CHALMERS

Comparing Agile XP and Waterfall Software Development Processes in two Start-up Companies

Master of Science Thesis in the Programme Software Engineering and Technology

JENNIFER DORETTE J.

Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, November 2011

Comparing Agile XP and Waterfall software development processes in two Start-up companies

JENNIFER DORETTE JACOB

Supervisor: WOLMET BARENDREGT
Examiner: MIROSLAW STARON

Chalmers University of Technology
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 November 2011

COMPARING AGILE XP AND WATERFALL SOFTWARE DEVELOPMENT PROCESSES IN TWO START-UP COMPANIES

ABSTRACT

The work done is an attempt to investigate a suitable software development process to be followed in a fluid entrepreneurial environment. To achieve this goal, the study is conducted in the form of two different case studies where existing software development processes namely the waterfall model and agile XP are applied in two different startup organizations with similar backgrounds.

The result of the work is that the two software development processes are engineered to be suitable for applying in different entrepreneurial setups, one in the presence of a client and another without communication with the client.

From the results obtained in the study the author concludes the agile software development processes to be more suitable to be applied in a startup company taking into account important factors such as knowledge sharing and communication within the team.

Keywords: Startup Organization, Waterfall Model, Agile XP

Table of Contents

List of Figures

List of Tables

1. Introduction

A startup company or "a startup" is a company which has a low operating history or in its first stage of operation. These companies are newly formed and are in the stage of capitalizing its product/service and research for markets [1]. Startup companies generally lack a well defined organizational structure and rarely have a matured/structured software development process in place. Software development processes play a vital role in the development of quality software and shapes the organization to meet employee goals such as performance and co-ordination. This thesis report therefore analyses the outcome of different software development processes that would best fit a startup organization or any other fluid entrepreneurial* company. The drawbacks of adopting these processes if used in such an environment and the other measures that can be undertaken to overcome these drawbacks are also analyzed. This analysis is done in the form of two case studies conducted in two separate startup companies which have similar management backgrounds. The companies involved in the study are Tesenso which was formed within Encubator AB, Gothenburg and Krizp system, India. The first case study is coupled with the simultaneous development of an end-user web based solution for Tesenso clients to handle the collection and disposal of waste from trash cans. The second case study conducted at Krizp System includes a series of interviews and does not involve the author developing any web based solution for the company.

*Fluid Entrepreneurial: A start-up company with a flexible management. The management is not involved in the activities of the team and also does not have a strict schedule for the employees to follow. The work carried out in such organizations is unstructured.

1.1. Problem Statement

To investigate a suitable software development processes to be followed within a fluid entrepreneurial environment, in the form of two case studies conducted in two different organizations with similar backgrounds but applying different processes: the waterfall model and agile XP.

1.2. Research Question

RQ1: Which software development process is suitable to be followed within a fluid entrepreneurial environment: the waterfall model or agile XP?

This question will be addressed in Chapter 7 of the report where the author concludes agile XP to be a better development process to be followed within startups under similar situations based on the results obtained from the case studies.

1.3. Purpose

The goal of the thesis therefore is to analyze the pitfalls and advantages of working with a specific software development processes such as the waterfall model or agile XP, in small projects within fluid entrepreneurial* environments.
Thus this thesis provides a detailed exposure to software development in real life providing sufficient knowledge on processes which are likely to be followed in startups. A by-product of this thesis is to provide a complete web based solution where Tesenso clients can manage

their accounts and retrieve information on schedules and routes for garbage collection by mentioning their custom criteria. The analysis on the waterfall model on Tesenso was done while developing this by-product of the thesis.

1.4. Disposition

The rest of the report describes the following main topics: Project Methodology, State-of-the-art, Previous work done on research, Results, Conclusion and Discussion.

Method:

This chapter describes the methodology carried out to proceed with the project. The planning of the two main parts of the project which are research and case studies are explained here as well. A brief overview of the two case studies that will be performed during the thesis to investigate processes is also provided in this section of the report. In-depth details on the case studies will be provided in the Results section of the report.

State-Of-the-Art:

This chapter gives an overview of the major theoretical concepts that are related with this project. Introduction to the basic concepts of the sequential software development model (i.e.) the waterfall model and agile methodologies is done here. Furthermore the chapter explains about the theoretical aspects of an entrepreneurial environment. This section therefore gives the reader necessary knowledge on basic concepts on which this thesis is based upon.

Previous Work on Research:

The fourth chapter of the report summarizes all the findings from previous research on the waterfall model and agile XP process and its effect in projects. A general comparison to both the software development processes is made in this section. The chapter also describes the motivation behind the research question for the thesis work. It is a frame of reference from which the research question is formulated based on data which were found lacking in already existing research articles and papers.

Results:

This section of the report talks about the two case studies conducted in detail. Case study one talks about the waterfall model and case study two talks about agile XP. The analysis from these case studies is also made here. It summarizes the planning of interviews and the observed outcomes. This chapter can also include results/screenshots from the implementation of the web portal.

Conclusion:

This chapter summarizes the thesis by describing the learning outcomes and main experiences in the project. It gives a brief description of the limitations/demarcations of the project. The thesis is wrapped up by providing some suggestions for future research and summarizing the whole research and project.

2. Method

In this thesis the following two strategies will be adopted to address the aforementioned problems for the research of suitable processes and development of solution.

2.1. Research Methodology

"Research can be termed as a logical and systematic search for new and useful information on a particular topic. The outcome of a research is to verify and test important facts, to discover new facts and to overcome and solve problem occurring in our everyday life" [2].

In order to get enough information, the study was divided into the following parts:
a) Literature survey
b) Case studies

a) Research on software development processes is accomplished by extensive literature review on available software processes and to understand the nature of these processes. In this thesis the main focus will be on the waterfall model and eXtreme Programming an agile methodology. The study was intended to get some insight into processes in organizations which is later compared with the results obtained in the case studies

b) "The essence of a case study is that it tries to illuminate a decision or a set of decisions [3]." Case studies can be exploratory, descriptive, explanatory or improving. Since the thesis involves two case studies there is need to describe the differences in both case studies in the way it is conducted and how the results obtained are finally compared to arrive at a desired solution. Yin describes a case study research as a linear but iterative process [3]. It involves the following phases as shown in Figure 1: Plan, Design, Prepare, Collect, Analyze and Share.

In the planning phase of the case study research the research question is identified and the method in which the case study will be carried out is determined. Since the present thesis involves two case studies, the author follows an exploratory and comparative method for determining the results. Using an exploratory method we determine what is happening in the present situation, look for new ideas and insights and finally determine a hypothesis for the problem. The results obtained in both the studies are later compared to see which solution is more applicable for the current research question [3].

The planning phase is followed by the design phase of the study and it includes selecting cases and design of data collection protocol [3]. Here the organizations to perform the research on are chosen and the data collection method is identified. In the first case study data is collected while working on an experimental solution were the author observes the results at the end of each stage of development and in the second case study data is collected in the form of interviews and questionnaires. Interviews are conducted with the company's personnel mainly to understand their view on software development processes and the retrieve results while performing the scenarios in the relevant process.

The next step in case study research is to prepare, collect and analyze the data. These steps follow the case study protocol from the design phase. Information is collected in the form of interviews, direct observation and participant-observation. "Participant-observation deals with interpersonal behavior and motives. Here we may assume a variety of roles within the case

study situation and directly participate in the events being studied [3]." Interviews in this study will "…be a guided conversation instead of structured queries" [3]. Participant-observation is performed in case study 1 since the author is an active participant in the organization whereas interviews and direct observation is performed in case study 2.

The final stage of the case study is to analyze and compare the results obtained in both the case studies. The end result of this research should provide the companies involved with an overview of the process and a successful method to adopt these processes into practice. While the waterfall model is evaluated in Tesenso, the agile process is investigated in Krizp system and the results are tabulated.

Though the two case studies adopt different methods, the results obtained from both studies can be compared to arrive at a desired outcome.

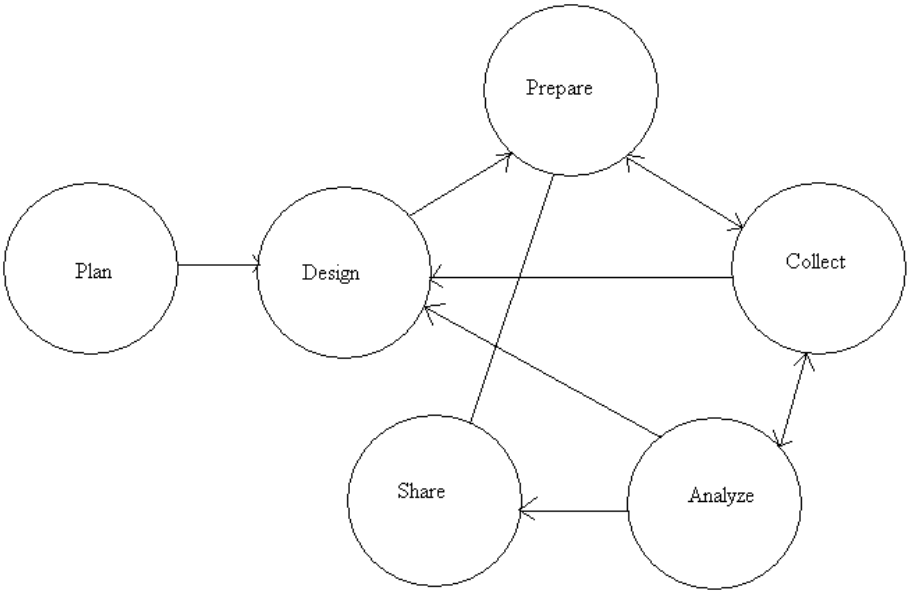

Figure 1: Elements of Case Study Research [3]

2.1.1. Sample Selection

The targeted organizations will be observed as probable users of the introduced processes. The organizations were chosen based on two main options: the company should be a startup company with not many employees in place and with very few experimental projects in hand, the company should until now not be following any well defined software development process because the idea here is to introduce a new structured process in an environment which has no knowledge on software development processes before and having a very flexible management.

The first case study was based in a company in Sweden because in general Swedish startups have agile software development process in place and as there was a need to introduce the waterfall model for the study a company from Sweden was chosen for case study 1. However, the second case study was based in India because agile is a new concept to the Indian industry

and most startups follow the waterfall model by default. Secondly it is also rare to find a startup in Sweden which does not follow any structured software development process.

2.1.2. Validity

"Validity is the best available approximation of the truth of a given proposition, inference or conclusion [30]". There are four different validity types such as construct validity, internal validity, reliability/conclusion validity and external validity [30].

Construct Validity:

"Here we observe if we implemented the program we intended to implement and did we measure the outcome we wanted to measure [30]". During the study this treat was mitigated by choosing the right people for interviews. The selection of candidates for the interviews in Case study 2 was done based on their role in the organization and candidates were made to perform activities pertaining to their role to ensure that the final results were not biased. The second was to perform the right data interviews because there was a high chance of misinterpreting the data collected. To address this problem some questions were repeated more than once to determine the correct answer. If clarifications on the interview were required, the same questions were revisited during the study. However in case study 1, the author is directly associated with the organization and not an external observer. Therefore the actions performed within the organization were not biased.

External Validity:

"External validity is the ability to generalize the finding to other persons, places or time [30]". One potential threat to this validity is the study was conducted only in two organizations each performing either the waterfall model or agile XP. The author did not have the chance to perform the waterfall model/agile XP in two startups to compare the results because of the time constraints and less availability of startups under similar circumstances for the study. The results obtained during the study are in view of a single startup following the specific process which is either waterfall model/agile XP. Also, the same results can only be applied to startups having a flexible management but may not be applicable to all startups in general having a dedicated management and a team with several people functioning below it.

Conclusion Validity:

This threat to validity is basically concerned with the possibility of arriving at the same results if we re-do the study in the same environment. In other words, it is associated with replication of the study. One specific way of going wrong in this study is the interpretation of collected data. There is a risk that the results are influenced by the interpretation of the researcher. This is particularly observed in case study 2 since the author is not an active participant as in case study 1. The results from case study 2 are partially based on the views of the members of the organization and how the author observed the occurring outcomes. The results therefore are greatly dependent on external views on the process.

Internal Validity:

One particular threat to internal validity is selection bias [31]. This was observed in case study 1 where the selected test users for the study belonged to different backgrounds from the intended users of the system. Therefore, there is a chance this may have impacted the outcome of software requirements gathered from these users.

2.2. Structure of the Project

The work done in this thesis is not the first time processes to be followed in projects are investigated. However there is not much research on processes to be followed in projects within entrepreneurial companies which have no definite structure in place. Such companies hardly have a process in place due to a lot of factors which will be later dealt in this report. Therefore the idea of the project is to perform case studies on two of such organizations and observe the results.

The following chapters give a detailed description of the case studies.

Case Studies:

Both case studies will include the following concrete activities which are: overview of processes and applying the chosen processes to observe the results.

Overview of the process:

Organizations targeted for this research should get a good overview of what the particular process is about and how it applies to their nature of work.

Applying the process:

The company representatives should be introduced to step-wise scenarios before the study, for successful implementation of the respective process in their firm. This has to be done in order to carry out the study and observe the results. These scenarios describe the nature of the process and are determined from available theoretical sources. The final result of the thesis should however include some kind of concrete activities which are to be followed for process adoption based on the observed results and tailored to the needs of the organization.

Case Study 1:

The first case study will be targeted on applying the waterfall model in Tesenso AB and involves the author being one of the developers in the company. Tesenso started within Encubator AB a year ago and ever since it has undergone several organizational changes. The management team has changed since last year leading to several other ongoing changes on the goals and focus on development of the business. Due to these ever occurring changes the company has not adopted a specific organizational process to be followed. There are a few clients linked to the company who are hoping to see a good web based solution in the near future. The company currently has three developers. The requirements and other technical functionalities of the system are reviewed only with the management team and a previous developer. The project requirements therefore experience change constantly. This part of the report therefore analyzes possible software development cycles and processes that could be best adopted in such a situation taking into account these and several other factors (which will be discussed later in the report) of the organization. This is done while simultaneously developing a web portal for Tesenso clients. The envisioned customers/clients for the system will be the people working for a Swedish municipality who are responsible for clearing trash in garbage bins across the city of Gothenburg. The development of the Web portal will follow the concrete scenarios of the waterfall model and the results of each of the scenarios within this startup are observed. By doing the study the author also observes how the introduced process affects the quality of the developed product and also the efficiency of

the process within the startup in terms of employee satisfaction and co-ordination with test users.

The web based project started as an experimental concept within Tesenso. The aim was to provide scheduling and route optimization services to the waste management industry in Gothenburg, Sweden. A sensor-based application is in place which stores and retrieves information from few waste collection spots. The statistics of the bin are therefore retrieved from these sensors. An end-user interface for the entire project was previously implemented but was found to be too trivial and ineffective with a lot of inaccuracies and requirements that were not met. Therefore, to begin with development of the web based solution it is also required to have a thorough understanding of the existing application and its limitations. The changes in requirements are identified from the previous to current application. It is also required to perform extensive requirement engineering to determine the limitations and possibilities of the new solution. The existing end-user interface is explored and the extent to which it can be modified to add new functionalities so that it is applicable for the new requirements is identified. A suitable web-logic pattern is then identified to implement the current solution. Finally the solution is based in a way that its design and architectural patterns can be reused for different purposes in the future.

Case Study 2:

In the second case study the agile software development process: XP is applied in a similar startup company (Krizp system) in India. Krizp Solutions is a web based development company found in India. Their business plan is to create web portals for other small companies or educational institutions. The company began a year ago with administrators who held employment at other major IT firms. The company therefore began as a part-time business with plans of launching into a separate concern of its own if it becomes successful. Currently there are a few more employees/developers in place. Being a startup and with not many projects in hand the company also does not have a relevant software development process in place. There are a few similarities found here to Tesenso. The author tries to introduce agile XP software development process in this company and analyses the results observed. The results are observed while the company's developers simultaneously develop a web portal for an educational institution. The user of this web portal will be the management of the educational institution.

The results for the study are obtained while following several scenarios defined for extreme programming. The results of the software development processes in this startup are observed by conducting interviews with the company personnel. Therefore, data collection was in the form of interviews. At the first interview it was verified that the company is not following any particular software development process. Few questions asked were:

"Does the company follow a structured approach to software development?"
"How are requirements gathered?"
"How are changes identified?"
"Is there positive team collaboration/communication within the organization in view of developing software?"
"How is communication established with the customer?"
"Is there any form of documentation from previous projects available of the company?"

To achieve the goal of the thesis it is first necessary for the management of the company to become familiar with what software processes is all about and the benefits of using software

processes. This will be done by giving a presentation at the beginning of the interview. By the end of the second meeting, the interviewee's perspective of the process will be observed and further doubts are clarified. The main part of the second meeting is to go through every detail or concept areas within the software development process. In this case study the author is a passive observer of results and will not be directly involved in any implementation of web solution. During the case study the author observes the efficiency of the introduced process in the organization in terms of employee satisfaction and employee-employer collaboration. The quality of the product was not taken highly into consideration here since the author did not have an active development role in the organization. However, by interviewing the candidates it was observed that the introduced process also promoted the quality of the developed product.

3. State-of-the-Art

The following chapters explain the two different processes adopted to perform the case studies in detail. A short description about entrepreneurial organizations is also provided in this section of the report. These descriptions are based on the available theoretical sources.

