RADBOUD UNIVERSITY NIJMEGEN

Agile User Experience

Identifying and solving problems that occur when integrating User Experience into an Agile environment

by

Beau Verdiesen

August 2014

A thesis submitted in partial fulfillment for the degree of
Master of Science of Information Sciences

At the
Faculty of Science
Institute for Computing and Information Sciences

Beau Verdiesen

Studentnumber: *0813648*

Date: *12-08-2014*

First Examiner: *dr. L. G. Vuurpijl*

Second Examiner: *prof. dr. ir. Th. P. van der Weide*

Abstract

This thesis categorizes problems that might occur when integrating User Experience in an Agile environment into three problem areas. It then addresses these problem areas by matching techniques that are used within the fields of User Experience and Agile Software Development to the problem areas. This results in the creation of ten heuristics that originate both from the categorization of the problem areas as well as the matching of possible solutions. The heuristics were presented to experts in the form of an online questionnaire. The result of this thesis is a generalization and categorization of the Problem/Solution space that helps shed some more light on the subject.

Preface

The following research was conducted as part of the Master Information Science at the Radboud Univerity in Nijmegen. This thesis was written as a graduation assignment from Info Support B.V.
I would like to thank René Hietkamp and Stefan Jansen for granting me the opportunity and support to graduate at Info Support as well as the other interns who provided constructive feedback during my internship. Lastly I want to thank Louis Vuurpijl for his advice and guidance, giving me support and motivation when I needed it most.

Table of Contents

1 Introduction

The reason for writing this thesis was a question posed by the Dutch Software company Info Support. They got an increasing demand from their customers to make applications for touch devices. They noticed that users of these applications where far more demanding when it comes to the User Experience of these touch applications. This lead them to pose the question how they could come up with a good User Experience while working in an Agile environment.

It has proven to be difficult to integrate User Experience (UX) in an Agile environment which leads to bad collaboration of the software development team and overall lower efficiency and effectiveness (Isomursu, Sirotkin, Voltti, & Halonen, 2012) (Ferreira, Sharp, & Robinson, 2011). Meanwhile more and more software development projects in a business environment have an end-product where UX plays a key role in getting customer satisfaction. The most prevalent example of such an end-product would be a touch-enabled device (smartphone, tablet, etc.) The combination of these two developments can be quite problematic. Although existing design methodologies may be suitable for, e.g., designing for UX, Agile software development, or touch-enabled "Apps", what is lacking is an integrated methodology that provides the required elements of the individual methods. This thesis proposes a "best practices approach" consisting of a combination of existing methods that will: (i) ensure a good user experience when developing (ii) software that requires a high level of user satisfaction in (iii) an Agile environment.

It is important to note that a good user experience is not always the most important requirement when developing software. Compare, for example, a bank that has a system that handles transactions made by its customers as well as an app that allows the customers to make these transactions. For the first system efficiency, uptime and maintainability are far more important requirements then the User Experience. More than likely it will have a very basic interface that can tweak some variables. The app on the other hand will have customer satisfaction as one of its main priorities to make sure the app is even used in the first place.

Many companies want to get an app for mobile devices in order to deliver more service to their customers and extent their brand identity. It is imperative that the app delivers a good experience to the customer as a bad experience will also reflect poorly upon the company.

This thesis helps further the fields of User Experience and Agile Software Development and helps bring to the attention of software developers that giving UX a more central role in Agile teams will lead to better user satisfaction and better efficiency (Najafi & Toyoshiba, 2008). The development of a method that allows for the effective creation of a good User Experience when developing within an Agile environment will help software companies in their struggle to match a good design with good functionality. This, in turn, improves the overall User Experience of software which makes using it more comfortable for the end users.

1.1 The touch society

As the amount of computer interaction grows in our every-day life (smartphones, wearables, car-computers) so do our demands of the software we encounter. Small screens with limited input options such as touch or voice-activation pose new challenges for software development teams to make sure the user not only gets the functionality he wants but also has a good 'experience' when working with their software.

Touch applications are the ultimate test when it comes to User Experience. People have certain expectations when using an application with touch input: screen changes have to be smooth, buttons have to be just the right size and input should be handled flawlessly. Preferably errors should be handled without input from the user and typing should be reduced to a minimum . In short: the experience is expected to be flawless. These expectations are inherently different than when using, for instance, a personal computer and require a set of specific design principles.

Touch-enabled devices come in a wide variety of shapes and sizes and making a specific design for all those different devices individually is an enormous undertaking. Furthermore, new devices are still being produced which increases the already vast amount of different screen sizes and resolutions. At the moment screen sizes vary between 4.0" and 13.3" and can be separated into three different groups (Wroblewski, 2012): Smartphones, Tablets and Convertibles/touch-enabled Ultrabooks. Each of these devices have different areas where controls can be reached quickly and easily.

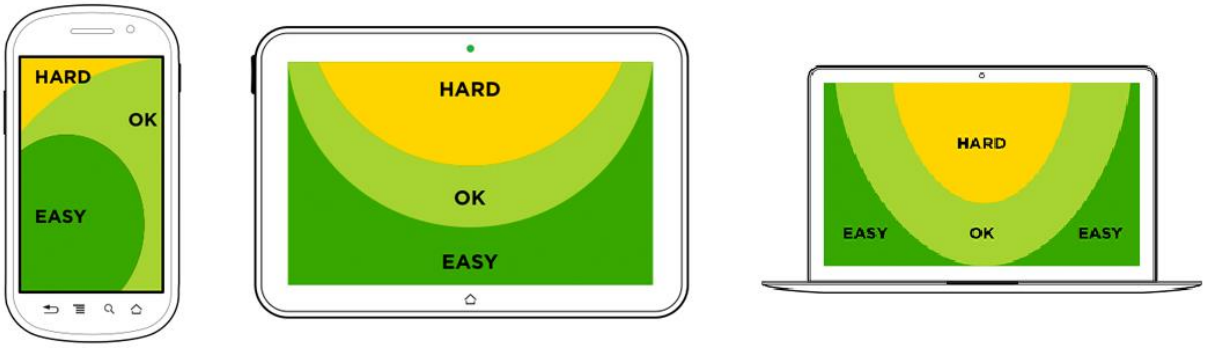


Figure 1 – Easy and hard to reach areas on different touch-enabled devices (http://www.lukew.com)

1.1.1 Design Principles

The three largest operating systems for touch-enabled devices (Android, iOS and Windows Phone) all have design principles to help developers create a good User Experience for their OS. The table below shows a comparison of these principles of Android (Android, sd), iOs (Apple, 2014) and Windows Phone (Microsoft, sd). Below is a rough comparison based on information that the three Operating Systems have openly put on their websites to help guide developers creating apps for their platform.

		Android	iOS	Windows Phone
1	Interact with objects	X	X	X
2	Use pictures where possible, instead of words	X	X	X
3	Avoid clutter, keep it simple and clean	X	X	X
4	Make sure the application is consistent (with itself and the OS)	X	X	X
5	Use the right typography	X	X	X
6	Use subtle visual and sound effects	X	X	
7	Make suggestions, but do not decide for the user	X	X	
8	Use established input (such as swiping)	X	X	
9	Use clear recovery instructions in case of an error	X	X	
10	Focus on the content, the UI is supportive		X	X
11	Use negative space to focus on what is important		X	X
12	Use a clear hierarchy		X	X
13	Prioritize certain actions over others	X		X
14	Focus on the User Experience	X		
15	Be smart (get to know the user)	X		
16	Be brief, use as short a phrase as possible	X		
17	Save content created by the user	X		
18	Only interrupt if it's important	X		
19	Split complex actions into small tasks	X		
20	Integrate behavior and appearance with purpose		X	
21	Provide feedback for actions (sound or visual)		X	
22	Make use of the whole screen		X	
23	Use translucent UI elements		X	
24	Use color to simplify the UI		X	
25	Use depth: Layers convey hierarchy and position		X	
26	Take advantage of the digital medium			X
27	Minimize typing input			X
28	Operate quickly, minimize the time it takes to respond			X
29	Use a clear Information Architecture			X

Table 1 – A comparison of Design Principles of Android, iOS and Windows Phone

Looking at the table above we can summarize that that in order to design a good application for a touch-enabled device you have to focus on:

1. The way the user interacts with the application [1,6,8,13]
2. Make sure the content you display is well visible, without any clutter [2,3,5,10]
3. Make it easy for the user to use the application [4,7,8,9,12]

1.2 Structure

The main research question this thesis aims to solve is:

> "*What are the problems when integrating User Experience in an Agile environment and how can you solve them?*"

The following sub questions have led to the conclusion of the main question:

I. *"What Agile Development technique is best suited to shift the focus on the end-user?"*
II. *"What are the best-practices for creating a good User Experience?"*
III. *"What are the problems when integrating Agile Development and User Experience?"*

This thesis starts in Chapter 2 by describing the results of a literature study on the two main research areas: Agile Software Development and User Experience. This second chapter also aims to explain some of the concepts that will be discussed further in the thesis and is divided into three parts:

- **Agile Software Development**
  What is the history behind Agile Software Development? How do you develop software in an Agile way, what methods are being used and how do the differ from each other?
- **User Experience (UX)**
  Where does User Experience come from, what does it add to the software development process and what methods are available?
- **Integrating UX and Agile**
  What are known problems that occur when trying to integrate UX in an Agile environment?

The third chapter summarizes the problems that have come forward during the literature study into three main problem areas. Chapter 4 then combines these problem areas with possible solutions in the form of techniques and methods that have been researched in the literature study. In order to write Chapters 3 and 4 the most suitable techniques and methods from the fields of Agile development and User Experience have been taken and expanded upon where necessary. The research done in these chapter has led to the creation of ten heuristics that are explained in Chapter 5. Validation of these heuristics took place in the form of a questionnaire. Chapter 6 covers the method of the questionnaire as well as the questionnaire itself. Chapter 7 then discusses the results of the questionnaire followed by a conclusion in Chapter 8. This thesis ends with a discussion in Chapter 9.

2 Literature study

The literature study in this thesis consists of three parts. First a study on Agile Software development is conducted examining its history, the Agile manifesto, Agile methodologies and Agile techniques. Next is a study on User Experience. Here traditional Usability evaluation is compared to User Experience and methods and techniques related to User Experience are studied. The third part is a study on the problems that you can encounter when trying to combine Agile and User Experience.

2.1 Agile Software Development

2.1.1 History

Agile methods are methods for developing software incremental. Developing software in an Agile manner ensures flexibility during the development process. The origin of Agile software development can be traced back to the 1930's but took a rise in popularity with the creation of iterative and incremental development (IID) methods in the late 60's (Larman & Basili, 2003). The idea behind IID is that you build software by means of repeated cycles (iterative) and create small portions at a time (incremental). The advantage of this is that the system is built up slowly which means that mistakes can easily be fixed and the lessons learned can be applied to the rest of the project. Furthermore you can get feedback at an earlier stage of the project allowing the project to stay on track.

Before the term Agile ever came into existence many big IT companies such as IBM and Microsoft were already experiencing the benefits of using iterative software development techniques. Synchronize and Stabilize, for instance, is a Microsoft-style iterative development technique that they have been using since the late 80's (Cusumano & Yoffie, Software development on Internet time, 1999) and has many similarities with Extreme Programming, a technique that became popular around 1997 (Cusumano, Extreme programming compared with Microsoft-style iterative development, 2007).

An important techniques that predates Agile techniques but has laid the groundwork for several of them, such as ASD and DSDM (see 2.1.4) is called Joint Application Development (JAD) (Paetsch, Eberlein, & Maurer, 2003)**.** During a JAD session people from various disciplines come together to discuss the project. This includes, among others, executives, project managers, users, outside experts and documentation experts. In these sessions product features are discussed as well as possible solutions to problems that have occurred. JAD sessions also help to monitor the project. These sessions promote understanding and communication between the various disciplines involved in the project.

The term 'Agile' came into being with the Agile manifesto that was published in 2001 (Beck, et al., 2001) by a group of software developers discussing such IID methods. A number of the authors later formed the Agile Alliance that promotes the usage of Agile techniques.

According to the manifesto, Agile processes are based on four main statements with the notation that "*while there is value in the items on the right, we value the items on the left more*":

I.	Individuals and interactions	over	processes and tools.
II.	Working software	over	comprehensive documentation.
III.	Customer collaboration	over	contract negotiation.
IV.	Responding to change	over	following a plan.

Additionally the Agile Alliance defined 12 principles in support of these values. These principles should help agile teams get a grip on how to adhere to the four statements. These principles are not dead-set. Meaning that any team can choose which principles they find to be more important than others. They might even form principles of their own. These are the principles as stated by the Agile Alliance:

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

2.1.2 Concepts involved in Agile software development

It is important to note that 'Agile' is not a single technique, it is a way of working that is based on the statements and principles mentioned in the manifesto. This means that there are many Software Development Methods that are Agile in that they are based on these principles, but still differ from one another in a number of ways. This thesis explains Agile in four sections, these four sections all take one of the statements from the Agile manifesto and link them to the Agile principles. The sections are named as to reflect their content:

- Collaborative, self-organizing, multi-disciplinary teams
- Continuous Integration
- Customer Focus
- Changing Requirements

Numbers in parentheses refer to one of the twelve principles.

