CHALMERS

Agile Solo

Defining and Evaluating an Agile Software Development Process for a Single Software Developer

Master of Science Thesis in Software Engineering and Technology

ANNA NYSTRÖM

Agile Solo
Defining and evaluating an Agile Software Development Process for a single Software Developer

NYSTRÖM

Supervisor: SVEN-ARNE ANDREASSON
Examiner: MIROSLAW STARON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2011

Acknowledgement

I would like to thank the following people for supporting me in my thesis work. I am very grateful to you all.

Maria Fridén, Sam Alnashi and everyone at Abou.

Björn Johansson and Amanda Johansson.

Sven-Arne Andreasson and Miroslaw Staron at Chalmers Tekniska Högskola.

Erik Josefsson and Erik Stenbäcka.

Sammanfattning

Målet med examensarbetet är att definiera en agil arbetsprocess för att utveckla mjukvara. Arbetsprocessen ska vara anpassad för en ensam utvecklare som samarbetar med en kund eller beställare.

Såkallade smartphones har slagit igenom kraftigt på marknaden de senaste åren. Både Apple’s produkt iPhone och ett stort utbud av telefoner med operativsystemet Android har sålt i stora mängder. Den här typen av telefoner kan köra många små applikationer, till exempel spel eller nyttoprogram. Många utav dessa applikationer utvecklas av en ensam utvecklare i samarbete med en beställare. Detta ökar behovet av en väl anpassad arbetsmetod för ensamma utvecklare.

Arbetsprocessen döptes till Agile Solo och innefattade ett antal obligatoriska moment i utvecklingsprocessen. Eftersom arbetsprocessen är en agil process utvecklas mjukvaran i iterationer om en vecka. Inför varje iteration träffas utvecklare och kund och planerar vad nästa vecka ska fokuseras på. Kunden är kravställare och avgör vad som är viktigast att fortsätta arbetet med.

Efter att Agile Solo definierats testades utvecklingsmetoden i ett verkligt projekt. En applikation för operativsystemet Android utvecklades hos mjukvaruföretaget Abou AB. Applikationen gav föräldrar till förskolebarn möjligheten att sjukanmäla och ledighetsanmäla sina barn via telefonen.

Arbetsprocessen fungerade bra att använda under utvecklingen med vissa anpassningar. För framtida soloprojekt rekommenderas utvecklaren att använda en skräddarsydd version av Agile Solo där utvecklingsprocessen och dess moment anpassas under projektets gång.

Abstract

The purpose of this thesis is to define an agile development process for software development. The development process is to be adapted for single developers working in collaboration with a customer.

The market for smartphones has grown very quickly in the last few years. Both Apple's product iPhone and a large range of smartphones running Android have reached market success. This type of phone can run many small applications for example games or utility programs. Many of these applications are developed by single developers. This increases the need for a well-adjusted development process for single developers.

The process model was named Agile Solo and consisted of a number of compulsory practices to be carried out during the implementation phase of the project. Since the development process is agile the software is developed in weekly iterations. Preceding each iteration a meeting is held between the developer and the customer where they plan which tasks should be included in the coming iteration.

After the definition of Agile Solo was completed the development process was tested in a project. An application for the operative system Android was developed in collaboration with software development company Abou AB. The application gave parents of preschool children the opportunity to report sick leave and vacation leave using their mobile phone.

Using the development process during the implementation of the application was successful although some adjustments were made. For future solo projects the developer is recommended to use a tailored version of Agile Solo where the development process and its practices are adjusted during the project's course.

Table of Contents

Table of figures

1 Introduction

When producing software it’s common practice to follow a predefined process model. Increased predictability, higher quality and long term traceability are examples of benefits that can be achieved from doing so. There are many process models to choose from and it is important to find one that fits the developers and the project well. Following the breakthrough of smartphones a new market has rapidly opened up – the market of mobile applications. It is likely that many of these small programs are developed by a single developer and as a result the need for a process model tailored for the solo developer may also be greater than before.

As early as 1992 IBM in collaboration with Bellsouth introduced *The Simon Personal Communicator* - the world's first smartphone. It had built in features like a calendar, note pad, file manager and even fax functionality. It also featured a touch screen, so it was in many ways similar to today's smartphones. It did not, however, find market success, probably because it was very bulky and expensive. Perhaps it was also ahead of its time. (1)

The BlackBerry smartphones were the first to become popular and widespread. From 2004 to 2006 the number of BlackBerry users had increased from one million to five million. They were used primarily by companies due to their focus on email (2) (3) (4).

Below is a recent definition of what a smartphone is; it is not that different from what a smartphone was in 1992:

> ”[A smartphone is] a cellular telephone with built-in applications and Internet access. Smartphones provide digital voice service as well as text messaging, e-mail, Web browsing, still and video cameras, MP3 player and video viewing. In addition to their built-in functions, smartphones can run myriad applications, turning the once single-minded cell phone into a mobile computer.” (5)

Today, Apple's iPhone and numerous smartphones running Google’s mobile operating system Android have entered the scene making smartphone sales grow exponentially. According to *Canalys Smart Phone Analysis* of Q4 2010 the total number of smartphone shipments rose to 101.2 million units, an increase of 89% from Q4 2009. The total number of shipped smart phones almost reached 300 million units in 2010, a growth of 80% over 2009. (6)

1.1 Introduction to software process models

A Software Process Model is basically a description of the software producing activities and the roles of people involved in producing it. There are a few general categories of process models; among them we find waterfall methods and iterative methods. (7)

The first mention of the waterfall method was in 1970 by Winston W. Royce. In an article he discredited the method, saying that it was too rigid to work in practice. He expressed that the waterfall process should be carried out twice before a satisfying end result could be achieved. Despite this it moved on to become the most widely used software process model in the 1980s. It is possible that the success of the waterfall method partly relies is it simplicity. It is easy to teach as it is a linear process. Soon after the publication of his article other writers and people using the method had simplified it further to only contain a single iteration of the steps. (8)

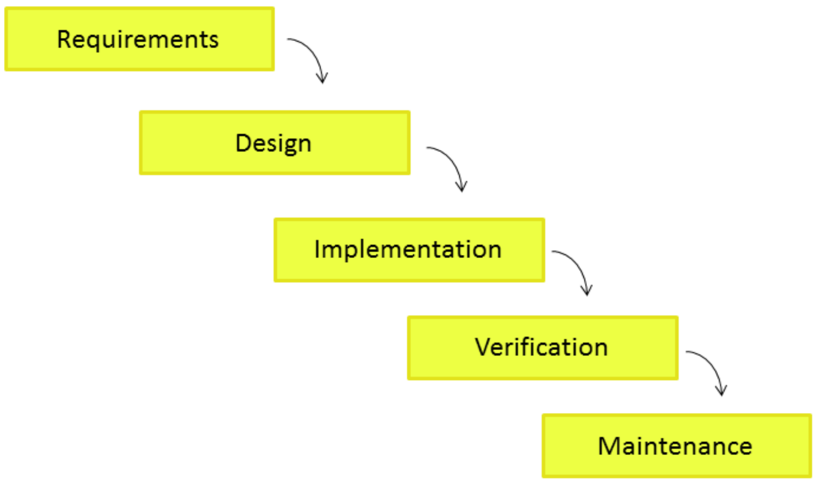


Figure 1: The Waterfall Method

1.1.1 Iterative development

Iterative process models have been in use since the 1950's but have only recently gained popularity as the waterfall method has been the dominant model for many years. (8) The thing that sets iterative process models apart from waterfall methods is the practice of working in iterations. This basically means that, throughout the development phase, pieces of functionality are added to incrementally improve the software. Agile development processes are all iterative processes and will be discussed further in the Theory chapter of this thesis.

1.1.2 The Personal Software Process

In 1993 Watts S. Humphrey developed a methodology called the Personal Software Process (PSP). The aim of this methodology is to help professional software developers improve their planning skills and reduce the number of defects in their products and thereby improve the quality of their work. One of the main concepts of this methodology is to collect data during the implementation phase. The size of a product part, the estimated and actual time to complete it and the number of defects are recorded. The collected data is later used to tailor the development process to a new project and to improve the planning of the project.

The PSP has three phases – planning, development and postmortem. The development phase is in turn based on a number of steps, for example, requirement, design, coding and testing. The PSP can be used in an iterative way by making each increment of the program into a PSP project.

2 Purpose

In the 1970's companies started to produce software using the waterfall method, or as it was also referred to: the Life Cycle Method. The waterfall method has been widely criticized during the last twenty or so years. However, in a scenario where the development team can know with certainty exactly what needs to be in the finished product, it has merit. Currently, working under circumstances like that is rare. Initially, software projects were almost exclusively large systems, requiring low error rates, such as military or banking systems. This is in high contrast to what we experience today. On the Android Market there are at the time of writing more than 360 000 available apps, the corresponding number for Apple's App Store is over 350 000. Few of these applications have requirements for low error rates, none of them are large. (9) (10)

In the context of mobile software development it is even more rarely the case that an extremely detailed and rigid description of the desired result exists. The reason for this is that the description of a typical software customer has changed radically. Software has plainly become highly accessible to regular people. It can also be argued that the software produced today is to a larger extent produced for entertainment, information and marketing purposes. That means that a typical software customer has changed from basically only large organisations, such as militaries and corporations to practically anyone. Software developers are now tasked to work with a new type of customer, in many cases one that is unsure what she wants and needs.

When developing a mobile application many developers would vouch for using an agile method and there are many to choose from. They vary in their practices and execution but they all have the same philosophy:

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

That is, while there is value in the items on the right, we
value the items on the left more.

This is the Agile Manifesto, agreed upon by Kent Beck,
Mike Beedle, Arie van Bennekum, Alistair Cockburn,
Ward Cunningham, Martin Fowler, James Grenning, Jim
Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian

Marick, Robert C. Martin, Steve Mellor, Ken Schwaber,
Jeff Sutherland and Dave Thomas in 2001. (11)

My goal in writing this thesis is to define an agile software development process for a single developer that is derived from this manifesto; so that the single developer who chooses to follow it can work in an agile but controlled way in collaboration with their customer.

3 Method

3.1 Defining Agile Solo

Defining an Agile software development process will be done with great respect towards the Agile Manifesto and towards already established agile development processes. The purpose of this thesis is not to reinvent the wheel but rather adapt established working practices and philosophies to function smoothly with solo development. In order to find useful practices a literature study will be conducted, not limited to only traditional agile software development practices. The literature study aims to give a deeper understanding about what the benefits and drawbacks of different software development practices are. It aims so clarify what practices are suitable and what practices can be adjusted to be suitable in a single developer project. It will aim to answer the following questions: What happens to a process model if some of its key practices cannot be performed? Can these practices be replaced or compensated for in a way that does not have negative effects on quality or execution time of the project? The definition of Agile Solo will be written based on the results of the literature study.

3.2 Evaluating Agile Solo

In evaluating the defined development process, titled Agile Solo, a project will be carried out. During the course of this project Agile Solo will be practiced with as few compromises as possible. Each practice in Agile Solo will then be evaluated separately as well as the process model as a whole. The evaluation will aim to determine whether using a particular practice is advisable in a single developer setting and whether the process model as a whole is useful to single developers.

The project that will be carried out entails developing an application for Google's mobile operating system Android. This will be done in collaboration with Stockholm based software development company Abou AB, hereafter referred to as Abou.

