Usability Requirements in Agile Development Processes

Lars Lynge Nielsen

Kongens Lyngby 2011

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

Abstract

Agile development process, as most other software developments, is focused basically on how to move from requirements to the final system. However, the focus is on technical and functional requirements not on end-user interaction, usability is usually only dealt with on the side.

The goal of this thesis is to examine how usability can be integrated more tightly into agile software development processes. The examination is conducted by adapting existing usability and interaction design methods and by introducing new ideas into an agile software development process.

First, the problem is analyzed theoretically by a literature study. Based on the study I develop a method for integrating usability requirements and developing usability requirement artifact. This method includes means for specifying and testing the requirements.

Second, the resulting process will be evaluated in a practical case study. In the case study a prototype of a new version of the EASEWASTE software with the focus on usability of the new version is implemented.

Resumé

Agile udviklingsproces, som de fleste andre software udviklingsmetoder, er dybest set fokuseret på , hvordan man bevæger sig fra specifikationskravene til det endelige system. Fokus er på de tekniske og funktionelle krav, ikke på slutbrugerens interaktion, og brugervenlighed er normalt kun klaret ved siden af.

Målet med dette eksamensprojekt er at undersøge, hvordan brugervenlighed kan integreres mere stramt i agile udviklingsprocesser. Undersøgelsen er foretaget ved at tilpasse eksisterende usability og interaktionsdesign metoder og ved at indføre nye ideer ind i en agil software udviklingsproces.

For det første er problemet analyseret teoretisk ved et litteraturstudie. Baseret på dette udvikler jeg en metode til at integrere usability krav samt udvikle usability krav artefakt. Metoden omfatter midler til at specificere og teste kravene.

For det andet vil den resulterende proces blive evalueret i et praktisk case study. I denne case study bliver en prototype af en ny version af EASEWASTE software med fokus på brugervenlighed af den nye version implementeret.

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling, IMM, at The Technical University of Denmark, DTU, in partial fulfilment of the requirements for the degree of Master of Science in Engineering, M. Sc. Eng.

The thesis has been prepared in the period from January to June 2011.

Contents

Abstract ..... **i**

Resumé ..... **iii**

Preface ..... **v**

Contents ..... **vii**

1 Introduction **1**

1.1 Overview 2

1.2 Prerequisites 2

2 Agile software development **3**

2.1 Agile background 3

2.2 Agile methodologies 7

2.2.1 Discussion 9

2.3 Agile life-cycle 9

2.4 Agile practices and methods 10

2.4.1 Artefacts in agile development 12

3 Interaction design **13**

3.1 Interaction design background 13

3.2 Interaction design life-cycle 14

3.3 Interaction design practices and methods 15

4 Combining agile development and interaction design 19

4.1 Related work 19

4.2 Bridging usability requirements with interaction design 21

4.2.1 Summary of characteristics 24

5 Case study 27

5.1 The project 27

5.1.1 EASEWASTE 27

5.2 The approach in action 29

5.2.1 Process overview 30

5.2.2 Roles 31

5.2.3 Phases 32

5.3 Evaluation 35

5.3.1 Pros 36

5.3.2 Cons 36

6 Conclusion 39

References 45

Appendix A - User stories 47

Appendix B – Artefacts 48

Appendix C – User interface 50

Appendix D – EASEWASTE guide 53

1 Introduction

Software development is about creation of a software product. It covers the process from conceptualisation of the product to the final release of the software and the activities in between. Several models for streamlining the development process have been conceptualised throughout the years from the traditional waterfall model to more iterative and incremental development processes and recently to agile development.

Agile methodologies were created by practitioners in the software industry and the focus was on empowering software developers to continuously deliver software that meets customer needs. As it were practitioners in the software industry who created these methods it meant that focus was mainly on technical aspects of developing software and as such did not focus as much on non-functional aspects of software development such as usability. This means that agile processes are at high risk not to sufficiently address usability concerns. The main focus of agile processes is how to organize the required task to reach the overall goal of delivering working software. Delivering working software is obviously a mandatory condition for any usable system. Though agile development focus on making coding more efficient and involving customers in the process it does not mean that customer involvement creates a more usable product since focus is on functional requirements and usability issues can potentially be ignored since an explicit usability focus is lacking.

On the opposite, the field of interaction design has been pursuing the development of usable software products for a long time. Interaction design offers techniques such as prototyping or usability inspection which may be applied for the aim of producing usable software.

Agile focus on how the software should be developed and interaction design focus on how the user will work with the software through the user interface and should hypothetically be a good match to ensure usable software. However, though some have tried in different ways to combine agile processes and interaction design, they have not yet been combined pragmatically or seen combined as an acknowledged and standardised new method for software development.

Considering this the main objectives of this thesis are to:

- examine how usability can be integrated more tightly into agile software development processes.
- adapt existing usability and interaction design methods to work in an agile context as a new pragmatically focused method for software development and developers.
- test and evaluate the proposed development process on a case study where a working version of the EASEWASTE application is developed.

1.1 Overview

Chapter 2 gives background information on agile including its origins. Agile methodologies are discussed and the two methodologies Extreme Programming (XP) and Scrum are presented in more depth. Following the agile-lifecycle is described and finally agile practices and methods are presented.

Chapter 3 introduces interaction design by giving background information on interaction design. The interaction design life-cycle is described and finally practices and methods are presented.

Chapter 4 gives a literature overview of existing methods for combining agile development and usability. The existing methods are discussed and a new method for combining agile methods with interaction design is presented.

Chapter 5 describes the case study in which the proposed method is used to create a new version of EASEWASTE and presents the evaluation of the method.

Chapter 6 summarises the findings and their relation to usability in agile development processes and a conclusion is given.

1.2 Prerequisites

It is assumed that the reader poses some basic knowledge on the concept of agile processes as well as general knowledge on software and development. Knowledge about interaction design and the concept of usability is an advantage but not a requirement.

2 Agile software development

This chapter introduces agile software development by establishing how it came about. The underlying values of the agile mentality are high-lighted in order to compare values of interaction design in a later chapter based on the assumption that it is important to know where they match the best when developing my new method. After that, a look at agile methodologies and the agile life-cycle are presented in order to be able to discover how and where interaction design might be integrated in the processes. Finally agile practices will be explained including the use of artefacts because they are crucial for selecting practices and techniques from interaction design afterwards. My main goal of this chapter is to establish and understand agile software development as a contextual framework in which I aim to integrate interaction design.

2.1 Agile background

The name “agile software” came about in 2001 when a group of process methodologists held a meeting to discuss future trends in software development. They noticed that their methods had many characteristics in common so they decided to name these processes agile, meaning it is both light and sufficient.

In a traditional software development process activities are performed in a sequence of separate steps where preferably each step is finished before the next one begins. A main characteristic of this approach is that the software is detailed up-front. The requirements are defined, the design of e.g. the user interface is documented and passed on to development followed by implementation, integration and delivery. This up-front style results in the need for a lot of documentation since each project phase must be signed-off before proceeding to the next phase [5]. This is typically known as the waterfall model which is illustrated in Figure 1a.

An alternative to the sequential approach to software development is the iterative and incremental model. Iterative development is an approach to software development in which the project is composed of several iterations in sequence. Each iteration being a self-contained mini project composed of all work activities (requirements analysis, design, programming and test) illustrated in Figure 1b. The goal for an iteration is to release an integrated, tested and partially completed but working system. Usually the partial system grows incrementally with new features iteration by iteration [5].

With these traditional approaches to software development it appears that an inefficient treatment of requirement can be identified and the issues are according to [13]:

- Lack of user input
- Incomplete requirements and specifications
- Changing requirements and specifications

The way I see it, it might relate to that traditional approaches to software development typical has limited customer interaction during the process of creating software and it is difficult to know the complete complexity of the software before you start to implement it, new requirements and problems will always emerge.

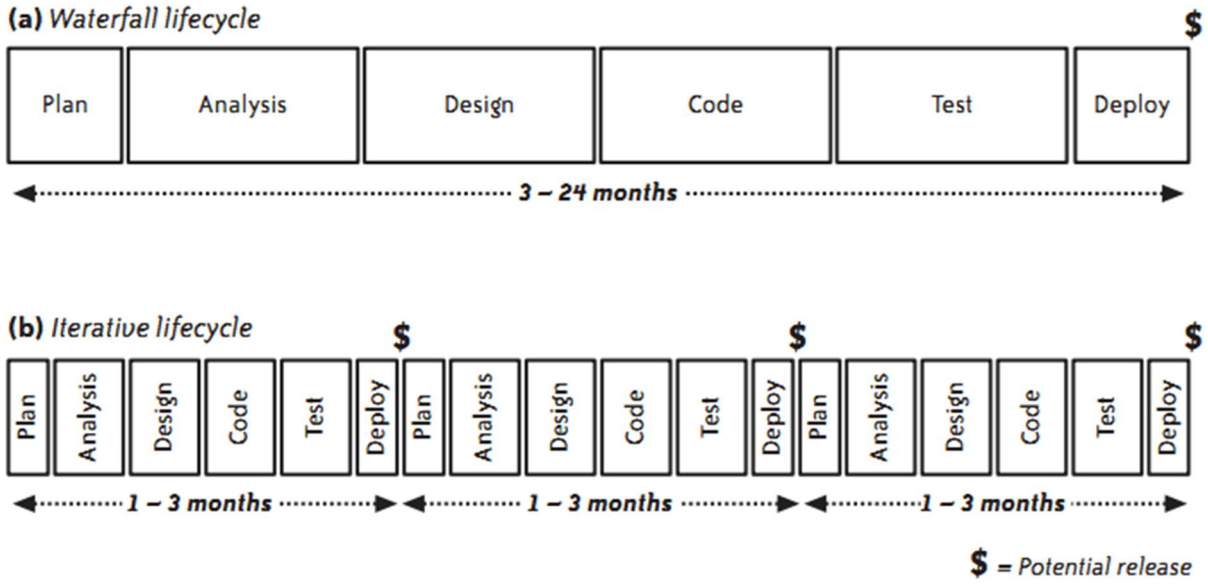


Figure 1: Traditional software development models [4]

Agile software development is a different way to develop software – compared to the traditional approach which is referred to as “heavyweight”, it can be described as “lightweight”. The one fundamental difference is that rather than assuming that the development process is predictable, agile methodologies assume that the process is

unpredictable. Agile is not a process or a method, but rather a philosophy, a movement covering different methodologies. The origin of agile makes it clear why it is a way of thinking [4].

In February 2001 the representatives from the different agile methods decided to form an Agile Software Development Alliance [12] to better promote their views and the result was the Agile Manifesto. Most of the agile techniques have been used by developers before the alliance but is not till after the alliance these techniques were grouped together into a workable framework [11].

The four key values of agile software development which constitute the Agile Manifesto [10] are described below. Each of the following values should be interpreted thus "while there is value in the items on the right, we give preference to the items on the left"

- ***Individuals and interactions*** *over processes and tools* [10]
  It is people, not processes and tools, who develop software. Therefore, each individual's skills and interpersonal communication are crucial – dialoguing face-to-face is the most effective way to communicate [11].
- ***Working software*** *over comprehensive documentation* [10]
  Documents showing requirements, analysis and design and other intermediate products are only relevant as means of communication during development. Although they can be very practical, they should only be worked on as long as they serve a purpose in delivering working and useful software [11].
- ***Customer collaboration*** *over contract negotiation* [10]
  This value describes the relationship between the people who want the software and those who are building the software. Instead of depending only on contracts, the customers work in close collaboration with the development team [11].
- ***Responding to change*** *over following a plan* [10]
  Plans are useful in software development, and each of the agile methodologies contains planning activities. The agile approach supports planning for and adapting to changing priorities [11].

Behind the manifesto itself are a set of core principles that serve as a common framework for all agile methods [10].

- *The 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 timescale.*
- *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.*

These four values and twelve principles constitute the Manifesto's definition of agile software development. However, in order to understand the essence of agile, this definition requires greater precision. Other definitions could help to achieve a broader understanding.

According to [5] agile methods can be defined as:

- **Iterative** – Delivers a full system initially, and then changes the functionality of each sub system upon each subsequent release.
- **Incremental** – The system, as specified in the requirements, is partitioned into smaller subsystems by functionality. New functionality is added upon each new release.
- **Self-organizing** – The team has the autonomy to organize itself in order to best complete the work items.
- **Emergent** – Technology and requirements are "allowed" to emerge through the product development cycle.