Collaborative, self-organizing, multi-disciplinary teams

'I. Individuals and Interactions over processes and tools'.

The first statement of the Agile Manifesto touches one of the key focus points of Agile processes: people. Rather than everyone on the same Software Development team working on their own little piece of software, Agile enforces that team members collaborate. This is not the same as simply communicating. Where communication is simply sending and receiving messages, collaboration means actively working together in product creation and decision-making (Cockburn & Highsmith, 2001). Communication however, is essential when trying to collaborate effectively.
Agile promotes the idea of self-organizing teams (#11). Such self-organizing teams should consist of members of all disciplines such as developers, architects, designers, team managers, testers and

perhaps a representative of the customer (#4). Preferably the whole team should be together in the same room (Moe, Brede, Dingsoyr, & Dyba, 2009)**.** What this creates is an environment where face-to-face communication flows freely (#6). This allows problems to be identified quickly and decisions on how to deal with these problems can be made on the spot in an equally quick manner. Such close lines of communication raises the level of trust (McHugh, Conboy, & Lang, 2012) between the team members and allows for a responsive team. To further enhance the responsiveness team members are encouraged to develop themselves as broadly as possible and expand their competences. If one team member would not be able to finish his tasks another member should be able to pick up the work. Self-organization also means that control from outside of the team should be limited (#5), the team should be trusted to handle the project themselves.
Another key point is shared responsibility. The whole team should feel responsible for the software that is being delivered. This encourages team members of different disciplines to comment on each other in order to identify issues before they become problems (#9). In order for these teams to stay up-to-date on the progress that is being made and adapt to new situations it is important they reflect on the progress at regular intervals (#12).

Continuous Integration

'II. Working software over comprehensive documentation'.

This is a view that is the opposite from traditional plan-based development methods, such as the waterfall method (Bell & Thayer, 1976), where documentation is a major part of the project. Writing documentation is necessary if you have a low-communication project. However, because Agile teams are all about collaboration, and thus communication, a lot of documentation becomes obsolete. Why write something down when you can just talk to the person you're writing it for? Agile processes are not *against* documentation, documentation serves it purpose when people not affiliated with the project need to know how the system is build and how it works. However, writing *comprehensive* documentation is not an efficient way of communication and time could better be spent on creating working software. This working software can then be presented to the customer who would much rather comment on something tangible then comment on pages of abstract documentation (#1). Because new working pieces of software are continuously added to the project progress is much more visible to both the customer and the team (#7). The shorter the iterations, the quicker a customer can comment on the progress and steer the project (#3). Again, this goes against the traditional, sequential, way of software development where delivery does not take place until the end of the project meaning customers cannot provide feedback during the development process (Olsson, Alahyari, & Bosch, 2012).
Continuous Integration means that new, tested and verified, code is being added to the software continuously. This means that at any given time the software could be built en shipped to the customer. Another interesting concept is continuous delivery. Here, instead of waiting for the project to complete before shipping the product the software is already available to the customer and he is actively working with it while it still under development. This allows for continuous customer feedback and provides real usage data that can be used to enhance the product.

Customer Focus

' III. Customer collaboration over contract negotiation'.

Note that it says "collaboration" meaning active participation of the customer in the development process. Traditionally a requirements document would be made that would be part of a contract the customer would then sign. After the project is complete and the software is delivered if there were any problems the developers could simply say "Sorry, but that's not what it said in the requirements

document which you signed". This way of working does not facilitate the creation of a product that is as useful to the customer as possible (#2).

In the Agile way of working the customer is involved in the development process and is able to steer the direction of the project by constantly adding new requirements and changing old ones if that it necessary (#2). This way you focus on delivering software that is useful to the customer by sharing the responsibility of the project with the customer.

Changing Requirements

'IV. Responding to change over following a plan'.

Agile is about creating and responding to change. This means that working in an agile way actually *facilitates* change (#9). While this might sound daunting to traditional, plan-based software developers, change in an agile environment is nowhere near as destructive as it is in a traditional environment. Traditionally, meaning the development team does not focus on User Centered Design, changes do not come to light until the customer gets involved in the project, usually in a very late state, and then it is often hard, time-consuming and expensive to make changes. In an Agile environment customers are encouraged to change their priorities during the entirety of the project to ensure that they get the best possible end-product (Highsmith & Cockburn, 2001). Because code is written iteratively and in small bits it is far easier to adapt to change.

2.1.3 Critique

Ever since the Agile Manifesto has been published there has been criticism coming from the traditionalists who advocate extensive planning and processes that make software development predictable and efficient (Boehm, 2002).

2.1.3.1 *The planning spectrum*

A good way to differentiate Agile from other traditional software development methods is to view them as different ends of a planning spectrum. The term 'plan' in this sense means documented processes, milestones, strategies, requirements, designs, architectural plans, etc. On one end of the spectrum you have Agile development methods that, although they do use planning, do not focus on it nor do they shun it as much as, for instance, unplanned and unorganized hacking. On the other end of the spectrum you have projects that use micro milestones (often called inch pebbles), where every little step that is taken is planned in advance and documented extensively. In the middle you might find risk-driven spiral methods such as the Rational Unified Process (RUP) (Kruchten, 2004) that incorporate both agile and plan-based techniques.

2.1.3.2 The people factor

Another important thing to note is that working Agile requires a certain kind of people. Working in an Agile team requires amicability, talent as well as good communication skills (Cockburn & Highsmith, 2001). It is necessary for team members to criticize each other's work without taking it personally. Because the team should be self-organizing and multi-disciplinary it requires team members to be experts in their respective field and, ideally, also capable of taking over tasks of other team members. As Larry Constantine states:

"All of the agile methods put a premium on having premium people and work best with first-rate, versatile, disciplined developers who are highly skilled and highly motivated. Not only do you need skilled and speedy developers, but you need ones of exceptional discipline, willing to work hell-bent-for-leather with someone sitting beside them watching every move" (Constantine, 2001)**.**

Taking on these responsibilities takes a certain amount of maturity and experience making agile experts hard to come by.

The emphasis Agile methods place on communication can also make it hard for international teams to work together in an agile way. Transferring knowledge gets harder when people are not working on the same site or even speaking the same language. In such a case the use of documentation is necessary but this often leads to misinterpretation.

2.1.4 Agile Methods

There are many Agile Methods available, each with their own spin on the Agile way of working. Some of the more well-known methods are *Scrum, Extreme Programming (XP), Lean, Rapid Prototyping and DAD*. In practice, many development teams use their own form of Agile development by combining aspects of different Agile methods to get a way of working that works best for them. This chapter lists some of the most well-known Agile techniques and their characteristics. The next chapter will compare the techniques listed here and answer sub question I: "*What Agile Development technique is best suited to shift the focus on the end-user?*"

2.1.4.1 Extreme Programming (XP)

Extreme Programming (Beck, 2000) advocates frequent releases to deal with changing customer requirements. XP also says something about how the developers should work. XP advocates pair programming, a technique where two developers are behind the same screen programming together. Furthermore developers should do extensive code reviews, unit-test all the code, keep the code as simple as possible and avoid the programming of features until they are actually needed.

2.1.4.2 Scrum

Scrum (Schwaber, 2002) assumes the development process is unpredictable and one of the key principles of Scrum is that the customer can change his mind about what he wants. Requirements should therefore be expected to change during the project. Scrum focusses on small, highly experienced teams that are co-located so that the team may communicate verbally. The three core roles of a scrum team are the Product Owner, who represents the stakeholders, the Development team, who are responsible for creating a shippable product and the Scrum Master, whose responsibility it is to remove impediments for the team so that they may work as effectively as possible.

The scrum process works in 2-4 week sprints in which a working increment of the software is delivered. Furthermore there is a 'daily stand-up' where all team members tell the rest of the team what they have done, what they are currently working on and if they foresee any problems in the future. In the beginning of the sprint the whole team decides what features will be created during that sprint in a sprint planning meeting.

Reflection and self-improvement is an important part of scrum and after each sprint a sprint review meeting takes place where the team reflects on what went well and what could be improved upon.

2.1.4.3 Adaptive Software Development (ASD)

ASD (Highsmith, 2013) is a dynamic speculate-collaborate-learn software development life cycle (Chowdhury & Huda, 2011). Objectives and schedules are fixed in the Project initiation phase. In the collaboration phase several components are under concurrent development. Because components are constantly being refined the planning cycle is a part of the iterative process. The final phase reflects on the work that has been done, including the status of the project and customer input, and how the process could be improved upon.

2.1.4.4 Feature Driven Development (FDD)

FDD (Palmer, 2001) has two stages: the first stage assess which features will be implemented and the development work takes place in the second phase (Chowdhury & Huda, 2011). The preciseness, maintainability and extensibility of the project hinges on the quality of the work done in the first phase and customer involvement is high at this point. UML diagrams are used to express functionality and the feature list is derived from the UML diagrams. These features should be small enough to develop in short iterations. Before the development work begins in the second phase the features are grouped into packages. One iteration of one to three weeks should be enough to complete one work package. Once one of these packages is completed it is given to the customer for testing.

2.1.4.5 Lean

Lean Software Development (Poppendieck, 2003) is not a method or a process you can follow (Anderson, 2012). Instead Lean suggests that the pursuit of perfection can be achieved by eliminating waste. Systematically identifying and elimination wasteful activities is one of the core principles of Lean Development. Much like the Agile manifesto, Lean has a number of core values that determine whether the Software Development Process you are using is lean or not. The six lean values are:

I. Accept the human condition
II. Accept that complexity & uncertainty are natural to knowledge work
III. Work towards a better Economic Outcome
IV. While enabling a better Sociological Outcome
V. Seek, embrace & question ideas from a wide range of disciplines
VI. A values-based community enhances the speed & depth of positive change

In the first years of the 21st century Lean principles were used to explain the use of Agile methodologies. Lean can explain that Agile methods produce very little waste and therefore produce better and more value. Software development methodologies can be both Agile and Lean at the same time. In recent years Lean has been the basis of a number of software development processes that use Kanban (further explained in 2.1.4.12) systems to focus on improving flow, risk management and decision making rather than (just) on eliminating waste.

2.1.4.6 Crystal

Crystal is part of a collection of methods and processes called Crystal Family developed by Alistair Cockburn (Cockburn A. A., 2001). This family of methodologies is segmented into color depending on the number of people that take part in the project. The methodology for 2-6 person projects is Crystal Clear, for 6-20 person projects is Crystal Yellow, for 20-40 person projects is Crystal Orange, then Red, Magenta, Blue, etc.
Crystal methods focus on security, efficiency and usability (developers have to be able to use the methodology). They have a number of principles in common, the most important being the delivery of products, feedback on improvements and good communication between team members.

2.1.4.7 Dynamic Systems Development Method (DSDM)

DSDM (Stapleton, 1997) was first released in 1994 and sought to provide some discipline to the Rapid Application Development method. It became a generic approach to project management and solution delivery in 2007. DSDM uses an iterative an incremental approach that embraces the Agile principles.

DSDM consists of three phases:

- *The Pre-Project*: Identify candidate projects, project funding and project commitment.
- *The Project lifecycle*: This includes a Feasibility and Business Study, Functional Model iteration, Design & Build Iteration and Implementation stages.
- *Post-Project*: This phase ensures system efficiency and effectiveness by means of maintenance, enhancements and fixes according to DSDM principles.

2.1.4.8 Disciplined Agile Delivery (DAD)

The DAD process framework (Ambler, 2012) is an agile approach to IT solution delivery that puts people first and is learning-oriented. The characteristics that are most important are:

- People first
- Learning-oriented
- Agile
- Hybrid
- IT solution focused
- Goal-driven delivery life cycle
- Risk and value driven
- Enterprise Aware

DAD is a hybrid framework that uses common practices and strategies from Scrum, Extreme Programming and Agile Modelling among others to address the full delivery lifecycle. Individuals and the way they work together are the key to success for IT projects. Furthermore DAD is enterprise aware, motivating teams to leverage their existing organizational ecosystem and improve upon it.

2.1.4.9 Timeline

Below is a timeline of the various Agile methods mentioned above. Keep in mind that many of these methods have been changed a number of times since their origin.