3.1 Sequential Software Development: The Waterfall Model

The waterfall model is a traditional software development process that is commonly used in most software development projects. It is a sequential model; therefore the completion of one set of activities leads to the commencement of the next activity. It is termed waterfall since the process flows "systematically from one stage to another in a downward fashion [4]". It forms a framework for software development. Several variants of the model exist, each using different labels for each phase. In general, however, the model is considered to have six distinct phases as shown in Figure 2 namely: Requirement analysis, design, implementation, verification, installation and maintenance [5].

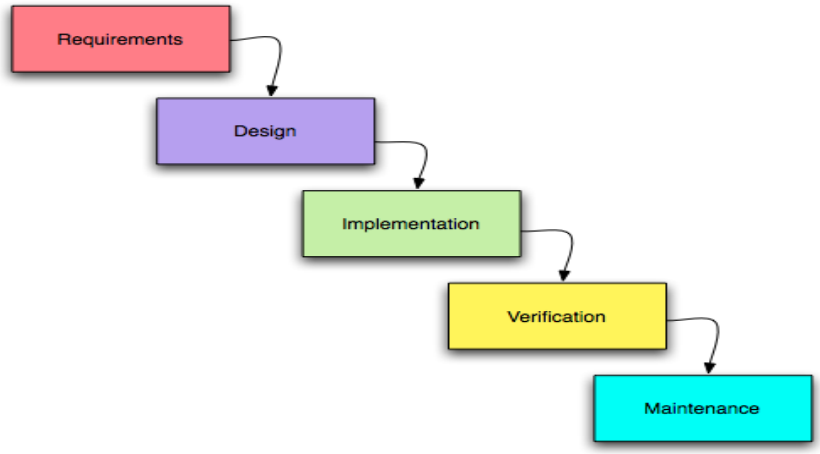

Figure 2: Waterfall Model [4]

3.1.1. Requirement Analysis

This is the first and most important step of the waterfall model. This involves gathering information regarding the end solution from the customer and understanding customer needs. It involves clear definition of the customer goals, expectations towards the project and the problem the end product is expected to solve. "Analysis includes understanding the customer's business context and constraints, the functions the product must perform, the performance levels it must adhere to and the external systems it must be compatible with"[5]. Requirement elicitation is the process of gathering information from the stakeholders of the system. Some of the techniques used for elicitation are customer interviews, rapid prototyping, use cases and brainstorming. Use cases are generally the functional requirements of the system. The results obtained from analysis are captured into the software requirement specification (SRS) of the system. The SRS describes the behavior of the system entirely. This serves as input for the next stage of the process model [5, 6].

“Captured requirements must be quantifiable, relevant and detailed [6]”. It should be defined in detail such that it is sufficient for system design. Requirements can either be non-functional, functional, design or behavioral requirements. Requirements analysis forms a very integral part of project management [6].

3.1.2. Design

This step commences using the information captured in the SRS. This can be considered as giving a solution to the problems in scope using the available resource. This stage comprises of how the software will be built, in other words planning of the software solution. The stakeholders involved in this module are the designers of the system. The software design may include the system design and the component design [7]. “The design phase involves defining the hardware and software architecture, specifying performance and security parameters, designing data storage containers and constraints, choosing the IDE and programming language, and indicating strategies to deal with issues such as exception handling, resource management and interface connectivity [5]”. This is where the user design of the system is also established depending on the ease of access and navigation. The output of this stage is one or more software design description (SDD) which serves as input to the next phase [5].

3.1.3. Implementation

The input of this phase is the SDD of the system. “This is where the actual development of the system takes place as per the design specification. This step is carried out by the developers, interface designers and other stakeholders using tools such as compilers, debuggers, interpreters and media editors. The output of this step is one or more product components which are built based on the pre-defined coding standard and debugged, tested and integrated to satisfy the system architecture requirement” [5].

3.1.4. Testing: Verification and Validation

In this phase both the individual components and integrated solution are verified to see it is bug free and meets the software requirement specification. The tester is the stakeholder involved in this phase of the model. Test cases are written to evaluate whether the system fully or partially meets the requirement of the system. Testing can be categorized into unit testing (performed on specific modules of code), system testing (to see how the system reacts when all the modules are integrated) and acceptance testing (performed with or behalf of the customer to see if all the customer needs are satisfied). Defects found at this stage are provided as feedback to the developers who in turn fix the issues. This is the stage where the developed product is documented. The user manual is published at this phase [5].

3.1.5. Installation

This phase takes place after the end-product has been tested and approved by the customer. “This step involves preparing the system or product for installation and use at the customer site”. Delivery of the product is done through the internet or through physical methods. A revision number is normally tagged alongside the deliverable to facilitate updates or changes at a later stage [5].

3.1.6. Maintenance

This is the final stage of the waterfall model and occurs after the installation of the product/system at the customer site. "This stage involves making modifications to the system or individual components to alter attributes or improve the system's performance". The modifications arise because of change requests triggered by the customer or defects that were discovered while using the system in real time. The revision number is updated during every maintenance release [5].

3.2. Agile Software Development

3.2.1. Overview of Agile methods

Unlike the waterfall model, agile software development is iterative or incremental. In agile software development (ASD) requirements and solutions are said to evolve throughout the course of the project.

The group behind ASD suggested building a first prototype which is to be like a demonstration prototype. This "prototype" is examined and discussed with the customer. By doing so, changes to the first prototype are identified leading to the development of a second prototype and so on. The changes are made until the customer is satisfied with the product. These changes are regarded as the driving force for the team giving them more insight on providing quality products for the customer. Therefore instead of trying to follow a predictable process to obtain good quality software at the end, ASD focus on maintaining a high quality prototype throughout the entire software development cycle [8].

In February 2001, a group of seventeen people met to talk about agile software development and thus formed the Agile Alliance. They agreed on the agile manifesto and several principles to be followed for agile software development. These set of principles were given by Fowler and Highsmith on August 2001 [9].

3.2.2. The Agile Manifesto

The manifesto of agile software development according to Agile Alliance is as following [9, 10]:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

1. Individuals and Interactions over processes and tools.
2. Working software over comprehensive documentation.
3. Customer Collaboration over contract negotiation.
4. 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."

3.2.3. The Agile Principles

The agile software development principles according to the agile alliance are as follows [9, 10]:

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 customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to a shorter time scale.
4. Business people and developers must work together daily throughout the project.
5. Build project 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 design emerge from self-organizing teams.
12. At regular intervals, the teams reflect on how to become more effective, then tunes and adjusts its behavior accordingly.

3.2.4. The Agile Approach: Extreme Programming (XP)

There exist several agile methods of software development, some of which are Scrum, Extreme Programming, Agile software development, Rational Unified process (RUP), Lean development, Crystal methods and Pragmatic Programming. But the most popular among all these is extreme programming (XP).

Coding is considered to be the primary aspect of extreme programming. XP was introduced by Kent Beck and Ward Cunningham. Together they realized the potential of man within loop pair programming. "XP describes an approach to development that combines practices used by many successful developers that got buried under the massive literature on software methods and processes [11]". The following are the approaches in XP:

3.2.4.1. Process

The lifecycle of XP includes five phases namely [12]: Exploration, Planning, Iterations to release, Productionizing, Maintenance and Death. Beck describes these phases as below [12] and Figure 3 illustrates it.

In the **Exploration phase** write the features they wish to have in the first release into user story cards. There can be several user stories each describing multiple features of the system. By doing this, the project team gets familiarized with tools, practices and technology used in the project. The development team simultaneously explores the architectural possibilities of the system by building the first prototype of the system. This phase takes roughly few weeks to few months depending on how well the team is familiar with the system [12].

The **Planning phase** prioritizes user stories and an agreement of contents that goes in the first release is made. The programmers estimate the effort for each user story and the schedule for progress of work is established. The time span for the schedule of first release does not exceed two months. The planning phase is completed in couple of days [12].

The **Iterations to release phase** includes several minor iterations to the system before the first release of the system. The schedule made in the planning phase is broken into several iterations each taking around one to four weeks to implement. The architecture of the whole system is developed in the first iteration. Relevant user stories are prioritized and chosen for the respective task. The stories to be implemented per iteration are chosen by the customer

based on his preferences. At the end of the last iteration the system is ready for production. Functional tests developed by customers are executed at the end of iteration, thereby making XP better known as test-driven development [12].

The **Productionizing phase** involves checking the system's performance and additional testing of the system, before the system can be released to the customer. Changes in the system can be even detected at this phase and a decision has to be made if these features will be included in the current release. The postponed ideas will be later implemented in the next phase and will be kept documented until then. Iterations in this phase will take one to three weeks.

After the product is delivered to the customer, it is vital to keep the system running and also to look for new iterations. This maintenance of the system is done in the **Maintenance phase**. New iterations are identified from customer support tasks. This may result in the development velocity to decelerate, thereby arising need for new development teams and team structure.

The **Death phase** begins when the customer has no longer any stories to be implemented. This implies that the developed system satisfies customer need and other aspects such as performance and reliability. As this phase indicated closure of the project lifecycle, necessary documentation relevant to the system is made. Death may also occur if further development to the system becomes too expensive.

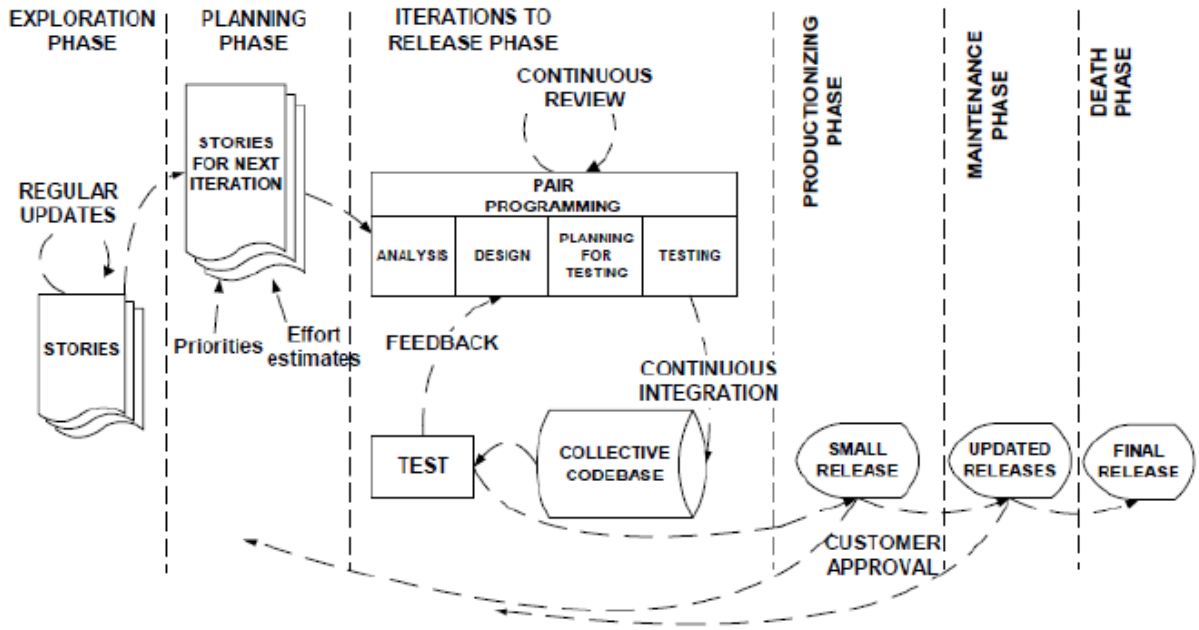

Figure 3: The Life cycle of a product according to XP process [12]

3.2.4.2. Roles and Responsibilities

According to Beck (1999b), "Extreme programming explained: Embrace change", XP comprises of different roles for different tasks and practices. These roles are presented in the table 1 below [8, 11, and 12]:

Role Name	Description
Programmer	Takes the responsibility of writing tests and code. Maintains the program code as simple and definite as possible. Programmers within the team need to coordinate and maintain good communication among team members.
Customer	The customer takes the responsibility of writing user stories and functional tests. A decision on a satisfied requirement and prioritization of requirements to be implemented is done by customer.
Tester	Responsible for helping the customer write functional tests and also maintains the testing tool.
Tracker	Responsible for giving feedback in XP. The tracker records the effort estimates of the team and provides feedback on improvement. He also traces the effort in every iteration, keeps check if the goal is attainable within the available resources and time and identifies changes needed to be made in the process.
Coach	Person in charge of the entire process. This role calls for deep understanding of XP thereby providing knowledge for the other members in the team.
Consultant	Member external to the team who possesses sound technical knowledge. Renders support to members in the team whenever needed.
Manager	Responsible for making key decisions. He communicates with the team to resolve any issues and solve difficulties or deficiencies in the process.

Table 1: XP's roles and responsibilities [12, 28]

3.2.4.3. Practices

XP is a culmination of ideas and practices obtained from existing methods and techniques. For example, XP's idea on usage of user stories was obtained from the practice of drawing use cases proposed by Jacobson. "Some of the major characteristics exclusive to XP include small releases, rapid feedback, customer participation, limited documentation and pair programming [12]". The following is a list of practices followed in XP. This list of practices is also explained in Extreme Software Engineering: A hands on approach [8] and by Kent Beck (1999a), "Embracing Change with Extreme Programming, IEEE Computers 32(10), 70-77 [8, 11, 12].

Planning Game:
"This requires close interaction between the customer and programmers. The programmer makes total estimate of effort needed to implement the customer user stories and the customers makes decisions on the scope and timing of releases". [12]. Pair programmers can choose the module of code they prefer to work on during the given day.

Small/Short Releases:
A prototype or a simple system is created frequently, at least once in two or three months. There is extensive possibility for new versions of the product to be released daily, if not monthly.

Metaphor:

"The system is defined by a metaphor/set of metaphors between the customer and the programmer. This "shared story" guides all development by describing how the system works [12]".

Simple Design:
The developers aim to keep the solution as simple as possible so that it provides momentary implementation. Extra code and unwanted complexity is avoided.

Testing:
XP is a test driven development strategy. Unit tests are executed before the original code and are run continuously. All tests are written in a test framework such as JUnit and hence become fully automated. Functional tests are written by the customer with the help of the tester.

Refactoring:
This is a practice in XP which involves restructuring the system by eliminating duplicated functionalities, improving methods of communication and enhancing simplicity and flexibility.

Pair Programming:
The code should be written by two people in one computer. This practice of XP helps spreading information amongst team members, skill sharing which directly promotes the quality of code written.

Collective Ownership:
All team members are given the authority to change any part of the code at any time. Thus it is necessary that no individual takes pride on the quality of the code that he/she wrote, but instead takes credit for the entire program.

Continuous Integration:
Integration of a new piece of code to the existing code takes place as soon as it is available. Integration of new code can occur several times in a given day. Automated tests are executed in order to patch/accept the new piece of code to the existing prototype.

40-hour Week:
The team should maintain a sustainable pace of development. A maximum of 40 hours per week is allowed. This is to ensure the team does not get drained of their energy reserves. Overtime for two consecutive weeks is treated as a threat or a problem that needs to be solved immediately.

On-Site Customer:
It is required for the customer to be present full-time for the team. This allows the team to clarify doubts sooner and improves communication regarding features and requirements of the system.

Coding Standards:
Programmers should emphasize communication through the code developed. In order to bring this into practice, the programmers should adhere to certain rules, norms or standards while developing code. This coding style should be exhibited in all parts of the system so that other members of the team are able to interpret what has been written.

Open Workspace:

The development team should be provided with a large room with small cubicles. Pair programmers are placed at the center of the room to provide flexible communication within the pair and across the team.

Just Rules:
Team members and pair programmers can collaborate with self defined rules. These rules should promote healthy work relationship and accelerate understanding within the team. The impact of these rules however needs to be assessed from time to time.

Customer tests (Acceptance tests):
The customer or the person representing a customer is in charge of writing functional tests. These tests verify whether the system has met the customer needs and satisfaction. If the customer is not available on-site, a person within the team can function as a customer.

3.2.4.4. Scope of Use

XP development process is aimed for usage between small and medium sized teams which are limited from three to twenty project members. This development process produces high quality code due to constant refactoring. However, the applicability of XP is restricted by a few factors. The physical environment for carrying out the process plays a primary role. Face-to-face communication is the prime motto of XP. This requires all the team members involved in the project to be present within the same room. Geographical distribution of the team on various floors will also be a hindrance. Secondly, any technology that does not support "graceful change" is not applicable for XP. This behavior can be observed in safety critical systems which require longer feedback durations [8, 12].

3.3. Organizational Structures

"An organizational structure can be considered as a viewing glass or perspective through which individuals see their organization and its environment. An organizational structure plays two predominant roles in an organization's actions. First, it provides the foundation on which standard operating procedures and routines rest. Second, it determines which individual get to participate in which decision making process, and thus to what extent their views shape the organization's actions [19]". It is basically represented as a series of vertical and horizontal lines. The vertical shows whom a particular position reports to and the horizontal shows the employees on the same level in the hierarchy [18]. There are several organizational structure types, one amongst which is the pre-bureaucratic (simple/entrepreneurial) structure [18].

