Table B.6: Questionnaire results – Discard

Discard		
Satisfaction	Transparency	Importance
3	3	5
4	3	4
4	4	4
5	4	2
5	5	5

Table B.7: Questionnaire results – Refine

Refine		
Satisfaction	Transparency	Importance
3	3	3
4	4	4
4	4	4
4	4	5
4	4	5

Table B.8: Questionnaire results – Split-up

Split-up		
Satisfaction	Transparency	Importance
3	4	5
5	4	4
4	4	4
3	4	4
4	5	5

Table B.9: Questionnaire results – Merge

Merge		
Satisfaction	Transparency	Importance
3	2	5
3	3	4
4	4	4
4	4	3
4	5	5

Table B.10: Questionnaire results – Reconfirm

Reconfirm		
Satisfaction	Transparency	Importance
3	3	1
4	4	2
3	3	3
5	3	2
5	5	3

How would you enhance the representation of the current requirements?

- The layout deserves some attention but that's probably because Iris overall deserves a makeover. It is very busy and cluttered
- The requirements are clearly displayed in a list. I would like to see more of the requirements than just the description.
- Identifiability of a requirement with one word would be a good improvement. Then when you are in a conversation it is easier to refer.
  - Mark discussed requirements better.
  - If a action has been coupled this has to be visible to the team-members.
- You could enhance the format of the requirements-list in the user interface of the backlog grooming session by printing keywords from the user-story in bold.
  - Moreover, the list is not clickable to their parents. Example, what is the epic with a particular user story? Maybe you can improve this by integrating the FogBugz graph functionality in to the list.

How would you enhance the visualization?

- The view is nice, but since most requirements start with the same text and only a short text is shown, it is difficult to find the 'right' requirement.

- The actions of the requirements are properly displayed, you have the ability to self-organize requirements so you can create a clear overview. Personally, I would appreciate it if it was immediately clear which items are currently open. (e.g. by displaying them at the same level)
- Perhaps use more color for the key actions. Identifiability of requirements would also have to come back in the visualization.
- Within the FogBugz graph it is very clear, however, in the requirement list it is not complete (as described in the previous point, I recommend a integration).
  - Something that I would improve is that currently discarding a user story and finishing a user story are represented by the same action. Some differentiation between these would be nice.

How good is the coupling between conversation fragments and the requirements? How would you enhance this coupling?

- Display when the selected requirement is changed, perhaps with a brief message in the chat. 'All comments from now on will be assigned to requirement: XYZ'
- At this time, all messages which are automatically related to the taken action coupled. I would appreciate it if it would be possible to make a selection of these messages, and possibly also comments on the action to place.
- Improved readability. But otherwise I'm happy here. I can find why this action was taken. There is important information in it that may not be clear directly from the requirement / user story.
- This is one of the things that do not work as well. The part of the conversation that took place while a particular user story was selected is properly attached to the tree, however:
  - Users still often talk about other user stories or more user stories simultaneously. It should therefore be made easier to edit or improve this afterwards, to ensure that the correct user story is selected during the conversation (no idea how you can reach the latter)
  - The interview portion is (partly due to 1) is often very long. I would add an option for users to summarize this (manually). Another option is by default the last x sentences as a summary record and to make an arrangement to always say the essence just before performing an action.
  - I would make the text not only accessible via a tooltip. I also want to be able to search and copy the text.
  - I would like to be able to see specific parts of the conversation that are linked to an action and jump to the entire text of the conversation where this part has focus to see the context in which it was discussed.

- To use all of this with audio/video conversation would off course be great.

What is your opinion about the support within Iris?

- You can do everything from Iris, thus the support from within Iris is good. Sometimes some things go wrong, like the order of messages or messages that disappear for a short period of time.
- Personally, I think this concept contributes to the overview within a project team.
- Ok. Going in the right direction. We can easily talk about the requirements because everyone immediately sees what requirement we have. And the requirements are updated immediately after a decision about them is made. And the reason why is also saved.
- Fun! I think it's a very good idea and the limitations are to a large extent related to the limitations of Iris itself because it is a tool that is still under development. I think it's a very good idea to make the evolution of requirements / user stories traceable / understood and I think people are more involved in the grooming sessions because of this insight.

What would you change about the support within Iris?

- Better layout, notifications when selected requirement is changed.
- The overview of the requirements needs to get a more prominent role within the groups;
  - It should be made clear what action is currently being taken and by whom.
  - The visual representation can be improved by clearly indicating which items are currently still open
- see "How would you enhance the representation of the current requirements?" and "How would you enhance the visualization?"
- Next to the improvements I have discussed at the other questions the only thing I can add is the ability to add new user stories from Iris.