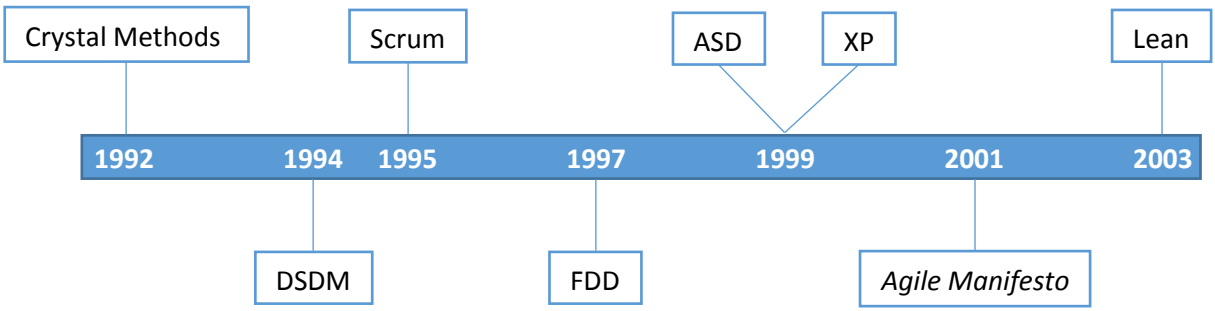


Figure 2 – A timeline of Agile Methodology

2.1.5 Comparison

Comparing one agile method to another is not as easy as it sounds. Many of these methods are derived from each other, removing certain aspects and replacing them with others. DAD, for instance, is a methodology comprised of more than eleven different methods and software development processes.

Table 2 lists the abovementioned Agile methods and compares them on five variables that are characteristic for the software development process. The variables come from a master thesis by Martina Šimičić from 2013. Deciding that method fits best for what project type is not easy in such a

diverse landscape. The main focus points seem to be the scale of the project and customer involvement. Some of the methodologies offer multiple solutions depending on the scale of the project (most notably Crystal and DAD) others are meant for smaller teams and do not scale very well (XP, ASD and to some degree Scrum). XP, Scrum and DAD are the only methodologies to proactively involve the customer in the development process.

	Customer involvement	Communication	Team size	Iteration length	Documentation
XP	Various methods	Daily stand-up,	Small (<20)	1-6 weeks	Basic documentation
Scrum	By means of Product Owner role	Daily stand-up	All sizes (multiple teams possible)	2-4 weeks	Basic documentation
ASD	By means of release feedback	Face-to-face	5-10 members per team	4-8 weeks	Basic documentation
FDD	By means of reports	By means of documentation	All sizes (more than one team)	Days-weeks	Extensive documentation
Lean	/	/	/	/	Minimal documentation
Crystal	By means of release feedback	Face-to-face	All sizes (depending on the selected method)	Depends on the selected method	Basic documentation
DSDM	By means of release feedback	By means of documentation	All sizes	Time boxing (days-weeks)	Extensive documentation
DAD	By means of Product Owner role	Depends on the chosen lifecycle	All sizes	2-4 weeks	Basic documentation

Table 2 – A comparison of Agile methodologies and their characteristics (adapted from an existing table at: http://agile-only.com/master-thesis)

The answer to the second sub question: *"What Agile Development technique is best suited to shift the focus on the end-user?"* is Scrum. The reason for this is that scrum already has great tools and processes in place that can be adapted to put the focus on UX such as the daily standups (2.1.6.1) and the sprint review (2.1.6.2). Communication and collaboration is important within a Scrum team and emphasized by the techniques above.

2.1.6 Agile Techniques

Following are a number of techniques that can be used when working with any of the Agile methodologies above. None of these are mandatory and it will depend on the software development team if they decide to use them or not.

2.1.6.1 Daily Stand-up

A daily stand-up is a short meeting with the members of the project team that are available. It takes place daily, at a set time and at a set location. Each member in turn briefly tells something about what they have done since the last stand-up, what problems they ran into and what they are currently working on. The point of these meetings is that they short, to the point and can make each member of the team aware what is going on and perhaps help solve some problems that other team member have run into. 'Daily stand-up' is a term used in the Scrum methodology. Other methodologies might have similar events but with a different name.

2.1.6.2 Sprint Review

A Sprint Review session takes place after an iteration is completed. In such a session the team discusses the progress that has been made this iteration. Often, within this session, this progress is shown to the client who can then immediately comment on it. The session is also used to evaluate the process the team is using and if anything has to change. 'Sprint Review' is a term used in the Scrum methodology. Other methodologies might have similar events but with a different name.

2.1.6.3 Continuous Deployment

Continuous Deployment (Humble, 2010) is a design practice, rather than a development practice, that extends on the principles of agile development. CD aims to automate the process of software delivery. It does so by means of a deployment pipeline. After code has been completed a number of validations take place: automated build and unit tests followed by automated acceptance tests. If these tests succeed manual approval takes place and the latest build may be deployed at the customer. Developing software with continuous development gives you the ability to repeatedly push out enhancements and bug fixes to customers at low risk and minimal overhead.
Developers have to be wary when using CD that any code that is committed may be released to the customer at any point. However, the quick feedback you get is of great value when working with short lifecycles.

2.1.6.4 Kanban

Kanban (Sugimori, 1977) is a method that allows you to manage work in such a way that promotes just-in-time delivery. An important aspect of Kanban is that you 'pull' work from the demand rather than work being 'pushed' to the work floor. Work is often visualized by means of cards placed in columns on a board that represent the different states or steps in the workflow.
The core practices of Kanban include limiting the work-in-progress, visualizing and managing the workflow, making policies explicit and implementing feedback loops.

2.1.6.5 User Stories

A user story consists of one or more sentences and describes a need for a certain piece of functionality. A well-used template for user stories was thought up by Connextra in the UK in 2001 and uses a role-feature-reason format:

> 'As a <*role*>, I want <*feature*>, So that <*reason*>'

For example: 'As a shopper in a hurry, I want to quickly navigate the shopping cart, so I lose as little time as possible finishing a purchase'

These user stories are created by the customer, with the help from someone from the project team. The stories are written down on cards and these cards can be tested by means of the INVEST guidelines (Wake, 2003). User stories are an important part of many Agile development methodologies and define what the system has to be able to do. They can be prioritized by the customer and help the developers to get an idea of the reason they are creating a certain type of functionality and what type of user will be using it.
Because the stories themselves use everyday language they can convey the story through multiple disciplines (Business Analyst, Developer, Tester, UX Designer, Manager, User, etc.).

2.2 User Experience (UX)

This section starts with traditional usability evaluation and why it has become more and more important to focus on the user experience. UX terminology is discussed followed by an explanation of User Experience itself by means of the book 'The elements of User Experience' by Jesse James Garrett (Garrett, 2010). This section ends with a number of techniques that can help improve the UX.

2.2.1 Traditional Usability Evaluation

Traditionally evaluating the usability of a product does not come into play until the project is finished or nearly finished. Once finished, end-users would be asked to sit down and perform certain tasks on the system. Their performance would then be measured, often using the five usability attributes of Nielsen: Learnability, Efficiency, Memorability, Errors and Satisfaction (Nielsen, 1994). This way of usability testing does not involve end-users until very late in the project and focusses on the product rather than how the *user experiences* the product.

"Racing toward launch without looking back might have seemed like a good idea back when the launch date was set, but the result is likely to be a product that meets all the technical requirements for the project but does not work for your users. Even worse, by tackling user experience evaluation on at the end, you might end up launching a product that you know is broken but have no opportunity (or money left) to fix." (Garrett, 2010, p. 158)

This way of usability evaluation is often called 'user acceptance' testing. The word *accepting* is key here, indicating that the focus lies on whether the client accepts the product, rather than whether he can actually use the product.

2.2.2 Focus on the user experience

Technological and social advancements in mobile and ubiquitous computing have created an opening for human-computer interaction to be a large part of our everyday life. This has caused a shift from traditional usability engineering to so called User Experience, where the feelings and motivations of the user are given a more central role than traditional usability metrics such as efficiency and effectiveness.

User Experience is a rather ambiguous term though. There have been surveys about defining User Experience amongst UX professionals that have proven inconclusive (Law, Roto, Hassenzahl, Vermeeren, & Kort, 2009) and these are just within the scope of Human-Computer interaction. One could argue that the principles behind UX could be applied to any production process. The closest we can get to a definition is written in the ISO 9241-210 standard on the Ergonomics of human system interaction - Part 210: Human-centered design for interactive systems (ISO, 2008):

"A person's perceptions and responses that result from the use or anticipated use of a product, system or service"

This sentence emphasizes the experience the user has whilst using or anticipates using a certain product. Technically 'perceptions and responses' covers everything from smell to touch to emotional processes so it's no wonder there is confusion about the terminology. The concept of 'anticipated use' empowers the idea that UX can play an essential role in the early development stages of product development meaning that studying UX before an actual working product is present can provide valuable input.

UX professionals strive to make the product User-Centered. This means that during the development of the product the user is always the central focus point and drives the development process. According to the ISO standard there are 6 principles that ensure your product is User-Centered:

1. The design is based upon an explicit understanding of users, tasks and environments.
2. Users are involved throughout design and development.
3. The design is driven and refined by user-centered evaluation.
4. The process is iterative.
5. The design addresses the whole user experience.
6. The design team includes multidisciplinary skills and perspectives.

Principle two ('Users are involved throughout design and development') implicates that UX teams should involve users in all design phases: not just by running a focus group at the start of design or by administering a survey at the end of design.
For this thesis the fourth principle ('The process is iterative') and the sixth principle ('The design team includes multidisciplinary skills and perspectives') are especially interesting because it closely relates UX to Agile processes, which are iterative by nature and use teams that are multidisciplinary.

2.2.3 Terminology

There are a lot of different job description when it comes to User Experience as explained by Jeff Patton.

It all starts with a problem that needs solving. The people that research users and problems and suggest features that solve those problems are Interaction Designers. Information Architects were responsible for static information but because these days nearly most content is dynamic those two have sort of blended together. People that focus on usability metrics (Efficiency, Learnability, etc.) are called Usability Engineer. The people responsible for the looks of the product are the Visual Designers.
All of these people are part of User Experience and most of them specialize in one area but also know quite a bit about other branches of the User Experience tree. An Interaction Designer, for instance, will likely know quite a bit about Usability as well as the visual aspect.

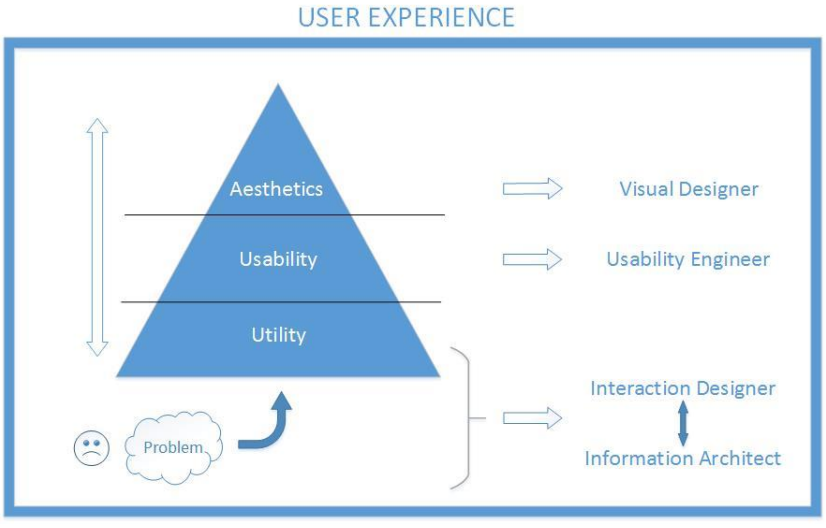


Figure 3 – Derived from an explanation by Jeff Patton (source: https://www.youtube.com/watch?v=-wOLMVbFGCQ)

2.2.4 The Elements of User Experience

Jesse James Garrett has written a book called 'The Elements of User Experience (Garrett, 2010). Here he describes five planes of User Experience: The Surface Plane, Skelton Plane, Structure Plane, Scope Plane and the Strategy Plane. Each of these planes can be divided in a functionality side and an information side.

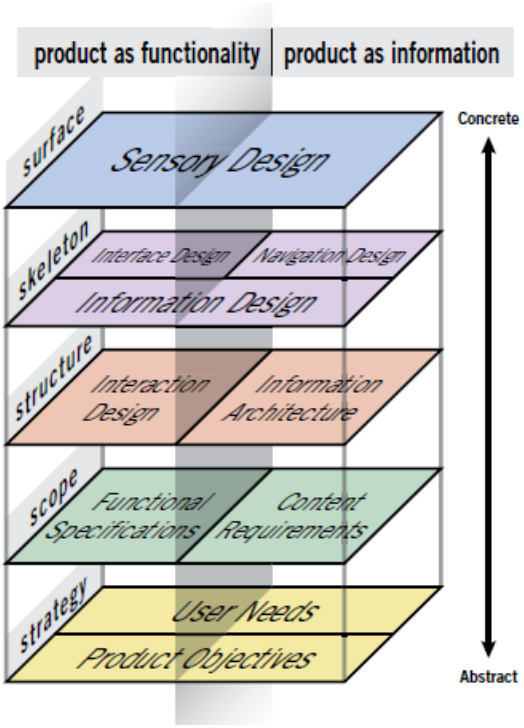


Figure 4 – The Elements of User Experience (Garrett, 2010)

2.2.4.1 Strategy Plane

The strategy plane is all about basing your strategic decisions on knowing what the product has to accomplish for the organization, what it has to accomplish for the end-users and how it will affect the User Experience. You should explicitly state what the client wants from the project and what the users want from the project. A client might want to increase his revenue or get more visitors to visit his website. Most of the times these wishes stem from some business goal and can be measured by means of metrics (such as a visitor count of the website). It is important to note that brand identity is inescapable. Users will always form an impression about an organization based on their interactions with the product, be that employees or external users. It would therefore be wise to make that impression a result of conscious choices rather than making it happen by accident.