Different agile software development methodologies share these values and principles and follow them in their unique practices.

That is, these values and principles are the corner stones of agile methods in general and therefore necessarily influence the aims of my method development. Feedback might be the

most important criteria because it addresses functional challenges and errors and as a routine of the development process and relates to the most valuable dimension of how I expect usability to attribute: To create knowledge and deliver it as feedback about challenges and errors in interaction.

However, that values and principles are basically to be considered abstract. They give only guidelines but no actual recipe or method. Therefore you have to operationalize these guidelines into actual agile methodologies as I will do in the following section.

2.2 Agile methodologies

Agile methodologies share many common practices like iterative and evolutionary development and delivery, adaptive planning and emphasise open face-to-face communication and people before documentation, processes and tools [5]. They all embrace change by adapting to the situation rather than doing detailed predictive planning and locking down requirements [4].

Agile methods favour an iterative process characterized by a succession of incremental small releases containing a small set of features for that specific iteration [4]. This approach allows for changing software requirements discovered during the iterative process. Close customer collaboration during the iterative development process supports this flexibility.

Agile teams include an on-site customer representative that works with the team daily to give feedback and define requirements for the software [4]. This close collaboration allows the team to develop the software without needing a detailed written documentation up-front [4].

Agile teams are self-organizing and work as one. Problems are solved together regardless of roles thereby eliminating documents as the primary means of communication.

Two of the more popular agile methods being practiced today are extreme programing and Scrum or a combination of the two [15] and a brief description of these follow.

Extreme Programming (XP)

XP is a widely used agile software development method created by Kent Beck [1] that stresses “customer satisfaction through rapid creation of high-value software, skilful and sustainable software development techniques, and flexible responses to change” [5]. XP can be broken down into values, principles and practices.

XP promotes five values: Communication, simplicity, feedback, courage and respect [1]. It can be characterised by its five basic principles: rapid feedback, assuming simplicity, incremental

change, embracing change and doing quality work [1]. There is a considerable set of practices in XP and they are envisioned to work in synergy to give software teams some control and guidance over the work they do [1]. Some of these practices are:

- Planning game
- Small releases
- Simple design
- Incremental design
- Coding standard
- On-site customer

As the name indicates and the practices shows there is much focus on programming, but it also rely on communication and the whole team including customer working in a common project room. XP does not rely on detailed artefacts such as requirements specification documents, the preferred way of working with requirements and design is through face-to-face communication [1].

To build better software, XP also has certain roles that team members play. The customer's role is to document software requirements as user stories (a simple description of some functionality), prioritize these stories by business value and write acceptance test. Developers, that be programmer, tester and so on, work as a team and shares responsibilities for design and development tasks. In addition they are also responsible of making work estimations for the user stories [1].

Scrum

Scrum is an agile project management method and it can be combined with or used to compliment other methodologies [2]. Scrum and XP are often integrated and used together as a single comprehensive software development process.

Scrum teams work in an iterative manner during the software development. These iterations are called sprints and each day a short meeting, called daily Scrum, is held where the team assesses how it is doing [2].

The roles on a Scrum project are the Scrum master, product owner and Scrum team. The Scrum master is part time developer, part time leader that keeps the work on going and sees that the Scrum practices are followed and mediates between managements and the Scrum team. Product owner creates and maintains a product backlog, which is a list of features that are planned to be included in the project, and also chooses the features to be implemented for each sprint and finally reviewing the system at the end of each sprint. Team members are responsible for delivering potentially shippable software in every sprint [2].

2.2.1 Discussion

XP and scrum show two aspects of agile methods: While XP is a method with a technical focus and addresses techniques and practices used in software development, scrum is a method that establishes a framework for project management. What they have in common is that they both anchor responsibility to software developers. By responsibility in this case I mean ensuring progress and quality assurance and incorporating business aspects into software development. This focus on the software developers as central to both ensuring progress and quality assurance as well as integrating business aspects is also in focus when I develop my method combining agile software development processes and usability. That is, the software developer themselves should be able to operationalize an interdisciplinary approach software development.

2.3 Agile life-cycle

Agile methodologies follow an iterative and incremental life-cycle. The process is composed of iterations, which can be described as self-contained mini project, and releases. The project is divided into multiple releases that each have their own scope and schedule. A generalized agile life-cycle is illustrated in Figure 2 and described below.

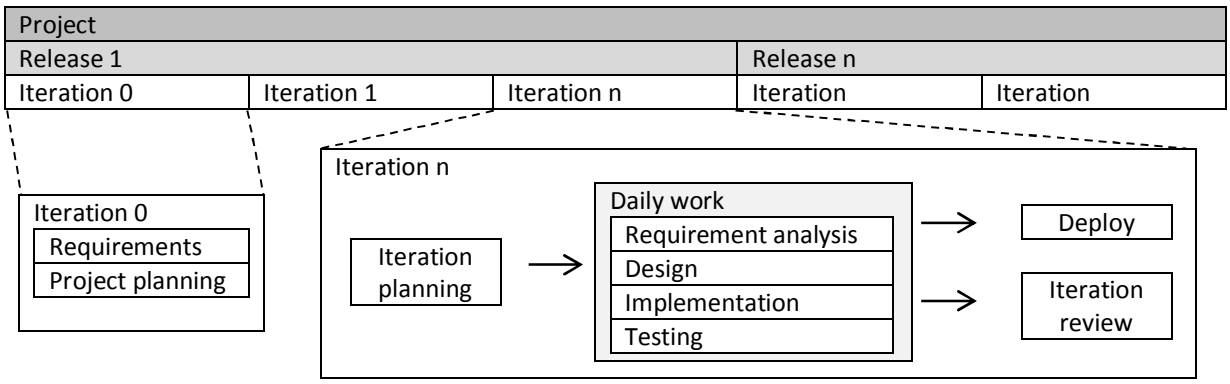


Figure 2: Agile life-cycle. Modified after [4] and [5]

The project starts with an iteration 0 that focuses on discovering initial requirements and project planning. Initial requirements are gathered in a backlog which is a list of estimated user stories and a vision for the project to which the whole team can commit [5].

Planning in an agile project is also iterative. At the beginning of each release, a release plan is created that defines the scope of the next operational release, with maximum value (to the customer) software. A release plan contains those user stories (chosen by the customer) that are going to be implemented in the release. Each iteration begins with planning meeting where the user stories to implement are chosen and the release plan is updated accordingly. All planning is done in collaboration by the whole team but it is the customer who decides what is included in the iteration based on the estimates of and the discussion with the developers [5].

Iterations include work in all of the activities necessary for software development. The team works with requirements, design, code and test every day to keep their software ready to deploy at the end of any iteration [4].

Iterations are timeboxed, deadlines cannot be exceeded even if that means dropping work that had been planned to be completed during the iteration. This is accepted since work is done in priority order, however, it leads to re-planning since the user story is important enough to be part of the iteration and has to be done at one point. Iterations last for a fixed length, during which the team works on those user stories agreed in the iteration planning. During iterations customers are not allowed to make changes to the current iteration only the team can add work if they find room for it [2]. At the end of each iteration the team delivers working, potentially shippable software in order to receive feedback from the customer [2].

2.4 Agile practices and methods

Agile software development methodologies include a number of different practices and techniques that define how requirements, design, development, testing and project management should be done. Many of the practices are shared between the different methodologies.

The most significant ones are presented in Table 1 and have been collected from various sources [1][2][3][4][5]. The table is divided into sections according to the five activities performed in software development. Each section has a number of agile practices and a description of the agile practice. When the practice requires an artefact to be produced is this mentioned as well.

Table 1: Agile practices and methods.

Practice	Description	Artefact
Project management		
Small releases	The development progresses in a series of short, timeboxed iterations, each producing new fully functional features and working software.	
Planning	Planning is done on several levels in agile projects and it usually means the process of creating, choosing elaborating the next cycle (product, release, iteration). Planning usually involves a backlog which is a documented list of prioritized story cards and their work estimate.	Backlogs: • Product • Release • Iteration Story cards
Sustainable pace	Work should be done at such a pace that can be sustained indefinitely.	
Self-organizing teams	During iterations, the team itself is responsible for fulfilling the planed goals and has the authority to plan and execute its work as it sees fit.	
Retrospective	Retrospectives are meetings held after iterations, releases and projects where the team reflects its experiences and decides on possible actions for improving the process.	
Daily meeting	Each day a short meeting is held where the team coordinates its work, synchronizes daily efforts and assesses and revise its plans.	
Requirement analysis		
On-site customer	The whole team work together in a common project room with an on-site customer. The on-site customer is expected to be subject matter experts and empowered to make decisions regarding requirements and priority. Having an on-site customer improves communication aiding to develop better quality requirements and reducing documented overhead.	Story cards
User stories	User stories are a requirements engineering tool in agile projects. They are short written descriptions of functionality used for planning and as a reminder, conversation to flesh out details of the functionality and test that documents details and determine when story is complete.	Story cards
Design		
Simple design	Simple design means that the team aims to have the simplest possible design to meet the functionality that the customer needs. This means that the team should eliminate duplication and avoid speculative design for possible future changes.	
Incremental design	Incremental design means that team invest a little in the design of the system every day as their understanding of the system increases.	
Metaphor	Metaphors aid design communication, capture the overall system or each subsystem with memorable metaphors to describe the key architectural themes.	
Programming		
Pair programming	Pair programming is a practice where production code is created by two developers at one computer in order to produce better quality code and share an understanding of the code. They rotate frequently and the observer is doing real time code review and thinking more broadly than the typist.	Program code
Refactoring	Refactoring is the continual effort to simplify and clean the code and larger design elements without changing functionality.	Program code
Collective code ownership	All programmers share responsibility for maintaining and the quality of the code so anyone on the team can improve any part of the system at any time as long as it is not out of scope for what that person is doing right now.	
Coding standards	The team agrees and follow the same coding style which improves maintainability and readability of the code.	
Testing		

Test-driven development	Unit test are written by the programmer before the code the passes the test. Programmers follow a rapid cycle of testing, coding, and refactoring when adding a feature.	Test code
Continuous integration	All checked-in code is automatically and continuously re-integrated and tested on a separate build machine. This helps the team to react to defects more quickly and be able to deploy at any time.	Automation scripts
Customer acceptance test	Acceptance tests are specified by the customer which will be used to know when a story has been correctly developed.	Story cards Test code
Iteration review	At the end of each iteration, a meeting is held where the results of the iteration is demonstrated in the form of working software. The goal is to share information, evaluate design and functionality of the software.	

As shown in the table, agile processes focus on the technical aspects of software development and organizing the overall structure of the project. It is obvious that usability is not considered at all. While the customer is part of the process nothing guarantees the focus on other aspects than the purely functional aspects of the software. By technical aspects I mean for instance code and system architecture. The user interface and usability as such is not central in itself. At the same time the end-user might differ from the costumer therefore this kind of user participation might not be sufficient for developing usable software.

2.4.1 Artefacts in agile development

Artefacts have long been used to facilitate collaboration in software design and development. They are used as a basis for organizing discussion and collaboration. Artefacts are a central part of design and analysis phases in traditional software development as opposed to agile development which to some extent tries to incorporate them in the development itself. Agile methods has a small set of predefined, informal artefacts, such as paper index cards for summarizing feature request called user stories. The most valued artefact in agile development is working software.

Agile methods are not restrictive in its use of artefacts and allow any other artefact of value to the project. As with all agile methods, non-code artefacts are expected to add real value, rather than be created for the sake of following a process formula.

I therefore expect artefacts to work as a way to capture issues related to usability because usability is cross-cutting and cannot be capture in a single user story.

3 Interaction design

This chapter introduces interaction design by describing how it originated and how it is related to usability. After that, a look at the interaction design life-cycle is presented. Finally interaction design practices and methods will briefly be explained.

3.1 Interaction design background

Interaction design as a field originates in experimental psychology and computer science and its roots can be traced to the 1970s. With the advent of monitors and personal computers, interface design came into being. Novel interfaces presented novel design challenges, and a new breed of designers emerged to accept it. The goal of Interaction design is to produce usable products that meet the users' needs [6].