3.2.1 Abou AB

Abou is a company that specializes in improving work processes. Inspired by Toyota they practice and teach Lean production methods in order to deliver the most innovative solution possible. Their IT-solutions always aim toward improving their customers' organization and communication processes. This is often done by eliminating wasteful activities and making information more visible. Abou have a great deal of experience working with government digitalization including communication between government and citizens as well as internal communication within government organizations and municipalities. (12)

Abou will play the part of customer in the evaluation of Agile Solo. They will also provide me with two supervisors to support me in my thesis work; a project manager and a technical supervisor.

3.2.2 The Preschool Portal

Abou have in collaboration with the city of Stockholm developed a communication solution for the parents and teachers of preschool children. The idea behind the portal is to simplify the communication between parents and teachers. Reporting illness, vacation leave and schedule changes is made easier through new channels of communication in the system.

3.2.2.1 Web portal

The broadest supply of services can be found in the web portal. After logging in the parent can manage permanent or temporary schedule changes, read information about the preschool and follow a news feed, report sick leave and vacation leave, edit their personal info and see the contact information of the other children and their parents. There is also functionality available only to teachers, such as managing substitute teachers, viewing contact information and posting news etcetera.

3.2.2.2 Touch screen system

Touch screen monitors have been placed in the preschools' entrance halls. Most of the services available in the web portal are also available here. Functions that have high security requirements are not accessible here since there is no support for e-identification in the touch system. The touch system is also used by the preschool teachers to get a quick overview of today’s schedule and the number of absent children as well as to send information to the parents via text messages or e-mail.

3.2.2.3 Progress

At this time the project is at the end of its trial run, two preschools have been testing a full installation of the preschool portal since October 2010. It not yet decided that all preschools in Stockholm will be included in the project, but in the fall of 2011 fifteen more preschools are planned to be given access to the portal.

The trial run has been very successful, with teachers as well as parents responding positively to the amount of time that has been saved using the portal instead of making phone calls and sending notes home. (13) According to Maria Fridén, project manager at Abou, on the 19th of May 2011, 120 out of 125 children had reported their regular schedule in the preschool portal. More than 60 percent of the parents reported their children’s Christmas absence via the preschool portal and in total, the number of absence leaves that have been reported is 687 since the introduction of the system.

3.2.3 The Preschool Android application

The single developer project that is the basis of the evaluation of Agile Solo is an application that will serve as a complement to the Preschool Portal. The most important functionality available on the web portal and on the touchscreen monitors is to be included in the Android application; reporting sick leave and vacation leave. The flow of the user interface is to be based on an iPhone application that has already been developed for the preschool portal.

3.3 Scope

The evaluation of Agile Solo will be performed as a case study. During the development all practices of Agile Solo will be implemented, analyzed and assessed. This is done based on the developers experience using the practice as well as the feedback from the customer and the end-users.

3.3.1 Limitations

The evaluation of Agile Solo will be limited to a single trial. Evaluating it in a larger scale would naturally be more interesting but including that in the thesis is considered too costly and time consuming to be achievable. This also means that the process model will not be revised during the trial period. Since the project is relatively small there will only be time for evaluating one version of the process model.

4 Theory

4.1 Agile Development Processes

In the 1990's many new process models based on agile principles were established. These processes all have their focus on delivering working software quickly to the stakeholders of the project in order to get early feedback and thereby produce a more customized and higher quality product. These models are commonly used in software development today.

4.1.1 Scrum

Scrum was founded in 1995 by Jeff Sutherland och Ken Schwaber

4.1.1.1 Roles

There are three different roles in a scrum project; the Scrum Master, the Product Owner and the Team. There are usually 5-9 developers in the Team. The Scrum Master's role is similar to that of a project manager. He is in charge of following the process and keeping the team focused on their goal. His job is not to lead the Team, they should be self-organized, but rather to remove obstacles for them, do their Velocity Tracking and make sure that their work runs smoothly. The Product Owner, preferably the customer or a person representing the customer is responsible for making a product backlog. This document contains a prioritized list of the functionality that the customer wants in the finished product. The document is not rigid and can be changed by the Product Owner at any time during the project. Decisions on when to release finished functionality to the customer are also made by the Product Owner. (14) (15)

4.1.1.2 Sprint

Sprint is the term used in Scrum for iteration. The length of a sprint is usually two to four weeks but can be anything between one and six weeks; that is up to the team. Before starting a sprint, a planning meeting is held and items from the product backlog are moved into the sprint backlog. The sprint backlog is basically a list of tasks, the goal of the sprint is to finish all tasks in the list. The product Owner should attend the meeting and discuss what items should be included in the upcoming sprint with the team. The goal of the sprint is that the software is in a deliverable state. At the end of each sprint a demo and review of the completed sprint is held in order to get feedback from stakeholders and to discuss how the execution of the next sprint can be improved. (15)

4.1.1.3 Daily Scrum

The team meets every day to discuss the progress made since the last daily scrum. Anyone can attend the meeting but only members of the team are allowed to talk. The topics are "What has been done since the last meeting?", "What will be done before the next meeting?" and "What are the foreseeable obstacles?" The daily scrum is a short meeting, intended as a status update for everyone involved in the project, it is sometimes done standing up to ensure that it is kept short and fast-paced. (Having stand-up-meetings is practice that is commonly used in other agile process models as well.) (16)

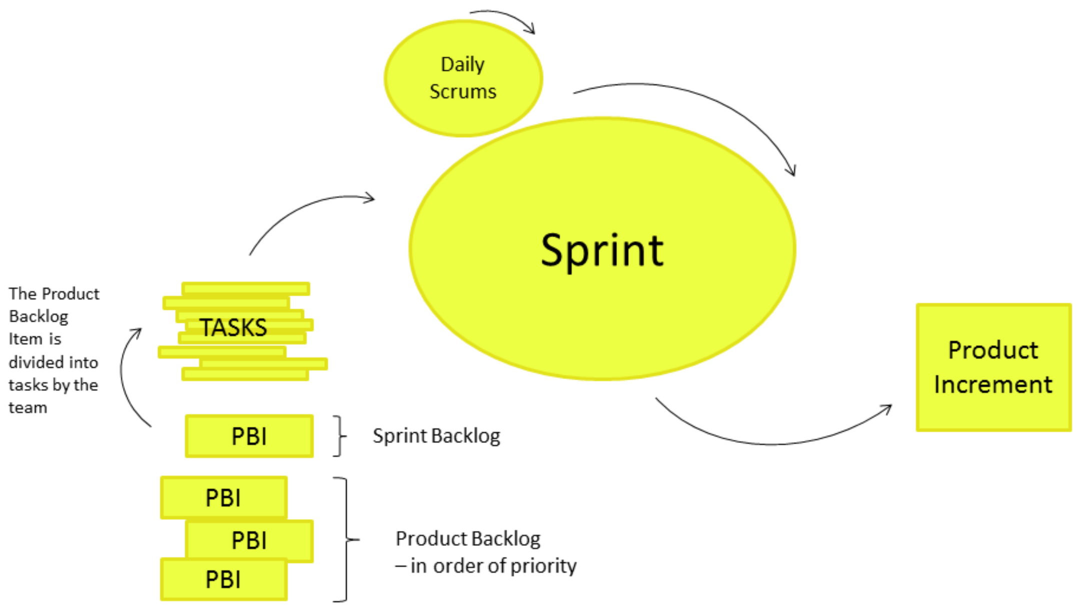

Daily Scrums
Sprint
The Product Backlog Item is divided into tasks by the team
TASKS
PBI
Sprint Backlog
PBI
PBI
PBI
Product Backlog – in order of priority
Product Increment

Figure 2: Scrum

4.1.2 eXtreme Programming

The first eXtreme Programming project was carried out in 1996. (17)

4.1.2.1 Customer Representative

In eXtreme Programming a Customer Representative is considered a member of the development team. She makes the decisions on what user stories should be included in the next iteration and the priority amongst them. A user story is a short description of some functionality that should be added or altered in the program. It is very important that the communication between the customer representative and the rest of the team is open and continuous.

4.1.2.2 The Planning Game

The planning game is what the iteration planning meeting is called in eXtreme Programming. Preceding the meeting, the customer formulates the user stories which he finds most important. After that, during the meeting with the team, collaborative decisions are made about which users stories should be included in the coming iteration. After this is done the team members continue the meeting without the customer representative. First they break down the user stories into smaller tasks and after that they divide them amongst themselves. Depending on their individual velocities from the last iteration some commit to more work than others. In order to decide who should do which task the programmers place "bids" on each task. The person who bids the lowest amount of time to finish a task wins it and gets to implement it. (18)

4.1.2.3 Test Driven Development

Writing tests first is a central practice in eXtreme Programming. It basically means that before any other code is written the programmers write a test. If it is done correctly the test fails since the code making it pass has not yet been produced. Only after the test is written, run and has failed are the programmers allowed to start working on the initial task. (18)

4.1.2.4 Pair Programming

In eXtreme Programming every line of code that gets delivered to the customer is written by two collaborating programmers working on the same problem at the same computer. One of the programmers is the "driver", the other one is the "copilot". The driver writes the code and the co-pilot helps by discussing possible solutions and correcting mistakes. If the copilot at any point has an idea of a different approach it is acceptable for the two programmers to switch roles and the driver then becomes the copilot. (18)

4.1.3 Kanban

Kanban is a method invented by Toyota in 1953. It was originally used in the industry to schedule production in a “Just in time”-manner. By not producing anything that could not immediately be sold, costs for stock and insurance could be lowered and the amount of capital tied up in raw materials or stocked finished products could be minimized. Kanban can also be used in the context of software development. (19) (20)

4.1.3.1 The Kanban-board

As a process model Kanban is more flexible than Scrum as the tasks for the current iteration can be changed at any time. Task management in Kanban is usually carried out using post-it notes. Every task is written down on a post-it and put on a Kanban-board. The board has columns illustrating different stages that the task can be in, such as “ready for implementation”, “in progress” and “finished”. Using this system makes progress visible and works as a motivation for the team carrying out the work. (21)

4.1.3.2 Limiting work in progress

The column named “work in progress” has an upper limit, that limit is a key aspect of Kanban. Limiting work in progress is a way of making programming time more efficient. Having a clear image of what the current tasks are is very beneficial to a programming team since it allows them to start working in a focused way immediately. Jim Benson wrote “You have two hands. You can only juggle so many things at a time. The more you add, the more likely it is that you will drop something.” This perfectly illustrates that an unoccupied mind works better than a mind juggling too many tasks. (22)

4.1.4 The Crystal family of methodologies

Crystal was defined in 1992 and has evolved greatly since. The name crystal was taken in 1997. The reason for calling Crystal a family of methodologies instead of simply a methodology is that its proponents believe that when team size and fault tolerance change, the method should also change. (23) (24) The methodologies are named after geological crystals representing the hardness of the process: Clear, Yellow, Orange, Orange Web, Red, Magenta, Blue etc. Projects with fewer developers and less critical parts should use a less rigid methodology such as Crystal Clear or Yellow. (25)

4.2 Time Boxing

Time boxing basically means dedicating a block of time for a specific task. It could be three hours every Thursday morning spent cleaning your apartment. In the context of software engineering it could be looking over what should be done the coming week and devoting blocks of time to be used for uninterrupted programming blocks. By scheduling meetings and other administrative tasks in a way that doesn't chop up all the work days in a week, focusing on one thing at a time is made easier.