To know what users want to get out of a product we need to understand who they are by means of User Research. This can be quite a daunting task because it is not always clear on first glance who your users will be. User segmentation, dividing your users with certain key characteristics they have in common, can help to give you a good overview of your users. Often-used characteristics include demographic and psychographic similarities but you can segment your users any way you want. You only need as many segments as there are different sets of user needs. Once you have got your segments clear you have the option to focus on one single segment or provide a way for different users to accomplish the same goal.

To get to know who your users are you can use market research methods such as surveys or focus groups. These methods can help you find out what your users are doing when they use a particular feature and why they're doing it. Clearly formulating what kind of information you want to get will help in using these techniques.

User testing is one of the most commonly employed forms of user research. Here you let a user test your product while monitoring him live. The product can be a fully operational system but also a variety of prototypes such as rough sketches, 'lo-fi' mockups of 'click-through' prototypes (Garrett, 2010) that create the illusion of a finished product.

To get a grip on how users see the information you present in your product you can use Card Sorting. Here you let users group cards that have a name, description and a piece of content on it. Analyzing how users organize these will help you get an idea how they think about the information you provide.
To actually use all the information you gather using the aforementioned techniques you create Personas. A persona, or user model, is a fictional character that you put together using the gathered data. They have a name, age and a certain backstory and represent one of the user segments you have established. These personas help the development team to take in account who they are developing for.
It is important that all participants of a project - designers, developers, managers – have a broad understanding of the strategy document so they can make informed decisions about their work.

2.2.4.2 Scope Plane

Next is the scope plane, where the functional specification document is created. This document contains the requirements for the project. UX comes into play because sometimes the things people think they want are not the things they *actually* want and moreover, people often do not know what features they want. A group brainstorming session with people from different parts of the organization and representatives of the various user groups can effectively open possibilities that were not considered before.
User stories are a way to clarify the user needs to the project team by means of a sentence that states the role of the user as well as the goal or desire or by means of a small scenario.
When looking at content you should always focus on the purpose rather than the format. To get some sense of how the user will experience the product before it is finished it is wise to make estimates of all your content requirements such as word counts for text features, dimensions for images or file sizes for downloadable content.

2.2.4.3 The Structure Plane

Developing a conceptual structure shifts the focus from abstract issues into more concrete ones. Making decisions about performing and completing tasks is called interaction design whereas information design deals with conveying information to the user.

Interaction design is all about anticipating what a user will do next. An important factor in this area is consistency. When a user works with a piece of software they should expect is to at least be consistent. Conceptual models can help with this challenge, getting a grip on how the system handles certain objects and features can improve consistency. Using conceptual models users are already familiar with, such as conventions, will improve the overall UX. Correctly preventing and handling errors is another important factor in contributing to a positive experience for the user.

In order to structure the information used in the project it might fruitful to create a categorization scheme. In such a scheme you group and organize the different concepts used in the project in order to get the terminology clear and consistent for the project team. This terminology can, for instance, be used for metadata to provide a smooth experience for the user.

2.2.4.4 The Skeleton Plane

This plane is all about the positioning of elements and design. Details will not be discussed because it falls outside the scope of this thesis.

2.2.4.5 The Surface Plane

The user experience of a product is the combination of our five senses: smell, taste, hearing, touch and vision. The first two do not apply much to software development. Hearing has a role to play (i.e. playing warning sounds when something goes amiss) and touch is a nice way to give feedback when using touch-enabled devices but the most important sense when it comes to software development is vision.
It is important to note here that, when talking about user experience, we're not simply talking about the product looking good (good being subjective here). We should ask ourselves how effectively the design supports the objectives designed by the lower planes.

Although vital for the User Experience the surface plane, like the skeleton plane, is slightly outside the scope of this thesis and so will not be handled in-depth.

2.2.4.6 The Elements Combined

Thinking about the people who are going to use the actual product from the start of the project is vital in order to achieve a good user experience. These five planes can help a team to keep the user in mind throughout different phases of the project. It is not necessary for one single person to be responsible for each of the different planes. By having at least one person responsible for each of the planes you ensure that User Experience will be on the minds of the development team from start to finish.

When thinking about the User Experience small problems can get disheartening. Solutions to one problem will make you rethink other problems that you had already solved. Compromises and trade-offs will have to be made. Ground your decisions in the understanding of the underlying issues at play and always ask yourself: "Why did we do it that way?". Having the right frame of mind is most important when working on the User Experience, every other aspect can be adjusted to fit the time, money and people that are available.

Getting the information you need from your users can be a daunting task. More often than not people themselves do not know what they want or why they think something is difficult to understand. It is the job of the person responsible for the UX to understand the needs of the users better then they understand those needs themselves.

In order to deliver a good User Experience you have to make sound choices at the first few planes (Strategy, Scope and Structure) and keep those decisions in mind when designing the product in the later planes (Skeleton and Surface). Focusing too much on the visual design might yield you a good-looking product but if it's utterly unusable no user will ever want to work with it.

2.2.5 Techniques used in UX

There are a number of techniques that can be used to effectively test the UX of the product while it is still in development. The techniques described below answer our second sub question: *What are the best-practices for creating a good User Experience?"*

2.2.5.1 Prototyping

In the field of User Experience a prototype is most often a model of the UI of a product without, or with limited, functionality. This ranges from fully electronic mock-ups with a functional interface you can click through to paper wireframes where a person simulates the behavior of the system.
Users are asked to perform various tasks and the UX expert takes notes of any problem that might occur whilst performing these tasks. The feedback from these types of users test can prove very valuable to detect any problems in the UI. The earlier this form of testing can be conducted (in the form of paper wireframes) the earlier any problems can be worked out.

2.2.5.2 RITE

Rapid Iterative Testing and Evaluation (Medlock, Wixon, Terrano, Romero, & Fulton, 2002) is a technique to quickly get user feedback. The key is to test one user at the time and make changes in between test session. So if you notice a problem after the first test, immediately make changes so the next user is already testing a slightly altered prototype. This works really well with paper prototypes because they can be adapted on-the-fly with very little effort. It is still possible to use this technique with electronic prototypes or working software but those will require some more time to implement the changes.
The hard part is building up a big pool of ready users. If you are working on a project that already has a large user base you could try to have those users opt-in to (remotely) test prototypes and working software.

2.2.5.3 Story Mapping

A story map is a map consisting of user stories in a certain arrangement. At the top of the map there are so called 'user activities'. These are big stories consisting of something a person might do, something that has lots of steps and not necessarily a precise workflow, for instance: managing email. Below that are 'user tasks', smaller tasks that someone does to reach a certain goal, for instance: deleting an e-mail. If you can divide there user tasks in even smaller sub-tasks they go further down (Patton, 2008)**.**

The stories are arranged by time, from left to right. Something like 'logging into the system' would appear further near the left because it tends to happen before some of the other tasks. This temporality is not very strict but is supposed to give you an idea of how someone would use the system.

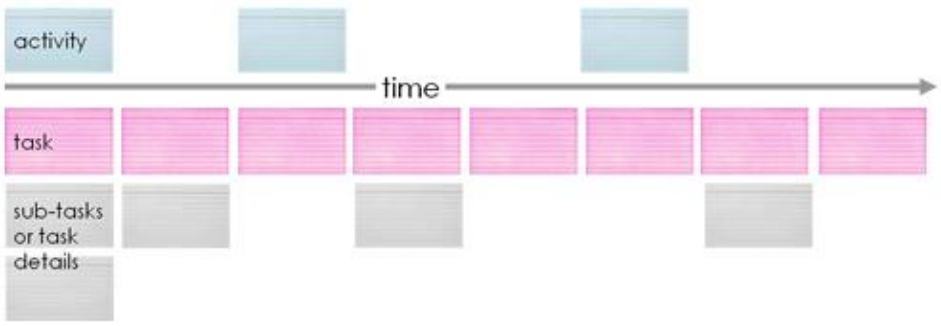


Figure 5 – The new user story backlog is a map (Jeff Patton, 2008)

The stories at the very top of the map are the backbone of your project, it does not require any prioritization as all of it is necessary. Just below the backbone are the user tasks that are absolutely necessary to get the system working. Together these stories describe the smallest possible system you could build that would give you end to end functionality. Alistair Cockburn refers to this as the 'walking skeleton'. You prioritize the story map by placing stories higher or lower on the map. Stories that are higher up are deemed more important than stories lower on the map.

2.2.5.4 Personas

As explained briefly in section 2.2.4.1 personas characterize a certain end-user. You create a persona for each of the end-users you have defined in your User research. A persona usually has a name, a picture, a description of who that person is and what that person values in the system you are building. Each of these personas are then printed and displayed near the area where the daily stand-up takes place. When certain user stories come up for discussion these personas can help steer the discussion and makes the team think about how a certain end-user might react to the problem posed.

The more team members you can get involved in creating the personas the better. This helps to get the entire team involved in thinking about the end-users rather than just the designers and project managers.

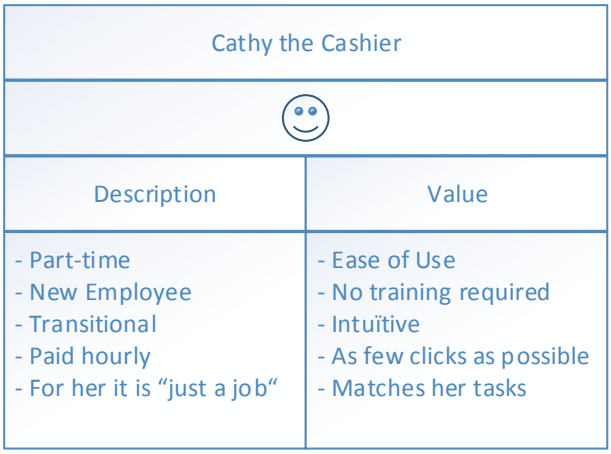


Figure 6 – An example of a pragmatic persona (template by David Hussman)

2.2.5.5 *Experience Canvas*

The Experience Canvas is based on the Business Model Canvas (Ostenwalder & Pigneur, 2010), which is a visual poster you hang on the wall during meetings that shows the value proposition, infrastructure, customers and finances of a company. The idea is that the canvas is filled with sticky notes and everyone is encouraged to question what is written on the sticky notes and suggest a change accordingly.

The Experience Canvas (Crothers, 2013) is inspired by Ostenwalder's Business Model Canvas and fulfills the same purpose: It helps to keep people aware of their environment, in this case the User Experience of the project. By hanging the canvas in the room where the team is located it becomes a talking point for the team that sparks discussion. As the project progresses and changes so will the Experience canvas change and keep people thinking about the User Experience.