3.3.1. Entrepreneurial Organization

According to Mintzberg [20, 29], a simple organization has a flat structure where in several employees report to few top managers. Such an organization is unstructured and highly centralized by having strong leaders who are usually their owners. This kind of organizational structure is flexible and can only be found in small or new organizations. Therefore the top managers have greater influence on the company's growth and bad decision making can possibly result in the company's downfall. The top management is also highly dependent on the individuals/ employees to achieve success [28].

4. Previous Research

4.1. Literature Study

There are a few articles and papers that describe the experience of applying software development process (Agile XP and waterfall model) in small startup projects. The author analyzes a few articles that discuss the experiences of applying waterfall and XP processes in startup projects or in any project in general. The results from these articles are closely observed and the reasoning behind the present research question is formulated.

One research focused on a controlled empirical study on the waterfall model and extreme programming conducted in Carnegie Mellon University by a team of experts [21]. The factors such as method (process used), programming language, project and number of teams were taken into account. Few other auxiliary factors that contributed to the research were design documents from previous years and number of lines of code written. By observing their results it shows that the waterfall and XP eventually produces almost similar quality software, but agile was preferred due to the limited time spent in documentation. The study involved data/statistics of software development collected over a period of five years for the same project [21]. However, a similar kind of research cannot be done in a startup for reasons such as fewer projects, lesser duration and limited human resource. The papers did not give much information on the organizational structure and the type of working culture adapted to perform this research. Therefore this research and results cannot be readily applied to a startup organization with unreformed goals and business plan. It is unclear whether the same process applied in an entrepreneurial setup would produce the same results.

While the previous study focused on the project in specific, there are other studies that focused on startup projects within a well established IT company. Striebeck talks about introducing a startup culture for a startup project within a well structured firm, Google [22]. Here he introduces one agile practice at a time to observe the results. Although it concerned a startup project the study conducted at Google cannot be compared to introducing a startup company to a structured software process since Google is not new to the concept of software development process and does not have a problem with introducing a new process [22]. While performing the study Striebeck retained the role of tech leads and UI designers throughout the project. These roles however adhered to a few agile techniques [22]. The results showed that the Google culture of allotting different people to carry out different tasks was not affected by introducing agile practices. The reason why the same results may not hold for an entrepreneurial company is that, here again, these companies do not have a well established structure and have fewer employees to carry out multiple tasks if one other fails.

4.2. Comparison (Pros and Cons) between the two models

Waterfall model:
The waterfall model has several advantages. The step wise development strategy forces discipline since each stage has a starting point and an end time simplifying identification of development progress throughout the software life cycle. Emphasizing requirements and design before coding reduces the risk of customers need not being met. Moreover, having a clear idea of requirements and design before implementation brings into the spotlight the notable flaws that might occur at the later stages. Identification of flaws in development after integrating the entire software is difficult and adopting waterfall helps reduce this risk [5].

In recent times, however, the waterfall model has faced a significant number of criticisms. The most relevant being, "customers do not have a clear picture of what they really need right from the start and requirements keep emerging throughout the course of the project". In this case, the waterfall model is not designed to adopt change in between the project lifecycle. Change needs to be implemented right from the top hierarchy downwards, thereby altering the entire project from start. The model therefore is applicable only in stable projects where customer needs do not vary a lot [5]. In such a case, the requirement and analysis part of the model should be greatly focused on when thinking in terms of a startup since most developers who work in startups do not have direct contact with customers. This hinders their requirement engineering activity. Developers do not really know what the customers really need and this gets complicated further when the startup does not have a customer at present and looking forward to win clients based on the product that is going to be developed. Most startup companies are likely to face this problem while implementing the waterfall model.

Secondly, the presumption that design made on paper can be effectively turned into real products arose some criticism. The real product implementation can be costly in response to the design made. The waterfall model does not address such risks in projects. Especially in a startup this poses greater risk as startups do not have much funding to begin with. So it is highly necessary for the design to be perfected right from the start. However in exceptional cases, time and cost may not be serious barriers in a startup as with other major IT firms if the startup management has employees working for free. Here, the lack of a proper design before development does not pose a serious threat. Re-design can take place at any point of time as employers do not really need to worry about funding for projects.

Critics also claim the waterfall requires division of labor such as: designers, programmers, testers. In reality such a division does not prevail in most software companies [5]. This aspect of criticism on division of labor holds true in a startup. The company dealt with in case study 1 has three developers in place working on different solutions for the company. It is therefore practically not possible to have separate roles to carry out the separate scenarios of the waterfall model in such an environment. The discussion on this will be carried out in detail in the case study chapter.

Extreme Programming:
Humphrey enumerates the advantages of using XP [14, 13]. Here he explains XP emphasis on customer involvement. This promotes customer-vendor relationship and improves clearer drafting of requirements from time to time. Priority is also given towards teamwork and communication improving employee relationships. In the practice of XP the programmers estimates the time for completion before committing to the task. This is a major plus to project planning and management. The design of the project module is kept as simple as possible so further additions can be made over the skeletal design, enabling regular refactoring and re-designs. Continuous reviews help keep the team's performance in check.

Like any other process XP also has several drawbacks. Some of its drawbacks are projected by Humphrey as follows [14, 13]:
a) XP exhibits a code-centered development instead of design-centered. This is safe in small projects but in big projects with larger teams it can be disastrous.
b) Lack of design documents reduces reuse opportunities.
c) The use of source code to document large projects is unrealistic and impractical since such documentation may involve thousands of pages.

d) XP does not support a structured review process. On screen reviewing of programs will allow engineers to locate 10-25% of errors. The same applies for pair programming. However, with structured reviews 60-80% of errors can be discovered.
e) XP does not maintain a quality plan. Having such a plan in place is said to reduce the time allotted for testing a product. XP spends enormous amount of time on testing.
f) XP does not have any data-gathering technique.
g) XP methods are not described elaborately. While some programmers will effectively implement all process details, most of the engineers will not reciprocate this behavior. Thus practitioners adopt only the methods they like and ignore the rest.
h) Though XP introduces several methods which have an impact on management, it does not provide relevant management training for successful adoption of these methods to be practiced.
i) XP does not address any of the CMM (Capability Maturity Model) perspectives. Hence, it plays very limited role in the management and organizational issues.

Additional to the aforementioned drawbacks, XP can also face several drawbacks when implemented in a startup.
a) If the startup does not have a customer to begin with then XP faces a major flaw during implementation, since extreme programming centers around the idea of customer involvement. Change in requirements in such a situation cannot be identified during the course of software development and this disturbs the ultimate motto of XP which is "change".
b) In a startup with a single software developer and few managers, the concept of pair programming cannot be practically implemented and discussion on time estimation cannot be done by a single developer.
c) The time allotted for testing increases due to the lack of many peers with relevant knowledge to review the code with.

4.3. What is missing in theory?

As seen in section 4.1.there are several papers and empirical case studies on what type of processes (whether waterfall or XP) will be suitable to be followed with small sized and large projects. However, there is limited research on the type of process suitable to be implemented within a firm taking into account the organizational factors/aspects. One prevailing criticism is that agile does not support the CMM (Capability Maturity Model) perspective of an organization. While there are startup projects existing within larger firms, which adapt sound software development cycles and case studies to implement a process within such project in such an organization, the author's opinion is that the current state of research lacks a source of reference on the steps to be followed while trying to adopt a software process into a startup which has no prominent business development strategies in place. The companies in this case study are new and trying to find a place in the market with the product that they are planning to develop and sell in the near future. These companies have fewer employees and limited access to resources like money, licensed software, etc, in comparison to major IT companies. Therefore, this case study on processes to be implemented within an entrepreneurial environment should provide some insight to the practitioners about a suitable software development process to follow and with a simpler way to adapt these processes into the organization so that it promotes their business goals and provide them with a structured approach to software development if it is applicable to do so.

4.4. Assumed prerequisites in software development lifecycle

The software development process undertaken to carry out the project plays a major role for the project to be completed successfully. It also contributes in the growth of the organization in terms of employee satisfaction and work culture. Almost all organizations that develop software have a software development process in place based on their managerial strategies, organizational culture and organizational structure. Below are a few factors which are assumed by the author to greatly contribute to organizational growth and performance. These factors are obtained from thorough literature research and will be used as basis of arguments in the results obtained by the case studies.

Factor 1: Knowledge Sharing

Knowledge is an indestructible asset of an organization. The more an organization's knowledge is expanded the greater are its chances of competition in the market [23].Knowledge sharing is the exchange of information among individuals and this "…is greatly influenced by conflict in interests, interaction among actors and the organization the actor resides in [23]." Present day software organization realizes the importance of introducing knowledge management strategies within projects. Irrespective of the organizational structure, transfer of information from amongst members of the team and from one project to another contributes to a major part in software development. Teams change more often than seldom due to reformulation of organizational plans. This results in the loss of information when older teams are dispersed. Organizations should implement a strategy where knowledge is always gathered. Knowledge can be classified as ones gathered from technical data and the other gathered out of experience. Knowledge gained from experience is difficult to be stored and hence it is of primordial importance the organization finds a way to collect this information and transfer it to another. Such knowledge based on experience once lost is lost forever since developers will continue making the same mistakes again during the software development process without any revision or guidance from senior or previous employees. According to Feyman [24] not only positive outcomes or success is knowledge. In order to avoid mistakes in the future or to see how a solution is formed one has to tell the failures and dead ends too. Knowledge is when we implement the lessons learnt at the appropriate time [24].Considering these aspects it is therefore important for any organization to adopt a good knowledge sharing technique (be it a knowledge database or a software development process that might contribute to knowledge sharing to some extent) which would later help in the development of good quality software

Factor 2: Communication within teams

Exchange of ideas with peers is a contributing factor to software development cycles. Doing so promotes the growth of the organization through better performing teams and eventually producing quality software. This determines the organizational behavior and it is greatly important when thought in lines of a startup organization. Startups generally have the advantage of having smaller teams with higher probability of communication. But what happens if the startup does not have a process or method in place to channelize this communication for the betterment of software development? It then becomes of high importance to introduce a software development process which contributes to this factor of an organization. There are several benefits one can think off when good communication of team members is established some of which are; strengthening of employee relationships, increase in employee performance, enhancing employee satisfaction etc. According to Cockburn [25],

"the core to agile software development is the use of light but sufficient rules of project behavior and the use of human-and-communication oriented rules". Considering this, good communication among team members was always emphasized in agile development processes. XP and Scrum development processes where greatest contributors to achieve communication within teams. In the case study we also analyze if the waterfall model can contribute to human-and-communication rules and which among the two processes is a greater contributor to this factor of the organization.

Factor 3: Freedom of thought

Another important aspect of software development is to encourage creativity throughout the software development lifecycle. The quality of the software created greatly depends on the extent of creativity and human ideas invested in the product. One of agile principles is "…Build project around motivated individuals. Give them the environment and support their needs, and trust them to get the job done [9, 10]."When in an organization other than a startup, employees have the privilege of not working directly under the supervision of the management team. In such a scenario freedom of thought is encouraged amongst members of the development team. Employees have the advantage of developing products based on creative styles and concepts. Changes in requirements are made based on what the customer wants and not what the management wants. In a startup however employees mostly work under the direct supervision of the management who in most cases will be owners of the firm. Here they slavishly follow whatever is given by the upper management without thinking of any external ideas or possibilities of arriving at a solution. Employees do not provide new ideas even out of fear of the upper management. Employees hesitate because they think their idea may not be good enough or it may be offensive to the management if they supply new ideas better than those of the management's. Employers trust their idea is better than the employee's idea. This poses a great threat to creativity and improvising existing solutions. In most cases the management is not aware of the consequences of not encouraging creativity. Therefore there is also a need to introduce a process that supports change and creativity within a startup organization and allows building of projects around motivated individuals. This change should be brought about smoothly within a startup without causing a threat to the upper management of the firm. In both the case studies we analyze which software development process is encouraged for use by both employee and employer, which process encourages flexibility for employees and management and which process produces more motivated employees.

5. Results

5.1. Case Study 1: Waterfall model in a startup

The case study at Tesenso AB began with implementing the stages of the waterfall model in the company. These stages include requirements, design, implementation and testing. The results are observed during the course of study. The waterfall model is assessed by a single software developer in this startup and therefore will be applicable in view of a single developer working to develop a software solution for the company. These stages are engineered by the author to suit single developer projects in a startup and can be used when the developer does not have communication with the intended client.

REQUIRMENTS

Requirement is the initiator activity of the waterfall model and the most important scenario whether it in a startup or any other organization and is usually done in collaboration with the customer. The customer and the development team involve in the requirement elicitation process to determine the general requirements of the system. In this case study there were clients associated with the project, although, the development team was not given the opportunity to interact with the clients to determine the requirements of the system. Therefore, requirements are gathered by face-to-face interaction with the management who in this case is the acting client. This activity took place during the first week of project startup. These requirements were captured in the Software Requirement Specification (SRS) document and are presented in Appendix B of the report. Tesenso experienced a problem of having a change in management over the year. The change did not happen during the study; however there are possibilities of change in management in the future. Therefore it is probable for the requirements to change every time the management changes. The requirements defined by the previous management where found to be different from the requirements defined by the present management. Considering that the management's choice of requirements may be biased and of lesser scope for the present solution there was a need to address the usability criteria of the solution with actual users of the system. Since it was not possible to the test the solution with actual users of the system there was a need to address to this problem faced by the company and hence we assessed the usability of the solution by testing with test persons who can stand model for the actual users. Usability testing was carried out with five different test users and their view of the product was taken as additional requirements while developing the solution.

Developing a user interface for a startup faces several challenges. One such challenge is that a single person is responsible for doing multiple tasks, or rather represents multiple roles. Therefore the time allotted to perform each task is fairly low [17]. User Centered Design is used to address these challenges to get a better response on the product during development and capture changes in requirements. Therefore one aspect of usability testing is performed with customers before and during the development of the product to obtain a user-centered design (UCD). UCD is the termed used in recent times for human factors engineering, ergonomics and usability testing [15]. Woodson defined human factor engineering as "the practice of designing products so that users can perform required use, operation, service and supporting task with minimum of stress and maximum of efficiency" [16, 15]. There are three important principles of a user-centered design which are [15]:

- An early focus on users and tasks

- Empirical measurement of product usage.
- Iterative design whereby a product is designed, modified, and tested repeatedly.

Here the first principle of UCD is predominantly used in the requirement stage of the waterfall model.

An early focus on users and tasks:

The goal here is to have direct contact between the user and the development team during the course of software development. By doing this changes in requirements are captured then and there and are reflected in the implementation of the product. To do this we need a "systematic and structured approach to collect information from and about the users".

Rubin [15] describes a variety of techniques and methods available to perform UCD at different stages of software development. Some of which include surveys, questionnaires, usability audits, field studies, design or structured walk-through, paper-pencil evaluations, etc. Paper prototype is the technique adopted to perform usability testing in this study.

Paper Prototypes:

In this technique, "the user is shown an aspect of a product on paper and asked questions about it, or asked to respond in other ways". These questions or scenarios are made to allow the user find certain kind of information or options on paper [15]. The user's reaction to the prototype is observed and suggestions on change are recorded.

There are four types of tests involved in usability testing which are exploratory tests, assessment tests, validation tests and comparison tests. Exploratory tests are those which are conducted early in the software development cycle when the project is in its preliminary stages. The purpose of an exploratory test is to explore the efficiency and ease of use of the earliest designs of the product made. It requires extensive interaction between the participant and test prototype (which in our case is static representation of the screen on paper) for creating the preliminary design concepts. Exploratory tests give an idea only on the high level design concepts [15]. Below are a few examples of paper prototypes used during the project.

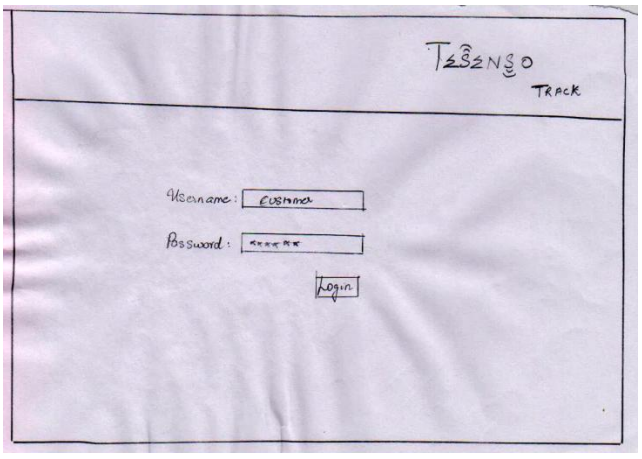

Figure 4: Paper prototype for login page

To perform usability testing the user was given a paper prototype as shown in figure 4 and observations were made on how the user interacted with the page. A few example scenarios listed to the test users were:

- Perform login operation

- Interaction with home page

The users belonged to age group 25-27 and were as follows:
User one: A student with different engineering background but knowledge in computer science.
User two: A student with background in computer science.
User three: A student from a different engineering background.
User four: A student from a different engineering background.
User five: A student from a different engineering background

Scenario 1: Perform login operation

Here the author asked the users to perform the login operation by presenting the paper prototype shown in figure 4. Since the initial step was to perform only exploratory test, most of the underlying operations were hidden from the user. The author notes the actions of the user on the prototype and records additional suggestions.

User one: When the first user interacted with the prototype he suggested the use of the "forgot password" link to be included in the design. This option was missing in the initial design of the system. Otherwise he did not find it difficult to understand the first page of the web portal.