Interaction design largely deals with activities related to improving usability. Usability is generally regarded as ensuring that products are easy to learn, effective to use and enjoyable form the user' perspective[6]. Nielsen [7] defines usability as a sub-attribute of usefulness (which is a part of the acceptability of a system) and divides it into five different properties that apply to a system:

- Learnability (easy to learn).
- Efficiency (efficient to use).
- Memorability (easy to remember how to use).
- Errors (users make few errors, good error handling so users can recover from any error).
- Satisfaction (pleasant to use).

In interaction design usability is not a separate feature or something that can be added to a product at the end. The usability of a product is a result of systematic work that starts at the beginning of the project.

3.2 Interaction design life-cycle

Interaction design work is not a one-time activity where the user interface is created before the release of a product, but a set of activities that are carried out throughout the life-cycle of the product's development [7].

The process of interaction design involves four basic activities [6].

- Identifying needs and establish requirements
- Developing alternative designs that meet those requirements.
- Building interactive version of the designs so that they can be communicated and assessed.
- Evaluating what is being built throughout the process.

In addition to the four basic activities of design, there are three key characteristics of the interaction design process.

- Early focus on users and task.
- Empirical measurement.
- Iterative design.

The model incorporates iteration and encourages a user focus. A simple interaction design model is illustrated in Figure 3

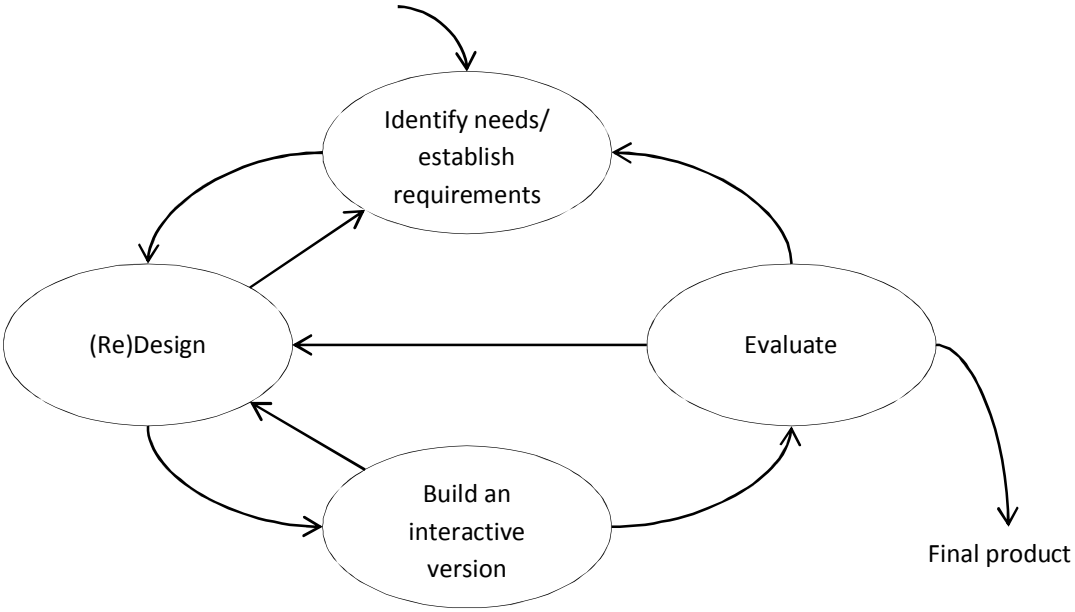


Figure 3: A simple interaction design model. Source [6]

The project starts with identifying needs and requirements. From this activity a design is created in an attempt to meet the needs and requirements that have been identified. Then an interactive version of the design is created and evaluated. Based on the feedback received from evaluations, the team may have to return to identifying needs and redefine requirements, or it might go straight into redesigning. Implicit in this cycle is that the final product will emerge in an evolutionary fashion from a rough initial idea through to the final product [6].

3.3 Interaction design practices and methods

Work in interaction design general consists of user research and the iterative design and evaluation of the product. A number of specified techniques can be used to accomplish different interaction design activities. Practices and techniques relevant to software development are presented in Table 2 and have been collected from various sources [6][7][8][9]. The table is divided into sections according to the activities performed in software development (project management, requirement analysis, design, programming and testing).

Each section includes a number of interaction design practices with a short description, list of different techniques that can be used and any artefact they produce beside documentation.

Table 2: Interaction design practices and methods.

Practice	Description	Techniques	Artefact
Project management			
-	-	-	-
Requirement analysis			
User research	The purpose of user research is to discover who the actual users are, what goals they have and in which context the product is used. The information is used to direct the design and development of the product to match the users' needs.	• Contextual interview • Focus group • Interviews • Observation • Site visit • Surveys	• Mental model • Personas • User profiles • User scenarios
Task analysis	Task analysis takes user research one step further by discovering and analysing specific tasks the user perform with the product to reach their goal.		• Flowcharts • Diagrams • Sitemaps • Use cases • User scenarios • User stories
Setting usability goals	The product should have an agreed set of qualitative and quantitative usability goals that can be used to measure how useful the product is.		
Design			
Conceptual design	Design at the conceptual level usually involves envisioning the proposed product, based on the users' needs and other requirements in a low-fidelity model at low level of details.	• Design patterns and guidelines • Parallel design • Participatory design • Prototyping	• Flowcharts • Diagrams • Use case • User scenarios • Wireframes
User interface design	Design at the detailed user interface level involves the drawing of high-fidelity UI mock-ups to model and describe the UI in detail. Functional prototypes can be built to analyse the design in practice.		• Functional prototypes • Mock-ups • Style guides • Wireframes
Programming			
-	-	-	-
Testing			
Expert evaluation	The goal in an expert evaluation is to identify usability problems through a review of the product conducted by a qualified specialist. Usually the review is made by evaluating the product against a list of design principles or heuristics.	• Cognitive walkthrough • Heuristic evaluation • Usability inspection	• Evaluation report
Usability testing	The goal of a usability test is to identify any usability problems by testing the product with representative users completing typical tasks and determine the user's satisfaction with the product.	• Interviews • Observation • Think aloud • Wizard of Oz	• Test report

As shown in the table, interaction design practices concentrates on research, producing design solutions and testing those solutions. Practices related to actual building the software are almost non-existent.

As mentioned in an earlier chapter agile software development processes deals especially with the technical and functional aspects such as code, system architecture and so on. Agile software development in itself does not address user participation in broader terms, only involving the customer. On the other hand user participation is absolutely central to interaction design. I consider three dimensions of interaction design to be relevant because they add most value to the work of software developers in order to operationalize interdisciplinary approaches themselves. By using these three dimensions of interaction design you can provide and apply important knowledge about the users that can be crucial to creating a software product that is in fact usable and not just functional:

- Who are the users: Understanding the actual needs of the users considering the scenarios in real life where they will be using the software. Thereby identifying main success criteria for the final solution.
- How do users use the software: Examining how users actual interact with the software through the user interface. Thereby discovering challenges and errors in usability.
- How to evaluate the software: Creating a methodical framework for analysing and reporting usability feedback in order to apply it in the agile software development process.

4 Combining agile development and interaction design

This chapter goes into details on how to combine agile development and interaction design. First existing methods are presented and discussed. Afterwards my method of combing agile development and interaction design is presented.

4.1 Related work

Agile methods and interaction design have in recent years received increased research focus on how these two methods integrates in software development. Several studies exist that examine various aspects of the integration of agile methods and interaction design. In the following I will present and discuss profiles within researchers and practitioners illustrating the different approaches to the subject.

Patton [18] discusses adapting interaction design into his agile development process. He presents a ten-step process that that has been added to their current development process to produce an agile usage centered design process. The process has similarities with interaction design approach by identifying user roles and user task initially. He does not claim that this process builds better software but they felt confident the resulting software would meet end-user expectation. What he does not mention is how this design phase fits into the complete life-cycle of the project. It seems that the approach more dictates implementation instead of evolving together as requirements emerge throughout the project.

Sy [16] describes a method which includes interaction designers, where interaction design and implementation work as two separate tracks to accommodate an agile approach to development. The interaction design track work one iteration ahead so it is able to feed the implementation track and when implementation is done it is tested by the interaction design track. Sy reports that this method produced better-designed products than the “waterfall” version of the same techniques. This approach of handing design over to developers who then turns the design in to working software and hands it back to interaction designers seems to contradict with agile idea of working close together to solve requirements to the system. A solution to this does not seem to be feasible when part of the team can be categorized as specialists compared to agile teams normally being categorized a generalists.

Kane [19] discusses that none of the main agile development methods explicitly incorporates interaction design practices, and suggest that by incorporating discount usability techniques as part of agile development it can improve the usability of the software. He stresses that there are still many issues that would have to be addressed to make the combination effective. It seems that these lightweight techniques can be a good match with agile methods as these techniques does not require any formal usability training and can thereby possibly easily be adapted to work in an agile context. Worth mentioning is the use of heuristic evaluation which can be adapted to work along the line of coding standard as it is prescribed in XP.

Wolkerstorfer et al [20] presents an approach to combine interaction design techniques (user studies, extreme personas, usability expert evaluations, usability test and automated usability evaluations) with XP to take advantages of both approaches. They have combined the advantages of extreme programming methodology (on-time delivering, optimized resource investment, short release cycles, working high quality software, tight customer integration) with the advantages of an interaction design process (usable, accessible, and accepted products, end-user integration).

This work shows the continuing efforts to combine agile methods and interaction design, it also shows the diversity on how the usability issue is addressed in agile methods.

Sy suggest to use interaction designer which means a change to the organization, Patton suggest to change the whole process, Kane and Wolkerstorfer both simply use techniques from interaction design in a an agile software development process.

As my ambition is to enable software developers to integrate interaction design in the agile software development process, organizational changes would cause a shift in focus from the interdisciplinary work of software developers themselves to the composition of an entire interdisciplinary team involving both software developers and interaction designers

Patton process is too rigid and fixed in certain steps and is therefore not compatible with agile processes as a whole. However certain elements can inspirational: first and foremost the

identification of user roles and user tasks initially. This provides a more solid and user centred foundation to build requirements on.

Kane and Wolkerstorfer seem to be the most useful inspiration for my methodology development because they provide specific and rather simple usability techniques that require no formal training and because their approach is likely to make the integration of agile software development and interaction design smooth. Unlike Sy their approach does not require organizational changes but can be operationalized in existing team organization thereby allowing continuously focus on interdisciplinary work performed by software developers instead of dividing functional and usability competencies.

4.2 Bridging usability requirements with interaction design

This section accounts for my approach to incorporate interaction design into agile methods based on my literature study so far. My aim is to develop a method that brings together the advantages of both agile and interaction design approaches to creating software that can be used in my following case study.

This method can be characterizes by incremental planning, short development cycles, evolutionary design and an ability to respond to changing business needs. The process itself builds around practices used in agile software development. As highlighted earlier this means small releases, planning, simple and incremental design, close collaboration with customer and refactoring enabling the development team to continuously receive feedback and if necessary change requirements as they go along. The method combines agile guidelines with practices used in interaction design including user research, task analysis, setting usability goals and usability evaluations. By usability evaluation I mean a conversation between software developer and customer about the user interface with focus on usability while trying to solve selected tasks with working software.

My method is designed to increase focus on usability in the daily work on several levels, both on an organizational level and on a technical level. By influencing both levels the responsibility of usability will be shared amongst the whole team including software developers. The increased focus on usability is done by introducing instruments found in interaction design and tweak them to work in an agile development context so the agility of agile methods are preserved. Two very important aspects of the method is that it is user story driven, user stories are the centre of everything, and face-to-face interaction driven.

Interaction design stresses that to create usable products you need to know your users and this should have an early focus. In this perspective doing user research early in the process inspired by Patton and Wolkerstorfer therefore establishes an essential understanding of who the users are and what their goals are. My methodology will be based on the assumption that this knowledge is an advantage when building requirements. Interviews, questionnaires and other quantitate or qualitative methods could be relevant to apply to my method if the scope and time period of my case study had been wider. However, the chosen technique for doing this is by observing users because I had exclusive access to a group of target users during an educational session. The case study is accounted for in the following chapter.

Interaction design's strength is analysing and provides feedback to the software development process about what users are trying to achieve in an existing situation and how they are going about the tasks. The information gathered from task analysis thereby establishes a foundation of existing practices on which to build new requirements and ensures an overall vision for the project.