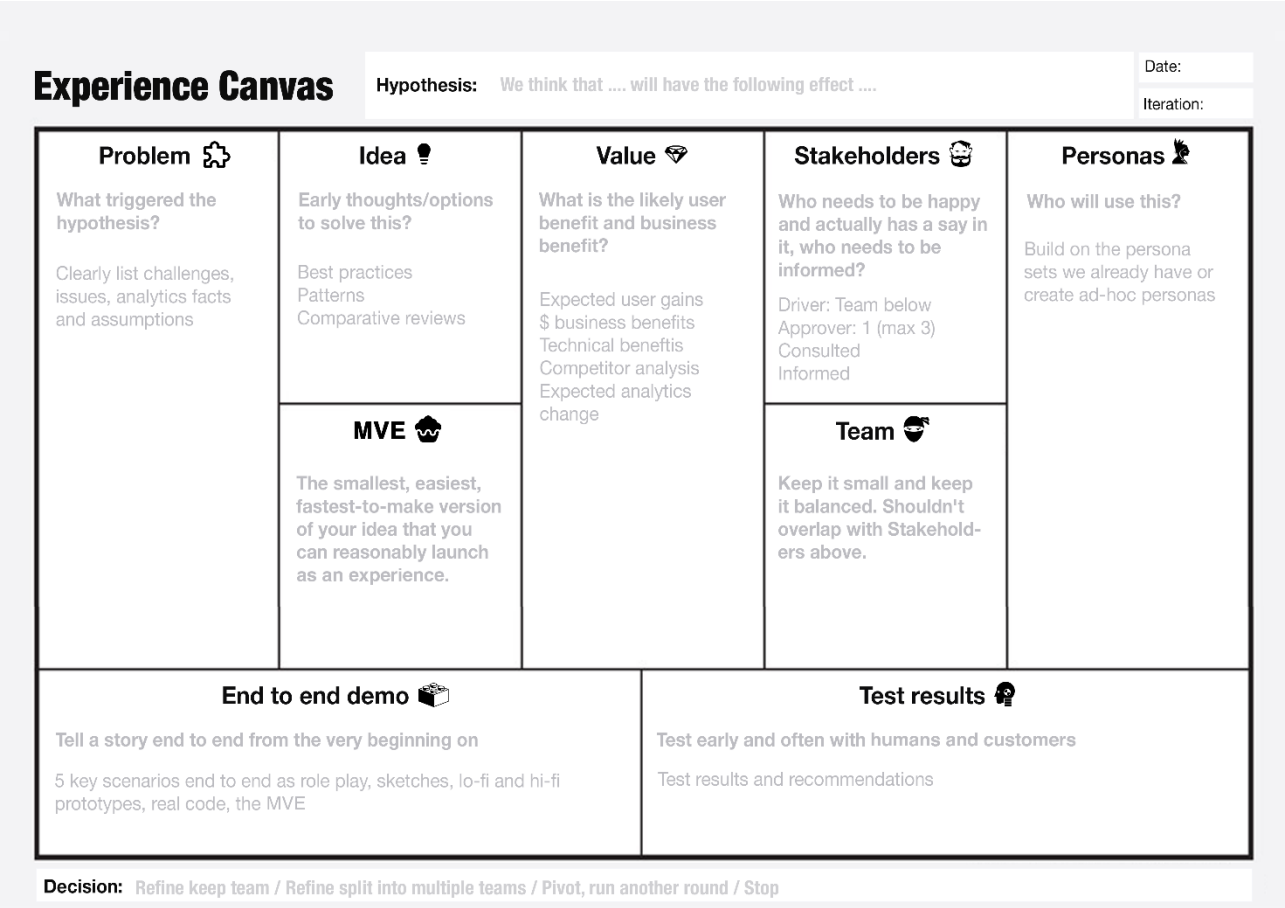


Figure 7 – The Experience Canvas by Ben Crothers (source: https://blogs.atlassian.com/2013/10/fight-the-dark-side-of-lean-ux-with-the-experience-canvas)

2.3 Integrating UX and Agile

"Agile methods enable software development teams to create software that is valued by the customer. UX design methods allow software development teams to create software that is usable for the user" - Jennifer Ferreira (2011).

Effectively combining the two fields means a win-win situation. This chapter takes a closer look at the integration of UX and Agile. It starts by listing a number of known difficulties that persist within the literature. It then continuous by discussing some key areas that need attention when trying to integrate UX and Agile.

2.3.1 Known difficulties

This chapter aims to answer the third sub question: "*What are the problems when integrating Agile Development and User Experience?".* There are a number of known difficulties in the existing literature. The most prevalent difficulties are listed here (Miller & Sy, 2009). Even though there are many challenges in integrating UX with Agile development methods there is also an opportunity to bring UX design closer to software engineering and enhancing the interaction between these two disciplines.

Sprints are too short

There is not enough time in a single sprint to finish designs, test usability or set up customer contact. Traditional User Experience tasks often stretch over more than the duration of a single sprint. Developers are often waiting on designs before they can move on, this leads to designs that are finished in a hurry which has a negative effect on the quality. Because of the lack of time these designs are often unverified and untested. A way to remedy this would be desynchronize some of the work, meaning that some of the UX work is done one or two iterations ahead.

Not enough user feedback

If you follow traditional usability techniques feedback comes too late in the process. Conducting user or usability sessions and field studies requires making appointments sufficiently in advance en typically have a lead time of several weeks. It is not feasible to schedule user tests in the same iteration, even though the information gained would have benefited from a more just-in-time approach.
Even if user feedback is acquired it is not always used. If the feature set is cast is stone, and no re-ordering is allowed than this could mean that changes are implemented too late, or not at all. Usability testing easily gets overridden by urgent development goals (Isomursu, Sirotkin, Voltti, & Halonen, 2012).

UX is not full-time on one Agile team

The User Experience people spend their time in many meetings instead of on designs and iterations and are often demoralized by the drop in quality this inevitably results in . Studies show that when UX specialists are members of the scrum team they were better able to use incremental and agile ways of working whereas this agility was not achieved in centralized UX design (Miller & Sy, 2009). Having the UX team centralized is often preferred by UX managers as it gives them more control over the UX work. On the other hand, UX professionals may value decentralized UX work as it gives them better visibility and opportunities to influence the design decisions early during the development process. Furthermore the communication with the development team is far better if they are co-located. This improves the understanding the UX professional has of the project increasing the quality of the work. If UX is not a part of the scrum team 'gap analysis' has to take

place, which compare's the designs to the implemented software and identifies mismatches between them. However, this takes valuable time.

Missing the 'big picture'

Because functionality comes in small pieces there is hardly any vision or end-goal and too much focus on details. This makes it hard to prioritize of make design decisions. This becomes an even bigger problem when new people join the team and old project members leave.
User Story mapping (described above) helps to keep the eye on the big picture because it provides an insightful map of the entire system.

2.3.2 UX centralized VS decentralized

Research has shown that having UX experts in the Agile team is beneficial for the development process (Najafi & Toyoshiba, 2008). When UX professionals are present in the same room as the rest of the Agile team this encourages face-to-face interactions. Participation in the Sprints and Scrums is vital when focusing product development on the User Experience.
Decentralized teams lack the face-to-face communication that keeps everyone on the same page. Often they would use shared documents and tables but those can be misinterpreted and it could potentially take a very long time before the teams even realized they were not on the same page.

2.3.3 Synchronized sprints VS desynchronized sprints

There are two options when considering having UX work within the confines of a sprint (Najafi & Toyoshiba, 2008). The first option is to have them work within the same sprint as the development team. This means that the UX work (design, user research, testing, etc.) for certain features takes place within the same sprint those features have to be implemented. The advantage of setting up your team this way is that both the UX and the developers are working on the same features at the same time leading to meaningful discussions and a general consensus that the entire project team is on the same page. The risk however is that the UX team does not have enough time to finish their work because of changes during the sprint meaning features might have to be completed in the next sprint.

A second option would be to have the UX team work ahead. This would mean designs would already have been tested before the feature is implemented meaning the development team would not have to wait on the UX team to continue with their work. The UX team would still participate in the daily scrums so any questions or misinterpretations could be dealt with swiftly. The UX team however has to realize that their work does not end when they hand off the designs to the development team and have to be equally committed to the sprint the development is in as they are to their own sprint.

Sprint 0 is a term that is often coined to indicate the work that takes place before the actual development work starts. Typically this is when the initial requirements are gathered and the product backlog is identified en prioritized. The developers use this sprint to decide what environment or platforms are going to be used. The UX team uses this time to get a better understanding of the users and their context.

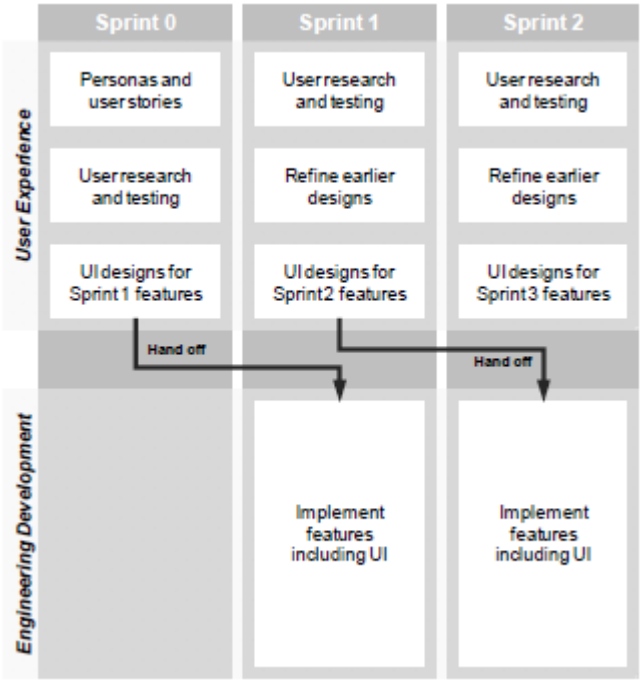


Figure 8 – Two Case Studies of UX Design and Agile Development – (Najafi, Toyoshiba, 2008)

2.3.4 Culture

Combining two different fields of expertise requires a certain mentality from both sides. In order for the merge to be successful all participants must want it to be successful. In the beginning there will inevitably be frictions but this is part of the learning process. If, for instance, the UX team only creates designs, delivers them to the developers and then get on their merry way it will be very difficult to get any sort of integration going. Active participation in all aspects of the project is required to make the merging of Agile teams and User Experience a success.

3 The Problem Space

This chapter focusses on the problems that are encountered when trying to integrate User Experience in an Agile environment. Chapter 4 will offer possible solutions for these problems.

The problems found in the literature study can be summarized into three main problem areas:

I. The Big Picture
II. Communication
III. User Focus

This thesis aims to give you the best possible chance to deliver a product that the user will experience as functional and pleasant to use. It does so by integrating a holistic view or 'an approach to design which considers the system being designed as an interconnected whole which is also part of something larger' (WiseGeek, 2014) into the design process. Every development project starts with a problem that needs to be solved. Rather than just following the specifications of a client and create his solution to the problem this method aims to get the whole development team to take a critical look at the problem at hand and the environment in which the solution has to be placed. According to Rahun Sen (Sen, 2010) this holistic view is not easy to come by. To get a good understanding of the environment the product is placed in you need opinions from people with various backgrounds. Luckily in software development people with various backgrounds are readily available: developers, managers, designers and of course users are all involved in creating a piece of software. The hard part is to get the entire project team aware of these views and preferably actively discuss them to get more input and an even broader understanding of the problem space. We call this '**(I) The Big Picture**' and it is the first problem area. Chapter 4.1.1. goes into detail on how you can solve the issues in this problem area.

The reason this broader view is necessary is because Agile teams inherently tend to focus on the smaller details of the project as they work in short iterations creating small pieces of functionality at a time. Working in this fashion makes it easy to lose sight of the project as a whole. Integrating the big picture into Agile methods allows the team to switch between the broader view and the small details. This way the project team will not get tunnel vision working while on details but will be able to place them in a broader context ensuring the work they do benefits the project as a whole.

In order for such a holistic view to stay relevant it has to be adjusted during the entire project. New insights or new viewpoints could very well have an effect on the problem, the solution, or both. This leads to the second problem area: '**(II) Communication**'. Communication is essential if you want the team to work collaboratively (see 2.1.2). New insights might get lost if they're not openly discussed or cause members of the project team to have a different mental images of the same problem leading to inconsistencies in the software negatively effecting the User Experience. The importance of good interaction is emphasized by the agile manifesto in their first statement: 'Individuals and Interaction over processes and tools' (see 2.1.1).

The third and final issue is **'(III) User Focus'**. In order to get the best User Experience you have to know who is actually going to use the software and continually ask for their input during the project. Traditionally end-users are only consulted at the beginning of a project, to see what problems they have, and at the end of the project, to test the usability. This leaves very little time to fix any problems that might occur during the usability tests. This method aims to get user input while the

project is still under development in order to catch potential problems earlier in the project when they can be solved relatively easily. More on this in chapter 4.1.3.