4.2.1 The Pomodoro Technique

The Pomodoro technique is a way of managing your own time on a very low level. Its purpose is to improve concentration and reduce the number of interruptions and distractions during a fixed block of time. The idea is basically that you make a detailed list of the tasks that you want to complete during the day and prioritize them. When you are done you set a timer for 25 minutes and when it rings you have completed your first "Pomodoro". Between each Pomodoro you should take a short break from all work related activities and after every fourth Pomodoro you take a longer brake.

The method was invented by student Francesco Cirillo in the 1980’s who used a kitchen timer to keep track of his Pomodoros. (26) Now there are many different Pomodoro software applications available. They are substituting the kitchen timer as well as providing additional functionality such as keeping prioritized lists, tracking the number of finished Pomodoros and showing statistics.

5 Results

5.1 Agile Solo

This is Agile Solo; a software process model specifically engineered for single developer projects. It is the result of analyzing established agile process models and adapting practices for single developers. The process model will be evaluated in a case study and its strong points and weaknesses will be established and discussed in later chapters of this thesis.

5.1.1 Weekly Presentations and Updated Priorities

A key practice in all agile development is requesting frequent feedback. Therefore, in Agile Solo, the software is presented in its current state at the end of each week to a supervisor, a manager or directly to the customer. At this time priorities are updated so that the most important and urgent work is done first. Agreements made prior to this time are no longer relevant.

5.1.2 Monthly Deliveries and Customer Test

Working software is delivered monthly to the customer in order to get direct feedback from the users. An accessible communication channel is set up for the customer to give their input, ideas and comments. It is of high importance to be perceptive and responsive to this type of input between deliveries. The feedback should be taken into account in the weekly meetings and reprioritizations.

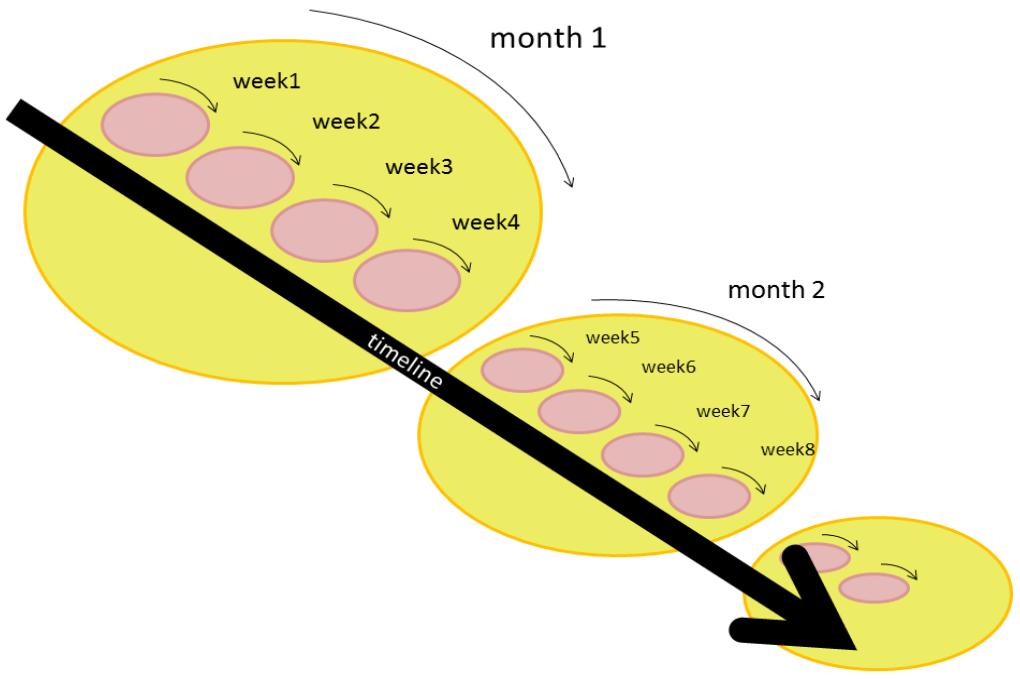


Figure 3: This is a model that illustrates the two types of iterations in an Agile Solo project

5.1.3 Planning an iteration

In Agile Solo there is a shorter iteration where software is presented and a longer iteration where software is actually delivered to the customer. Goals for both iterations should be updated in the weekly meetings. This is done by prioritizing what is most important to finish first, based on the customer's needs and the feedback from the users. An estimation of what can reasonably be finished within the timeframe of the iteration should also be done. The estimated time to finish a task should be written down so that when the iteration has ended it can be compared with the actual time it took to complete the task. This is done in order to increase the accuracy of the estimates done in the future.

5.1.4 Test Driven Development

This is perhaps the most fundamental practice of all. One of the most important effects of this practice is maximizing the work not done - doing only what *has* to be done. Test Driven Development also helps the programmer visualize the end result of the task at hand and really think about the problem before implementing a solution. In addition, having a comprehensive test suite specifically engineered for the project is very valuable and helps guarantee that refactoring doesn't change features in an unwanted way.

5.1.5 The Pomodoro Technique

The Pomodoro technique was not originally intended as a software development practice. In Agile Solo the Pomodoro technique serves as a low level time-boxing method, partly to fill some of the void caused by the absence of pair programming (see 5.1.10) and partly to maintain a good work flow. It is also useful in auto code review (see 5.1.7).

5.1.6 Peer Code Review

Peer Review is a common practice in many different development processes, not only in agile processes. The main objective of this practice is finding bugs and poorly structured code at an early stage. At least twice a week, preferably every day, a fellow programmer or a supervisor should review the produced code in an Agile Solo project. This is important in order to maintain quality code and to remove the illusion that the author is the only person who needs to understand the code. It also helps with writing comments and code concurrently.

5.1.7 Auto Code Review - inspecting your own code

At the beginning of each Pomodoro a quick review is made of the code produced in the previous Pomodoro. The idea is that the Pomodoro break provides some distance to the work carried out and an opportunity to look at solutions with fresh eyes. This is a practice that aims to substitute an important part of pair programming which is the continuous code review performed by the copilot while the driver writes the code.

5.1.8 Visual Control

Visual Control basically means keeping track of your work in a visual way. It is a lean concept that is used in many different organizations as a way of increasing efficiency. In Agile Solo it is important that a visual representation of the current iteration is kept available and updated. This way it is easy for the developer as well as the customer to get an overview of what has been completed and what still needs to be done.

5.1.9 Modeling

It is usually a good idea to do some modeling before the implementation starts. However, it is important to keep in mind that these models might not be valid throughout the entire project; therefore they should be concise rather than overly detailed. Not all agile process models promote modeling, but in this case, it will work as a way of prompting communication with the customer. Making some simple diagrams in collaboration with the customer is a good way of getting started working together as a team.

5.1.10 Compensating for pair programming

"Pairing does four things: it reduces the number of mistakes that programmers introduce into their code; it helps avoid everyday programming distractions; it forces thought and evaluation of possible solutions; and it provides an opportunity for team members to exchange programming knowledge and project-specific information." (18)

Using a development process that excludes the practice of pair programming means missing out on all these benefits. In the case of a solo project there is no choice but to try to fill the void with other practices.

In Agile Solo, Code Review and Peer Code Review work to reduce mistakes in the code. Using the Pomodoro technique helps with handling distractions in a good way and focusing on one thing at a time. Test Driven Development will enforce thinking about a solution before implementing it. Exchange of programming knowledge within the team is not relevant in a solo project.

5.1.11 Iteration Task Management

To keep control over the weekly tasks eXtreme Programming uses user stories, paper cards on which the Customer Representative formulates the program’s desired functionality. After agreeing on a number of user stories to be completed during the next iteration the programmers break down the user stories into smaller tasks and divide them amongst themselves during the planning game. In Scrum the customer formulates "Product Backlog Items" or PBI's that also are broken down into tasks. It is a similar method although it is much stricter in the sense that you do not add or subtract PBI's or tasks in an ongoing sprint. Also, if at the end of the sprint some tasks were not completed, the sprint is considered failed. For Agile Solo, the decision about which task management process that should be used is up to the developer and the customer. However, some guidelines are set. It is important to keep it as simple as possible. Striving for a system that offers some type of easy overview is also beneficial. This means that that the customer and the developer should easily be able to follow the progress of the project and be sure that it moves forward and in the right direction. Being flexible during iterations is also encouraged as long as a proper review of a finished iteration is carried out. A review should contain comparison between estimated and actual times to finish tasks as well as a more general discussion about what could be improved on in the next iteration.

5.2 The case study

In this chapter the results of using Agile Solo as a development process is presented. The process model was used during a case study where the development of an Android application described in 3.2.3 took place. Two supervisors at Abou acted as customer representatives and the parents of the preschool children tested the application as representatives of the end-users.

The goal of the case study was to evaluate how useful the following practices are in a single developer project:

- Modeling
- The Pomodoro Technique
- Weekly code reviews
- Daily auto code reviews
- Monthly user tests
- Task Management
- Test Driven Development

5.2.1 Modeling

Preceding the first iteration a simple model of a more long term goal was created in the form of an activity diagram. It was not made to be definitive but rather intended as an approximate target and inspiration point. The customer liked it but wanted a diagram that also showed the communication between the application and the webservice. To fulfill this, three sequence diagrams were created. The diagrams modeled the three main activities: logging in, reporting sick leave and reporting vacation leave.

An iPhone application containing the same functionality as the Android application to be developed had already been completed. It would be a waste of time to spend more time modeling when that application in itself could function as a model as far as confirming that the customer and developer agree on what a possible successful end-product could look like.

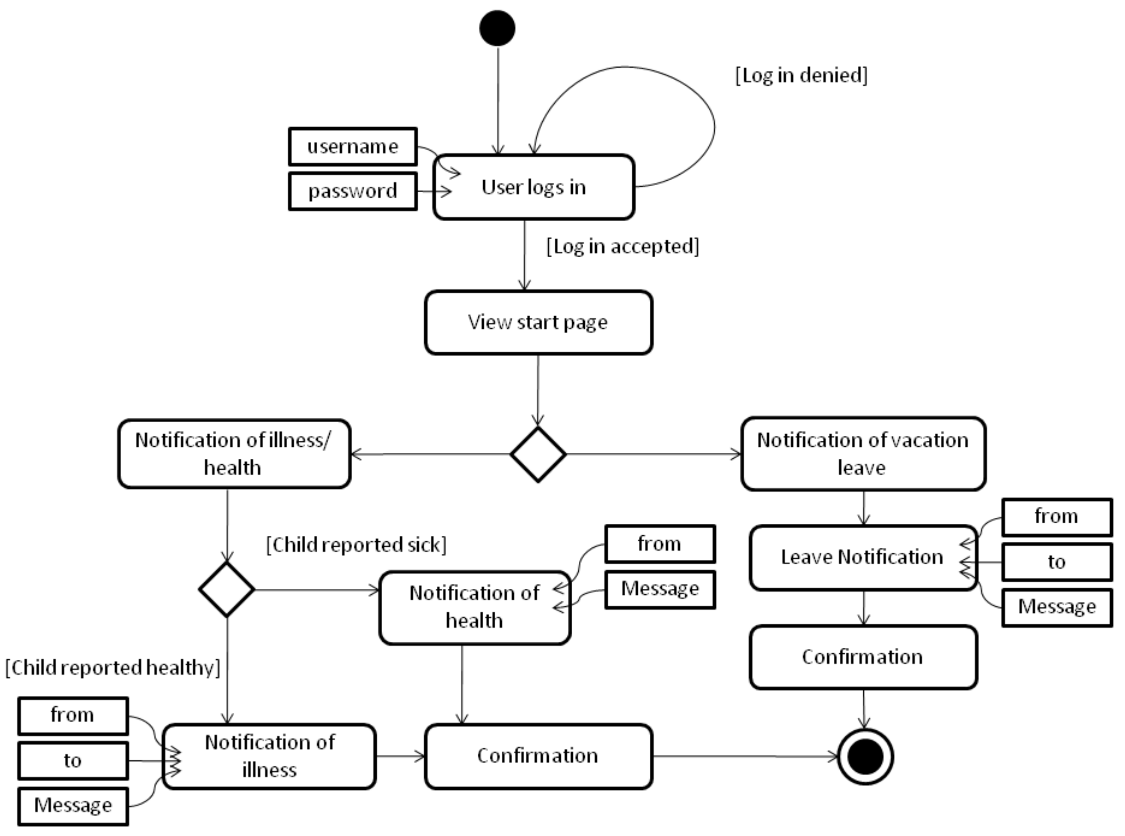


Figure 4: This is an Activity Diagram created before the first iteration. It shows the flow of the application from logging in to reporting sick leave or vacation leave.