I will incorporate the use of artefacts associated with both agile software development processes and interaction design as a way to capture and document issues related to usability: User profiles are traditionally used to create a basic understanding of the end-user whereas usability goals are used to highlight the users wanted achievements.

In my opinion a way to improve basis for decisions is to partly base them on usability goals as an artefact designed to highlight issues early in the process without using mock-ups or prototypes and instead placing the end-user in relation to the design. They can also be used as a measurement when evaluating the design.

My approach to improve usability during development is to use heuristic evaluation as discussed by Kane because it is designed to improve the usability of software by applying a small collection of rather simple usability principles to the design and development of software. This type of technique does not require any specialised usability training so it can be applied by developers on agile teams who is normally generalists.

Close collaboration between developer and customer as stressed in the typical XP practice makes it easy to incorporate usability evaluation on a daily basis. The customer possesses invaluable knowledge about business and should at the same time contribute by participating in the discussions about the user interface. The aim is that each user story goes through a usability evaluation as part for the process of validating user stories before they can be accepted as done.

The process follows an iterative and incremental life-cycle. The process is composed of iterations and releases. The project is divided into multiple releases that each have their own scope and schedule. It is illustrated in Figure 4 and is described below.

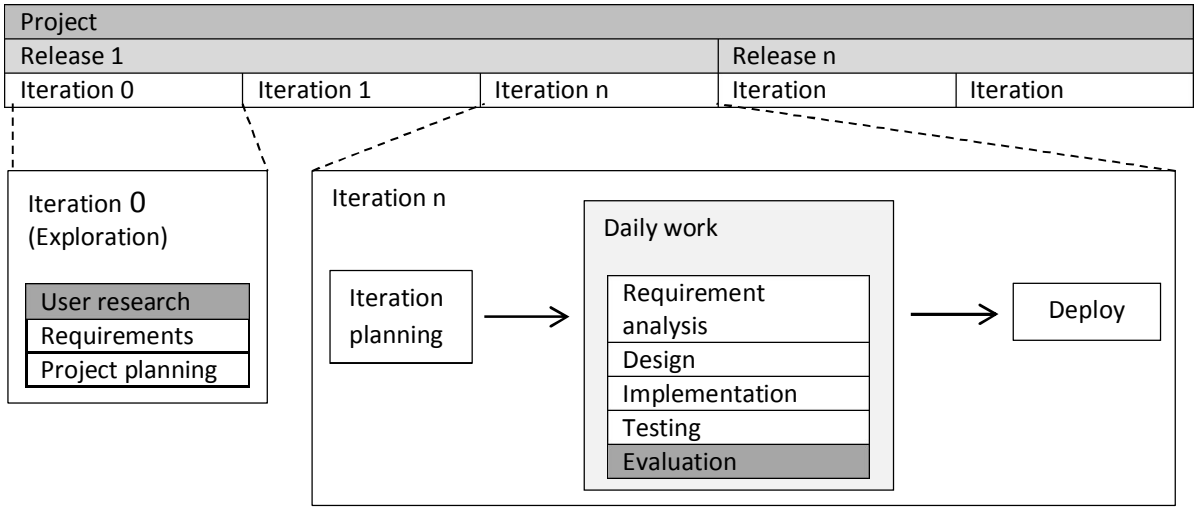


Figure 4: A future process. A modified version of **Figure 2**.

The project starts with a short iteration 0 meant for exploration iteration. The exploration phase is common in XP projects but do not clearly state how to ensure feasible knowledge and if users should be involved in this phase. Focus is on user research including task analysis and on discovering initial requirements and project planning. The outcome should be "just enough" user research and task analysis to inform the design in two ways: knowledge for creating some initial user profiles and usability goals and input to the creation of user stories.

Subsequent iterations begins with a planning meeting where the user stories to implement are chosen and prioritized by the customer – and thus steer the project - based on their latest priorities for the release which is a common practice in agile development. All planning is done in collaboration by the whole team but it is the customer who decides what is included in the iteration based on the estimated user stories and the discussion with the developer.

Iterations include work in all of the activities necessary for software development. The team works with requirements, design, code, test and evaluation every day to keep their software ready to deploy at the end of any iteration. There are two elements to the evaluation activity: Developers use heuristic evaluation during implementation and before a user story can be considered as done after it has been implemented, developer and customer perform a usability evaluation on the working software together.

Artefacts

Some new artefacts are introduced in the process to support a focus on usability when having system requirements as user stories. The used artefacts in the process are:

User profile: Some initial user profiles are created in in the exploration phase. They are used in the subsequent iterations to guide design solutions. When new requirements emerge that is not covered by current user profiles it can lead to two actions: the new knowledge can suggest slightly change to current profiles and they will be refactored or to the creation of a new profile.

Usability goals: Some initial usability goals are created in the exploration phase. They are used in the subsequent iterations to guide design solutions and to verify usability evaluations. When new knowledge emerges that is not covered by the current usability goals it leads to two outcomes: the usability goals are refactored or new ones are added.

Prototype: Simple prototypes are created during development for making quick designs decisions. Working product is used for evaluation of user stories during iterations.

User stories: User stories constitute the requirements for the system and are used from the beginning of the project and throughout the entire life-time of the project.

Organization and roles

The process relies on a customer being part of the whole team and participates in activities and closely collaborates with the rest of the team from the beginning of iteration 0.

Usability is a collective responsibility that is shared amongst the whole team. The customer has the vision for the project and steers the project in the desired direction where most business value is gained. The customer writes user stories and engages in a conversation with the developer about requirement details and finally performs usability evaluation. Software development is responsible for design and code, refactor, estimating user stories and supporting usability evaluations as this method is aimed at enabling software developers to integrate interaction design in agile software development processes.

Responsibilities for the evolution of the artefacts are also shared amongst the whole team and also for making sure that usability has focus in daily conversations when discussing requirement details.

4.2.1 Summary of characteristics

The most important differentiating characteristic between my method and those accounted for in the literature study is the use of usability evaluation per user story. The method uses this approach because I expect it to be easier to evaluate as the design of the user interface

evolves rather than after the complete user interface have been implemented. This approach is well known from interaction design when developing and testing prototypes. However, in agile software development processes one of the most important ideas is to have working software product while developing it. Thereby in my method the working software product acts as a prototype as well in order to provide feedback about usability issues.

Other characteristics worth mentioning includes user studies at the beginning of the project that is different to for instance Patton's approach because my method allows the artefacts representing the results of user studies to be refactored while developing the software in an agile process. This means that you can update the user studies whenever you get a cause to do and include new experience and requirements that is not already covered in existing user studies.

Another characteristic is integrating usability heuristic as a supplement to existing coding standards already used by XP practitioners. The technique is probably one of the easiest procedures to secure a minimum of usability in the development phase resulting in a more useful implementation of the software product in the end.

5 Case study

This chapter illustrates my experiences when trying to incorporate interaction design in agile software development. First the EASEWASTE project is described and its usability issues are discussed. Then my proposed approach and method is used to create a new version of EASEWATE. Finally my proposed approach and method is evaluated from the development of the new version of EASEWASTE.

5.1 The project

5.1.1 EASEWASTE

EASEWASTE stands for *Environmental Assessment of Solid Waste Systems and Technologies* and is a Life-cycle-assessment model for waste management developed at the Technical University of Denmark. EASEWASTE models resources use and recovery as well as environmental emissions associated with waste management in a life-cycle context [14].

The project involves a new version of EASEWASTE because several usability defects have been identified with the current version. One of the main issues with the current version is that it has a steep learning curve so it do take some time from users see the software for the first time till they can accomplish desired tasks. The main window is shown in Figure 5 and illustrates very well that from a usability point of view it can be difficult to figure out how to move from the idea of the system to creating the system and finally assess it.

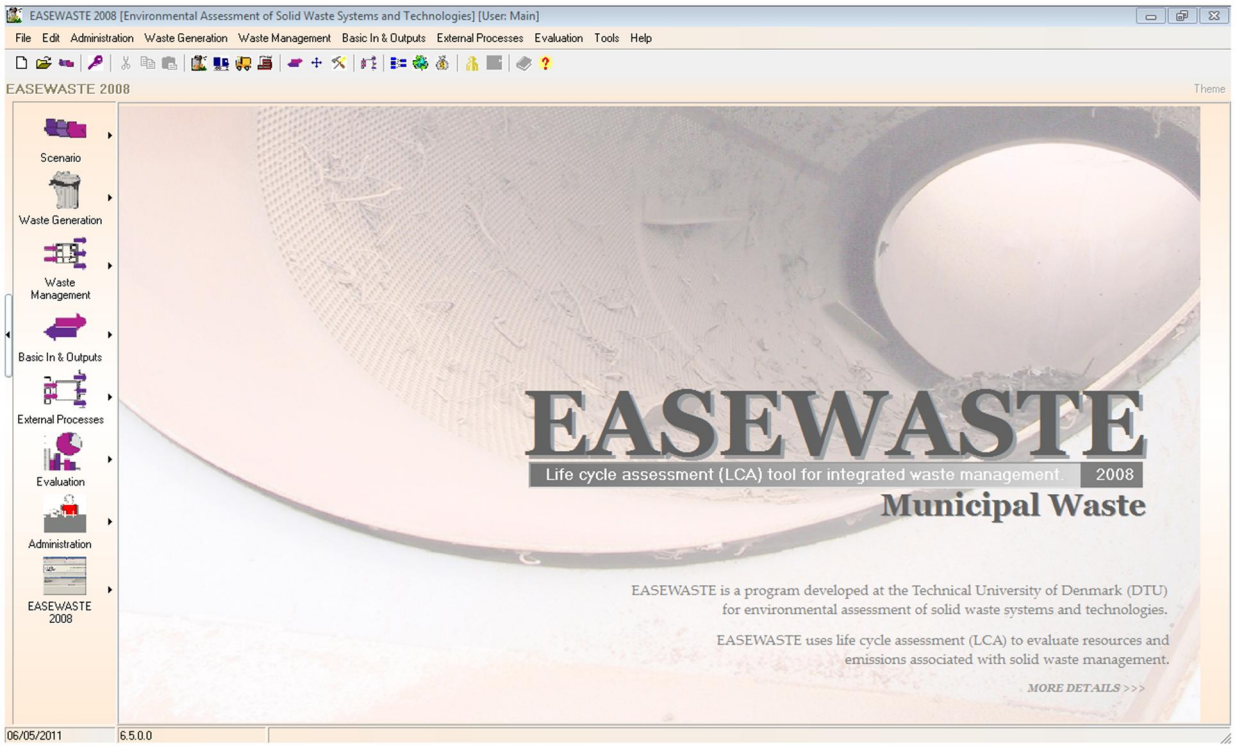


Figure 5: EASEWASTE main window.

The system the user wishes to assess is represented as a scenario in EASEWASTE and the scenario builder which is used to construct scenarios is illustrated in Figure 6. Users are confined to fill in fields instead of freely putting the system together. This lack of freedom when users create systems they desire to assess is another issue with the current version

The aesthetic and design of the UI is structured in such a way that it is very easy for the user to get lost or feel overwhelmed especially when using the scenario builder where users have to flip through windows when creating scenarios and dealing with popup windows when assessing the system they have constructed.

In an attempt to make the UI more efficient to use, the same functionality is presented both as a menu item and a button, sometimes even a third button is added to the UI. This clutters the UI, and instead of supporting the user, the UI confuses the user.

There are some constraints on the EASEWASTE development influencing scope and possibilities present for my case study:

- Development team consisted only of me:
    - The speed of development was limited.
- The timeframe for the project was tight:
    - It was realistic to aim for a part of a complete agile project lifecycle.

- Focus on only one release to be finished.
- Iterations only involved one user story.