Below is an overview that maps the proximity of the concepts and techniques that have come forward in the literature study to the three issues:

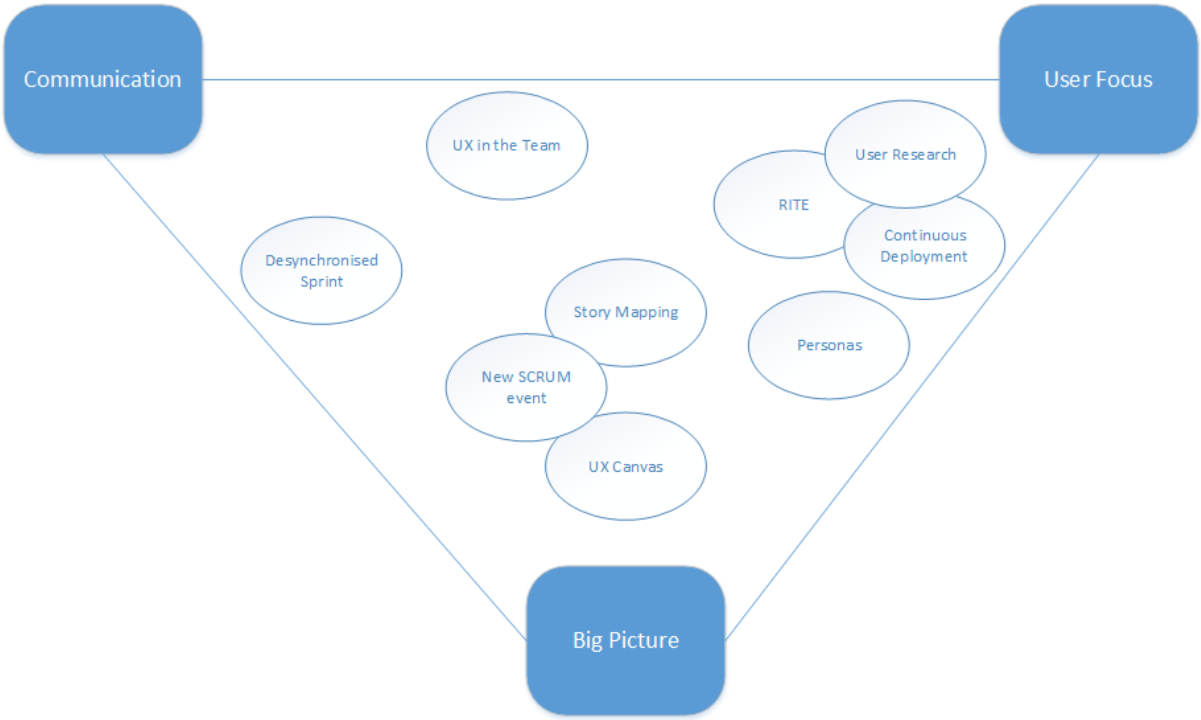


Figure 9 – Problem/Solution Space

In this image you see the collection of concepts and techniques mentioned in the previous pages of this thesis mapped in a two-dimensional space. The closer one of the concepts or techniques is mapped to one of the three issues the more it can help solve that particular issue. 'User Research' (2.2.4.1) for instance can help a project team get a good grip on who their users are and thus help them create a good experience for those users. However it does little to improve the communication within the team or help the team with their mental image of the big picture. Story mapping (2.2.5.3) on the other hand can help solve all three issues. Having a story map on the wall sparks discussion thus improving communication, seeing how the stories relate to each other gives you a better idea of the problem you are trying to solve (the bigger picture) and the temporal nature of the map gives you an idea of how a user would use the software you are developing (user focus).

How exactly these methods and techniques helps solve some of the problems within each problem area is explained in the next chapter.

4 The Solution Space

In this chapter each problem area is discussed in detail. Techniques and methods that have been found in the literature study are used to solve the issues of each problem area. These techniques and methods are explained in chapter 2 and this chapter will refer to them when necessary.

4.1 The Big Picture

There are two parts to be considered when thinking about 'The Big Picture'.

I. Context of the problem
II. Context of the solution

4.1.1 Context of the problem

The first part is making sure you are fixing the right problem by looking at the environment of the project. Think about stakeholders, a description of the problem (or problems?), who is eventually going to use this piece of software, etc. Creating the right piece of software is essential in order to satisfy the user. After all, if your solution does not fix the problem of the user it is meant for, their experience of the software will not be satisfactory. The goal is to get the entire team aware of the problem they are trying to solve, and instead of it being set in stone allow the problem statement, as well as the solution, to be subject to change as the project progresses.

The Experience Canvas (2.2.5.5) is a helpful tool when trying to achieve this goal as it forces you to write down a number of things mentioned above that help define the environment of the solution such as the problem, stakeholders and users but also makes you write down any idea's that might help solve the problem and what exactly the added value is of what you are trying to build.
The whole idea of the canvas is that it is hung on a wall in the room the team is working and is subject to change at any moment. If any member of the team thinks that what is written on the canvas is no longer up-to-date he can put it up for discussion at any time and if the rest of the team agrees it can be changed to better reflect the current situation. The power of the Experience Canvas lies in the fact that it is a constant presence that is subject to change and resembles the reason why you are doing what you are doing making you aware of 'the big picture' and consequently helps solve a common problem of Agile teams:

"One of the biggest problems we have seen on many agile teams is forgetting the real business value of the feature they are developing. We call this risk "forgetting the big picture". Teams break up a feature into tiny stories and then make sure they test the functionality on each individual story. Even with all the testing done for each story, the team may deliver a feature that has outstanding issues, or does not meet the customer's expectations. We have to balance the fast feedback of working in small increments and iterations with ensuring that the overall feature delivers adequate business value" (Gregory & Crispin, 2012).

4.1.2 Context of the solution

The second part is about knowing where the feature fits into the system as a whole. Story mapping (2.2.5.3) can help the team put their current work into a broader perspective by giving a visual representation of where the story fits into the system as a whole. Because the story map is temporal from left to right you can see where in the timeline the story sits and what steps the user has already taken to get to that point. This really puts you in the mindset of the user thus helping the User Experience as well. By looking at the vertical axis you can quickly see what activity the task you're

working on is a part of. This enables the team to keep the big picture in mind, even when working on little pieces of functionality.

4.2 Communication

One of the focus points of the Agile Manifesto is 'Individuals and Interaction' (see 2.1.1). In order for an Agile team to be able to focus on the end-user all team members have to collaborate. To achieve this they have to communicate openly and often. Communication by means of documentation is slow and ambiguous, often raising more questions than it answers (Jan & Tabrez, 2013). Documentation in and of itself is not bad as long as it is precise, on-point and relevant. Agile values 'Working software over Extensive Documentation' (2.1.1). This, in combination with the fact that Agile is people-oriented methodology, means that there is more interaction between developers, customers and users and as such less need for documentation that needs to be maintained and updated. It is essential that the team is working in an environment where the lack of documentation can be countered by the increase in personal communication.

4.2.1 Co-located team

To create such an environment all the project members should be situated together in a single room. This encourages face-to-face communication and removes the ambiguity that stems from documentation such as e-mails and the like. Furthermore, if someone on the team has a problem and needs help he can simply ask a team sitting across or next to him, increasing collaboration. This way little UX problems such as inconsistencies in nomenclature or in the way certain elements behave can quickly be solved simply by asking other team members how they have done these things.

It has proven beneficial to have the person responsible for the UX of the project be situated with the team (2.3.2). Keep in mind that in Agile teams people should have multiple responsibilities meaning that even if someone is specialized in UX he should also contribute in other ways, for example by testing various functionality or even help with developing. The benefit of this situation rather than the UX being centralized somewhere else is that the UX expert is far more connected to the project, is quickly notified about any problems that might occur with the UX and can fix them just as quickly because he is in constant communication with the rest of the team. The disadvantage is that a form of tunnel vision might occur. In such a situation the UX of the project might depend too much on one person's ideas. A solution to counter this is to continually ask for input from the rest of the team as well as ask for peer-reviews from UX experts from other projects.

4.2.2 De-synchronized sprints

There are two ways for a UX expert to work within an Agile team:

I. Work within the same sprint as the team (sprint n)
II. Work one sprint ahead (sprint n+1)

UX work such as testing or designing typically starts after or before a piece of functionality has been completed. Testing a piece of functionality when it hasn't been is a bad idea because you would be testing something that is still subject to change. Waiting for the functionality to be finished means testing cannot begin at the beginning of the sprint but rather halfway through or maybe not at all if the work is not finished before the sprint ends. Even if the functionality is finished within the sprint it leaves only a few days for it to be tested within the same sprint. Hardy an optimal situation.
The same goes for design work. Although it is entirely possible to develop and design at the same time it often helps to have some rudimentary design, such as wireframes, finished before development starts so the developers have a visual image to work with.

This makes it seem as though option (II) is the best choice. Working one sprint ahead of the development team means you can test functionality they have finished a sprint before as well as make designs for functionality that is planned for the next sprint. This way of working however also has its drawbacks. It detaches the UX expert from the problems at hand losing some of the synergy that you would get when working in the same sprint as the rest of the team. It would also mean that test results from users will not be available to the team until two sprints later and any adaptations would have to wait that long to be implemented.

In the end it will depend on the team which option to go for. If the team has a quick way of getting test results (such as Continuous Deployment or ready-and-waiting end-users to test) it might prove beneficial to have the UX work in the same sprint. It could even be an option to have the UX expert work ahead some of the time and work within the same sprint the rest of the time, Scrum is an Agile environment after all.

4.2.3 Story Mapping / Experience Canvas

As stated in 4.1.2 Story Mapping is useful for keeping the Big Picture in mind but it also helps the team communicate because the story map is a constant presence in the room. At any given time a member of the team can look at it and if anything is unclear can ask someone in the room for clarification. This constant communication helps to form one mental image of the project that is shared by everyone on the project team.

Although the Experience Canvas, like Story Mapping, is useful for clarifying the Big Picture it really shines when it comes to communication. As explained in section 4.1.1 the Experience Canvas is meant to be a tool for discussion. If a team member has any doubts about any of the subjects that the canvas covers they are encouraged to speak their mind and see what the rest of the team thinks of their comments, thus sparking discussion. And just as with story mapping these discussions make sure the team is on the same page. Where the story map mostly sparks discussions about functionality the Experience Canvas sparks discussions about the what you're building and who you're building it for, essential questions when trying to achieve a good user experience.

4.2.4 SCRUM event: UX meeting

In addition to the already existing Scrum events another event can be added: the UX meeting. This meeting would take place after the Sprint Review and is used to evaluate the test results from the UX testing that has taken place the sprint before. This way, any problems that are uncovered can be added to the sprint backlog prior to the sprint planning.

Issues that have arisen during the sprint review could lead to realizations that have an effect on the Experience Canvas, so besides reviewing test results the entire team should take a minute to gather around the Experience Canvas and discuss whether it is still up-to-date.

4.3 User Focus

One of the most important aspects to integrate into the Agile way of working is to get the team to focus on the user throughout *the entire* development cycle. Often the end-user gets some attention at the beginning of the project, when the requirements are taking shape, and at the end of the project, where traditional usability testing takes place. This thesis is centralized around the idea that the end-user should be a focus point for the *entire* team (not just the designer) throughout *all* stages of the project.

This chapter about User Focus can be divided into three different parts:

I) User research
II) User testing
III) User presence

4.3.1 User research

User research takes place in the very beginning of a project. In chapter 2.2.4.1 some methods and techniques that are mentioned in 'The Elements of User Experience' (Garrett, 2010) are discussed. It is impossible to focus on getting a good user experience if you do not know who is going to use your product. This means it is imperative that you get to know your user(s). First you define what different types of users the system should accommodate. Then, for each of these types of users, you need to get a good understanding of their frame of reference regarding the product you're making. This means thinking about their needs and motivation. Being thorough in this stage helps prevent problems down the road, when the software is put into production.

4.3.2 User Testing

Traditional Usability testing entails having a testing session in a usability lab when most of the project is finished. Putting the focus on the end-user means asking them what they think, as often as possible. The idea behind this is that if you ask your users for input early on you can prevent a lot of difficulties such as superfluous functionality or a hard-to-use interface further down the road. The best way to do this is by using Continuous Deployment (CD)(2.1.4.11). Using CD means that any code you commit will be live in minutes and will be used straight away. Because new functionality can be used straight away you are able to gather statistics quickly and find out in what way new functionality is being used. This information can then be used to change, for instance, the design and see if any improvement takes place. CD, however, requires an advanced infrastructure and will certainly not be the best solution for many projects. For example, updates to critical systems such as banking software need to be tested extensively to ensure nothing will go wrong.

In order to still get input form the users quickly without having to use CD you can use Rapid Iterative Testing and Evaluation (RITE). As explained in chapter 2.2.5.2 RITE enables you to quickly test your software in any stage without having to use extensive tooling to do it.

4.3.3 User Presence

A constant user presence can make sure the team keeps in mind who they are creating software for. This constant presence can be established by means of personas (2.2.5.4) that you place on the far right of the Experience Canvas. These help the development team keep in mind who their end-users are even if you place them out of the context of the experience canvas. When developing team member can ask themselves how persona X or persona Y would respond to certain functionality or if the sequence of events fits their expectations. They also help clarify UX problems by giving examples. Image a situation where a developers has labelled a certain button something that seems obvious to him, but maybe not to persona X who has a very different frame of reference. These issues are easy to fix when these personas are readily available but might be hard to notice if they're not.

5 Heuristics

The following ten heuristics are the result of combining the fields of User Experience and Agile Development and analyzing the problems and solutions that prevail in the literature. They are the result of extensive research into both these fields and are a translation of the work done in chapters 3 and 4. This means all ten heuristics can be linked to one or two of the problem areas defined in chapter 3 and are grounded in the literature described in chapter 2.

1. **IT projects need to focus on producing software to solve a problem rather than working out a solution proposed by the client.**
   The client is not the most suitable person so the project team should be critical and first review what the actual problem is instead of blindly building what they are being told.

2. **In order to successfully integrate a software solution and create a good User Experience you need information about the environment it will be deployed in.**
   If you want to build usable software that delivers a good User Experience you need to know how and by who the software is going to be used.

3. **In order to create the best possible solution to a problem and deliver the best User Experience you need input from as many different stakeholders as possible.**
   Gathering information from different stakeholders allows you to get a complete picture of the problem area and enables you to weight arguments against each other.

4. **Developers in Agile teams focus on the details of a solution and often lose sight of the project as a whole.**
   A side-effect of working on small pieces of functionality at a time is that you lose the holistic view of how it all fits together.