User two: When interacting with the prototype, user two suggested the inclusion of some company related text in the login page of the web portal. This was to provide any user some brief description of what the company was all about. Therefore his suggestion was to make the page a bit more informative and attractive than to simply have a username and password option.

User three, four and five: These users did not have much to say instead had almost the same suggestions as the previous users. Some suggestions made by them were part of assessment tests which will be explained further in the design chapter of the study.

Scenario 2: Interaction with home page

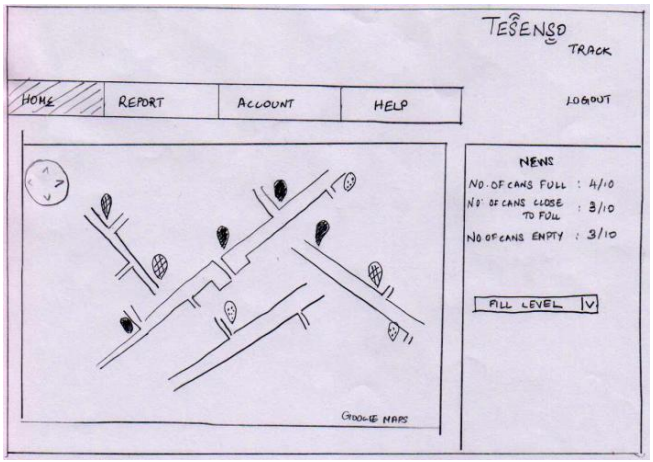

Figure 5: Paper prototype for home page

The five different users were made to interact with the mockup prototype for the home page in figure 5 as done with Scenario 1. Their suggestion were recorded and later used during the

implementation phase of the project. Similar paper prototypes were created for the admin page as well and user needs were identified.
In addition to paper prototypes requirements were also captured from the previously developed solution for Tesenso clients. The details missing in the previous implementation were taken into consideration during the current implementation of the project. The changes in the requirements between the old and present solution are elaborated in Appendix A.

Discussion:

Most startups normally face this difficulty of not having any clients to begin with to be associated to their projects. This is because the company is new to the business of software development and will be trying to venture into the market with the product being developed. In few other cases, the company may have clients but the development team may not be allowed to interact with them due to biased opinions of the management. This becomes a major flaw while considering working with the waterfall model since customers/clients are the key resources to initiate the first stage of the software development process. In such cases, following the regular trend of writing an SRS document at the requirement stage of the waterfall may not be possible at all times as the SRS is generated to establish a good understanding between the customer and developer. One might consider writing an SRS document in collaboration with the management. If that is the case the management may not have enough knowledge of what the customer really wants. It is not possible to imagine an appropriate solution for a customer taking into account only the management's view of the system. The worst case is the customer and the management may have two entirely different solutions in mind. In such cases where the customer is absent startups might consider adopting the technique of paper prototyping which is found to be the simplest and easiest method to capture requirements from random users. Moreover, producing an SRS document which contains use cases and use case description to a random user who does have any background in the technical jargons of software development is meaningless. The best way to approach an anonymous user is to provide him with data which is easily understandable and graspable to a layman. It is therefore the author's opinion that the technique of generating a user centered design using paper prototype is a suitable form of communication between a user and a developer.

The SRS document might have been put to use by the developer during the design stages. In this study since the design was also associated with paper prototypes, the use of SRS was not relevant. It was vaguely used to capture requirements in natural language from the management and therefore not revisited very often. Requirements gathered from management were given higher priority because the solution demanded the implementation of these requirements. Communication was well established with test users in spite of not using an SRS to gather requirements. In addition to this, it was the author's opinion that writing a detailed SRS document for a solo project was time consuming and may not have served its purpose during the lifecycle of this software development. The management or the test users would not have understood or be interested in knowing the technical background of the solution nor were there co-workers to share the SRS document with.

One of the greatest strength of the waterfall model in terms of reuse was documentation. Hence excluding an SRS document totally during the requirement stage of the waterfall has its own drawbacks. It minimizes the extent of knowledge sharing between individuals working in a project. This is something that was irrelevant in this context of study since the project was carried out by a solo developer. Considering other startups with two or three

developers working on a project and associated with a client, writing an SRS document might be useful as developers can have a common mode of understandable communication. Also, having an unclear SRS reduces future reuse by new employees of the firm. Future developers of the project will be expected to learn the system right from scratch to begin with further iterations to the web portal.

Paper prototypes did not provide much opportunity for reuse either as the future developers who might work with the system may not get an idea of the technical details behind the solution only by looking at these prototypes. Though usability testing was useful in various aspects, a successful method to accomplish reuse of requirements was not established during this stage of study.

Finally gathering requirements from a previously developed system was a tedious task to do, especially when requirements are gathered only from previously available code. The development strategy used by the former developer was complicated to be understood at first. It was unclear initially to know the software requirements to run the project. Several meetings were scheduled with the previous developer which however was unsuccessful. Having an undocumented solution can be a drawback in any software project.

DESIGN

This was the second activity conducted during the study. Normally the idea was to write the SDD from the SRS captured in the requirements stage. Since there was no definite SRS in this study, the design of the web solution was done using assessment tests of user centered design. The second principle of user centered design emphasizes as follows [15]:

Empirical measurement of product usage
Here we emphasize on behavioral measurements that give attention to the simplicity of use and learning, at the early stages of the design process. This is done by "developing and testing the prototypes with actual users" [15].

One complete week has been spent on developing the initial design of the product using exploratory tests. Capturing requirements using assessment tests began during the second week of project startup. Assessment tests were performed using paper prototypes. These tests were carried out midway through the project and gave the user an idea on the lower level designs. "If the intent of exploratory test is to work on the skeleton of the product, the assessment test begins to work on the meat and the flesh [15]".

Example Scenario 1: Perform login operation

The paper prototype shown in Figure 4 was once again revisited and presented to the five users to capture design requirement. The users were asked to perform the login operation successfully.

User one: When the user began to input data in the username text box, his immediate question was if the content in it was case sensitive. His suggestion was to keep this text box case insensitive. In the password text box, his suggestion was to enable the use of ASCII characters while setting the password.

User two: His suggestion was to make the captcha option as part of the login procedure. The captcha is a method used for authenticating a human as a user of the system. His suggestion was to keep it as a mandatory option for incorrect login. In case of a wrong entry of the

password three times, the user is asked to enter the captcha the fourth time for further attempts of login.

User three: This user also did not find difficulties during the login procedure. However, he suggested the use of a popup box stating “incorrect username/password” and to limit the number of login attempts to three if the login procedure failed.

User four: In case of incorrect login, this user’s suggestion was to stay on the same page by making the text box empty for further trails and without limiting the user to the number of attempts made for successful login.

User five: Did not make any notable suggestions.

Example Scenario 2: Customer account

After successful login, the users were given the task to proceed to the user account to view and perform options available in the page.

User one: Successfully clicked on the third tab to proceed to the customer account page. By doing so, he was able to view customer related information on the page. One notable suggestion made was to include the “change password” option for the customer account. This was to enable privacy of customer accounts. The password however can be viewed by the admin.

User two, three, four and five: These users did not have any particular comments. One suggested the inclusion of the logout option in this section of the portal. Some faced difficulty navigating to the customer account page which was later successful by the guidance of the author.

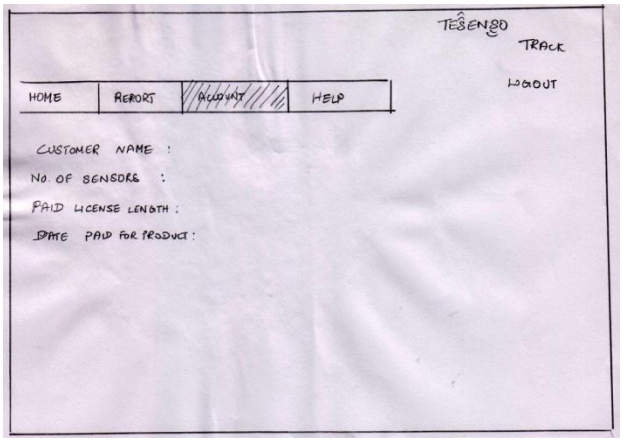

Figure 6: Paper prototype for customer account

Likewise, several other scenarios on various aspects of the web portal were presented as a task list to the users and the observations were taken into consideration during the development stages of the system. There is a slight variation to how assessment testsare carried out regularly than what is done in this study. Usually, the development of the system takes place simultaneously while capturing the design requirement through assessment tests. The user is made to perform operations on the prototype system than just commenting on screens made on paper. Since the study follows the waterfall model, the actual development of the prototype was saved for the implementation stage of the software development process.

The design suggestions were initially recorded in paper and later looked up during the implementation phase of the software development lifecycle. Changes in design observed after implementation were also considered to be part of assessment tests and as a design suggestion which was also reflected in the original implementation of the system.

Discussion:

If a developer were to follow the waterfall model design stage strictly, then it becomes mandatory to produce a relevant software design document. This document usually contains sequence diagrams, activity diagrams, other software and hardware specifications, etc and is primarily intended to give an overview of the solution that is going to be developed to the customer. Most of these however might have been considered to be technical jargons and will not be understood by an ordinary user. These documents are used to exchange technical understanding within team members and are usually revisited when there is a need to fix technical difficulties. In this study, since there were no other team members to review the SDD with, it became unnecessary to consider spending lot of time in writing this document. These diagrams which are part of the SDD were however created as a rough draft on paper to provide personal guidance for the developer providing her with an overview of the entire development plan to be adopted, but did not find its use elsewhere. The author considered providing a sample view of an activity diagram on paper to a test user who was disinterested in taking a closer look at it. The management was also only interested in the final product and did not want to know the technical details to arrive at the final solution. They were however interested in knowing the software needed to run the solution, which is not part of the design of the solution. Furthermore, they also had an overview of how the end solution will possibly look like since they had provided a skeleton view of the system initially in the form of a power point presentation. The initial paper prototypes were created based on this skeleton view of the system provided by the management.

Therefore including a detailed SDD was outside the scope of the project and this may be the case in most startups where the customer is not in communication with the developer. Even if the customer is in communication with the developer, presenting such a document to a customer without any knowledge in software development will not be worth the effort. An SDD can however be used in a startup where there are more than one developer. In such a case, developers can establish a stronger technical mode of communication to get the task done precisely. In addition, most startups have developers working in close collaboration with the management. Therefore there is a higher probability that the management provides the developer with an initial design plan of the system. Assessment tests can be used to make iterations over this initial design plan.

Hence, the technique of adopting assessment tests proved useful during the design stage. It was perceived as a step wise increment to the desired solution. The users were given a chance to modify the requirements of the prototype throughout the development lifecycle. The method was considered to be inexpensive, simple, effective, and not time consuming. Too much time was not wasted by the developer doing extensive documentation. This method can also be a solution to the criticism of the waterfall design stage, in turning design on paper to a real solution presented in section 4.2. Design on paper can be successfully made into real products without worrying about finding flaws at the end of the project since in assessment tests iterations are made then and there and those are reflected in the final solution.

IMPLEMENTATION

The third principle of user centered design is as follows [15]:

Iterative design whereby a product is designed, modified, and tested repeatedly:
Design iteration plays an important role in usability testing. Changes are not made late in the development cycle but rather during the development of the system. Therefore here we follow the process of design, testing, redesigning and retesting the prototype [15].

The third stage of the waterfall model began by implementing the requirements gathered using assessment tests in the design phase. Here the solution was developed incrementally and shown to the user to observe his reaction to the web portal. Changes observed were reflected in the implementation in the subsequent days. The development of the initial requirements captured roughly from the management through the SRS took place for nearly two weeks. The entire implementation phase of the preliminary design requirements lasted for four weeks. After development of the preliminary design requirements, the web portal was presented to the user once again and an assessment test was performed to capture more requirements. This time assessment tests were performed by allowing the user to interact with the web portal rather than interacting with the paper prototypes. Thereby, iterations to the primary design were made very often during the same week. There are however a few requirements that were provided by the management that are unmet. For instance multiple mapping of garbage cans. Currently only single entries of full, close to full and empty cans are retrieved by the web portal. The entire implementation phase of the web solution lasted for six weeks.

Some requirements captured during design phase were not implemented successfully due to technical difficulties and time constraints. For instance, the implementation of forgotten password option, route tracing and captcha inclusion was never done. This is due to the technical complexity that lies behind this implementation. Such requirements are kept as a long term goal for the web solution.

Comparison tests were also performed with the users during implementation to determine requirements of the system. Comparison tests are those which are conducted throughout the development cycle with the user to determine different interface styles or to measure the effectiveness of a single element [15]. For example, during the design and implementation stage comparison tests were performed with users to determine preferences over the usage of drop-down menu or radio buttons for selecting fill level of cans. The majority opinion was taken into consideration while developing the module.

Discussion:

The implementation stage began with the expectation of following the same implementation pattern of the previous developer at Tesenso. The existing web interface was developed using Java Springs and therefore the author was keen to develop the present solution using the same programming language thereby making it easier for future integrations. However there were some problems initiating development using java springs. The previous developer did not have any recorded information on the technical details that lies behind the web portal. In other words there was no knowledge sharing established from the previous development to initiate the present development. The management also did not have any information on technical details. Therefore the author did not get any guidance to resume implementation from where it was left in the previous solution. The management and the author tried to establish communication with the previous developer which was unsuccessful. Even when contacted the previous developer did not have much information to share regarding database issues and there was a vast difference in our knowledge considering our level of experience. Most

startups experience this problem of establishing good communication with ex-employees. This is because they usually have employees working unpaid and when they move on to new jobs these employees cease to show any interest in the startup they once worked for. The situation was the same here; therefore it took a lot of time for the management to retrieve the previous code from the previous developer. Even after getting the code, the previous developer showed minimum interest to get the project running.

These difficulties therefore forced the author to develop the system from scratch using a familiar but different programming language (enterprise java). The framework/architecture for the current solution is explained in Appendix C. In this case study it was not a problem since the management was more interested in the solution instead of the programming language being adopted. In some other cases where the startup company's management is strict about the language adopted, this might have been a problem. Therefore here we observe the importance of having some knowledge sharing strategy in a startup.

The method of adopting implementation simultaneously along with the design was very useful and simple. Errors and changes were made to small patches of code instead of doing it in the end when the entire project was completed. User centered design proved to be useful along the way and is suggested to be a suitable technique while dealing with solo developers in a similar situation. Unfortunately, this method also did not do any good for knowledge sharing. The drawback of not writing a detailed SDD is therefore reflected at this stage of the waterfall model. Without a detailed SDD technical knowledge cannot be shared with future developers.

Sample screenshots of the final web solution are shown in Appendix D.

TESTING

This was the final activity conducted during the study and was performed using validation testing during the release of the solution. Validation tests also known as verification testing is performed at the closing stages of the project development cycle (i.e.,) at the time of release of the product [15]. The five users were presented the web solution to observe the final usability of the product. The users navigated through the screens without facing much difficulty. Testing was done to see if the solution has the features captured during assessment tests and exploratory tests and if the test users are satisfied with the implementation and the final solution.

Apart from the above testing, the author also performed system testing and acceptance testing. Unit testing was performed to capture bugs in small modules of code during the development cycle. This was followed by system testing used to capture bugs after the entire system has been developed. Minor changes in the code were made to fix bugs during system testing. Test protocol consisting of test cases were written and used by the developer during testing the code. The final testing done was acceptance testing. Acceptance testing is similar to validation testing done with the test users and is basically done to get the user's approval. The current solution will also be presented to the management. Few functionalities of the system such as multiple location mapping will be done outside the scope of thesis before the final solution is presented to external users. The current solution is not fully complete to be ready to be sold to outside customers.

Discussion:

Testing of a system is usually carried out by testers in several companies. Startups experience the problem of not having many employees thereby forcing few employees to take up multiple roles. This criticism was pointed out in section 4.2 and it prevails in this startup. Therefore, in this startup all roles of a software engineer such as developer, designer, tester were carried out by a single person. Integration test was omitted in this case as different units of code were not written by several people. Also, integration of the current solution to the previous solution was not possible as both solutions had a different programming approach. The management can choose only one of the two solutions that are available. Bug fixing was a simpler task to do as a single developer was in charge of writing all the code and therefore was able to identify minor errors. Since the author was aware of the language used in programming and was responsible for the entire solution, she was able to detect minor issues in code. However, the process of finding bugs in the solution was time consuming since there were no other developers to review the code with making the quality of the code unreliable. A future developer making changes to the existing code may or may not find the current code satisfactory. The coding standards could not be compared to any other solution or with any other developer.

Though there were two other developers working on a different aspect of the solution, communication was rarely established with them. Therefore code review with them could not be done. The management and the test users also did not have the required technical knowledge to support the developer in carrying out testing. The only form of testing which the management could do was acceptance testing.

One notable aspect of performing testing in this case study compared to how it is performed in the regular waterfall model is that, bugs were reported throughout the development cycle. Changes were made during the entire development cycle instead of piling everything to the end. The test users did not find any major problems with the end solution since they had a participatory role throughout the development cycle. This is however not the case in the regular process of following the waterfall model. Changes are captured only at the end of development posing a risk of changing the entire solution thereby causing more errors and complexity.

Therefore it is the author's suggestion testing, bug fixing and reporting should be carried out in all stages of the waterfall model as it was done in this case study, to make it easier for the developer to solve issues regarding the solution then and there.