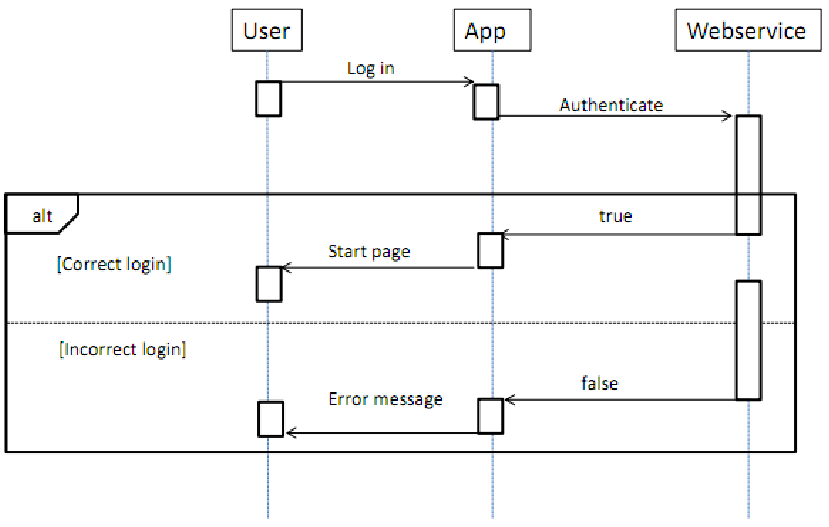


Figure 5: This is a Sequence Diagram showing the communication between application and webservice during a log in.

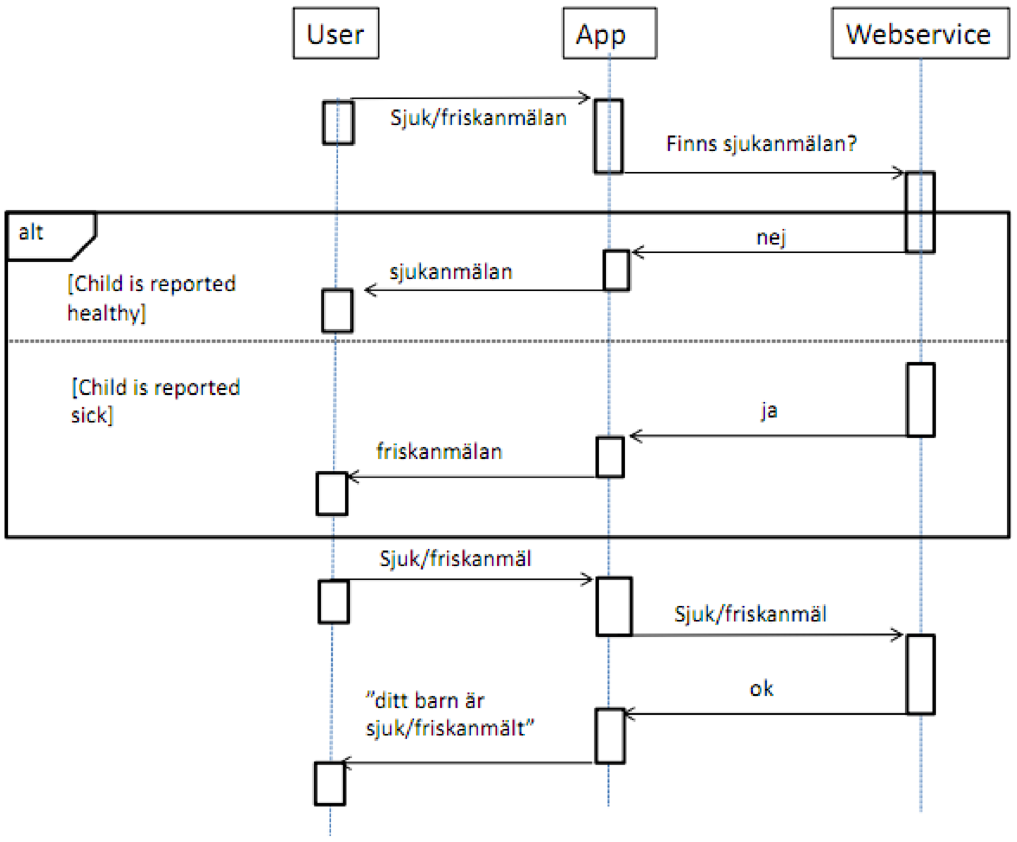


Figure 6: This is a Sequence diagram showing the communication between application and webservice while reporting sick leave.

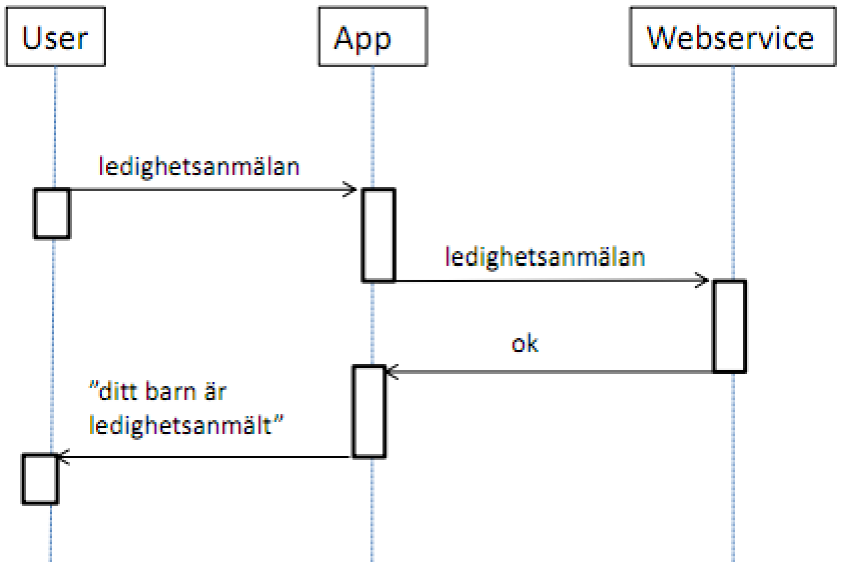
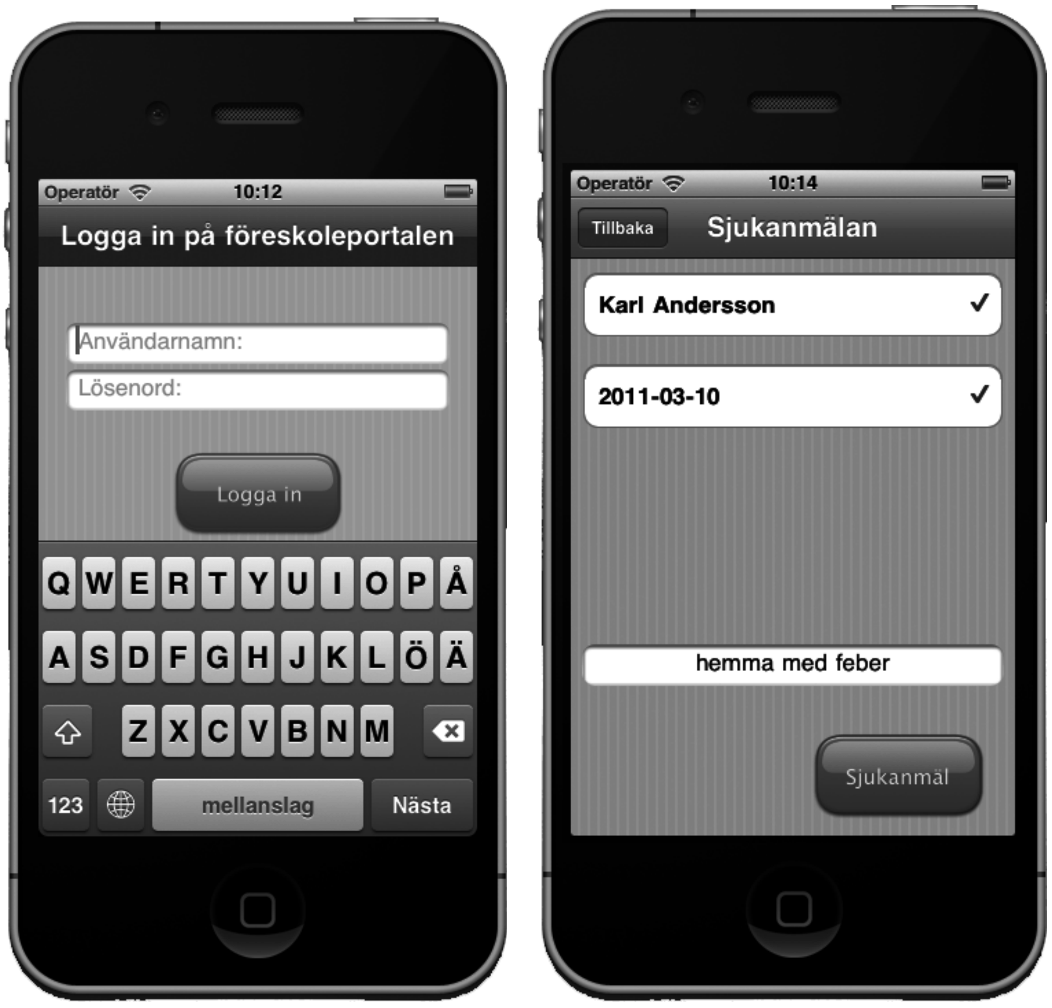


Figure 7: This is a Sequence diagram showing the communication between application and webservice while reporting vacation leave.