5. **When working in an Agile team, established insights should always be subject to change.**
   Discussion should be encouraged and project members should not be afraid to question anything they think is wrong.

6. **Face-to-face interaction between project members lowers ambiguity and increases the quality of the solution.**
   Being able to immediately en directly ask someone about a problem clears up things much faster and makes sure the team is on the same page.

7. **A good User Experience is essential when creating software that is going to be used daily.**
   Good software not only delivers all the functionality that is required but also makes sure it is usable and pleasant to work with.

8. **Thinking about the motivation and needs of your end-users helps to produce a better User Experience.**
   Knowing how your users behave and what their preferences are helps you build software that is pleasant to use and allows you to create a good experience for the user.

9. **Having a UX expert as a team member improves the quality of the software.**
   Decisions about UX should be made by the team itself and the best way to do this is to have a team member actively engaging in UX.

10. **Involving users early on prevents mistakes down the road.**
    Getting input from users that are actually going to use the software you are building helps to avoid creating functionality that is not going to be used or making designs that are confusing.

Justification

Five of the ten heuristics can be traced back to 'The Elements of User Experience' (Garrett, 2010). The first three stem from the idea that you should acquire as much information as possible before trying to create a solution. In this thesis this links to the problem area of 'The Bigger Picture' (chapter 3.2.1). The two other heuristics (7 and 8) stem from the same work and are based on the chapters about why User Experience is useful at all.
Heuristics 4 and 5 are based on articles and lectures from Jeff Patton (Patton, 2008). His work is about helping software development teams overcome some of the problems that occur when working with Agile development. One of these problems is losing sight of the Big Picture and another that requirements tend to be too rigid and there is too little freedom for criticism within Agile teams. A study by (Najafi & Toyoshiba, 2008) showed that software development teams that communicate face-to-face achieve better results than teams that communicate via documentation leading to heuristic #6.
The last two heuristics come from a study that lists a number of common issues that occur when trying to combine User Experience and Agile development (Miller & Sy, 2009). One of these problems was that the UX of the project was done by a team of experts that was not a part of the project itself (heuristic 9). The other problem was that Usability testing took place too late in the process meaning errors could not be prevented.
The table below shows what problem area relates to what heuristic, what literature it stems from and in what chapter in the thesis this literature is covered.

Heuristic	Problem Area	Chapter	Literature
1	BP	2.2.4.1	(Garrett, 2010)
2	BP	2.2.4.1	(Garrett, 2010)
3	BP / C	2.2.4.2	(Garrett, 2010)
4	BP	2.2.5.3	(Patton, 2008)
5	BP / C	2.2.5.3 & 2.2.5.5	(Patton, 2008) & (Crothers, 2013)
6	C	2.3.2	(Najafi & Toyoshiba, 2008)
7	UF	2.2.4	(Garrett, 2010)
8	UF / C	2.2.4.1	(Garrett, 2010)
9	UF / C	2.3.1	(Miller & Sy, 2009)
10	UF	2.3.1	(Miller & Sy, 2009)

Table 3 - C = Communication, BP = Big Picture, UF = User Focus

6 Questionnaire

To validate the heuristics mentioned in chapter 5 an online questionnaire was performed that asked experts in the fields of User Experience and Agile Software Development to what extend they agree or disagree with the heuristics.

Basic demographic was deemed irrelevant for this questionnaire because the questions asked refer to the fields of expertise only and any conflicting answers would not be explained by differences in gender, age, etc. Instead the questionnaire asked about the individual's experience within the fields of User Experience, Agile Software Development and their combination. Because these fields of expertise are quite broad the questions about experience were asked after the questions about the heuristics. This way the participants could get a good idea of their level of experience with the specific subjects that are covered within the heuristics.

The structure of the questionnaire, as well as how it should be filled out was explained beforehand. It was also made clear that the ordering of the questions was random and did not reflect any difference in importance or significance.

The purpose of the questionnaire is to measure the extend of which experts agree with the heuristics. For this reason a 5-point Likert scale was chosen with 'Strongly Agree' and 'Strongly Disagree' at opposite ends of the axis (Sharp, 2003). 'Neutral' was chosen as the middle of the scale to give the respondents an option to fill out if they neither agree or disagree with the heuristic.

Potential respondents were found by contacting researchers of within the fields of User Experience and Agile Software development as well as people working in Software Development that have day-to-day experience with these fields. Furthermore the questionnaire was put up on a forum for people specifically interested in Agile/UX. The questionnaire was online for a total of three weeks and within this period of time a total of 16 respondents filled out the survey.

The full questionnaire is added in Appendix A.

7 Results

A total of 16 subjects completed the online questionnaire. For all of the subjects the affiliation they have with the fields of Agile and/or UX is work-related. For nine subjects this means they are active in Software Development (SD), for one it means Research (R) and five subjects combine both in their work. One subject also claimed working in these fields is a hobby to him.

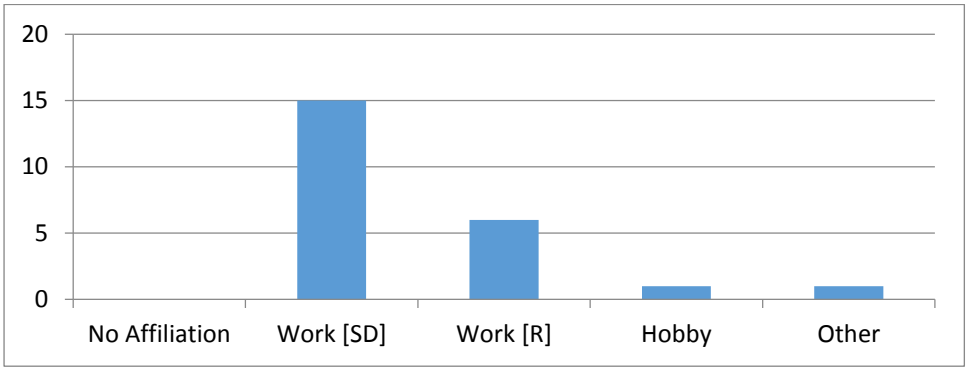


Table 4 - Affiliation with research areas. [SD] = Software Development, [R] =Research

When looking at the experience these subjects have with Agile and UX all of them are highly experienced. There was only one subject that had fewer than one year of experience with both Agile and UX and no experience with combining them. The rest of the subjects all have more than one year experience with either Agile or UX. The amount of subjects that have actual experience combining the two fields is also rather high. Six of them have between 1-3 years of experience, five between 4-6 and there are three subjects that have more than 6 years of experience combining Agile and UX.

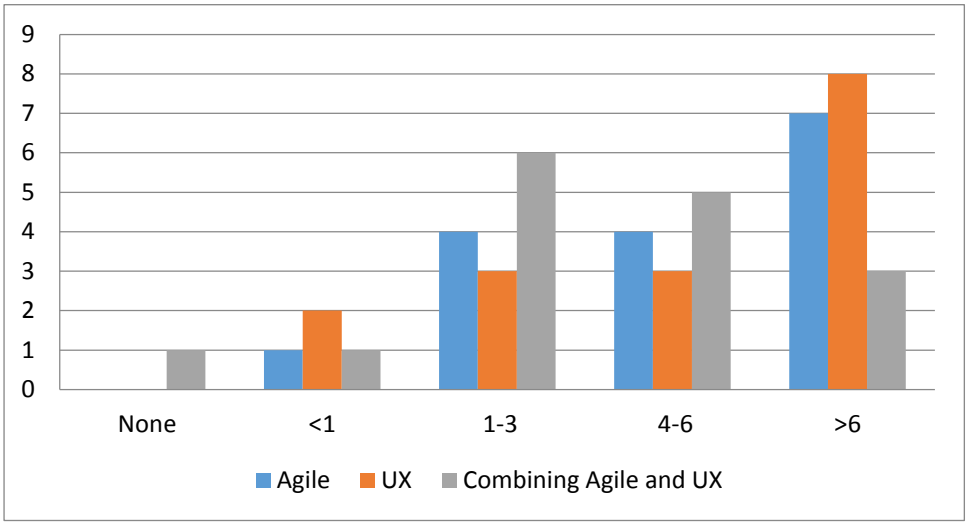


Table 5 - Years of Experience

When looking at how these experts evaluated the ten heuristics that they 'strongly agree' with seven of the ten heuristics and 'agree' with three. The standard deviation for the answers lies mostly between 0 and 1. The only two heuristics that resulted in a higher standard deviation were heuristics 3 and 4, with deviations of 1,29 and 1,26 respectively. Heuristic 2 has a standard deviation of 0, all respondents strongly agreed with that heuristic.

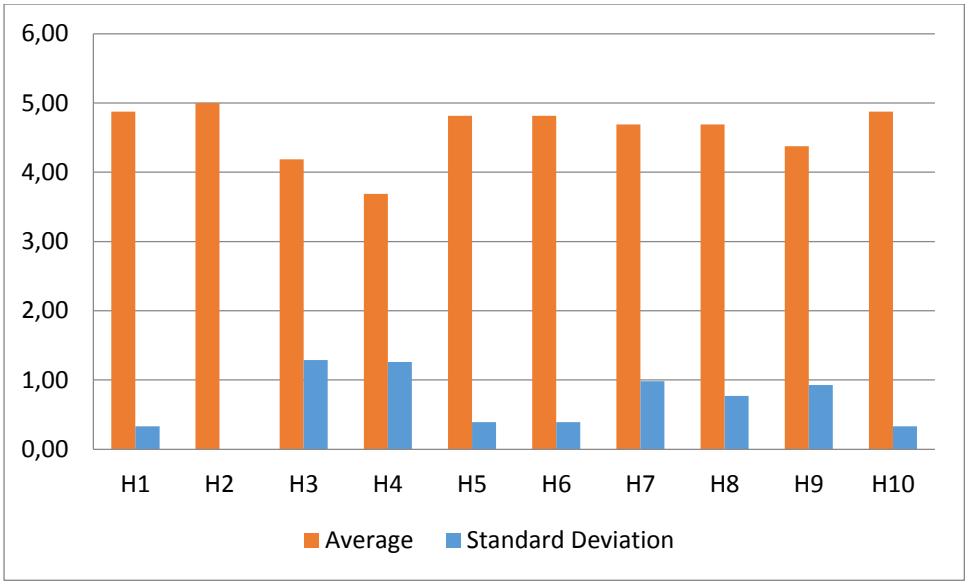


Table 6 - Average and Standard Deviation (1 = Strongly Disagree, 5 = Strongly Agree)

Comments

There were only three subjects that have written something in the open comments area. To the question whether they missed anything in the survey one responded that he was not quite sure from what perspective he was supposed to fill in the survey. Different roles such as 'scrum master' or 'product owner' may lead to different points of view.
The questions were not specifically designed for a single role. If any of the experts have filled out more than one role this will add to their expertise in general and allow them to better weight the arguments when deciding to agree or disagree with one of the heuristics.

When asked about any final comments in general one of the subjects remarked that some of the heuristics were quite obvious. This is certainly the case and while experienced practitioners of Agile/UX might see the heuristics as obvious, less experienced people may not.

One of the subject commented: "*Being open for everyone to challenge design decisions is all good but it can also lead to a lot of inefficiency since it is common that "everybody" (including the client and developers) think they know UX design. It is really hard to explain what UX design is, what value it contributes with and what skills it takes. There is often a need for educating the team in UX. And a need for defining roles in the team.*"

This is a very valid point. This means heuristics 3 and 5 should be taken into account 'to a certain degree' depending on the team and the context of the problem.

Another subject also commented on this issue saying that "'*Gathering information from different stakeholders takes time and money and does not necessarily contribute to a better solution. You can create a simpler solution by only taking into account the most important stakeholders.*"

8 Conclusion

At the start this thesis posed the following as the main research question:

> "*What are the problems when integrating User Experience in an Agile environment and how can you solve them?*"

The main problems that arise when integrating User Experience in an Agile environment can be divided into three problem areas: The Big Picture, Communication and User Focus. There are many techniques and methods that can be used to tackle these problems. For instance, if you're having difficulties with User Focus, you can use personas to create a constant user presence. Problem with the Big Picture can be solved by Story Mapping and the Experience Canvas can help you with the communication within the team. This thesis described the problem/solution space and created ten heuristics based on these findings. The heuristics summarize the knowledge in the rest of the thesis and were presented to experts with many years of practical experience. All of them agreed with the ten heuristics. This means the results of the questionnaire confirm that the theory described in this thesis matches the day-to-day reality of software development in practice.

The heuristics are based on all three problem areas. Heuristic #5, for instance, is based on problems within The Big Picture and Communication while heuristic #8 stems from problems with User Focus and Communication. Following all heuristics will certainly help a software development team to get achieve a good User Experience while working in an Agile way.

This thesis in and of itself does not offer a clear solution to the problem of integrating User Experience in an Agile environment. It does however provide an anchor point for others. This thesis points the finger in the right direction. It does not identify specific, context-driven problems but instead generalizes a lot of these problems into problem areas and discusses what sort of techniques might possibly help with these problem areas. This means there is a large basis that can be used to identify specific problems and help create specific solutions.