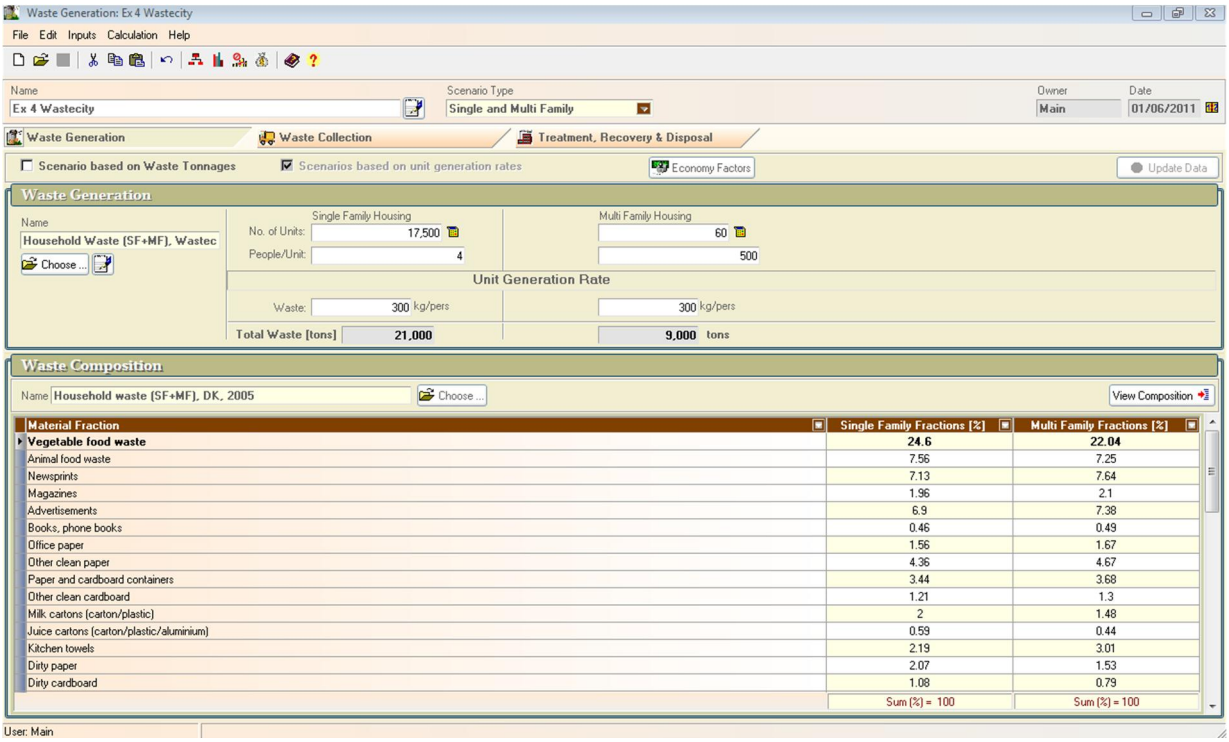


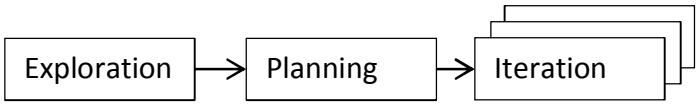
Figure 6: EASEWASTE scenario creation window.

The focus in the project will be on functionality and usability of the system. How the LCA model used in EASEWASTE works is given in ruby code and therefore does not have to be implemented.

5.2 The approach in action

This section summarizes how my proposed approach and method were used in practice by developing a new working version of EASEWASTE. It gives an overview of how my proposed approach and method were applied as a process, the applied roles and organization and finally how each phase was conducted.

5.2.1 Process overview



Exploration:

The purpose of the explorations phase was to get and understanding of the product and how users use the product. This was done by following a course on the introduction of EASEWASTE which taught the LCA model used in EASEWASTE. Though the customer is in this case an end-user, she was highly skilled and trained and therefore it was relevant to observe others trying to interact with EASEWASTE as well as learning to use it myself.

Planning:

The purpose of the planning phase was to have an overall plan for the next release. The release plan is a suitable amount of user stories that constitutes the customer's vision for the software and what they expect to see for the next release. This was done by having a meeting between developer and customer and then working together to complete user stories with estimates and then decide what to do for the next release. During this face-to-face communication process new requirements emerged which resulted in new user stories to be written and estimated.

Iteration:

The purpose of the iteration phase was to implement software in short cycles that lives up to the customer's expectation. This was done by having the customer pick stories – and thus steer the project - based on their latest priorities for the release. The system evolved one user story at a time and the requirement captured in the user story was first analysed by face-to-face communication between developer and customer to learn the requirement in details before implementation of the one user story started. Before the user story could be considered as being done developer and customer met face-to-face and conducted a usability investigation of the UI. Detected usability defects were discussed and if they were found to be small usability defects that were in the scope of the user story they were allowed to be refactored. Otherwise it required a new user story. The process then started over with the next user story.

In the process some additional artefacts beside users stories and code were produced which are not normally used in agile development. These artefacts including user stories were meant as a help to ensure the overall usability of the application and they were:

- User stories are the main artefact used to capture requirements in agile development and is used to feed continues development.
- User profile is a way of capturing different potential users of a system and in this context used as a representation for the end-user of the system.
- Usability goal is the instrument to keep focus on users' issues and desires during development especially when making decisions that have effect on the UI design.
- Visualisation of requirements regarding UI sketched as a sort of paper prototype as an addition to user stories enabling quick UI design decisions during implementation and working software for during usability evaluations.

No distinct usability requirement artefact was produced. However, the idea is that usability of the software is covered in the above artefacts and through usability evaluations of the software.

5.2.2 Roles

The roles involved in the case study process were customer and developer with responsibility for designates knowledge areas:

- **Customer**. The customer was the business representative of EASEWASTE that controlled the definitions and prioritization of user stories and had a detailed understanding of the business.
- **Developer**. The developer was me having to accept and implement user stories.

Even though we represented different areas we collaborated about finding the best solutions for the requirements of the system on a daily business as illustrated in Figure 7.

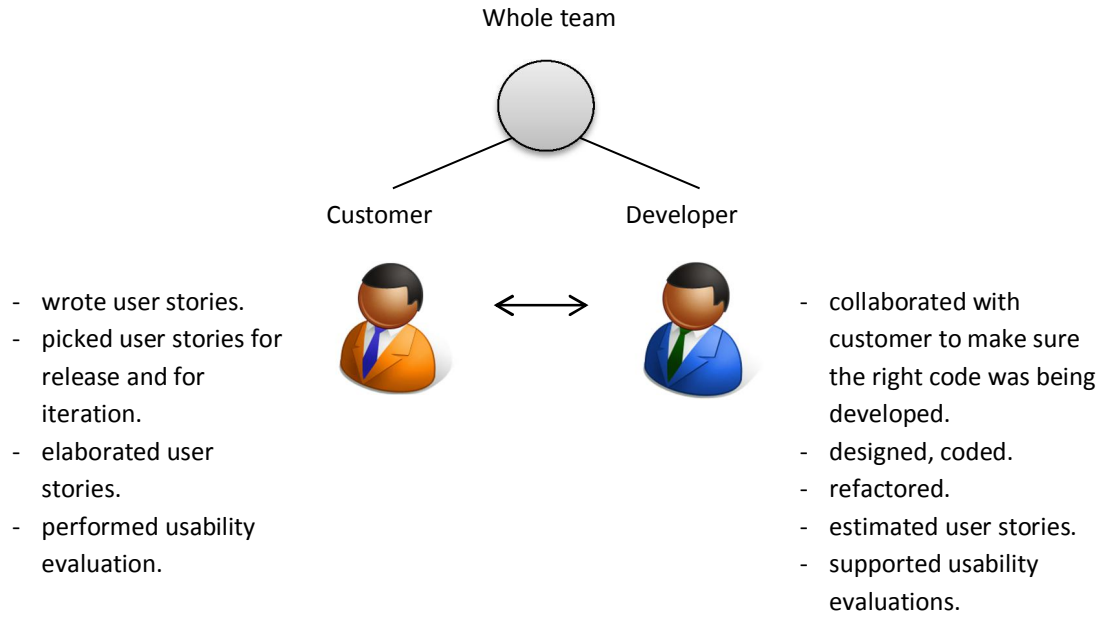
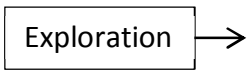


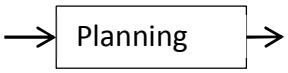
Figure 7: The whole team (customer and development) worked together to develop the system.

5.2.3 Phases

The different phases in the process varied in length and team involvement. The exploration phase took a day and only the developer took part of it. The planning phase was very short and duration was 3-4 hours and involved both customer and developer. The iteration duration was 5 weeks and involved both developer and customer.



At the beginning of the project I conducted a contextual observation to get a greater understanding of how users interact with the software to accomplish specific tasks. Then I did a task analysis from which I created some user profiles and usability goals. They can be found in Appendix B. These activities increased my knowledge about the system and the expectations to the system from which I created 14 initial requirements as user stories end established a vision for the software. These user stories are used to kick-start the planning phase which was the next phase.



We (developer and customer) sat face-to-face at a table with the 14 user stories in front of us. We picked up each user story and we had a discussion about the requirement for that specific user story. We realised during the discussion that not all user stories applied to the customer's vision of the system so we discarded those user stories. We created new user stories for parts of the system which was not already covered in the user stories. I as the developer estimated the user stories and split the ones that were too big into smaller ones. The initial 14 user stories turned into 15 user stories with estimation. Figure 8 shows one of these user stories and a complete list of all user stories can be found in Appendix A. We spend the last part of the meeting on planning the first release and we decided that it should constitute a workload of 5 weeks. The customer chose the user stories for the release based on her priorities and my guidance.

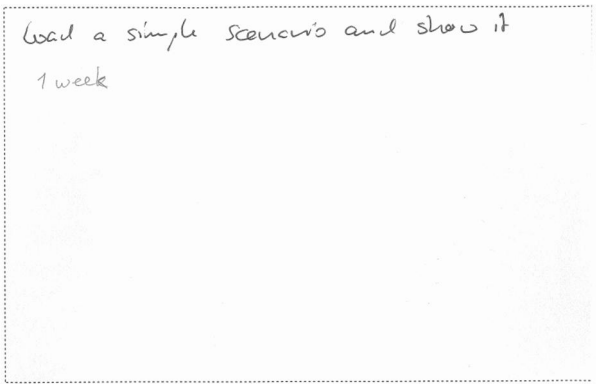
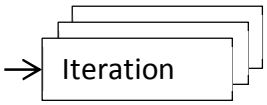


Figure 8: A user story with work estimate.



We (developer and customer) sat face-to-face and discussed the requirement details for the user story that I was about to implemented. Then I started to implement the user story. I made my implementation decisions by sketching the UI with pen and paper and consulting the user profile and the usability goal. I used heuristic evaluation to uphold a usability standard or to refactor the design if some was violated. I did not have style guide or other UI design guideline at my disposal. My design decisions were based on the user profiles and pre-established usability goals. The latest version of EASEWASTE can be seen in Figure 9.