Karl Andersson ✓

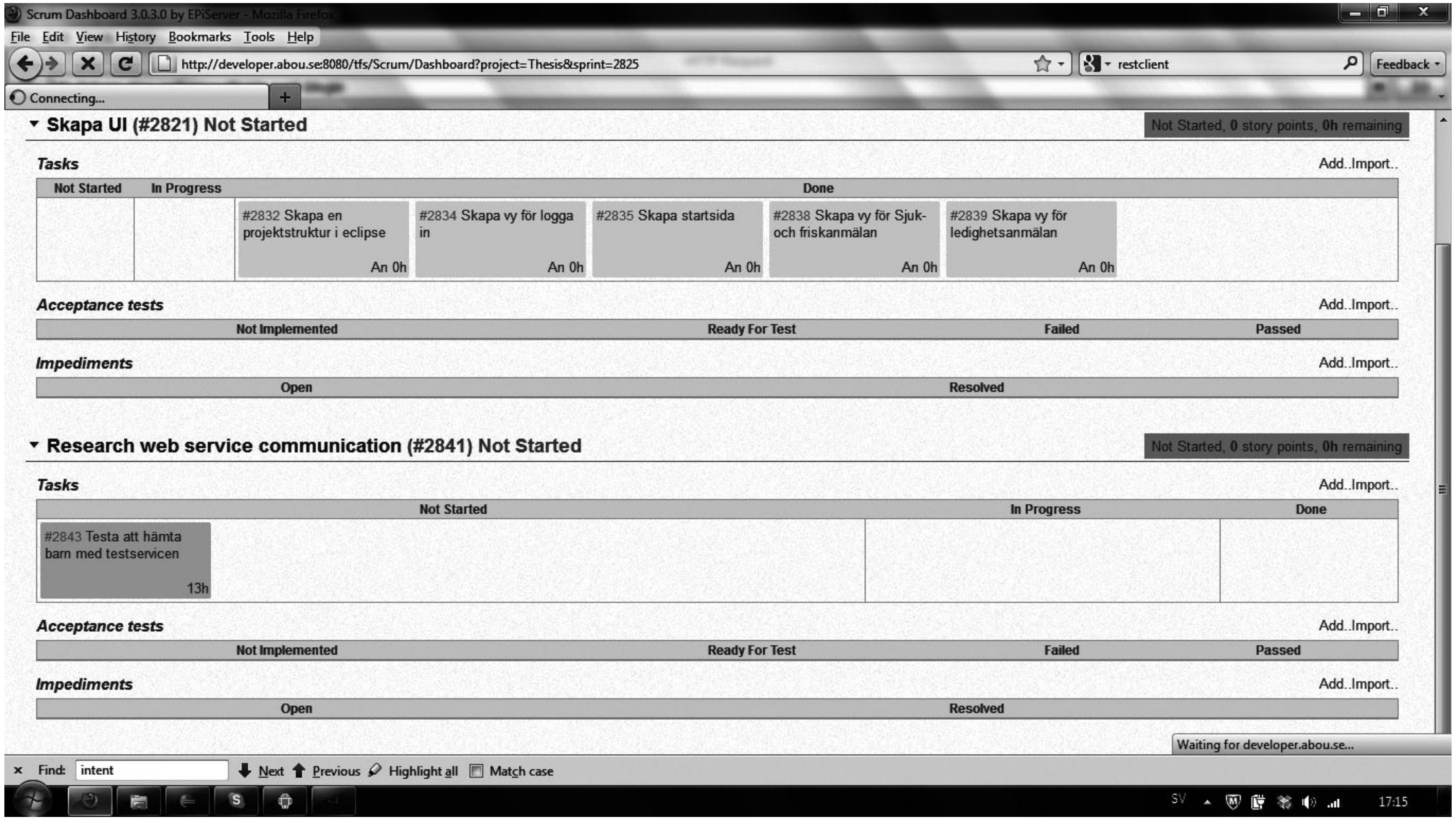
2011-03-10 ✓

hemma med feber

Figure 8: The iPhone Application already developed before the beginning of this thesis work.

5.2.2 Task Management

Beginning the first iteration a specific method for task management had not been established. A collaborate decision was made to use the Scrum backlog system available at Abou for at least one iteration as a trial. Evaluating it showed that it was not flexible enough to accommodate certain key aspects of Agile Solo Task Management. There was no functionality supporting velocity tracking in a visible way and there was no way of splitting PBI's if completing a whole PBI seemed to be overreaching. For a larger group of programmers working in two-week-iterations there would naturally be no need to split PBIs. However for a single programmer working with single week iterations a reasonable amount of work seemed to be only one or two PBI's, meaning it would be beneficial to be able to include partial PBI’s in an iteration. These drawbacks and a desire to move towards a simpler method led to a switch to writing down the iterations' tasks in a notebook.



Scrum Dashboard 3.0.3.0 by EPiServer - Mozilla Firefox

http://developer.abou.se:8080/tfs/Scrum/Dashboard?project=Thesis&sprint=2825

Skapa UI (#2821) Not Started — Not Started, **0** story points, **0h** remaining

Tasks Add..Import..

Not Started	In Progress	Done				
		#2832 Skapa en projektstruktur i eclipse An 0h	#2834 Skapa vy för logga in An 0h	#2835 Skapa startsida An 0h	#2838 Skapa vy för Sjuk- och friskanmälan An 0h	#2839 Skapa vy för ledighetsanmälan An 0h

Acceptance tests Add..Import..

Not Implemented	Ready For Test	Failed	Passed

Impediments Add..Import..

Open	Resolved

Research web service communication (#2841) Not Started — Not Started, **0** story points, **0h** remaining

Tasks Add..Import..

Not Started	In Progress	Done
#2843 Testa att hämta barn med testservicen 13h		

Acceptance tests Add..Import..

Not Implemented	Ready For Test	Failed	Passed

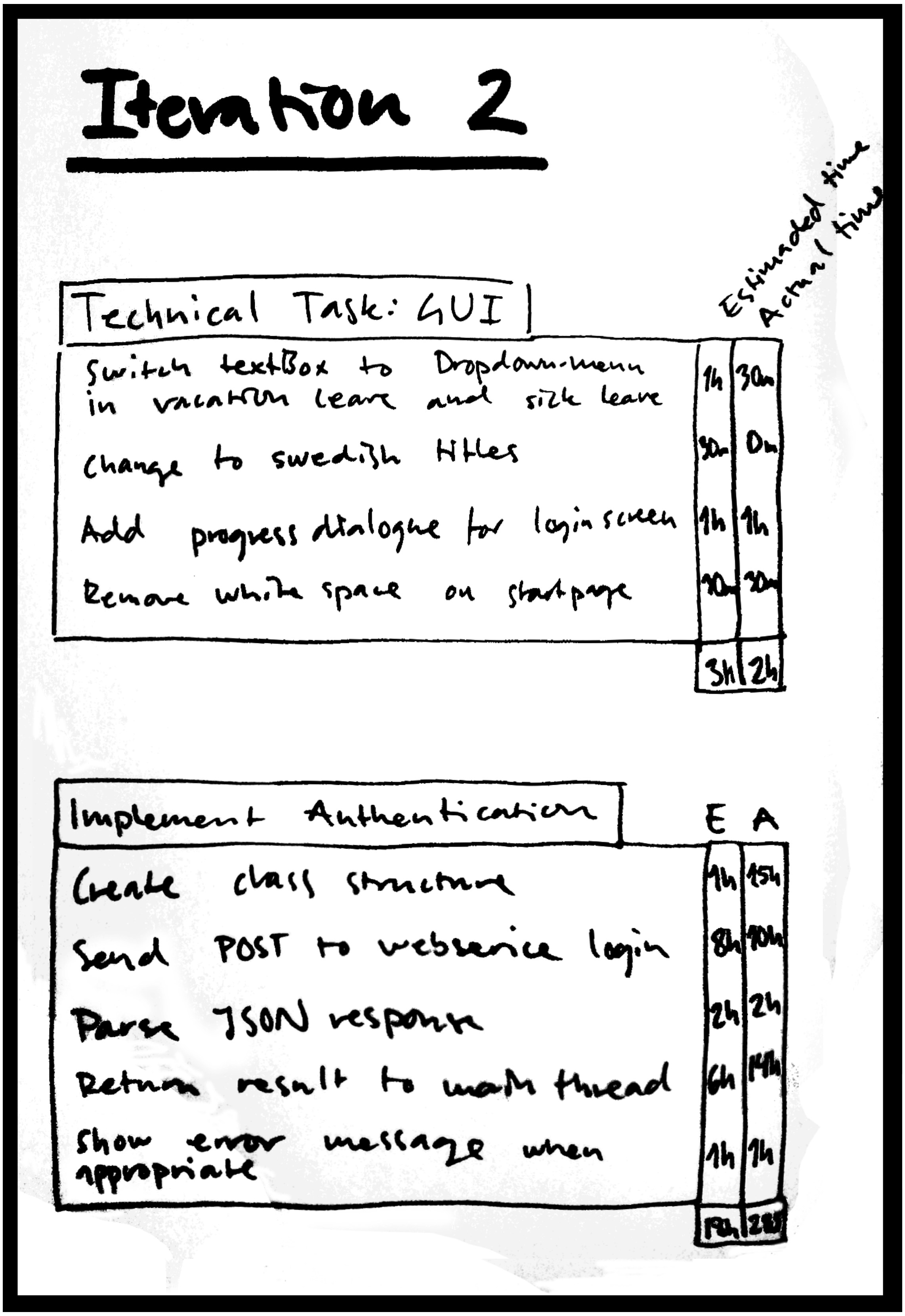
Impediments Add..Import..

Open	Resolved

Waiting for developer.abou.se...

Find: intent

Figure 9: The Team Foundation Server used for task management during the first week of implementation



Iteration 2

Technical Task: GUI

	Estimated time	Actual time
Switch textBox to Dropdown-menu in vacation leave and sick leave	1h	30m
Change to swedish titles	30m	0m
Add progress dialogue for login screen	1h	1h
Remove white space on startpage	30m	30m
	3h	2h

Implement Authentication

	E	A
Create class structure	1h	1,5h
Send POST to webservice login	8h	10h
Parse JSON response	2h	2h
Return result to main thread	6h	14h
Show error message when appropriate	1h	1h
	18h	28h

Figure 10: A simpler solution for Task Management used after the first iteration of the project.

5.2.3 The Pomodoro Technique

The main objective in using the Pomodoro technique is keeping focus throughout the day. Its first objective is to keep the mind from wandering; simply give the programmer an incentive to keep focusing even if the task at hand is uninteresting. On most days using the Pomodoro method seemed redundant as keeping focus was not a problem. However, using it gave a better sense of being in control of the big picture. It allows diving in to details and working hard at solving a specific issue and then after 25 minutes coming up for air and regaining focus.

5.2.4 Weekly Presentations and Updated Priorities

Every week the work was presented to two representatives from the company during what resembled a scrum meeting. These meetings usually lasted for about one hour and during that time three main topics were discussed. The first topic was the current status, whether the goals for the past iteration had been met and if the project as a whole was going according to plan. A comparison between the estimated time to finish the iteration and the actual time it took was made. The second topic was difficulties, covering problems that had been solved, unresolved issues and potential future difficulties. The third and last part of the meeting consisted of planning the coming iteration. Making a detailed plan had a high priority as visualizing the goals was considered helpful in achieving them. First the remaining work was prioritized by the company representatives and a common decision was made about how much work should be taken on for the next iteration. The work was divided into work items and the work items were in turn divided into tasks. Finally the time to finish each task was estimated by the programmer and the time frames and the reasons behind them were presented to the company representatives. During the process of planning the iteration discussions were kept open, as opposed to one party making final decisions, everyone was involved in making the goals of the iteration as rewarding as possible for the developer, company and the end-users.

5.2.5 Monthly Deliveries and Customer Test

After four weeks of developing the application a visit was made to the two pilot preschools. The visited preschools are the first two to try the new system and following a successful trial run another 15 preschools will follow in August 2011. The developed application is intended to be used by parents whose children are preschoolers so the application was demonstrated to roughly fifty parents who all got to use it in a test environment. Three fictional children had been added; "Erik", "Sara" and "Otto". The testers were asked to use the application to login, report sick leave and vacation leave and thereafter log out. During the process the testers were asked to think out loud in order to get as much feedback as possible. Still it was hard to get useful information from this customer test as few parents were familiar with Android devices. The main part of the testers found the application easy to use and asked that it would be kept as simple as possible. Only a few testers had more specific criticisms.