The test protocol written was put only for personal use as it involved the technical aspects of the solution and none of the test users were interested in reviewing it. Writing test cases was not time consuming in this case as these was just kept as to-do notes instead of writing a detailed formatted test protocol. If it was written in detail, it might have unnecessarily consumed a lot of time.

5.2. Case Study 2: Agile XP in a startup

The second case study began with introducing agile XP concepts to Krizp solutions. Like in case study 1 some of the scenarios of XP are tailored to suit the situations prevailing in this company. If necessary other agile concepts are introduced for the benefit of the organization. The author observed results of the study by interviewing company representatives and developers.

First Interview Analysis

By performing the first interview as mentioned in section 2.2 it was observed that the company did not have any structured approach to software development. Requirements were given to the developer on paper initially; clarification regarding requirements was gathered through phone calls with the customer. Major changes in requirement were not made during the development cycle but only after the solution was entirely developed, when the customer reviewed the solution. Both developers worked individually on different features of the solution without much communication on the alternate technical possibilities for the solution. They however interacted for minor bug fixing. No form of major documentation was maintained for previous projects and for the current project. The developers gathered requirements and developed whatever was asked of them. The developers did not have a project manager to oversee their work or keep track of time. The management did not have a participatory role and only observed whether the requirements supplied from the customers were being developed and delivered to the customer.

Second Meeting

The next step was to introduce the company personnel with the concepts behind agile software development. Interviews were held using Skype, taking about an hour or more each. During the interviews several concepts of agile such as Customer representative, the Planning game, Pair Programming, Test Driven Development, Continuous integration, Small releases, etc. were explained in detail to the employees and the CEO (Chief Executive Officer). These scenarios of ASD (Agile Software Development) were later implemented in the company and the results were observed by the author. The number of developers associated with the company is two and therefore the results obtained are in view of two developers working in close collaboration. The collected data was analyzed to see how well the team reacted to these scenarios, how it had transformed their way of developing software and how beneficial it was to the organization. The study was conducted during a period of one month when the developers were involved in developing a web based solution for the management of an educational institution.

Unlike in case study 1 there was a customer involved in this project, although he was not available for regular face to face meetings. Developers and management had the opportunity to contact him through phone calls to gather new requirements and give him updates on project progress. The management of this company also had background in computer science and was able to understand the ongoing proceedings of the process. During the study there was an active participation from the management.

Third Meeting

The third meeting was conducted over Skype to observe results obtained while implementing the agile XP practices. This was conducted in the middle of the month since the developers were given some time to work on the process. More meetings were conducted following this to clarify further doubts. The following are the results obtained from the third meeting:

CUSTOMER REPRESENTATIVE and TRACKER

In extreme programming the customer representative is one among the development team who represents the customer. In this case, the CEO/management of the company acted as the customer representative and was in charge to delegate prioritized user stories to the development team and have regular communication with the customer. The CEO also acted as the tracker wherein he was in charge of keeping track of time needed and time taken to complete specific tasks. It is similar to the role of a project manager. He made decisions on which user story was going to be implemented in the following iteration. He was therefore required to have continuous contact with the activities of the team. A user story represents a piece of functionality of an XP project and is documented on a small paper note called the story card. Tasks are developed based on the priorities of the user stories. Below is an example of a story card presented to the company [8].

#	Name	Estimated 1	Estimated 2	Actual
User Story				
Acceptance Test				
Name	**Test**		**Expected Result**	

Figure 7: Story Card (page 1)

Task	Who	Estimated	Actual

Figure 8: Story Card (Page 2)

A good user story is short, perhaps a sentence or two and easily understandable to the customers and the developers. For example, it is small enough that the programmer is able to implement half a dozen of it in the first iteration [8].

The CEO gathered requirements from the customer and presented it to the team which later developed these requirements of the system. The user stories were prioritized by the customer representative in collaboration with the customer based on what the customer wanted to be implemented in the first iteration. This prioritization was however not the final decision of the customer. He was allowed to change the priorities of the user stories anytime during the project lifecycle. The development team could also contact the customer if required and was able to approach the CEO to clarify doubts. After prioritizing user stories the customer representative took up the role as a tracker and initiated the planning game where the members of the team decide on which functionality of the system individuals were going to develop. During the study, the customer was available on-site for prioritizing user stories and subdividing tasks in user story one. He was also available for performing the first planning game with the development team.

Discussion:

Having a company member represent a customer within a team is a great way of channelizing activities within the organization. In startups this could be an added advantage since startups have few employees working in close collaboration and the management can oversee the work done by the employees in a positive manner. The management is given the opportunity to be part of the development team and monitor the work done by them more closely instead being someone who orders the staff to get work done without understanding the complications within the team and the project. This therefore creates a more friendly work environment. Major IT companies may not have the possibility for their management to have close interaction with the team, but startups have this feature and instead of looking at it as a problem as described in section 4.4. Factor 3, having the management as a customer representative could be a positive approach towards employer-employee relationship. In the regular process, the customer representative and tracker are two different people. Since this startup had very few human resources these two roles were taken up by one person and this may be the case in view of any other startup with few employees.

THE PLANNING GAME

The CEO of the company initiated the planning game every week or at least once in four days since the project was small and developers were able to complete tasks in a user story more quickly. The planning game is basically an iteration planning meeting where both the developers in the company decide on which tasks they are willing to take up during the week. It is recommended for the customer to be available on-site during the iteration meeting so he is aware of what is going to be done during the week. In this case, however the customer was able to be directly available in the office only twice during the entire month. The rest of the time he was available for contact through phone calls and e-mail.

During the planning game, the CEO presented the employees with the story card containing a user story. For example, during the first few days of project startup the user story was the login page. This was further subdivided into smaller tasks like creation of text boxes, login button, linking the login screen to home page, password authentication, working on the page layout, etc. Subtasks which were not purely technical were also gathered from the customer over the phone. The initial design was kept as simple as possible. These tasks were divided between the members of the team. The two developers chose whichever tasks they were comfortable working with and a rough estimate of time needed to complete the task was made. The entries were made in story card (page 2) as shown in Figure 8. The sum of all the subtasks estimates was recorded in Estimation 1 of story card (page 1) shown in figure 7. A

buffer time was also maintained during every planning game to compensate for tasks which took longer to implement than expected. After choosing subtasks the programmers began implementation of the user story.

The next planning game was initiated after the first user story was completed and it followed the same procedure. Since the customer was not on-site for the second planning game, the developers sent screenshots of user story 1 to the customer to get his approval before they began working on the second story. They also made a demo of user story one to the customer by sharing the screen in Skype video. Changes in requirements were captured and were implemented in the solution. If the customer was not able to be present on-site for the following week, the CEO gathered subtasks from the customer via phone calls/Skype and presented it the developers during the second planning game. The developers maintained the concept of small releases to the customer throughout the development cycle.

Discussion:

Since there were only two developers involved in the study the planning game conducted was much simpler. In normal circumstances where there are more developers, the developers bid on the tasks they want to do. The person who bids the lowest wins the task and gets to implement it [8, 26]. However, the developers in this startup felt they did not have many options to choose from since even if both developers did not want to do a task they were forced to pick one of the tasks as there was no other developer. Hence, they picked up alternating tasks and decided to implement it.

Having the CEO represent the tracker was the most convenient since he was able to perform better task and time management, something that was lacking before the process was implemented. The team felt they were on the right track taking up one task at a time in a more organized format. Also the vacuum of not having a project manager to guide and keep note on their efforts was replaced.

Most startups do not have a task management tool as with several other IT firms since they do not have the finance and software needed to support this activity. In such scenarios having story cards to keep track on tasks completed and in the process of completion is recommended. This however lacked traceability and in larger projects it might have been difficult to lookup previous iterations of the system. This conclusion was also presented by Nystrom in Agile Solo [26].

Communication with the customer over the phone after completion of first user story faced a few difficulties. The customer was not able to interact with the system to identify relevant changes. He could only comment on few functionalities and the look and feel of the system. Perhaps more changes would have been captured if the customer was on-site. Slow net connection during communication was also a problem. The call got disconnected several times during the demo which the customer and the team felt irritated about. The customer was wasting his time during such moments and the team felt inconvenient. Therefore it is always advisable to have the customer on-site, in this case and perhaps in most other startups this is often not the case.

The planning game was observed to increase communication within the programmers and the management, again something that was missing before the process was introduced in the firm. It raised personal satisfaction in the work culture and this reflected in the performance of tasks the following week.

PAIR PROGRAMMING

One of XP's greatest features is the concept of pair programming. Developers work in pairs in the same system while implementing the solution. One of the programmers in the pair is the driver and the other is the co-pilot. The driver writes the code and the co-pilot discusses other possible solutions of implementing the code. Whenever necessary the roles can be swapped [8]. In this study, the programmers worked in a pair throughout the project lifecycle. The driver worked on the tasks he had chosen while the co-pilot assisted him in providing alternate solutions. If one of the programmers was unable to complete his chosen task, the other developer assisted in completing the task for him.

Peer code review a technique from agile Solo [26] was introduced as an additional concept within pair programming. After completing a task, both the programmers reviewed the code with the CEO. This was to enhance the probability of finding more bugs in implementation and also to write structured, commented code in a way that everyone apart from the programmer is able to understand [26].

The CEO was always present to oversee the work carried out by the pair. After completion of a specific task, the actual time taken to complete the task was recorded in the actual column of story card (page 2). The total work estimate of completing all the tasks was recorded in estimation 2 of story card (page 1). The actual time for completion of the entire user story was recorded during the next iteration meeting before starting a new user story.

Discussion:

It was observed that adopting the method of pair programming increased knowledge sharing within the team and thereby promoting the development of better quality software. It also increased team work and communication. This was however only a verbal exchange of knowledge. Documented knowledge sharing could not be established by this method for future use. The pair exhibited good communication since their knowledge backgrounds were similar. Therefore there was greater co-operation between the two members. They supported and guided each other smoothly throughout the period of working together. This may not be the case in all circumstances were the members of the team exhibit different levels of experience. In such cases, this might cause conflicts in opinion. Choosing partners who are not compatible in performing a task can be a drawback. This drawback though not observed in this startup, is important to take into consideration while dealing with pair programming in a startup since conflicts in a smaller work environment can cause problems in the work culture of the organization. Furthermore, in a startup, people probably cannot choose who to work with. It is therefore important that the programmers are compatible. On possible solution for this in a startup is to involve the programmer in the recruiting process of a new programmer into the firm.

Additional bugs in the system were captured when both the programmers reviewed code together. Peer code review was also a positive approach in this startup since the CEO of the company was also from a similar knowledge background. He was able to understand the code that was written and give his comments on it. This cannot be applied to all startups were the management does not have similar technical backgrounds.

Reviewing code with the CEO was also a great way for the management to keep track on work progress periodically. There was a closer interaction between the management and the employees. Personally the programmers felt encouraged since their individual efforts in the

project were being recognized by the management. The management was able to track individual employee performance. This made the employees work harder and smarter which was reflected in team performance and quality of the solution.

Involving the management in reviewing the code also supported to knowledge management is some form. If the present developers were to leave the company, the management will be able to guide the new employees into resuming the project from where it was left off. This is an added advantage in a startup and can be used only if the management has previous knowledge in developing software and other technical backgrounds.

TEST DRIVEN DEVELOPMENT (TDD)

TDD is another fundamental practice of agile. Developers write unit tests before writing the code for the solution. Doing so helps the developer have a clearer view of the task they are implementing. This is primarily useful when refactoring code, during which it ensures the entire system is not changed producing bugs unnecessarily [26].

Discussion:

Developers found it difficult to begin writing tests for each module of code and they had never adopted this strategy elsewhere before. They found it time consuming and therefore wrote tests only for few functionalities. The author did not get much feedback on this aspect of agile from the developers.

However, it is recommended for other startups to adopt this practice regularly as part of the development cycle as it maximizes the efficiency of code produced. Though it is considered to be time consuming it can still be beneficial to the project as a whole. There cannot be many changes this aspect of agile brings to a startup; however it can be useful for individual projects.

CUSTOMER TESTS (ACCEPTANCE TESTS)

Customer tests were performed at the end of each iteration where the customer was presented with the implemented user story. Since the customer was not available on-site to perform every acceptance test, the management sent in the developed module by e-mail to the customer. At first a demo was made through Skype and later the customer was asked to work on the same in his local PC (Personal Computer) and give his approval. The tests and the expected results were recorded in a story card (page 1). This process was repeated for three weeks. The team faced some difficulties in guiding the customer to run the project on his PC.

During the last week of the project before it was completed the customer was available on-site to perform acceptance tests with the developers. Changes were also recorded at the end and reflected in the original solution.

Discussion:

One way of keeping the customer reassured throughout the project lifecycle is to involve him in every stage of development. This particularly works for a startup as winning more customers is their primary goal and this method was found suitable to keep the customers satisfied. Though the method adopted to perform customer tests was not ideal, most of the customer tests were done by the customer representative on behalf of the customer before presenting it to the customer. Since the CEO had relevant technical background to get the

solution in accordance to the customer's needs the process of acceptance tests was easily accomplished.

DESIGN IMPROVEMENT (REFACTORING)

The team did continuous improvement on design throughout the development cycle. After the customers test at the end of every user story, suggested changes were recorded and reflected for the immediate design of the system. Therefore there was continuous refactoring done in the project in the presence of the tracker. During the entire month of the project the team made changes for roughly around 12-15 times.

Discussion:

Refactoring supports creativity and in a startup when this is done in the presence of the CEO the team is able to suggest new ideas to the management in a more convenient way. Normally this may not happen in any organization, the management is never involved in the team activities and never promotes ideas emerging within teams. This technique is therefore recommended for usage in the presence of the management. If the management is not involved, then the benefits of this technique will only be reflected in the quality of the solution and satisfaction of the customer.

40-HOUR WEEK

The development team and the management maintained regular working times of eight hours a day. The 40 hour schedule was not strictly followed during two weeks since a few days the team spent more than eight hours a day or less than eight hours a day depending on the work they wanted to complete during that week.

Doing so helped the team finish the entire project before the required deadline and therefore they had time to make further iterations and improvements to the finished system in collaboration with the customer.

Discussion:

Working more than 40 hours during a couple of weeks was not a problem in this startup since the company did not have many projects in hand and when they had a project the team felt like contributing the most towards the project. Therefore this did not necessarily drain energy reserves; instead the team was enthusiastic about it. Since the management was now more involved in team work, the team took some extra time every day to reflect on activities carried by them. By interviewing one of the employees the author came to understand that the development team was now more satisfied since they had time to establish better communication with the CEO. There was more unity now within the organization than before, thereby raising employee satisfaction in the organization.

Fourth Meeting

The final meeting was conducted with the CEO and employees to get overall feedback on the process introduced in the organization. The CEO was completely satisfied with this new approach towards software development. First, since it brought the team and the management closer in terms of communication and second it supported knowledge sharing and creativity. The CEO now had an opportunity to be part of the development team without actually involving in any development of software. He was able to perform time and task management

more effectively in the absence of a project manager. Similar feedback was obtained from the programmers. They were also impressed with this new approach which did not focus too much on initial upfront design and extensive documentation. They were excited primarily because the only process they were familiar and worked with all this while was the waterfall model. This new approach of agile XP therefore gained quick popularity within the team in terms of developing software effectively and bringing the management and employees closer together. They now did not think of themselves as two different entities, rather as one development team working together for the future of the organization. One approach of agile which the team did not feel comfortable working with was Test Driven Development. This aspect as mentioned earlier, the team felt was time consuming and difficult to implement. They wanted to improvise on it in the future project if possible.

From the management the author understood that the customer was also completely satisfied with the final solution. The management and customer felt if the customer was on-site acceptance tests could have been better performed. The customer felt initial difficulties in doing the first acceptance test online since he faced problem installing the required software to run the solution in his PC. The second acceptance test was done more smoothly since by this time the customer got the hang of the system. From the management's perspective, the CEO was slightly unconvinced since he felt it was his duty to get the required software installed in the client's system. Time and location barriers prevented him to do so. Apart from these minor issues, the client was happy about the way the programmers approached him after completion of every module of the solution to get his feedback. This gave the company additional credit.

5.3. Recommendations

Some recommendations proposed for organizations in similar situations as in this study are as follows:

1. For startups working on small projects on multiple teams it would be advisable to have daily standup meetings between the teams to promote transfer of project related ideas between developers in various teams. This would support inter-project knowledge sharing in the organization. This recommendation aligns with Striebeck's way of conducting standup meetings in Google [22].

2. Even in case of a solo developer environment, if the management has sufficient technical knowledge, it would be advisable to involve them in peer code review. This is to enhance the quality of code written.

3. Test Driven Development, the practice of writing unit tests before the coding is a good technique to be followed in software development. Especially in a solo developer project where the developer has no support for external code reviewing, this technique can be adopted to minimize the number of bugs in the system raising the developer's confidence while coding the solution.

Some recommendations proposed for the case studies are as follow:

Case Study one:

1. To approach different test users from different age groups and backgrounds for performing case study one. The performed case study focused on users only with engineering

backgrounds. It would be helpful to get a clearer context for the web solution when ideas are gathered from user groups more similar to the intended users, preferably without any engineering or computer background.

Case Study two:

1. To use the realized user stories as documentation for future use. Since agile XP does not support a structured way of documentation, it will be helpful if the user stories from past projects are archived to be used as a source of reference in the future. This can also support traceability between user stories, tests and code sections thereby promoting reuse and knowledge sharing in future projects.