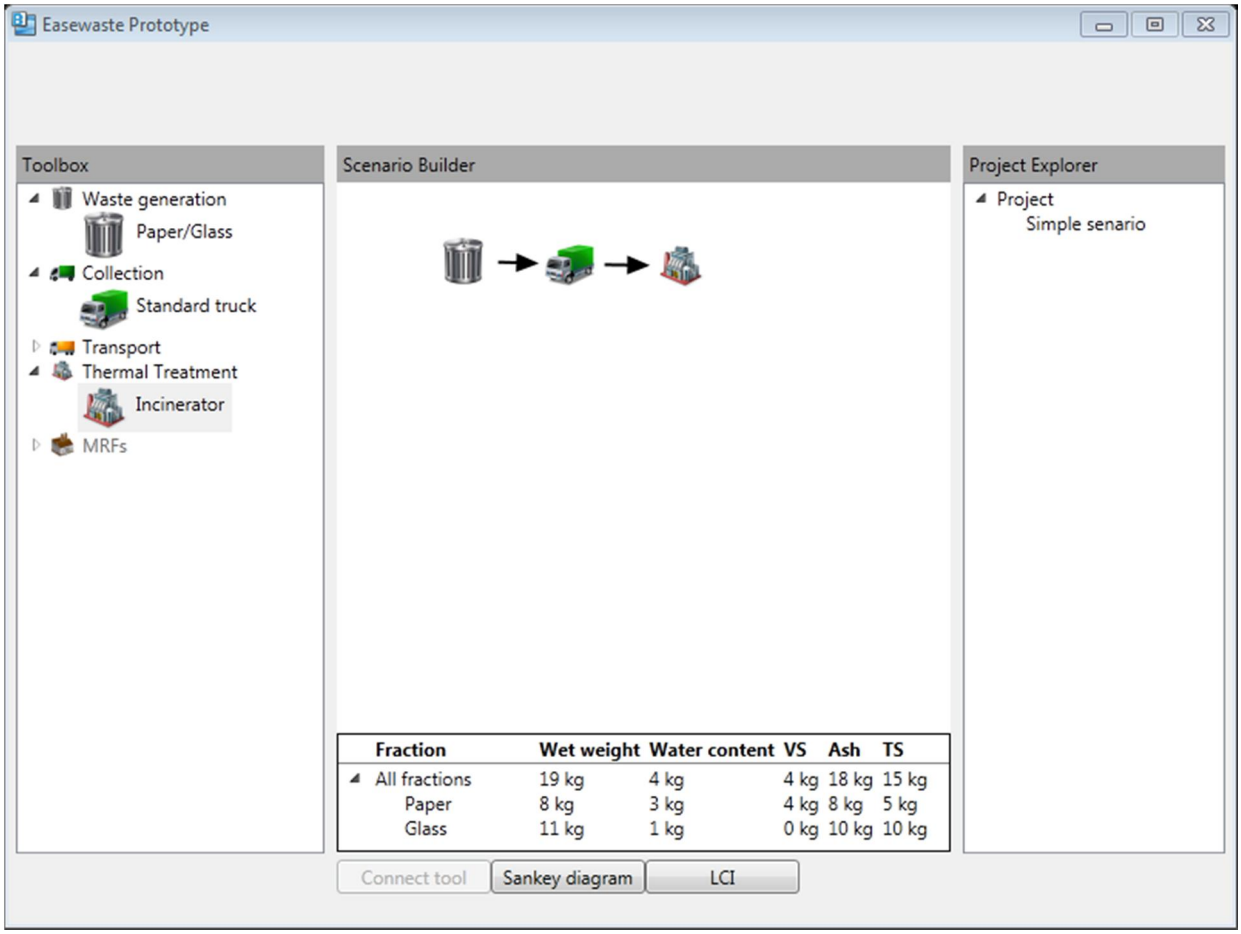


Figure 9: Latest version of EASEWASTE.

When I had finished implementing a user story, we (customer and developer) sat face-to-face and conducted a usability evaluation of the design with the working software. I refactored small usability defects and we created a new user story for large defects. After end evaluation the next user story was picked up and the process of discussing requirement details, implementation and evaluation started over. How part of the UI evolved over the process of implementing user stories and conducting usability evaluations can be seen in Figure 10.

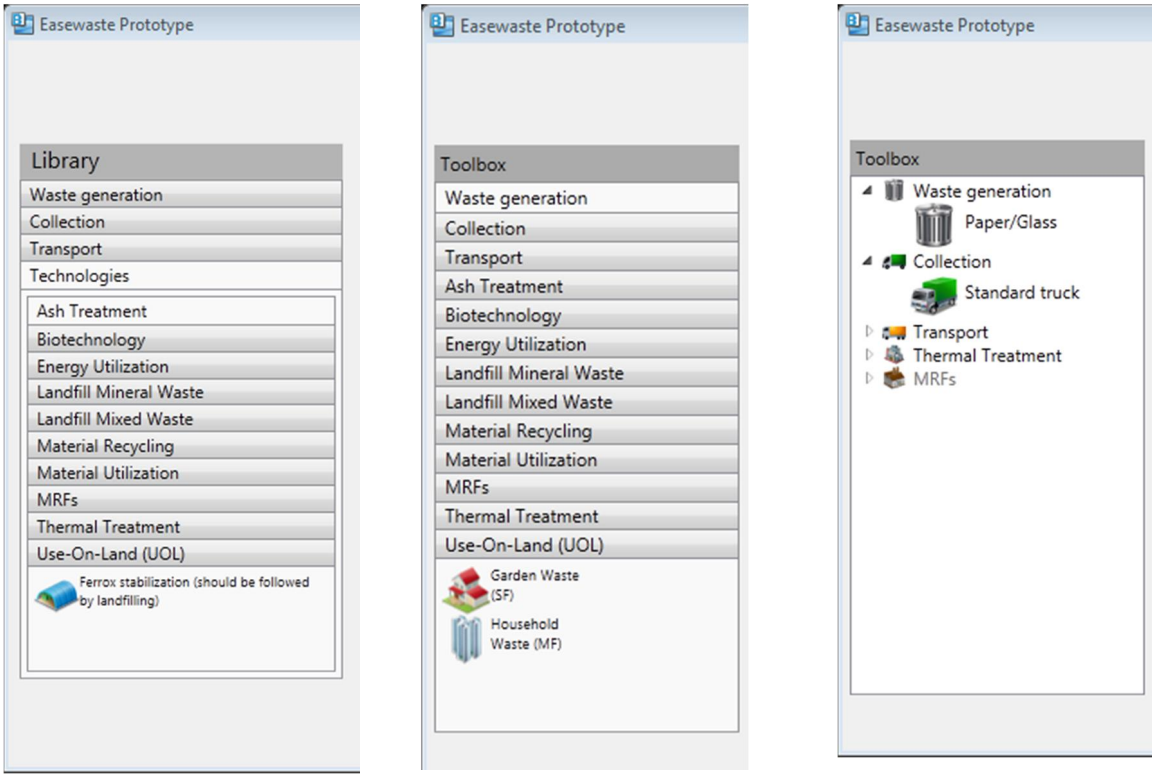


Figure 10: The user interface evolved over several use stories.

5.3 Evaluation

Evaluations are based on my experience from using the practices and techniques combining usability with agile software development processes during the case study project.

The process showed that the practice of user research and the usability evaluation of each user story are definitely something that adds value and should be considered part of the agile software development process.

First of all it helped focusing continuously on the user in the agile software development process.

Secondly it ensured continuously systematically evaluation of both functional and usability aspects throughout the process.

Thirdly the definition of method and process articulated and thereby anchored a mutual understanding of the importance of usability in any part of the agile development process.

All in all this provides a more solid and sufficient basis for decisions for the customer in collaboration with the software developers in order to release a software product that is in fact usable.

However, extensions and choice of techniques used for user research should be counterweighted with the typical agile processes quick and dynamical input and feedback. Extensive user studies such as observations as used in this case study can be time consuming and may impede the progress of the software development project causing it to be less agile.

I will elaborate pros and cons below.

5.3.1 Pros

Doing usability evaluation face-to-face to discover any usability challenges and defects using working software to establish if a user story could be considered done worked very well. It helped and supported the creation of an evolutionary UI design which suits very well with the agile approach aiming to have a system as close to production ready state as possible at all time.

Usability goals worked well to support decisions regarding UI design. Before making the final decision usability goals can be consulted to see if the decision is aligned or if the usability goals should be updated as a consequence.

Visualisation of requirements regarding user interface that resembles the format of simple prototypes, produced by pen and paper, worked well when trying to come up with different approaches for design solutions. Success rate can perhaps even be increased if they are used during the discussion between developer and customer about the details of a user story if it has any influence on the UI when it is about to be implemented.

Using heuristic evaluation during implementation was very suitable for making a consistent UI and can be seen as an extension to coding standard which means it is easy to operationalize for software developers without special training.

Creating user profiles and usability goals early in the project did help in making decisions regarding design of the UI. However, they should in fact not be set in stone, just like the understanding of the systems evolves throughout the project so does the understanding of user profiles and especially the usability goals.

5.3.2 Cons

Conducting user research by contextual observation in the exploration phase is very rewarding in regards to knowing different users and the tasks involved and in this case how the EASEWASTE model should be interpreted. However, it is also very time consuming in an agile

software development process before being able to set requirement and actually start implementing. One can argue that much of this knowledge could have come from the customer who in this case was a very good representative for the end user as well as she was trained and highly skilled.

Lack of focus on usability when discussing requirements in details before a user story was implemented might increase the number of usability defects found during usability evaluations. Perhaps if part of our discussion before implementing a user story also had focus on usability some usability defects would actually have been found before being implemented. This is an important lesson if I was to revise and reuse this method in the future as it stresses the need of focusing on usability in every part of the agile software development process.

6 Conclusion

This thesis focuses on finding ways for agile software developers and customers to integrate usability in agile software development processes by introducing practices and techniques from interaction design and using them to produce and test software requirements. Based on theories on agile methods, interaction design and related work I have proposed a method for integrating usability in an agile project life-cycle. It introduced early user focus by doing user research at the beginning of the project and introduced using heuristic evaluation during implementation. It also introduced the use of usability evaluation of each user story as part of acceptance test.

My literature study documents that agile software development processes is not likely to take usability aspects and the user interface itself into account. Agile processes focus on the technical aspects of software development and organizing the overall structure of the project such as code and system architecture. The strength of agile processes is short time span from initial phases to actually working software and be able to respond to change dynamically when identifying challenges, defect and scope as the projects evolves.

Feedback is a very important operationalization of a fundamental but rather abstract principle of agile processes because it addresses functional challenges and errors and as a routine of the development process. This routine also offers a link to my method developing of how to apply a valuable dimension of what and how usability can contribute in the process: To create knowledge and deliver it as feedback about challenges and errors in interaction.

The values and principles of agile processes are also operationalized through acknowledged methods such as XP and scrum. Roughly speaking, XP focuses on technical aspects and scrum focuses on project management. Though different in their focuses they both aim to enable the software developers to ensure progress, quality assurance and incorporating business aspects represented by the close collaboration with the costumer. The costumer might represent the end-user but agile methods do not guarantee user participation in broader terms or represented by other end-user. However, I have adapted the focus of enabling the software

developers in my method developing with integration of usability aspects in mind. That is, the software developer themselves should be able to operationalize an interdisciplinary approach to software development including optimising usability.

In order to optimise usability I have examined interaction design where user participation is considered absolutely central. I myself consider three dimensions of interaction design to be especially relevant because they add most value to the work of software developers in order to operationalize interdisciplinary approaches themselves: Who are the users, how do they use the software and how can we evaluate the software. By using these three dimensions of interaction design my method is more likely to provide and apply important knowledge about the users that proves to be crucial to creating a software product that is in fact usable and not just functional.

My literature study of how interaction design techniques can be used in agile software development processes led me to conclude that Patton's initial identification of user roles and user tasks provides a more solid and user centered foundation to build requirements on. Therefore I integrated this aspect in my method. Additionally Kane and Wolkerstorfer provide specific and rather simple usability techniques that require no formal training and because their approach is likely to make the integration of agile software development and interaction design smooth. Heuristic evaluation and usability evaluation are integrated in my method because of their easy-to-use characteristics. Unlike Sy the approach of Kane and Wolkerstorfer does not require organizational changes but can be operationalized in existing team organization thereby allowing continuously focus on interdisciplinary work performed by software developers instead of dividing functional and usability competencies.

The most important difference between my method and those accounted for in the literature study is the use of usability evaluation per user story. My method uses this approach as it seems to be easier to evaluate as the design of the user interface evolves rather than after the complete user interface have been implemented. Thereby in my method the working software product acts as a prototype as well in order to provide feedback about usability issues.

Other characteristics of my methods worth mentioning includes user studies at the beginning of the project but also allowing the user studies to be refactored while developing the software in an agile process. This means that you can update the artefacts representing the results of user studies whenever you get a cause to do so and include new experience and requirements that is not already covered in existing user studies.

My method also integrates usability heuristic as a supplement to existing coding standards already used by XP practitioners as mentioned above because the technique is probably one of the easiest procedures to secure a minimum of usability in the development phase resulting in a more useful implementation of the software product in the end.

The proposed method was used to implement of the new version of EASEWASTE. Time and resources available were significant constrains for the EASEWASTE project considering the fact that I was the only software developer and we focused on only one release to be finished within the current project period. Therefore it was realistic to aim for a part of a complete agile project lifecycle though my method aims at covering an entire lifecycle.

In overview my method consists of the following procedure aimed at software developers:

- **Exploration phase:**
  - **User research:** Conducting user research through observation to get a greater understanding of who the users are and of how users interact with the software to accomplish specific tasks.
  - **Task analysis:** Conducting analysis of actual user behaviour when interacting with the software identifying tasks.
  - **User profiles:** Stating user profiles as an artefact on the basis of user research and the task analysis.
  - **Usability goals:** Stating usability goals as an artefact on the basis of user research and the task analysis.
  - **Requirements:** Articulating requirements for the system as user stories.
- **Planning:**
  - **Discussion of requirements:** Meeting with the customer in order to discuss requirements for the system and capturing them as user stories.
  - **Estimation and Prioritizing:** Estimating the resources needed to develop each user story and getting the customer to prioritise the user stories according to business value.
  - **Release plan:** The customer created a release plan.
- **Iteration(s):**
  - **Discussion of requirements:** Meeting with the customer in order to discuss requirements in details for each user story to be implemented.
  - **Implementation:** Implementing user story using heuristic evaluation systematically as a supplement to coding standard.
  - **Evaluation:** Usability evaluation of the software and especially the user interface conducted by software developer and costumer when the user story is implemented.