From the beginning a second trip to the preschool was planned following another four weeks of development, in accordance with Agile Solo iterations. These plans were changed for two reasons. The first was that the parents had very little feedback to give concerning improvements to the application and the expectation was that they would not bring more to the table at a second demonstration. The second was that the graphical user interface of the application was nearly finished during the first demonstration and not very much visible change would be made in the following four weeks. Therefore it was decided that the second monthly delivery was to be made as a final demonstration at the Abou office instead and invitations were sent out to 20 employees who would hopefully share some interesting thoughts. A thorough test was also carried out with a project manager at Abou playing the role of a preschool parent. The outcome of this user test resulted in a more detailed confirmation being displayed to the user after reporting vacation leave in the application.

5.2.6 Test Driven Development

The practice of test driven development was an important part of following Agile Solo. For a developer who has never written a single unit test before, writing a test for each and every piece of functionality is a big challenge. Doing it in Android seemed like an even bigger challenge as the native test libraries in Android are rather complicated to use due to the Android lifecycle. An Android application consists of one or more activities and writing tests covering more than one activity proved rather difficult and time consuming. (27)

After working with native Android testing during the first iteration a third party library called Robotium was introduced which made testing easier. (28) However, testing this type of GUI heavy application is very time consuming so tests were not written for everything, but mostly for core functionality and for parts of the code that could possibly cause errors that would be hard to discover.

5.2.7 Peer Code Review

A peer code review was scheduled once a week, due to time constraints, not as often as was initially planned. A senior developer and supervisor would review the code. First a short demo of the application would be carried out in order to better understand the code. Following that a walkthrough of all code that had been introduced since the last review seemed to be the fastest way to get up to speed. During the whole process an open discussion was kept on how both the application and the code could be improved. The focus was kept on the code to be simple, readable and contain plenty of comments so that it would be easy for other developers to continue the work after the completion of the project.

5.2.8 Auto Code Review

After finishing a 25-minute Pomodoro in which code has been produced, the next Pomodoro would always start with a quick code review performed by the programmer. This activity would only be given a very short period of time, somewhere between two and ten minutes, with a preference for the shorter time-span. Following this rule required much discipline since it would be very tempting to start working on something new after having finished a task.

5.2.9 Visual Control Modeling

The only items that were kept for visual control were the planning sheets for the iteration, a fluid list of new ideas and a regular calendar. The three documents were considered to give enough overview of the current process state and were easy to maintain.

5.3 Result of the Preschool Portal Android Application

In total, nine weeks were spent developing the application. The goal was to maintain the concept and flow of the iPhone-application, the touch-screen system and the web portal. In doing this it was also important that no Android guidelines were broken so that a regular Android user would feel like the application was well designed and didn't feel contrived. The activity diagram shown in chapter 5.2.1 was followed with a few minor changes so that the application flow would be consistent with that of the iPhone-application.

5.3.1 Login screen

The login screen consists of two text boxes, a checkbox and a login button. The text boxes use what are called hints, making it easy for the user to understand what information should be entered in each text box. The check box is used to allow the user to choose between saving the login information and discarding it after logging out. The login function is protected by the use of regular expressions controlling the format of the social security number and the PIN-code entered by the user.

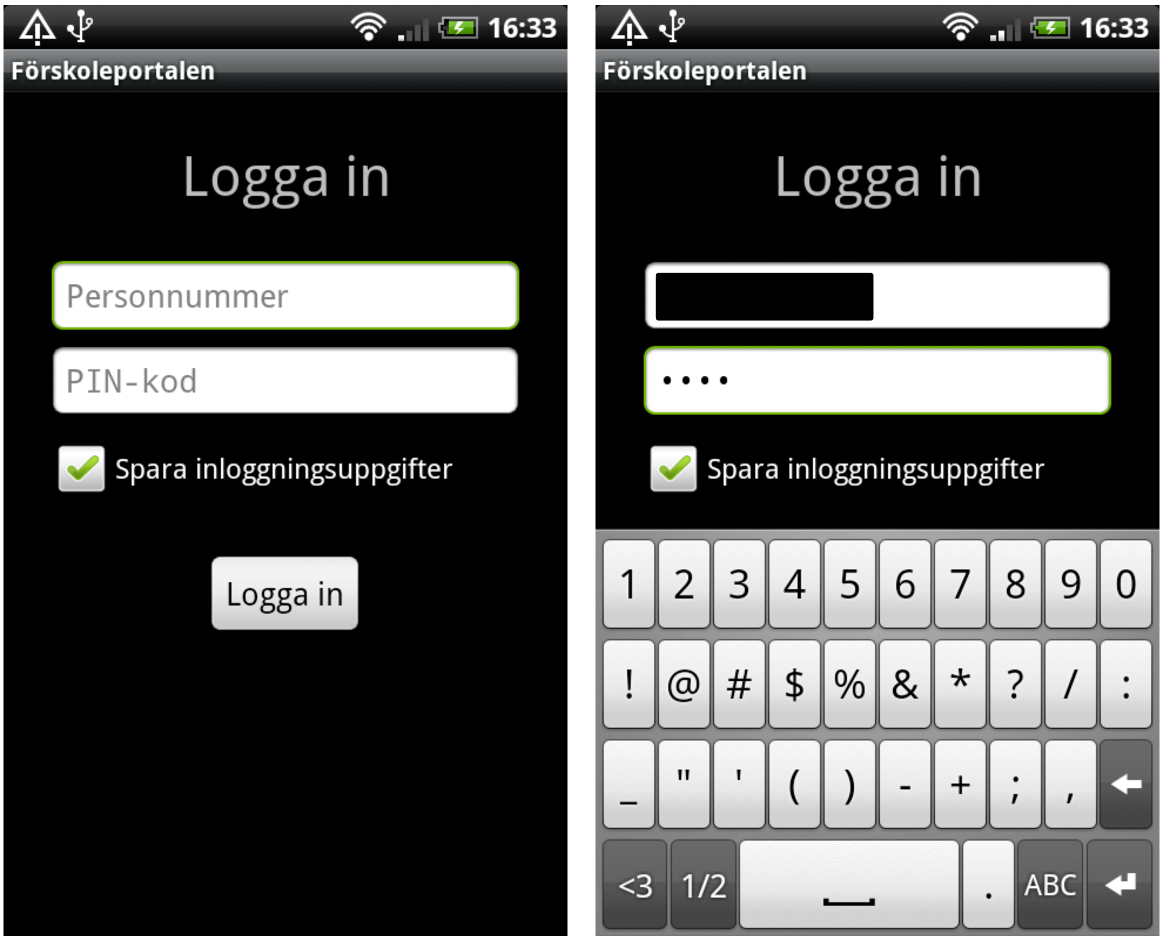


Figure 11: Screenshots from the login screen

5.3.2 Start page

The start page is a simple view giving the user a choice between two different functions. The first being reporting vacation leave and the second reporting sick leave or that a child is healthy and ready to go back to their preschool. In order to make the view more visually appealing and to make the application more usable and similar to the touch system, it features the same image buttons as the touch screen system. The image buttons show respectively a suitcase for vacation leave and a hospital cross for sick leave.

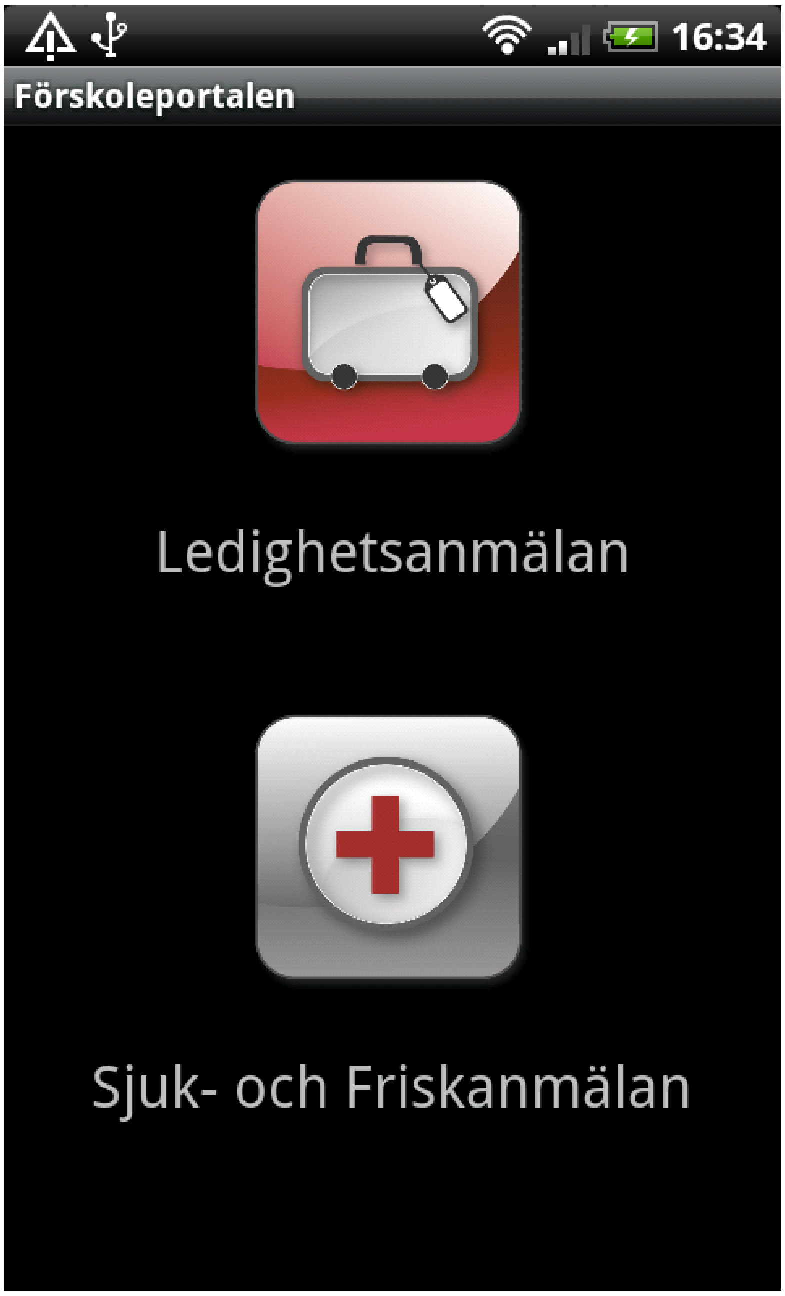
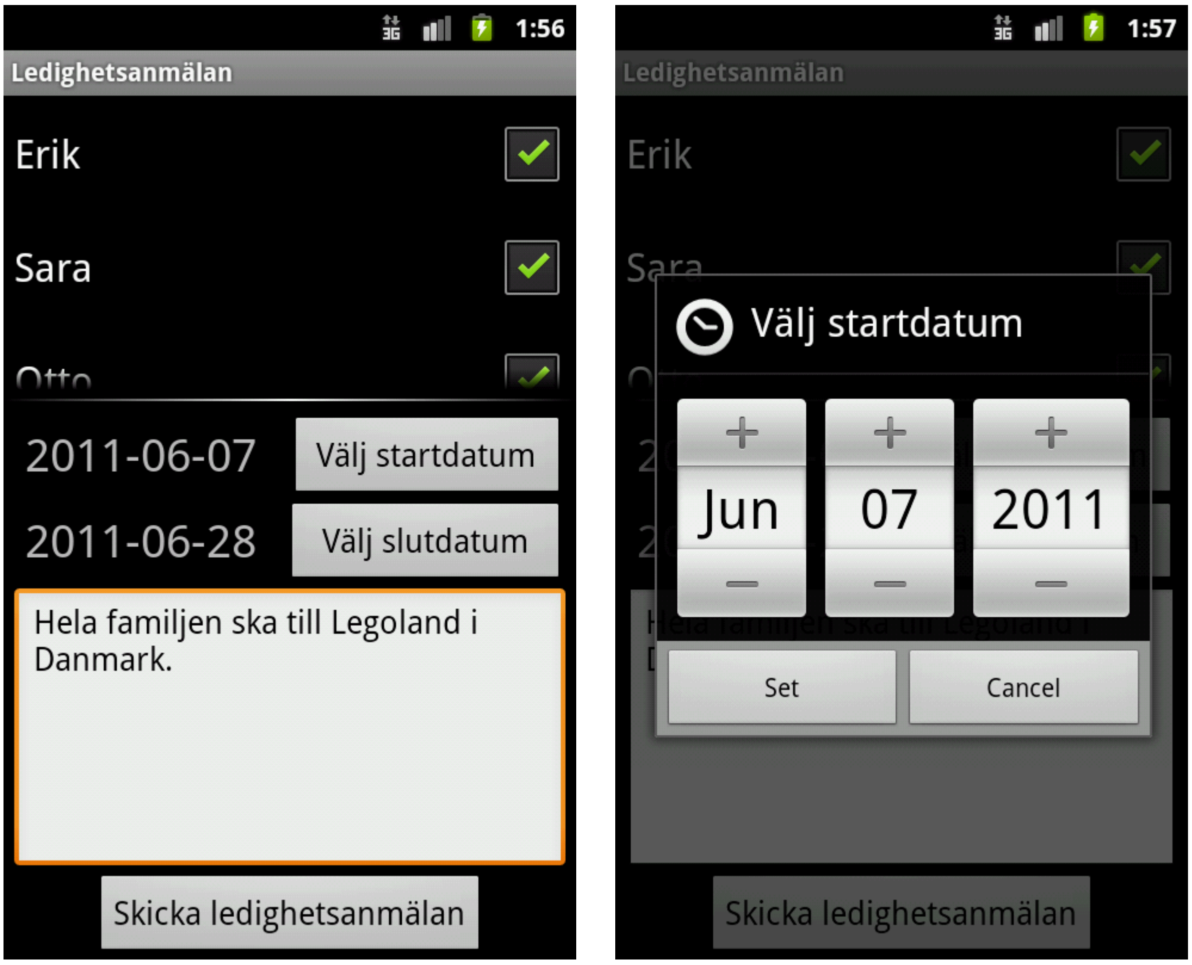