The problems and solutions associated with Agile/UX were too vast and extensive for this thesis to fully answer the main research question. The work done in this thesis does however shed some light on the subject and can help people get a better understanding of the subject. With ten heuristics that were agreed upon by experts this thesis has laid a basis for researching the combining of Agile Software Development and User Experience. Although the results are not ground-breaking in a practical sense they do add to the existing scientific knowledge and can become a useful starting point for other research that might lead to some new, uncharted territory.

9 Discussion

Agile Software Development and User Experience are both large and diverse research areas. Both are also very practical research areas. This means that a large portion of knowledge lies with software companies that actually work with Agile and UX and not with universities and institutes that do theoretical research, perhaps with a hint of practicality. This also makes it hard to write a thesis on the subject. Combining Agile and UX is something companies learn by trial and error but most of them will not be writing articles about their findings. There are still a lot of experts that do however, just not in a scientific way. This leads to a situation where you have three types of information:

I. Scientific Literature
II. Articles/Blogs written by practical experts
III. Restricted company knowledge

In the fast-paced world of software development scientific knowledge is often outdated as soon as it is published, especially in new research areas such as Agile and UX. Articles and Blogs by experts vary in quality but there are a lot of them out there. And maybe the problem has been extensively tested by a software company that has found the solution, but we'll never know because they will not share this information. There are many techniques out there that are not covered in this thesis because there was not enough time or manpower. For this reason someone else writing a thesis on this subject will largely come to the same conclusions regarding the problems but might find different solutions for them.

Further research could be to index all these big and little techniques that are associated with Agile and UX. Scientifically field-testing them would be another possible option. There is still a lot to learn about combining Agile and UX and because these areas develop so fast we will not be done learning about it for quite some time.

10 References

Anderson, D. J. (2012). *Lean Software Development*. Retrieved December 11, 2013, from http://msdn.microsoft.com/enus/library/hh533841.aspx

Android. (2014). *Design Principles*. Retrieved February 26, 2014, from https://developer.android.com/design/get-started/principles.html

Apple. (2014). *iOS Human Interface Guidelines.* Retrieved February 26, 2014, from https://developer.apple.com/library/ios/documentation/userexperience/conceptual/mobilehig/MobileHIG.pdf

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Jeffries, R. (2001). Manifesto for Agile Software Development. *Agile Alliance*. Retrieved November 12, 2013, from http://agilemanifesto.org

Boehm, B. (2002). Get ready for agile methods, with care. *Computer, 35*(1), 64-69.

Chowdhury, A. F., & Huda, M. N. (2011). Comparison between adaptive software development and feature driven development. *International Conference on Computer Science and Network Technology. 1*, pp. 363-367. IEEE.

Cockburn, A. A. (2001). Humans and Technology. Retrieved January 6, 2014, from http://alistair.cockburn.us/Crystal+light+methods

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *Computer, 34*(11), 131-133.

Constantine, L. (2001). Methodological Agility. *Software Development, June*, 67-69. Retrieved December 14, 2013, from http://www.drdobbs.com/methodological-agility/184414743

Crothers, B. (2013, October 3). *Fight the dark side of Lean UX with the Experience Canvas*. Retrieved March 24, 2014, from https://blogs.atlassian.com/2013/10/fight-the-dark-side-of-lean-ux-with-the-experience-canvas/

Cusumano, M. A. (2007). Extreme programming compared with Microsoft-style iterative development. *Communications of the ACM, 50*(10), 15-18.

Cusumano, M. A., & Yoffie, D. B. (1999). Software development on Internet time. *Computer, 32*(10), 60-69.

Ferreira, J., Sharp, H., & Robinson, H. (2011). User experience design and agile development. *Software: Practice and Experience, 41*(9), 963-974.

Garrett, J. J. (2010). *Elements of User Experience.* Pearson Education.

Gregory, J., & Crispin, L. (2012, May 23). *Testing the Big Picture on Agile Teams*. (informIT) Retrieved April 9, 2014, from http://www.informit.com/articles/article.aspx?p=1905881

Highsmith, J., & Cockburn, A. (2001). Agile software development: The business of innovation. *Computer, 34*(9), 120-127.

ISO. (2008). Ergonomics of human system interaction - Part 210: Human-centred design for interactive systems.

Isomursu, M., Sirotkin, A., Voltti, P., & Halonen, M. (2012). User Experience Design Goes Agile in Lean Transformation--A Case Study. *Agile Conference (AGILE), 2012* (pp. 1-10). IEEE.

Larman, C., & Basili, V. R. (2003). Iterative and incremental development: A brief history. *Computer, 36*(6), 47-55.

Law, E. L.-C., Roto, V., Hassenzahl, M., Vermeeren, A. P., & Kort, J. (2009). Understanding, scoping and defining user experience: a survey approach. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 719-728). ACM.

Marcotte, E. (2010). Responsive Web Design. *A list apart, 306*.

McHugh, O., Conboy, K., & Lang, M. (2012). Agile practices: The impact on trust in software project teams. *Software, IEEE, 29*(3), 71-76.

Medlock, M. C., Wixon, D., Terrano, M., Romero, R., & Fulton, B. (2002). Using the RITE method to improve products: A definition and a case study. *Usability Professionals Association*.

Microsoft. (n.d.). *Principles: The foundation that drives our design*. Retrieved February 26, 2014, from http://dev.windowsphone.com/en-us/design/principles

Miller, L., & Sy, D. (2009). Agile user experience SIG. *CHI'09 Extended Abstracts on Human Factors in Computing Systems.* ACM.

Moe, Brede, N., Dingsoyr, T., & Dyba, T. (2009). Overcoming barriers to self-management in software teams. *Software, IEEE, 26*(6), 20-26.

Najafi, M., & Toyoshiba, L. (2008). Two case studies of user experience design and Agile development. *Agile, 2008. AGILE'08. Conference.*

Nielsen, J. (1994). *Usability engineering.* Elsevier.

Olsson, H. H., Alahyari, H., & Bosch, J. (2012). Climbing the" Stairway to Heaven"--A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software. *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on* (pp. 392-399). IEEE.

Ostenwalder, A., & Pigneur, Y. (2010). *Business model generation.* John Wiley & Sons.

Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements engineering and agile software development. *IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises* (p. 308). IEEE Computer Society.

Patton, J. (2008). *The new user story backlog is a map*. Retrieved February 5, 2014, from http://www.agileproductdesign.com/blog/the_new_backlog.html

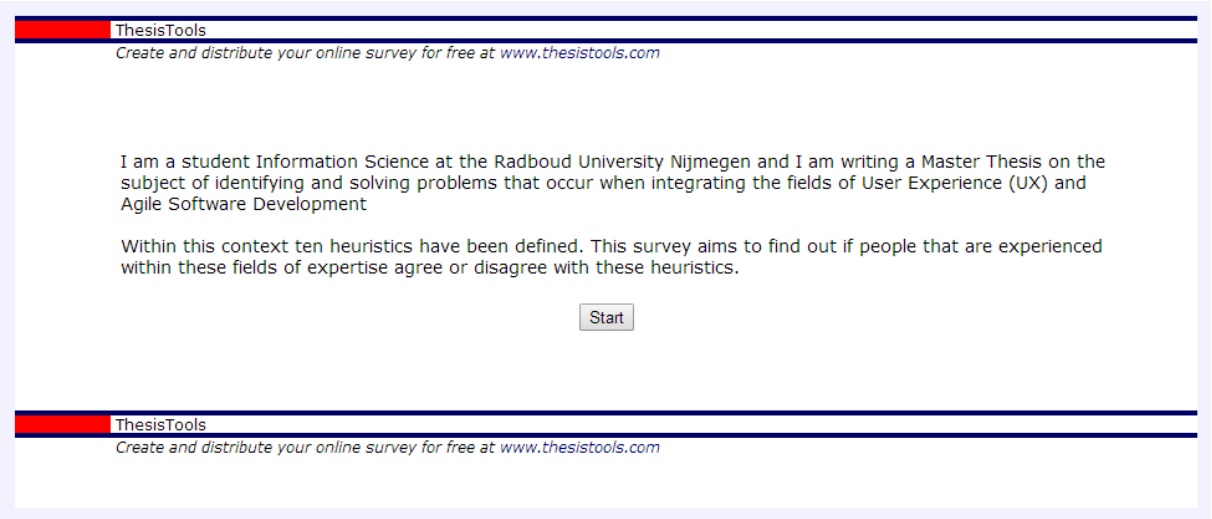
Sen, R. (2010, February 11). *From whole to hole: a recipe for a holistic design process*. (Johnny Holland) Retrieved March 26, 2014, from http://johnnyholland.org/2010/02/from-whole-to-hole-a-recipe-for-a-holistic-design-process/

Wake, B. (2003). *INVEST in Good Stories, and SMART Tasks*. Retrieved February 25, 2014, from http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/

wiseGEEK. (2014). *What is holistic design*. (wiseGEEK) Retrieved March 26, 2014, from http://www.wisegeek.com/what-is-holistic-design.htm

Wroblewski, L. (2012). *Responsive Navigation: Optimizing voor Touch Across Devices*. Retrieved March 4, 2014, from http://www.lukew.com/ff/entry.asp?1649

11 Appendix A



ThesisTools

Create and distribute your online survey for free at www.thesistools.com

I am a student Information Science at the Radboud University Nijmegen and I am writing a Master Thesis on the subject of identifying and solving problems that occur when integrating the fields of User Experience (UX) and Agile Software Development

Within this context ten heuristics have been defined. This survey aims to find out if people that are experienced within these fields of expertise agree or disagree with these heuristics.

Start

ThesisTools

Create and distribute your online survey for free at www.thesistools.com

Following are the 10 heuristics. The heuristic itself is put in bold text with an explanatory sentence put in brackets below it.
Note that the order in which these heuristics are displayed does not reflect any difference in importance or significance. Please indicate to what extent you agree or disagree with each heuristic.

1.

IT projects need to focus on producing software to solve a problem rather than working out a solution proposed by the client.
(The client is not the most objective person so the project team should be critical and first review what the actual problem is instead of blindly building what they are being told)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

2.

In order to successfully integrate a software solution and create a good User Experience you need information about the environment it will be deployed in.
(If you want to build usable software that delivers a good User Experience you need to know how and by who the software is going to be used)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

3.

In order to create the best possible solution to a problem and deliver the best User Experience you need input from as many different stakeholders as possible.
(Gathering information from different stakeholders allows you to get a complete picture of the problem area and enables you to weight arguments against each other)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

4.

Developers in Agile teams focus on the details of a solution and often lose sight of the project as a whole.
(A side-effect of working on small pieces of functionality at a time is that you lose the holistic view of how it all fits together)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

5.

When working in an Agile team, established insights should always be subject to change.
(Discussion should be encouraged and project members should not be afraid to question anything they think is wrong)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

6.

Face-to-face interaction between project members lowers ambiguity and increases the quality of the solution.
(Being able to immediately en directly ask someone about a problem clears up things much faster and makes sure the team is on the same page)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

7.

A good User Experience is essential when creating software that is going to be used daily.
(Good software not only delivers all the functionality that is required but also makes sure it is usable and pleasant to work with)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

8.

Thinking about the motivation and needs of your end-users helps to produce a better User Experience.
(Knowing how your users behave and what their preferences are helps you build software that is pleasant to use and allows you to create a good experience for the user)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

9.

Having a UX expert as a team member improves the quality of the software.
(Decisions about UX should be made by the team itself and the best way to do this is to have a team member actively engaging in UX)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

10.

Involving users early on prevents mistakes down the road.
(Getting input from users that are actually going to use the software you are building helps to avoid creating functionality that is not going to be used or making designs that are confusing)*

- Strongly Agree
- Slightly Agree
- Neutral
- Slightly Disagree
- Strongly Disagree

Next Page

11.

What is your affiliation with User Experience, Agile Software Development or both?
(multiple anwsers possible)

- No affiliation
- Work-related (Software Development)
- Work-related (Research)
- Hobby
- Other, namely

12.

How many years of experience do you have within the field of Agile Software Development?*

- None
- <1
- 1-3
- 4-6
- >6

13.

How many years of experience do you have within the field of User Experience?*

- None
- <1
- 1-3
- 4-6
- >6

14.

How many years of experience do you have combining the fields of User Experience and Agile Software Development?*

- None
- <1
- 1-3
- 4-6
- >6

15.

Is there anything not in this survey that you feel should have been included?

16.

Do you have any final comments about this survey and the heuristics specifically or the thesis in general?

Next Page

Thank you for participating in this survey. Your help is greatly appreciated.

Please press 'Submit' to record your results. You will then be asked by ThesisTools if you want the results of my study and if you want to participate in additional surveys. You are free to answer any subsequent questions which do not consider my survey.

For any further information you can contact me at [b.verdiesen@student.science.ru.nl]

- Beau Verdiesen

Submit