This literature study, method development and case study all together therefore clearly indicates that usability can be integrated more into agile development processes if it becomes part of the daily work both on organizational and technical level. Usability needs to be addressed when discussing user story requirements in details and accept that user focus can

be used as a tool to make valid decisions. On a technical level by means of practices and techniques that focuses on usability and not only technical functionality.

All in all the process of the case study indicates that the practice of user research and the usability evaluation of each user story are definitely something to be considered adding to the agile software development process. The approach provides a more solid and sufficient basis for decisions for the customer in collaboration with the software developers in order to release a software product that is in fact usable.

First of all it helped focusing continuously on the user in the agile software development process as I expected. The definition of method and process articulated and thereby anchored a mutual understanding of the importance of usability in any part of the agile development process. At the same time it ensured continuously systematically evaluation of both functional and usability aspects throughout the process when for instance heuristic evaluation are incorporated as a standard routine for software developers. In the following I will highlight the most important findings about the specific techniques tested in the case study:

- Usability evaluation is very good at discover usability challenges and defects when using working software. Thereby it provides means to establish if a user story can be considered done in agile software development processes. It helps and supports the creation of an evolutionary UI design which integrates very well with the agile approach aiming to have a system as close to production ready state as possible at all time.
- Usability goals support decisions regarding UI design. Before making the final decision usability goals can be consulted to see if the decision is aligned or if the usability goals should be updated as a consequence.
- Visualisation of requirements resembles format of simple prototypes of user interfaces making it easier to understand different approaches for design solutions. Success rate can perhaps even be increased if they are used during the discussion between developer and customer about the details of a user story if it has any influence on the UI when it is about to be implemented.
- Heuristic evaluation used during implementation is suitable for making a consistent UI and can be seen as an extension to coding standard which means it is easy to operationalize for software developers without special training.
- Creating user profiles and usability goals early in the project helps making decisions regarding design of the UI. However, they should in fact not be set in stone, just like the understanding of the systems evolves throughout the project so does the understanding of user profiles and especially the usability goals.
- User research conducted early in the agile process can be rewarding in regards to knowing different users and the tasks involved. In this case study I integrated contextual observation in the exploration phase. Though it might provide insight it is

also very time consuming in an agile software development process. The risk is that the exploration phase will be stonewalling creation of requirement and actually start implementing. In my case study one can argue that much of this knowledge could have come from the customer who was a very good representative for the end user as well as she was trained and highly skilled.

- Discussing requirements in details need to be based on usability aspects as well. I experienced this when trying to discuss them with a more traditional and thereby functional focus in my case study. This experience indicates that discussing requirements in details before a user story is implemented might increase the number of usability defects found during usability evaluations. This is an important lesson if I was to revise and reuse this method in the future as it stresses the need of focusing on usability in every part of the agile software development process.

As a result extensions and choice of techniques used for user research ideally should be counterweighted with the typical agile processes quick and dynamical input and feedback

In this thesis the case study was limited and the method was only tested on a part of an agile process. However, I have no reason to expect that my method would produce remarkably different results if used in the entire process. It would mean more iterations and releases which would not be likely to influence the overall results.

If the method I developed was to be used in a larger scale project I would assume it would face certain challenges because a lot of user stories would have to be tested simultaneously. As it is manual work it can be a bottleneck in the development. It would probably be necessary to make certain adjustments such as grouping the user stories in batches of priority order.

If I was to revise the method for a future project it would be possible to test the integration of other interaction design techniques. I would most likely consider to replace the observation technique used in this case study for user research in the exploration phase because I found it very time consuming thereby no ideal match for agile software development processes. However, I expect that it would require some experiments to find or perhaps develop a research format suitable for agile processes need of quick input as for instance both interviews and questionnaires conducted in a traditional manner require a considerable amount of time and resources.

References

[1] K. Beck, *Extreme Programming Explained: Embrace Change.* Boston, Mass.: Addison-Wesley, 2004.

[2] M. Cohn, *Succeeding with Agile: Software Development using Scrum.* Upper Saddle River, N.J.: Addison-Wesley, 2010.

[3] M. Cohn, *User Stories Applied: For Agile Software Development.* Boston, MA: Addison-Wesley, 2004

[4] J. Shore & S. Warden, *The Art of Agile Development*. Sebastopol, CA: O'Reilly, 2007.

[5] C. Larman, *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2003.

[6] J. Preece, *Interaction Design Beyond Human-Computer Interaction.* New York: John Wiley & Sons, Inc., 2002.

[7] J. Nielsen, *Usability engineering*, San Francisco, CA: Morgan Kaufmann, 1993.

[8] Usability.gov, Available: http://www.usability.gov/methods/index.html [Accessed: May 2011]

[9] UsabilityNet, Available: www.usability.gov/methods/index.html [Accessed: May 2011]

[10] Agile Manifesto, Available: http://www.agilemanifesto.org [Accessed: May 2011]

[11] A. Cockburn, *Agile Software Development*, Addison Wesley, 2001.

[12] Agile Alliance, Available: http://www.agilealliance.org [Accessed: May 2011]

[13] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, Addison-Wesley, 2011.

[14] EASEWASTE, Available: http://www.easewaste.dk [Accessed: June 2011]

[15] VisionOne, *State Of Agile Survey: The state of agile development 2010*, Available: http://www.versionone.com/pdf/2010_State_of_Agile_Development_Survey_Results.pdf [Accessed: June 2011]

[16] D. Sy, "Adapting Usability Investigations for Agile User-centered Design". vol. 2, no.3 *Journal of Usability Studies* May 2007 pp. 112-130 2007

[17] D Fox, J. Sillito and F. Maurer, "Agile methods and user-centered design: How these two methodologies are being successfully integrated in industry," in *Agile, 2008. AGILE '08. Conference,* 2008, pp. 63-72.

[18] J. Patton, "Hitting the target: Adding interaction design to agile software development," in *OOPSLA 2002 Practitioners Reports,* Seattle, Washington, 2002, pp. 1-7.

[19] D. Kane, "Finding a Place for discount usability engineering in agile development: Throwing down the gauntlet," Proc, *Agile Development Conference (ADC'03)*, IEEE Press, 2003, pp. 40-46.

[20] P. Wolkerstorfer, M. Tscheligi, R. Sefelin, H. Milchrahm, Z. Hussain, M. Lechner and S. Shahzad, "Probing an agile usability process," in *CHI '08 Extended Abstracts on Human Factors in Computing Systems,* Florence, Italy, 2008, pp. 2151-2157.

Appendix A - User stories

Prioritised user stories.

User stories:	**Estimate:**
Load a simple scenario and show it	1 week
Add one waste process to the scenario builder	1 week
Add two waste processes to the scenario builder and connect them	3 days
Sankey diagram for substances	3 days
A user can view the composition of a waste in a scenario at any place	3 days
A user can calculate and view LCIs of a waste process + a scenario	1 week

Remaining user stories:

A user can view detailed information on waste processes from a library of waste processes. More cards; at least one for each type of waste process. Look for simpler components like transfer coefficient.	1 week
A user can compare different scenarios with their analysis.	1 week
A user can save own waste processes to a library of waste processes.	2 days
Sankey diagram for materials	3 days
A user can view the external process. (“elec. production, DK, 2001”, “Diesel consumption, 2001”)	2 days
A user will be notified if a parameter is missing or waste processes are not connected correctly when creating s scenario. Likewise when calculating LCI/LCA.	2 weeks
A user using generic processes to construct a scenario should be able to put formulas in the waste transfer functions and the LCI calculations.	3 days
A user can manage different scenarios in parallel.	1 week
Improved waste generation module.	1 week

Appendix B – Artefacts

Task analysis:

What overall goal does users have when working with the software?

- Assessing a system.
- Comparing systems.
- Tweak a system and evaluate the result.
- Interested in the waste composition.

How are users currently achieving their goals?

- Users use the software to construct scenarios.
- Users are exporting data and using other tools to compare scenarios.

User profile:

Beginner:
Student who has taken the course so has a small background in waste and LCA.

Intermediate:
Researcher or consultant who works with waste and LCAs on a daily basis so there by have a great knowledge in this area.

Usability goals:

- Users, first time users as well and experienced users, can quickly and easily construct a scenario representing a system without any hassle.
- Users can easily compare different systems and interpret the result without any difficulty.

- While using the software users should feel that the software supports them efficiently in finishing their task at hand.

Appendix C – User interface

This appendix shows how the user interface evolved during development.

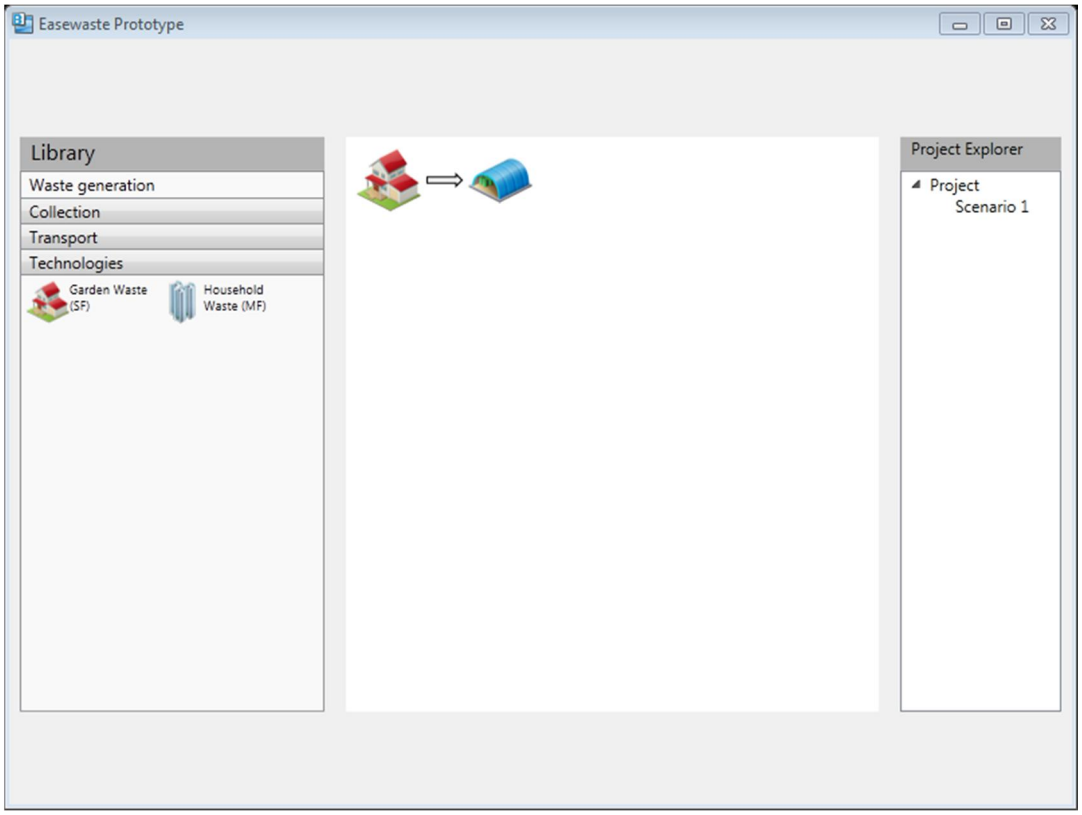


Figure C.1: Screenshot of the UI after first story "Load a simple scenario and show it"