2. If available within the same geographical location and if the customer is unable to be present on-site, the customer representative can take the initiative to meet the customer directly for performing acceptance tests. This of course is dependent on time and cost barriers and it's at the discretion of the customer representative.

6. Discussion

This section explains a little more about the threats to validity in the study. If case study 1 were to be redone again it would provide different requirements for the software solution based on the selection criteria of the test users. As explained in Chapter 7.2 the author choice in test users were biased. Therefore this is one aspect by which the results might vary if this study is replicated. The author also did not have the relevant technical background to continue development of the previous web solution without the help of documentation. This could have also influenced partially on the outcome of the result. Apart from this there will not be any other drastic changes in the results if the study was repeated

Case study 2 on the other hand might have changes if redone based on the views of the members in the organization. One major drawback in conducting case study 2 was the author was not part of the organization unlike in case study 1. Therefore this has a greater impact on the outcome of the results. The results are mostly based on the views of the members in the organization. The views are bound to vary for instance, pair programming can show different results based on the pair involved in programming.

7. Conclusions

The final section of the thesis discusses how the research question was addressed based on the results obtained from the case studies and provides suggestions for future research.

The purpose of writing this thesis was to analyze the pitfalls and advantages of working with a specific software development process such as the waterfall model or agile XP, in small projects within fluid entrepreneurial environments. By conducting the case studies in two different organizations, the author comes to the conclusion that the agile development process took a higher stand in comparison to waterfall model when applied in a startup organization.
By performing case study 1 (section 5.1) it was understood that the regular waterfall model may not be most suitable to be applied in a startup since it did not promote communication within the team, management or the customer. Therefore the waterfall was tailored using user centered design to be implemented in the organization. This method produced good results despite of all the prevailing obstacles within the organization. If we take a closer look, it is understood that the waterfall model following user centered design was also fulfilling some agile principles (#1 and #4 of agile manifesto). There was continuous release of the product throughout the course of the project. Also, there was regular interaction with the test users to receive constant feedback. However, this approach did not promote increased communication within the team or knowledge sharing because it was applied in the context of a single developer. Additionally, the agile XP process could not be applied to this startup in case study 1 since the management lacked the required technical background to communicate with the customer or represent the customer within the team. The developer lacking communication with the customer was also a drawback here as a customer is the key aspect of agile software development. The management could however have represented the tracker, but since there was only a single developer, spitting of tasks was irrelevant and time was not a constraint in this startup which was developing an experimental solution and who based their business on this solution. Perhaps if there were more developers the management could have overseen time and task management amongst team members. It is therefore the author's opinion to follow the tailored waterfall approach in situations where the organization has a solo developer and when the developer lacks communication with the customer.

On the other hand, the agile XP approach adopted in case study 2 (section 5.2) was found to have a higher impact in a startup as it supported the three factors explained in section 4.4. Knowledge sharing, though not promoted entirely was partially accomplished between the team and the management. The approach brought the management and the development team into closer team collaboration and communication was improved which was not observed in case study 1. The agile XP is a good approach to be adopted in a startup when there is more than one developer involved in projects. If the management has the required technical background, then it would be an added advantage while following agile XP in a startup.

In conclusion it is therefore suggested to tailor the agile approach either in terms of agile XP or waterfall model following user centered design, to adapt to different entrepreneurial organizational needs and implement it to have better performing teams and satisfied employee-employer relationships.

7.1. Future Work

1. To identify a suitable knowledge sharing technique while performing user centered design in a startup.
2. To test other available software development processes in a startup such as V-model, Scrum etc. and see how well it can be applied in a solo developer startup environment.

7.2. Limitations

While performing case study one the test users were not chosen from the Swedish municipality because of the language barrier between the author and the users. The author is also unsure if the intended customers are from the Swedish municipality.

Secondly, while implementing the web solution Java Springs was not the adopted programming language strategy. Since Java Springs was an entirely new concept for the author, it was difficult to jumpstart with this language without any formal guidance or recorded data on the previous solution. Learning a new language and start development was also time consuming since it was an entirely new concept and not familiar to everyone making it difficult to get proper guidance. Considering the limited time in hand it was not possible to adopt this language.

8. Acknowledgement

First and foremost, I thank Lord Jesus for giving me this wonderful opportunity of doing a Master degree at Chalmers University of Technology, Sweden. Without His mercies none of this would have been possible. I am forever grateful.

I extend my sincere thanks to my supervisor at Gothenburg University, Wolmet Barendregt for her able guidance, wise suggestions and effort throughout my thesis, without whom this would not have been completed. My special thanks to my examiner Professor Miroslaw Staron at Chalmers University for his valuable advice on the report during the course of the thesis. I am also immensely thankful to my family and friends for their motivation and selfless help which kept me going forward. I thank my friend, Madeshwaran Selvaraj for his constant support throughout my study here in Sweden.

Last but not the least, I owe my deepest gratitude to the following people who guided and encouraged me during my thesis work:

Jessica Thunman and everyone at Tesenso AB

Sreethar Chitambaram

9. References

[1] [Online]: http://www.investopedia.com/terms/s/startup.asp#axzz1cCOlmrRR

[2] Rajasekar, S., Philominathan, P., Chinnathambi, V.,"Research Methodology" page 1. http://www.scribd.com/doc/6949151/Research-Methodology

[3] Yin R. K., "Case Study research: Design and Methods", 2009

[4] [Online]: http://www.waterfall-model.com/

[5] TechRepublic [Online]: http://www.techrepublic.com/article/understanding-the-pros-and-cons-of-the-waterfall-model-of-software-development/6118423

[6] [Online]: http://searchsoftwarequality.techtarget.com/definition/requirements-analysis

[7] [Online]: http://www.coleyconsulting.co.uk/waterfall-model.htm

[8] Steinberg, D.H., & Palmer, D.W. (2004), *Extreme Software. Engineering–A Hands on Approach*. Pearson-Prentice Hall

[9] Fowler, M., Highsmith, J.(2001, August). *The Agile Manifesto*.Software Development. http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

[10] [Online]: http://www.agilemanifesto.org./

[11] Beck, K., "Extreme Programming Explained: embrace change", Addison-Wesley professional, 2001

[12] Abrahamsson, P., Salo, O., Ronkainen, J., &Warsta, J. (2002) Agile Software Development Methods, Review and Analysis, VTT Publications, Espoo, Finland. URL:http://www.vtt.fi/inf/pdf/publications/2002/P478.pdf

[13] Emery, P., "The Dangers of Extreme Programming" May 20, 2002. From: http://members.cox.net/cobbler/XPDangers.htm#_Toc530042780

[14] Humphrey, W., "Comments on eXtreme Programming", *eXtreme Programming Pros and Cons: What Questions Remain?, IEEE Computer Society Dynabook* (September 2001), http://www.computer.org/SEweb/Dynabook/HumphreyCom.htm

[15] Rubin, J., "Handbook of Usability Testing, How to plan, design and conduct effective tests", Wiley Technical communication library, ISBN: 0-471-59403-2

[16] Woodson, Wesley E., "Human factors design handbook: Information and Guidelines for the design of systems, Facilities, Equipments and products for human use", New York, McGraw Hill, 1981

[17] Allison L. Hansen, Reflections on I/Design: User Interface Design in a Startup, ACM, 22-27 March 1997

[18] [Online]: http://smallbusiness.chron.com/organizational-structure-definition-3804.html

[19] Jacobides., M. G. (2007). "The inherent limits of organizational structure and the unfulfilled role of hierarchy: Lessons from a near-war. Organization Science", 18, 3, 455-477.

[20] Mintzberg, H. (1983). "*Structures in fives: Designing effective organizations",* Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

[21] Feng Ji and Sedano, T., "Comparing Extreme Programming and Waterfall project Results", Carnegie Mellon University, Jan 2011

[22] Striebeck, M., "Ssh! We are adding a process…", Google Inc, IEEE, Agile 2006 Conference.

[23] Yang andWu, "Knowledge sharing in an organization- share or not? " , IEEE 2006

[24] Feynman, R.P., "Cargo Cult Science", June 1947

[25] Cockburn, A. (2002), "Agile software development", Boston, MA, USA Addison Wesley

[26] Nyström, Anna June 2011, Agile Solo: Defining and Evaluating an Agile Software Development Process for a Single Software Developer. Thesis, (Master Degree). Chalmers University of Technology

[27] Eckerson, Wayne W. "Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications." Open Information Systems 10, 1 (January 1995): 3(20)

[28] Salameh, Abdallah April 2011, On Process Tailoring, An Agile Example, Thesis (Master Degree), Chalmers University of Technology.

[29] Mintzberg, H., (1980), "Structure in 5's, A synthesis of the research on Organizational Design. Management Science"

[30] [Online]: http://www.socialresearchmethods.net/kb/introval.php

[31][Online]: http://www.une.edu.au/WebStat/unit_materials/c2_research_design/validity_threat_internal.htm

Appendix A

1. Previous Work on Web portal

As the company had started a year ago, a prototype system was previously created but not used in mainstream. There were no clients linked with the system. The system was developed using Java Springs/Spring Roo which followed the MVC (Model, View, Controller) architecture and hibernate to manipulate the database. The database used was Postgres database with pgadmin3 to manage it.

1.1 Existing System Requirements

a) The web system should have a common login page for both administrator and customer.
b) On successful login of the customer, the web page displays the home page which includes four tabs namely home, reports, account and help.
c) The home tab is the default tab that is enabled as clicked when logged into the system. The home tab should display a dynamic map of the city of Gothenburg showing all fill levels of the can which are full, close to full or empty bins.
d) The map should be dynamic such that it enables viewing of only full, close to full and empty cans when needed.
e) The map should be made full screen when necessary thereby hiding all unnecessary information of the page.
f) The home tab should also include a news column mentioning the percentage of the fill level of the cans. The percentage should be highlighted in various colors of red, yellow and green.
g) The second tab in the customer page should be the report tab.
h) The generated report should enable the viewing of the current statistics of the bin.
i) The third tab in the customer page should be the account tab. This should display information pertaining to the specific customer.
j) The fourth tab should be the help tab. On clicking the help tab it should display general direction with regard to the use of the web portal. This is mainly to guide the customer through the functionality of the website.
k) On successful login of the administrator, the web portal redirects to the admin page.
l) The admin should be able to add/modify/view/delete sensor groups.
m) The admin should be able to add/modify/view/delete sensor nodes.
n) The admin should be able to add/modify/view/delete customer information. (This requirement is hypothetically stated since the implementation included an account tab).
o) The web portal should reside on tnr.tesenso.com
p) The system should include logout functionality for both the administrator and customer.
q) The system should support validation wherever required.
r) The system should be secure and provide authentication of users.

1.2. Implemented System Requirements

a) The system had a login page but not a common login for both the administrator and client. Two separate login pages were used to access admin and customer page.
b) The system did not provide authentication of users.
c) The customer home page included four tabs which are home, reports and help. These were implemented according to the specified system requirements.
d) The home tab included a map which was not dynamic. On successful login the map statically displayed all the fill level of bins located in the city of Gothenburg.
e) The page did not have a customized view of full, close to full or empty bins. The page lacked a drop down menu which allowed the user to select sensor groups.
f) The home tab did not display a news column mentioning the number of bins which are full, close to full or empty. Instead it displayed a horizontal percentage bar of the fill level of cans.
g) The percentage of the fill level of cans was highlighted in various colors of red, yellow and green.
h) The map could not be made full screen when necessary.
i) The report tab was the second tab of the customer login page.
j) The report displayed a statistical graph on the cans during a defined duration. The page included four drop down menus which allowed the user to select longitude, latitude, start date and end date. The output of the report was based on the options that were selected. This report could not be downloaded, saved or printed. It was available to be viewed only within the web portal. Furthermore it did not include data suggesting the dates the particular bin was previously emptied nor the battery level of the sensors. This report section of the web portal was extremely user unfriendly as the users did not know which longitude and latitude option need to be selected to view a specific cans data. It required the user to memorize latitude and longitude positions of specific cans. The generated graph was not easily understandable. The email functionality of the report was also not included in this system.
k) The account tab was the third tab but lacked data within it.
l) The help tab was available as the fourth tab and it too lacked data within it.
m) The admin page was accessed by a separate login and included create/view/modify sensor node and sensor group functionality. The delete functionality was not included in the admin web portal.
n) The admin could not view or edit data relating customer information and license period of each customer. The admin therefore did not have privileges to access the customer data base. The admin was also unable to delete a customer from the customer database.
o) The entire requirements pertaining to customer information was not implemented in the system.
p) The system resided on tnr.tesenso.com. The customer page could be accessed through http://tnr.tesenso.com/trackandroute/dashboard.
q) The logout functionality was not implemented for both the administrator and customer. Logout was accomplished by simply closing the window.

1.3 Identified failures

The previous system was primarily built to address the scalability of the server when connected to multiple sensors. However not many sensors were in place and hence the system addressed the scalability of the server when connected to just one sensor. The results for multiple sensors were evaluated theoretically and not practically.

Since the goal of the previous project was different, it lacked a strong front end design. The portal was found to be not so user friendly and had poor usability criteria. The look and feel of the website was the default layout found in Java Springs. As Java Spring is a newly emerging technology most of the system features were not implemented due to the complexity of the language and time constraints. The system lacked user validation/verification testing. In comparison with the current solution, the previous solution did not include data in account tab to display customer related information like the customer's personal information and the duration of license period. The current solution supported extended/varied features based on the revised requirements of the new management currently running Tesenso.

The previous system had poor security features. The username and password were the same and the change password functionality was not provided. The system did not support language functionality. English was considered to be the default language. The system also did not deal with the transport issues for emptying of garbage bins.

Finally the system did not include any form of route optimization/scheduling technique. The shortest path of the route to be taken for emptying of garbage cans was not depicted in the map. Perhaps the previous management of the company did not want this functionality to be implemented.

Therefore, these identified failures and the need for different features in the system led to the development of the new web portal. The primary factors affecting the change in requirements were the new management that took over Tesenso. Their needs and view of the system changed, causing the change in implementation of the system.

Appendix B

2. Software Requirement Specification

2.1. System Requirements

The purpose of the web application/portal is to allow customers/users to identify the fill level of garbage cans and take necessary measures to deal with it, without contacting the personnel of the selling company. It is additionally used by administrators to manage the system.

2.1.1. Functional Requirements

1. Customer Account

In order for the customer to use the portal the web application should allow the customer to manage the web account.

1.1. Scenario "Login"

When the customer opens the web portal he is presented with a login form which consists of a username and password. If the customer is a registered by the organization then the customer is allowed to log into the application. If the username and the password are correct the user is authenticated. Otherwise the user receives an error message saying "login failed" and stays on the same page. Under this circumstance the user has to retry login again. The rationale of this requirement is the desire of the garbage collector to use the system. The requirement depends on 2. 4.

1.2. Scenario "Logout"

Wherever the user wishes to logout he can do so by choosing the logout function. The view is changed to the login page. The rationale for this requirement is the desire of the user to leave the application. The requirement depends on 1. 1.

1.3. Scenario "Select fill level of cans"

The home tab is the first tab in the customer page after successful login. It is kept enabled by default; therefore the customer need not click it to view the contents of the page. Here the customer is presented with the map of Gothenburg. A drop down box is provided with three options namely: full, close to full and empty. The customer can view the full, close to full and empty cans across the city by choosing the respective option in the drop down box. To the right of this page is provided a news column which gives the customer a quick look up on the fill level of the cans. The rationale of this requirement is the desire of the customer to spot the locations of the respective fill level of cans on the map. The requirement depends on 1. 1.

1.4. Scenario "View information on sensor node"

After the customer selects fill level of the cans the sensor nodes are visible on the dynamic map. By clicking the sensor node the customer is provided with a popup that shows the sensor address, the fill level and batter level of the node. The rationale of this

requirement is the customer's desire to view the sensor node's information. The requirement depends on 1. 3.

1.5. Scenario "Print report"

The report tab is the second tab in the customer view page. The customer can enable this by clicking on it. The customer is provided with a calendar function to choose the duration of the desired report to be generated from the database. The report contains the following columns: serial no, sensor address, battery level and fill level. The customer can print this report by clicking on the print function available in the report tab. The rationale of this requirement is the customer's need to print the generated report. The requirement depends on 1. 1. and 1. 3.

1.6. Scenario "Save report"

The customer can also save the report into a .pdf file by clicking the save function. The rationale of this requirement is the customer's desire to save the generated report. The requirement depends on 1. 1. and 1. 3.

1.7. Scenario "Email report"

The customer can email the report by clicking the email function. The rationale of this requirement is the desire of the customer to email the generated report. The requirement depends on 1. 1. and 1. 3.

1.8. Scenario "View account information"

The account tab is the third tab in the customer view page. The customer can enable this tab by clicking on it. This tab provides the customer with customer related information live the number of sensors associated with him and the paid license length. The rationale of this requirement is the desire of customer to view his account information. The requirement depends on 1. 1. and 2. 4.

1.9. Scenario "User Manual"

The help tab is the fourth and final tab in the customer view page. The customer can enable this tab by clicking on it. This tab provides the customer with the user manual which gives assistance on the various features of the web portal. The rationale of this requirement is the customer's need to get some guidance on the usage of the web portal. The requirement is dependent on 1. 1.