Figure 12: Screenshot from the start page

5.3.3 Vacation leave view

This view shows a scrollable list of children on top, the list will contain only the children of the logged in parent as opposed to all children at the preschool as seen in the touch-screens systems. On the lower half of the screen there is a control panel containing labels and corresponding buttons for choosing start and end dates for the vacation leave. There is a textbox with a capacity of 200 characters for writing a message to the preschool teachers regarding what the vacation leave is for. At the bottom there is a button for sending the information. In this scroll view at least one child has to be selected. Selecting multiple children is a new function that is not available in any of the other preschool systems. This function has been requested by many parents as they usually bring all of their children on vacations. Positive feedback was also given during the parent demonstrations four weeks into the development phase.



2011-06-07

2011-06-28

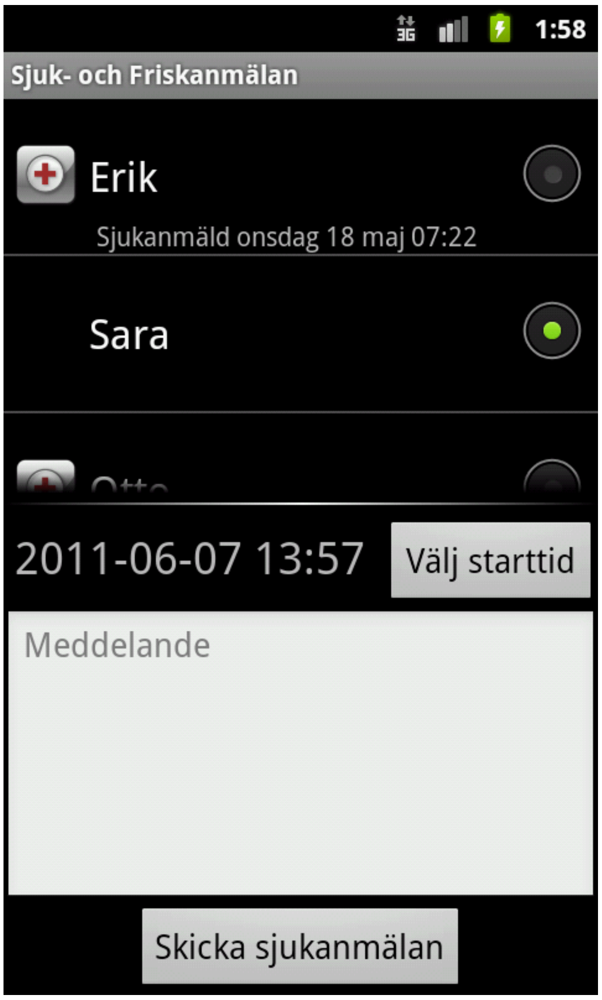
Hela familjen ska till Legoland i Danmark.

Jun 07 2011

Figure 13: Screenshot from the vacation view

5.3.4 Sick leave view

This view is somewhat more complicated as it contains functionality to report a child being sick and a child being healthy and returning to their preschool. The list of children is the same as in the vacation leave view with the difference that radio buttons are used as opposed to check boxes so that only one child can be selected at a time. The list also had an indicator showing on the rows of sick children as well as text showing on what date the child was reported sick. The big difference in the control panel is that the view changes if the selected child is sick or healthy. If the child is sick an opportunity to report the child healthy is displayed on the control panel, in this case the only choice that can be made is the end time when the child will be returning. If the selected child is healthy the view changes so that a sick leave report can be sent. In this case a start time can be selected and a message entered.



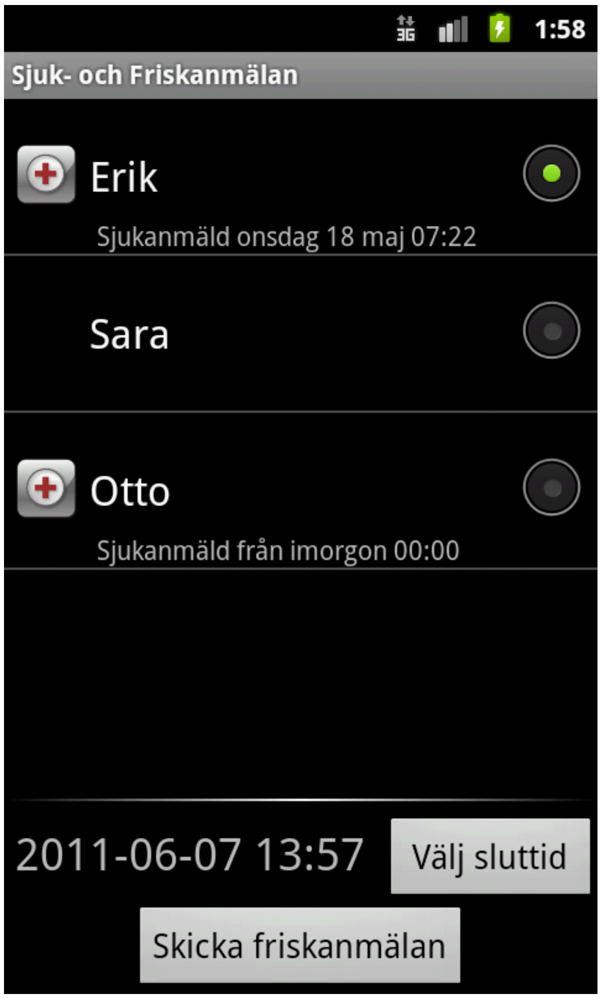


Figure 15: View showing if the selected child is healthy

Figure 14: View showing if the selected child is sick

6 Discussion

This chapter is written as a reflective account and evaluation of the process model Agile Solo in comparison to reviewed literature on the subject. The evaluation of the software development practices and the process model as a whole is based on the experiences gained from carrying out the case study as described in chapter 5.2.

6.1 Modeling

Before starting the implementation phase, two documents were produced in order to model the desired end-product; an activity diagram and a sequence diagram. The idea was that it would prompt the communication between the programmer and the customer and function as a visualization of a possible successful end-product. The expectation was also that the process of creating these documents would lead to a better understanding between the customer and programmer. Improved communication is one of the most important benefits of modeling as described in Scott W. Amblers book Agile Modeling. (29) In more complicated projects modeling improves communication in an efficient way but in this case there was no real need for it. In a single programmer project it might be the case that the communication between customer and developer is more efficient in a face-to-face discussion.

Producing these types of diagrams is time consuming and even if they provide an easy overview for the programmer they might not be so clear to a customer without a background in software development. Modeling in a large project greatly benefits knowledge sharing among developers, something that is irrelevant in a solo project. Perhaps it is better to use a language adapted for customer-developer communication rather than modeling that is primarily for developer-developer communication.

In the case study modeling did however confirm that we already had a good mutual understanding of what the application should look like. A different solution that might be attempted in order to reach that conclusion in the future is producing simple mock-up screenshots when there is a need for modeling. It is less time consuming and that way it is as easy for the customer to understand the model as it is for the developer.

Producing these diagrams could also have been justified by frequent use by the developer during the implementation. The expectation was that they would be useful to the developer since they gave a good overview of the system as a whole. However, during the entire implementation phase, these documents were actually not revisited once. I think that it was mainly because the tasks included in an iteration were well defined and that the system as a whole was always up for discussion during the iteration planning meetings so there were never any uncertainties.

6.2 Pomodoro vs. Montessori

The main reason for using the Pomodoro Technique in Agile Solo was to increase the number of effective work hours in a day. As described by Francesco Cirillo, the man who invented the Pomodoro Technique, we interrupt ourselves very often in our daily work and it makes us inefficient. According to Cirillo, working in Pomodoros increases the time spent working uninterruptedly. (26) However, using it in the process of developing the Android application gave a different result. I found that programming for 25 minutes only was rather unsatisfactory. When the bell rang indicating that it was time for a break it was never a relief, on the contrary I felt that it interrupted my concentration. About ten minutes in to a Pomodoro I would be in a state of deep concentration, a state where I would be most effective in my code writing, by that time only 15 minutes of time would remain before I would be interrupted again. Coming back to my work after a five-minute break I felt like I had lost momentum and had to start all over again. For me this was less than optimal so I did some more research. Maria Montessori had some interesting ideas regarding time boxing. Her philosophies were formed in 1917 to increase efficiency in educating children. Instead of having all children follow the same strict schedule she promoted letting the child herself choose what to work with for large blocks of uninterrupted time. She felt that a child should never be forced to take a break when they were in a state of deep concentration. (30) Allowing children to work this way showed that they had a great capacity for concentrating on a single task for a long period of time. After observing their behavior she saw that a child finishing a work period in their own time emerged from their concentration calm and very satisfied, almost as after a nap. (30) For me this way of working has merit even in the context of adults working with software development. I found that turning the Pomodoro timer off and allowing myself to dive in to code writing for as long as it took to finish my task was very rewarding. I was able to work hard for several hours and emerged satisfied with what I had completed. However, the Pomodoro method could also be a useful tool. When taking on a less complicated and perhaps less interesting task such a producing documentation or doing research I found the Pomodoro method rather helpful in warding off distractions and controlling impulses to go do something else. In these situations I could see the benefits described by Cirillo.