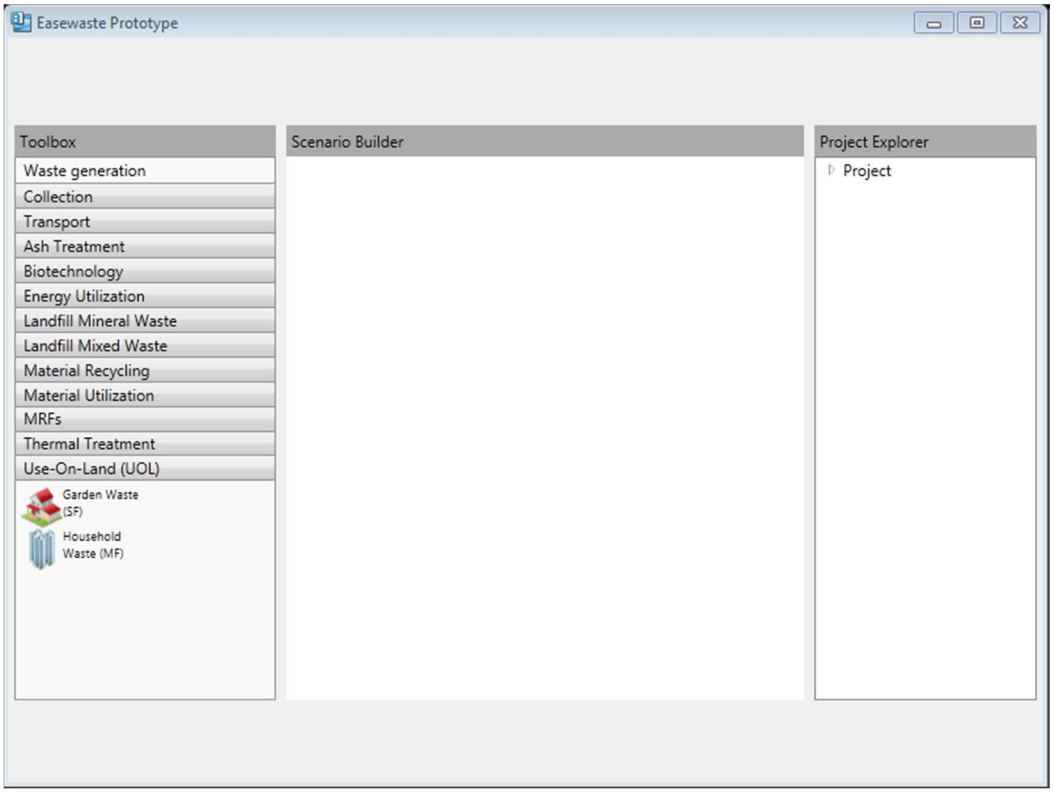


Figure C.2: Screenshot of the UI after second user story "Add one waste process to the scenario builder".

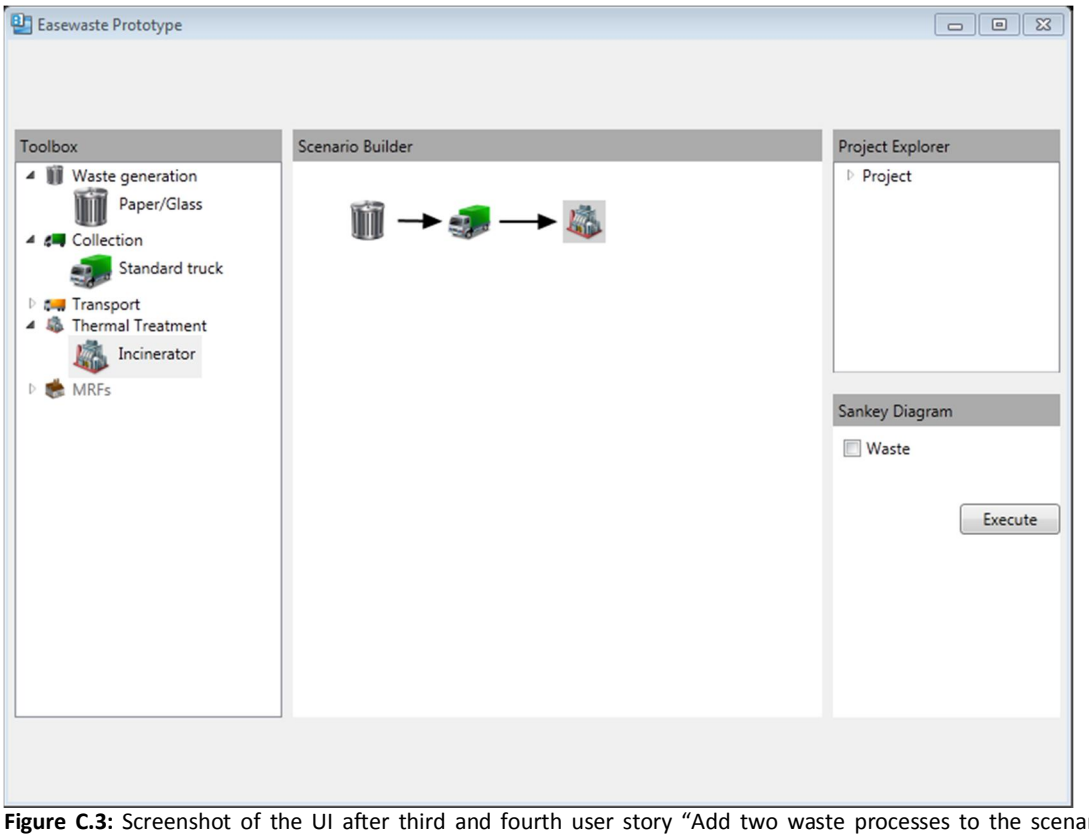


Figure C.3: Screenshot of the UI after third and fourth user story "Add two waste processes to the scenario builder and connect them" and "Sankey diagram for substances".

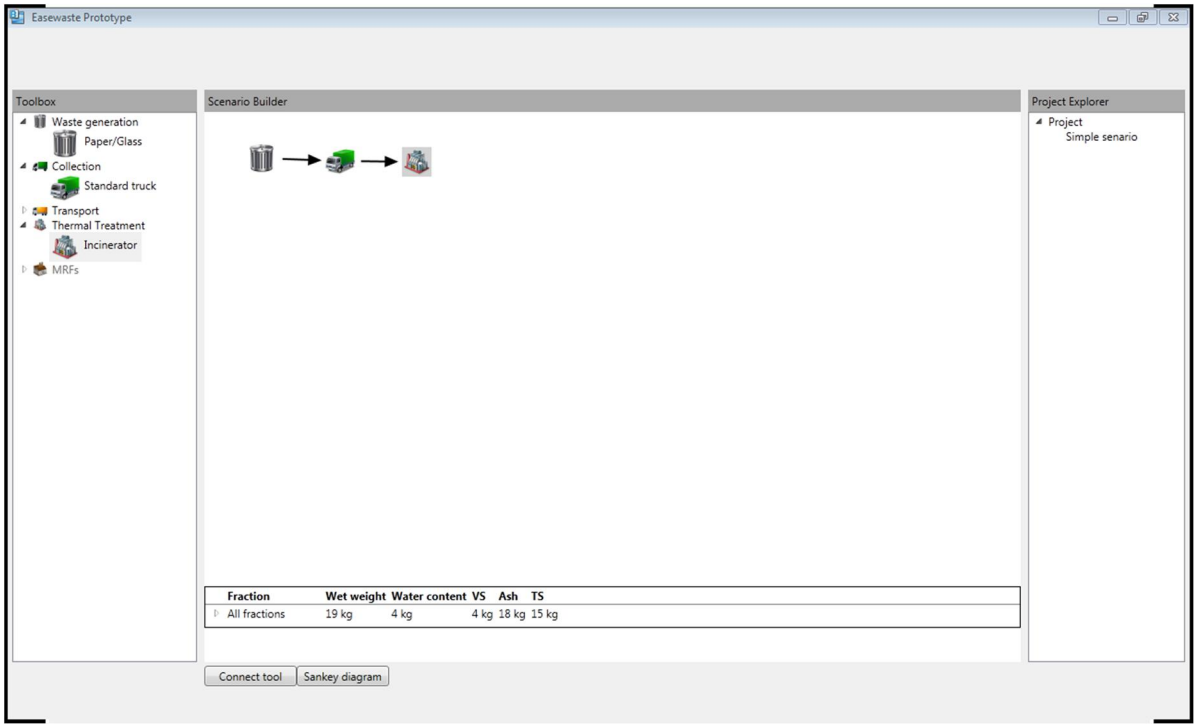


Figure C.4: Screenshot of the UI after fifth user story "A user can view the composition of a waste in a scenario at any place".

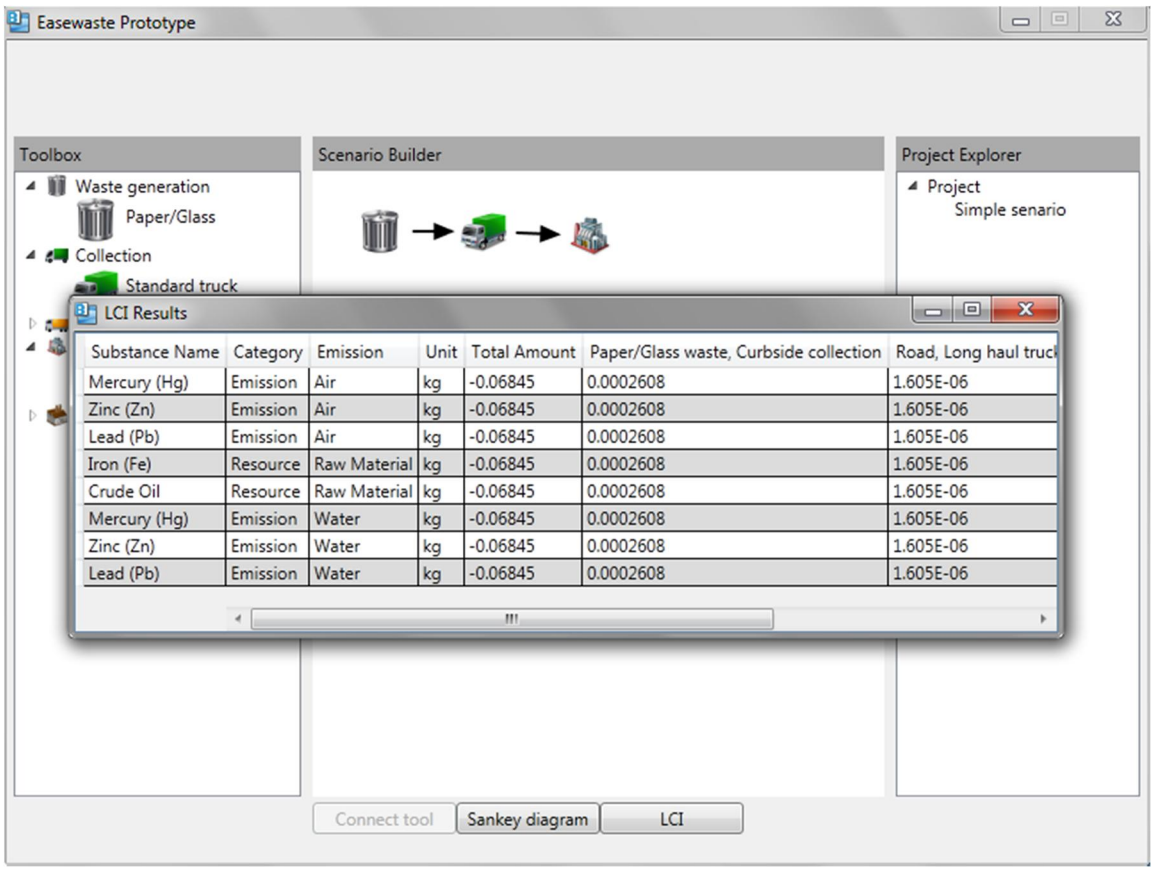


Figure C.5: Screenshot of the UI after sixth user story "A user can calculate and view LCIs of a waste process + a scenario".

Appendix D – EASEWASTE guide

The new version of EASEWASTE is located on the enclosed CD in the folder EASEWASTE.

EASEWASTE is created in .NET 4 so it is necessary to have Microsoft .NET Framework 4 installed on the computer before running the application.

Run EASEWASTE\Easewaste_prototype.exe to start the program.

Drag and drop items from the toolbox into the scenario builder to construct a scenario.

Connect items in the Scenario Builder by holding the SHIFT-key down + left mouse key then drag from one item to another item, they will now be connected with an arrow. Items in the scenario builder can be moved around and deleted by high-lighting them and then press the Delete-key on the keyboard.
Clicking on an arrow connecting two items in the scenario builder shows the compositions of the waste.