2. Administrator Account

In order for the administrator and customer to use the web portal the web application should allow the administrator to manage user account and the application's database. Note: The web application provides the customer and administrator with a common login page. Therefore the login requirement remains the same for both the users.

2.1. Scenario "Create sensor node"

In order for the company to install a new sensor node in a new location, the administrator should be able to create a sensor node in the web application. The admin can do so by clicking on the create sensor node link which opens a form. This form includes details

like the sensor id, the address, fill level and battery level of sensor. The entered information is saved by the admin clicking the enter function. The rationale for this requirement is for the admin to create a new sensor node. This requirement is dependent on 2.

2. 2. Scenario "View sensor nodes"

The admin is provided with the view sensor nodes link to view all sensors available. The admin is provided with this information on clicking the view link in the admin view page. The rationale of this requirement is the administrator's desire to list all the available sensor nodes at once. This requirement is dependent on 2.

2. 3. Scenario "Delete sensor node"

The admin can delete a specific sensor node by selecting the delete function. Checkboxes are available to allow the admin to select multiple sensor nodes and delete it all at once. The rationale of this requirement is the desire of the administrator to delete specific inactive nodes from the system. The requirement is dependent on 2.

2. 4. Scenario "Add new customer"

The administrator is given the privilege to create an account for a new customer. On clicking this function he is presented with the account details screen and fills in the required details. When the admin confirms the new customer creation, the customer details are directly registered into the system. The rationale for this requirement is the necessity to add a new customer to the system.

2. 5. Scenario "Delete customer"

The admin can delete a customer from the system by choosing the delete function. After he confirms the deletion the customer account is removed. The rationale for this requirement is the necessity to delete a customer from the web application facilities.

2. 6. Scenario "Edit customer information"

The admin can edit customer related information by choosing the edit function. After the changes has been made and saved, the page with latest information is shown. The rationale for this requirement is the necessity to edit customer account information. This requirement depends on 2. 4. and 2.

2. 7. Scenario "View customer information"

The admin can view all customer related data by choosing the view function in the admin view page. The rationale of this requirement is the desire of the administrator to view all customers associated with the system. The requirement depends on 2.

2. 8. Scenario "View sensor groups"

The admin can view all available sensor groups such as: full, close to full and empty by clicking the view function in the admin view page. The rationale of this requirement is the administrator's need to view all sensor groups.

2.1.2. Non-Functional Requirements

-The server must respond to user queries in 2 seconds.
-An adversary cannot eavesdrop on communication between the end-user application and the server during login operation.
-The server has to provide services to the web application 24/7, 99% of the time.
-The application should provide convenient user friendly look and feel.
-Password should be case sensitive. Its minimum length is 6 characters.

2.1.3. Additional Requirements

There are a few requirements outside the scope of the project which will be implemented if time permits. One such requirement is to trace the shortest route to traverse to reach the full garbage cans. To do this the current system needs to be integrated with 3rd party software which implements route optimization technique. The route is displayed in the map in red. Options should be provided in the web portal which allows the user to select the source and the destination.

Appendix C

3. Software Architecture

The development of the web portal followed the Multi-Tier (3-tier) Architecture. It is basically a client-server architecture where the presentation of data, processing the application and data management are considered to be independent processes.

The three-tier architecture has the following three tiers [27]:

Presentation tier:
This is the topmost layer of the application which includes the user interface of the portal. It outputs information to the web browser and the inputted data is used as input for the remaining tiers of the system.

Application tier:
This layer is also known as the business logic layer, middle layer or the data access layer. The data from the presentation layer is served as input to the application layer. This layer controls the web solution's functionality by detailed information processing.

Data tier:
This tier connects with the database servers and is responsible for storing and retrieving information. The data in this tier is independent to the application tier or the business logic of the solution.

Appendix D

4. Implementation: Screenshots of web portal

Below are few sample screenshots from the admin view and customer view of the web portal.