6.3 Weekly code reviews

Having code reviews with a supervisor was a great way of quality checking the code. Looking through the code written since the last review gave me a chance to consider my solutions and whether they were as clean, readable and effective as they could be. I found many bugs in my code during these sessions both while showing my progress in the application and while reviewing the code. A positive effect that I consider just as important, if not more important, was that during my implementation I tried harder to write perfect, well commented code since I knew that I would later have to show it to a senior developer. At first it seemed intimidating but I also saw it as a great opportunity to learn and in order to learn as much as possible I tried to write the best code possible. The only negative aspect was that the review sessions took a lot of time that my supervisor was hard pressed to find. In hindsight it might have been a good idea to exchange this service with a fellow thesis worker in addition to the supervisor review. That would have given me an opportunity to schedule more frequent reviews and to be more thorough in the code review since the time restraints would have been significantly reduced.

6.4 Auto code review

It proved more difficult to assess my own code than I expected. According to Agile Solo, the auto code review was to be performed a short time after writing the code. It might be the case that walking away for only a few minutes does not give enough perspective to review the code in an objective way. It was found that starting the code review I was predisposed to assume that my solution was the best solution and it was difficult to think of other possible ways of solving the problem. I did however find small bugs and uncommented code during these reviews so they were useful. If I were to use auto code review in a future project I would try extending the time between writing the code and reviewing it to at least 24 hours. Auto reviewing code is not a practice that is used in established process models and it is not discussed in the literature reviewed for this thesis work. Further evaluations are therefore needed to confirm these conclusions.

6.5 Monthly user tests

During my project I only had time to do two end user presentations. The first test was at the preschool and it was a large but not very thorough test as many of the parents were pressed for time. The collected information that I got from the parents was very useful and important to respond to, so taking the time to do it was definitely worth it. Even if I had hoped to get more feedback than I did, some of the things the testers wanted me to change were things that I would never have thought of myself. When developing a user interface it can sometimes be hard to see what parts of it that seem complicated to the user. Testing it on your peers only help so much as they are often more familiar with using software than some of the end-users.

The second test was carried out in collaboration with an Abou employee posing as a parent. That test was also worthwhile. The application was nearly finished at that time and was well received. I got some good feedback and made some minor improvements to the application. Having these presentations and tests helped me feel confident that the project moved along in the right direction and that the application was not hard to understand by the users.

6.6 Task Management

After the first week of development the task management system was changed from using a Scrum backlog system to a much simpler method where each task was written down on paper. The simpler method worked fine and served its purpose. It was easy to follow the progress of the iteration this way. It was also easy to calculate the expected implementation times and compare them to the actual times at the end of the week. In extreme programming these are the most valued benefits of using a task management system. (18) I would prefer to use a simple system in a future solo project. However, something that the simple system lacked was traceability. I would have liked to have a way of easily looking over the past few iterations and maybe create some statistics. The Scrum system that was used during the first week had a nice feature of connecting to the version control system that was used and thereby making it easier to make rollbacks to a specific state of the program. This is not something that I find necessary, but a feature that is nice to have.

6.7 Test Driven Development

Although I found it hard to get started with and more time consuming than I thought it would be I would never exclude it from a development process. Using Test Driven Development made me carefully consider what my goal was before I wrote any code and I think that it made me write less code with fewer errors. Although it was time consuming I think that with more practice it will feel like a natural way of writing code and it will be easier to do it in an efficient way. These conclusions align with what is described in Extreme Programming – A hands on Approach; test driven development is hard to get started with but later becomes a natural part of development. (18)

7 Conclusion

The purpose of writing this thesis was to define an agile software development process for a single developer. The development process was intended to be helpful for any single programmer in any project. After performing the case study, comprising the evaluation of Agile Solo, the conclusion was reached that no development process is a perfect fit for every project. There are practices in Agile Solo that were extremely useful under the specific circumstances that they were tested and there are practices that are not considered appropriate to use in similar projects.

Agile Solo is a good place for single developers to start at but it is recommended that they adapt the development process to fit their specific project and their style of producing software. The individual practices of Agile Solo should be adjusted so that they are tailored to fit the project, programmer, customer and situation perfectly. It is also suggested that the development process is evaluated and further adapted during the course of the project.

8 References

1. **Ha, Peter.** Time Magazine. [Online] Time Inc., 10 25, 2010. [Cited: 05 25, 2011.] http://www.time.com/time/specials/packages/article/0,28804,2023689_2023708_2023677,00.html.

2. **THE COMPUTER LANGUAGE COMPANY INC.** PCMag.com. [Online] [Cited: 05 25, 2011.] http://www.pcmag.com/encyclopedia_term/0,2542,t=Smartphone&i=51537,00.asp.

3. **RIM.** press.rim.com. *financial releases.* [Online] 04 06, 2006. [Cited: 05 25, 2011.] http://press.rim.com/financial/release.jsp?id=1013.

4. —. press.rim.com. *financial releases.* [Online] 04 7, 2004. [Cited: 05 25, 2011.] http://press.rim.com/financial/release.jsp?id=465.

5. **THE COMPUTER LANGUAGE COMPANY INC.** answers.com. [Online] [Cited: 05 26, 2011.] http://www.answers.com/topic/smartphone.

6. **Canalys.** [Online] 01 31, 2011. [Cited: 05 26, 2011.] http://www.canalys.com/pr/2011/r2011013.html.

7. **Sommerville, Ian.** *Software Engineering.* 8. Harlow : Pearson Education Limited, 2007. pp. 8-9.

8. *Iterative and Incremental Development: A Brief History.* **Larman, Craig and Basili, Victor R.** 6, s.l. : IEEE Computer Society, 06 2003, Computer, Vol. 36, pp. 47-56.

9. **AndroLib.** AndroLib.com. [Online] 05 26, 2011. [Cited: 05 26, 2011.] http://www.androlib.com/appstats.aspx.

10. **Apple.** Apple.com. [Online] 2011. [Cited: 05 26, 2011.] http://www.apple.com/iphone/apps-for-iphone/.

11. **Beck, Kent, et al., et al.** Manifesto for Agile Software Development. [Online] 2001. [Cited: 05 30, 2011.] http://agilemanifesto.org.

12. **Abou AB.** [Online] 2011. [Cited: 05 26, 2011.] http://www.abou.se/.

13. **Karlson, Helena.** [Online] 03 28, 2011. [Cited: 05 26, 2011.] http://pedagogstockholm.se/Utveckling/Forskola/Projekt/Tidstjuvarna-forsvann-med-digital-kommunikation/] .

14. **Schwaber, Ken and Sutherland, Jeff.** Scrum.org. [Online] 2010. [Cited: 05 26, 2011.] http://www.scrum.org/storage/scrumguides/Scrum%20Guide.pdf.

15. **Scrum Alliance Inc.** [Online] [Cited: 05 26, 2011.] http://www.scrumalliance.org/learn_about_scrum.

16. **Magno, Alexandre.** Scrum Alliance. [Online] 08 06, 2007. [Cited: 05 26, 2011.] http://www.scrumalliance.org/articles/62-the-daily-meeting-trap-.

17. **Wells, Don.** extremeprogramming.org. [Online] 09 28, 2009. [Cited: 05 26, 2011.] http://www.extremeprogramming.org/.

18. **Steinberg, Daniel H. and Palmer, Daniel W.** *Extreme Software Engineering - A Hands-On Approch.* New Jersey : Pearson Education, Inc, 2004.

19. **TOYOTA MOTOR CORPORATION.** [Online] [Cited: 05 26, 2011.] http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/just-in-time.html.

20. **Lean Velocity.** Kanban. [Online] 2007. [Cited: 05 26, 2011.] http://www.leanvelocity.ca/Content.cfm?C=6681&SC=1&SCM=1&MI=4379&L1M=4379.

21. **Crisp AB.** Kanban. [Online] [Cited: 05 26, 2011.] http://www.crisp.se/Kanban.

22. **Benson, Jim.** Modus Cooperandi - Performance Through Collaboration. *Why Limit Work in Progress?* [Online] 2010. [Cited: 05 27, 2011.] http://moduscooperandi.com/personalkanban/why-limit-work-in-progress/].

23. **Cockburn, Alistair.** [Online] 06 19, 2008. [Cited: 05 26, 2011.] http://alistair.cockburn.us/Crystal+methodologies.

24. **Fowler, Martin.** [Online] 12 13, 2005. [Cited: 05 26, 2005.] http://www.martinfowler.com/articles/newMethodology.html#Crystal.

25. **Cockburn, Alistair.** *Agile Software Development.* Boston : Pearson Education, Inc, 2002.

26. **Cirillo, Francesco.** *The Pomodoro Technique.* u.o. : FrancescoCirillo.com, den 19 10 2006.

27. **Google.** Android Developers. [Online] 2011. [Cited: 05 26, 2011.] http://developer.android.com/reference/android/app/Activity.html.

28. **Reda, Renas.** Robotium. [Online] 2011. [Cited: 05 26, 2011.] http://code.google.com/p/robotium/.

29. **Ambler, Scott W.** *Agile Modeling.* New York : John Wiley & Sons, 2002.

30. **Montessori, Maria.** *The Advanced Montessori Method - Spontaneous Activity in Education.* New York : Frederick A. Stokes Company , 1917.

31. Wikipedia. [Online] [Cited: 05 25, 2011.] http://en.wikipedia.org/wiki/kanban.

32. **BBGeeks.** BBGeeks.com. [Online] [Cited: 05 25, 2011.] http://www.bbgeeks.com/blackberry-guides/the-history-of-the-blackberry-88296.

9 Attachments

Område	Feedback	Prio	Kommentar
sjuk- & friskanmälan	Vill ha påminnelser i telefonen om att friskanmäla sitt barn	inte så viktigt	Kanske lägga till ett statusfält på startsidan i appen?
logga in	Vill att tangentbordet automatiskt visar siffror vid inloggningen	viktigt	Ska ändras
ledighetsanmälan & sjukanmälan	Direkt efter att man har lämnat en sjuk- eller ledighetsanmälan ska man komma tillbaka till startsidan	viktigt	Ska ändras
ledighetsanmälan & sjukanmälan	många tycker att det är svårt att se att det finns knappar för att ändra start- och slutdatum	ganska viktigt	Kanske lägga actionlisteners även på datumfältet? Överväga att skriva "välj startdatum" på knappen.
ledighetsanmälan	Det ska stå startdatum på knappen istället för starttid		Ska ändras
allmänt	Vill ha mycket fler funktioner, helst alla som finns i webportalen		
allmänt	Vill ha funktionen "ändra enstaka tid" i appen		
allmänt	Många ringer och sjukanmäler och vill sedan friskanmäla via webb-portalen och då finns ingen aktiv sjukanmälan.		

Figure 16: Feedback from the first user test