Admin View

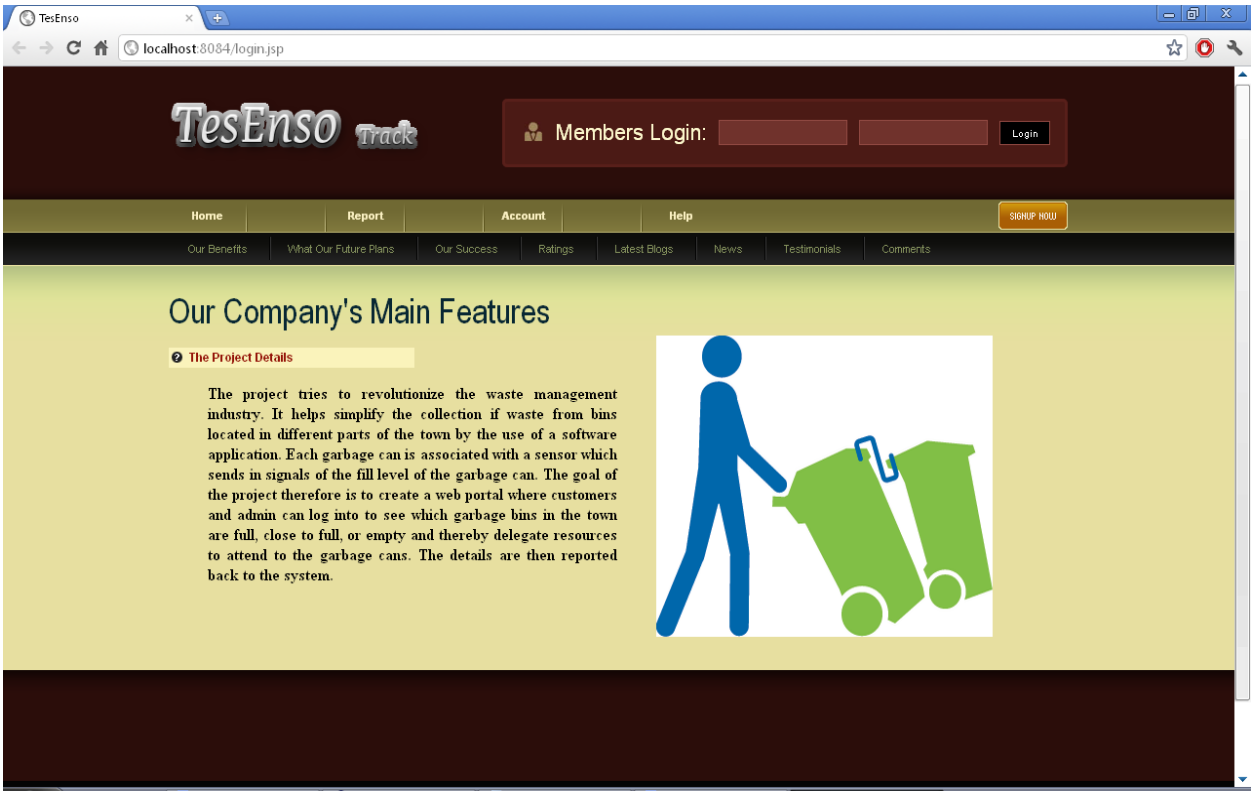

Figure 9: Login page common for both admin and customer

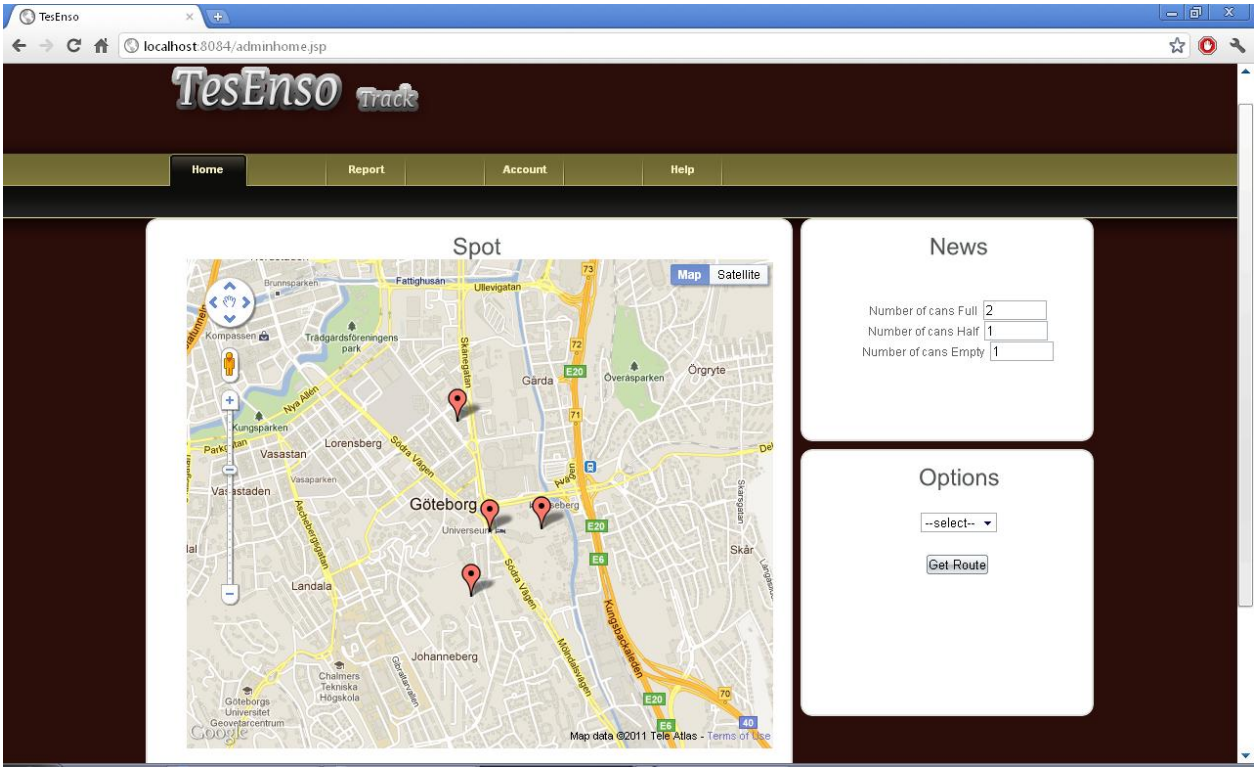

Figure 10: Admin home page displaying all fill level of cans

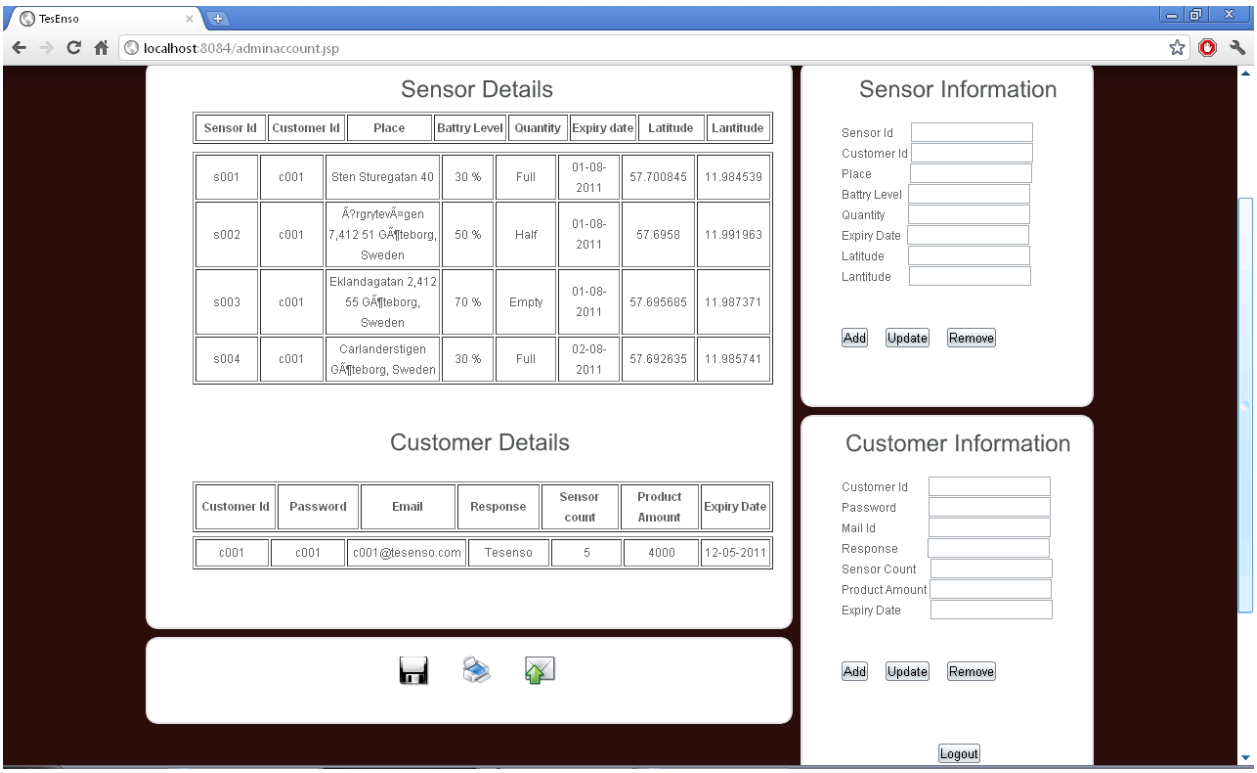

Figure 11: Admin account page showing all sensors and customer information

<u>Customer View</u>

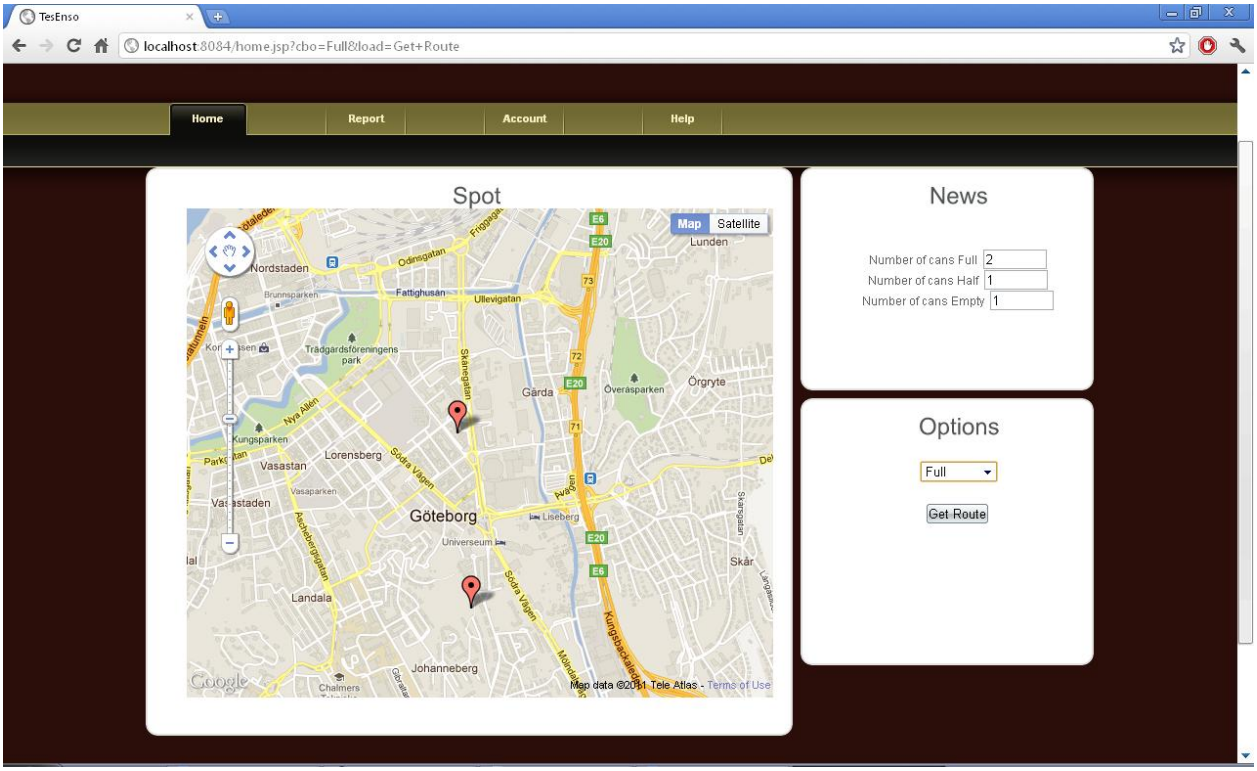

Figure 12: Customer home page displaying locations with full cans

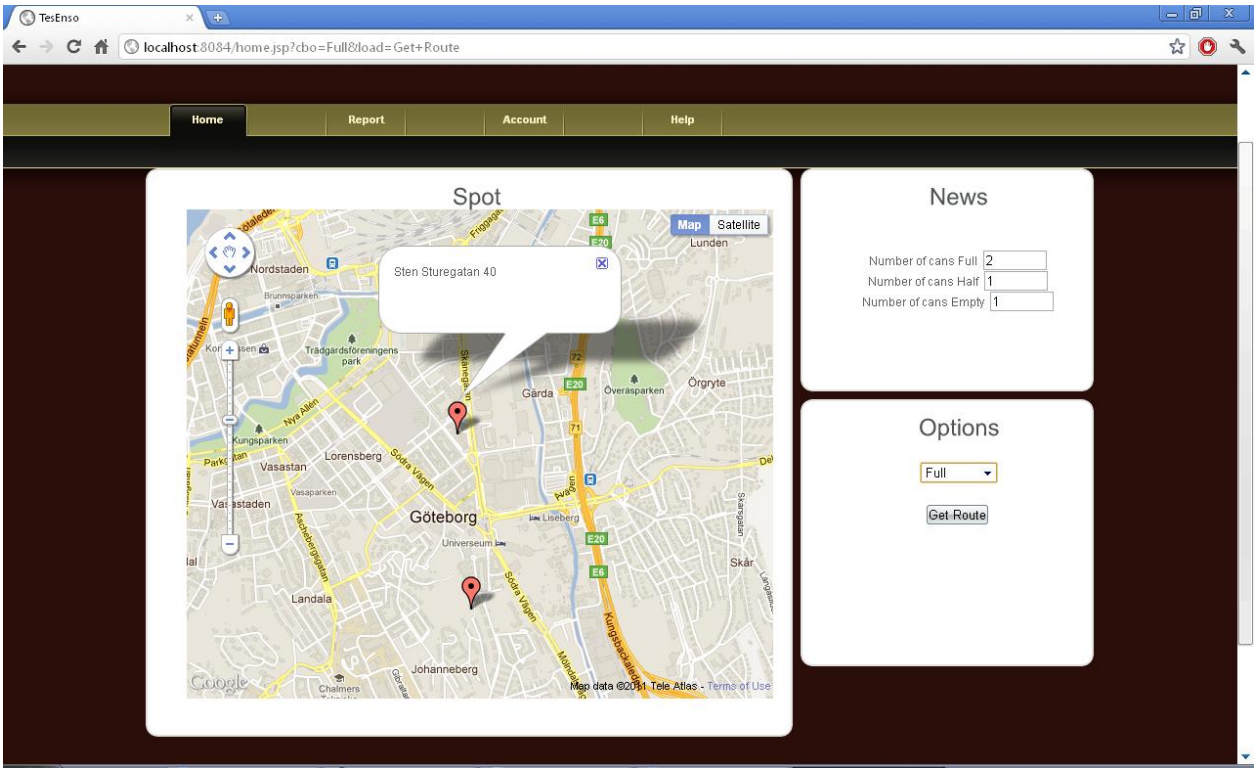

Figure 13: Customer home page displaying address of selected location

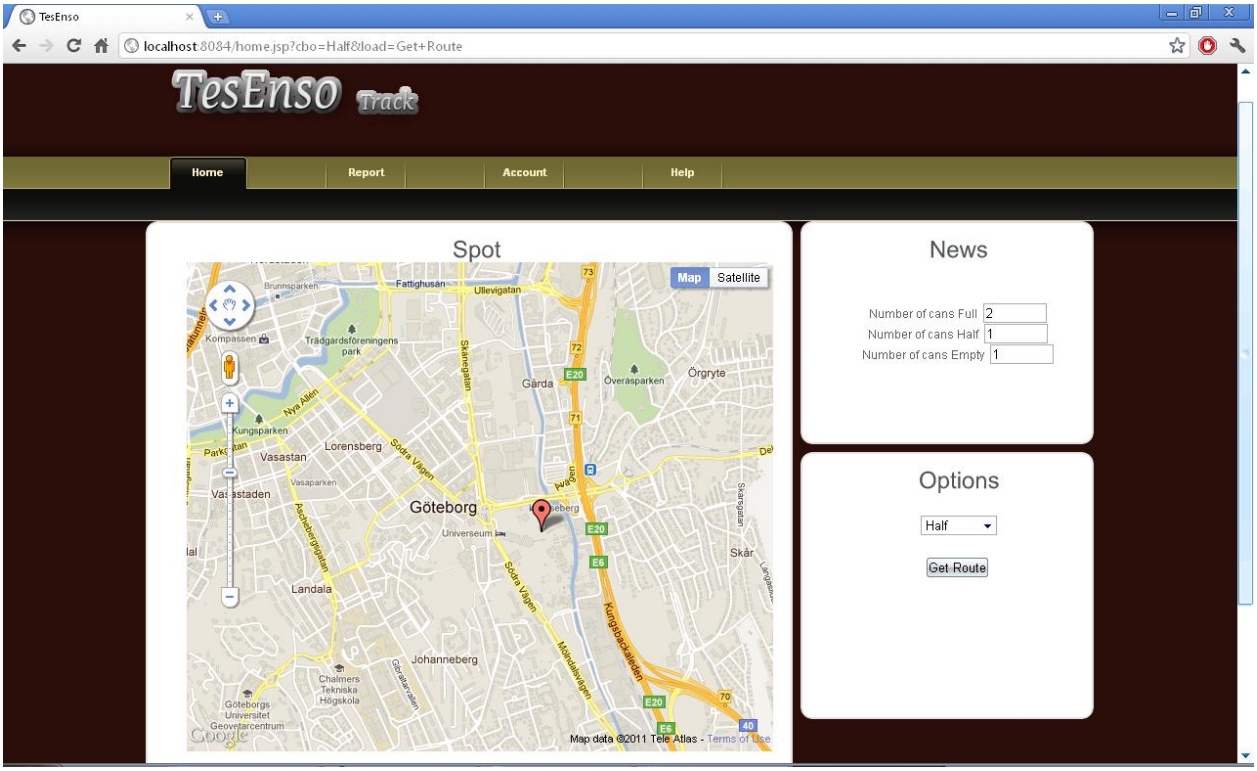

Figure 14: Customer home page displaying locations with close to full cans

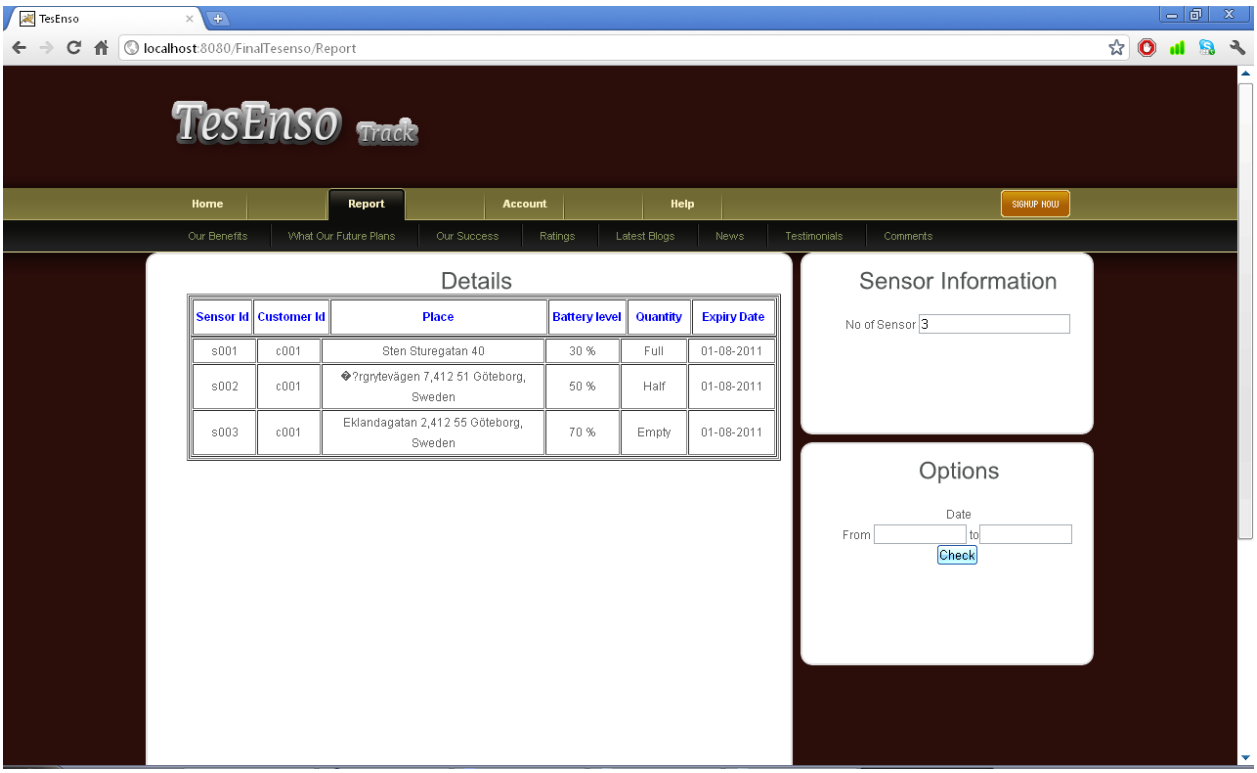

Figure 15: Customer report page displaying information related to sensors

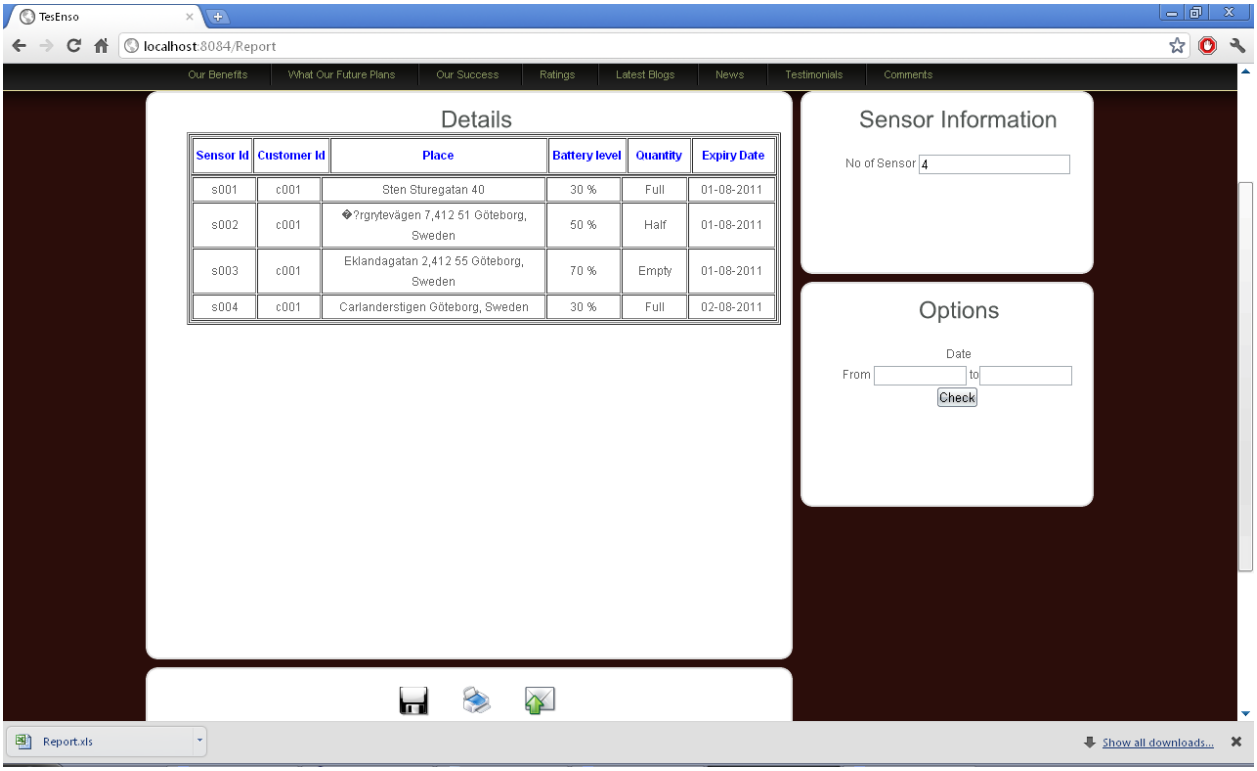

Figure 16: Save report

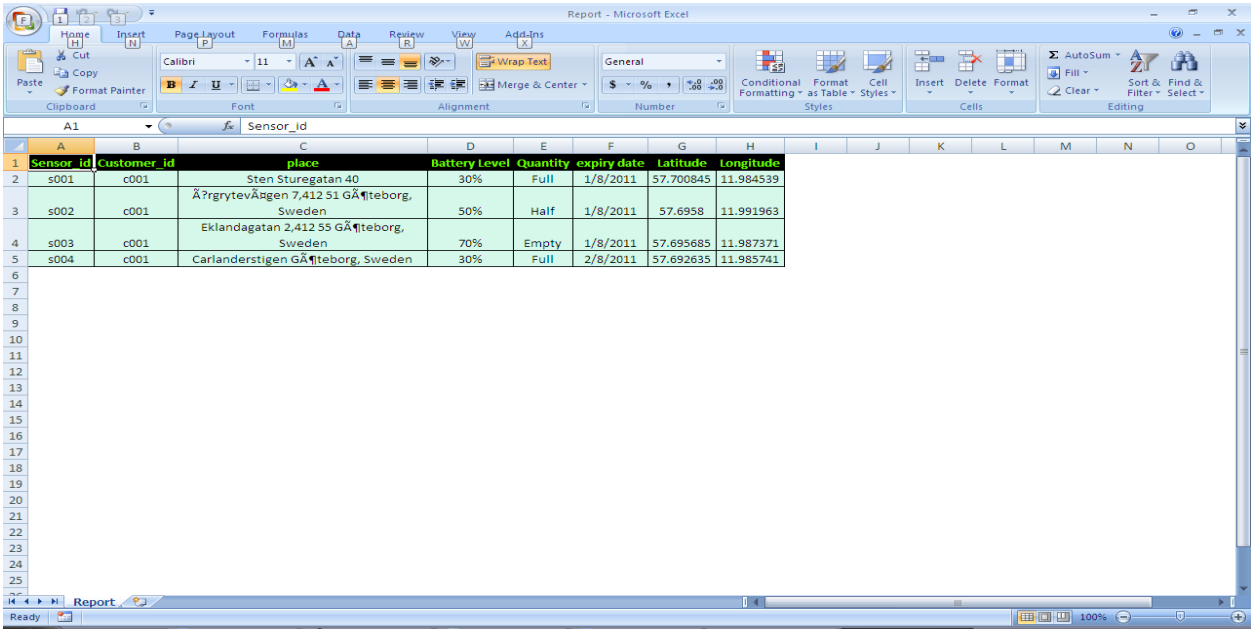

Figure 17: Downloaded report

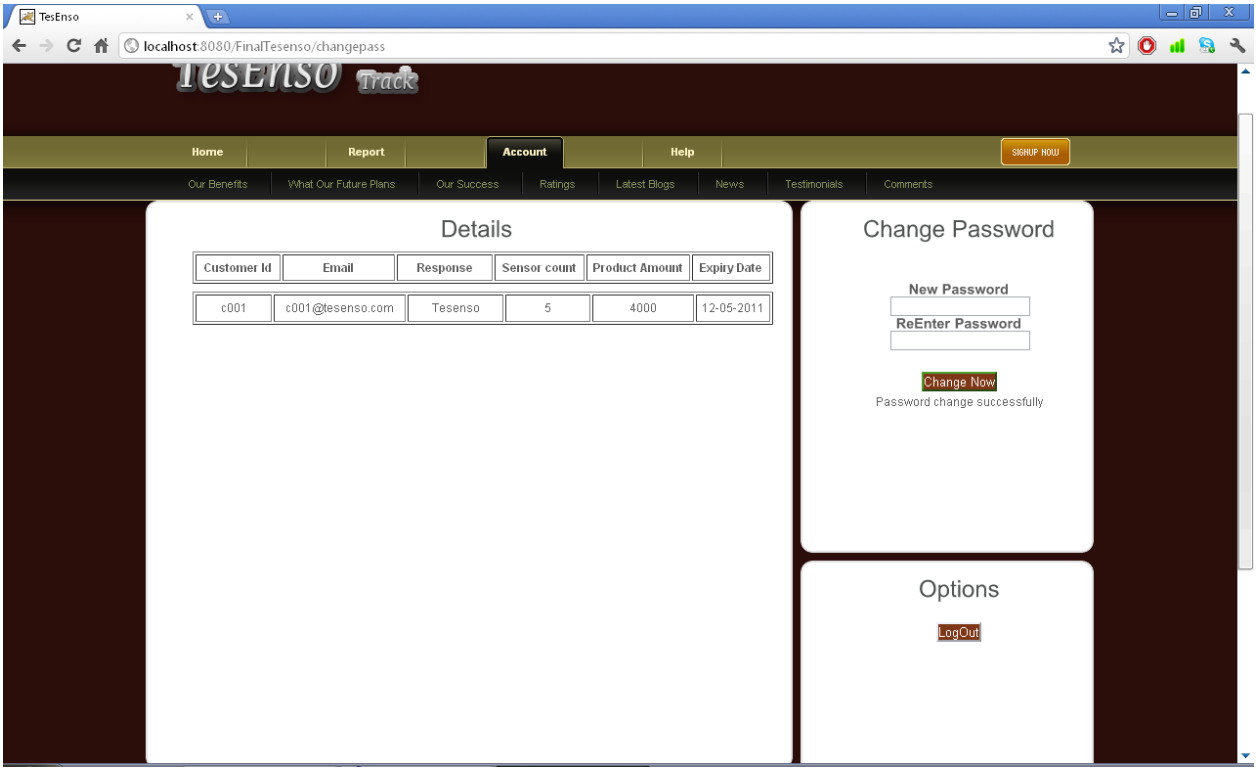

Figure 18: Customer account page