ARCADA

Theoretical and practical Requirements Engineering

Sebastian Lönnfors

Degree Thesis

Information Technology

2012

DEGREE THESIS	
Arcada	
Degree Programme:	Information Technology
Identification number:	3667
Author:	Sebastian Lönnfors
Title:	Theoretical and practical Requirements Engineering
Supervisor (Arcada):	M.Sc. Magnus Westerlund
Commissioned by:	Elisa corporation

Abstract:
The objective of the thesis was to produce a software system design for Elisa Corporation that included a requirements list, requirement specification and an architectural design. The idea was to use the UML model for this purpose. The UML diagrams would later be used as a basis for agile system development.

The main focus of this thesis is requirements engineering and architectural design. Requirements engineering consists of two main processes: Requirements determination and requirements specification. In the requirements determination process the requirements are gathered, negotiated, analyzed and determined to provide a describing definition of the software system. In the requirements specification process the requirements are modeled into specification requirements, in this case using use cases which are one example of behavior diagrams of the UML model. In the architectural design process a solution strategy is used to define the client and server components of the system. Software development including generic phases and processes models are also presented on a general level.

The result of this thesis is a set of UML design diagrams of a fictive software system to provide an overview of the methods used. The process to determine and specify requirements is described and also the architectural design for the system is presented. UML diagrams produced are use cases and a deployment diagram. Use cases are also documented.

Keywords:	Requirements Engineering, Requirements determination, Requirements specification, Architectural Design, UML, Use Case, Deployment diagram, Elisa Corporation.
Number of pages:	45
Language:	English
Date of acceptance:	11.5.2012

EXAMENSARBETE	
Arcada	
Utbildningsprogram:	Informationsteknik
Identifikationsnummer:	3667
Författare:	Sebastian Lönnfors
Arbetets namn:	Teoretisk och praktisk Kravhantering
Handledare (Arcada):	M. Sc. Magnus Westerlund
Uppdragsgivare:	Elisa Abp

Sammandrag:
Avsikten med examensarbetet var att producera ett system för Elisa Abp vilken skulle innehålla en lista på de krav som skulle uppfyllas, en kravspecifikation och en design på arkitekturen. För att uppfylla syftet för arbetet skulle UML modellen användas. Dokument och diagram producerade med hjälp av UML modellen skulle i ett senare skede andändas som grund för agil systemutveckling.

Examensarbetet fokuserar huvudsakligen på kravhantering och design av arkitekturen. Kravhanteringen består av två huvudsakliga processer: Fastställande av krav och kravspecifikation. I processen var kraven fastställs, ingår insamling, förhandling, analysering och fastställning av krav för att leverera en beskrivande definition av systemet. I kravspecifikationsprocessen ingår skapandet av definitioner på hur de fastställda kraven skall realiseras, i det här fallet med UML modellens användarfall diagram och dokumentation. I processen var arkitekturen för systemet skapas ingår användningen av en lösningsstrategi för att definiera klient och server komponenter för systemet. Systemutveckling vilken inkluderar generiska faser och processmodeller är också generellt presenterade.

Resultatet av examensarbetet är en samling UML diagram och dokumentation av ett påhittat system för att skapa en blick över de metoder som användes. Processen för att fastställa och specificera krav samt arkitekturen för systemet är presenterad. De UML diagram som är producerade är användarfall diagram samt deployment diagram. Dessutom produceras dokumentation av användningsfallens diagram.

Nyckelord:	Requirements Engineering, Requirements determination, Requirements specification, Architectural Design, UML, Use Case, Deployment diagram, Elisa Corporation.
Sidantal:	45
Språk:	Engelska
Datum för godkännande:	11.5.2012

OPINNÄYTE	
Arcada	
Koulutusohjelma:	Informaatiotekniikka
Tunnistenumero:	3667
Tekijä:	Sebastian Lönnfors
Työn nimi:	Teoreettinen ja käytännön vaatimushallinta
Työn ohjaaja (Arcada):	M. Sc. Magnus Westerlund
Toimeksiantaja:	Elisa Oyj

Tiivistelmä:
Tässä opinnäytetyössä tavoitteena oli suunnitella ohjelmistojärjestelmä Elisa Oyj:lle. Suunnitelma sisältäisi listan järjestelmän vaatimuksista, käyttötapaukset ja suunnitelman järjestelmän arkkitehtuurista. Tavoitteena oli käyttää UML mallia tuottamaan kaavioita jotka myöhemmin voitaisiin hyödyntää ketterässä ohjelmistokehityksessä.

Opinnäytetyö keskittyy vaatimuksien hallintaan ja arkkitehtuurisuunnitteluun. Vaatimuksien hallinta sisältää kaksi pääprosessia: Vaatimuksien päättäminen ja vaatimuksien määrittäminen. Vaatimuksien päättämisen prosessissa vaatimukset kerätään, sovitaan, analysoidaan ja päätetään, tavoitteena luoda kuvaus järjestelmän vaatimuksista. Vaatimuksien määrittämisen prosessissa päätetyt vaatimukset muunnetaan määrittelyiksi, tässä tapauksessa käyttäen UML mallin käyttötapauksia. Arkkitehtuurisuunnittelun prosessissa ratkaisu strategiaa käytetään määrittelemään komponentteja järjestelmän arkkitehtuurissa. Järjestelmäkehitys sisältäen yleisiä vaiheita ja prosessimalleja esitetään myös yleisellä tasolla.

Opinnäytetyön tulos on sarja fiktiivisiä esimerkkejä UML kaavioista ja dokumentaatiosta joiden tarkoitus on kuvata käytetyt menetelmät järjestelmäkehityksessä. Prosessi vaatimusten päättämiseen ja määrittämiseen sekä arkkitehtuurisuunnitteluun on kuvattu.

Avainsanat:	Requirements Engineering, Requirements determination, Requirements specification, Architectural Design, UML, Use Case, Deployment diagram, Elisa Corporation.
Sivumäärä:	45
Kieli:	Englanti
Hyväksymispäivämäärä:	11.5.2012

CONTENTS

Figures

ABBREVATIONS

ICT	Information and Communications Technology
UML	Unified Modeling Language
XP	eXtreme Programming
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
OS	Operative System
GUI	Guided User Interface
SQL	Structured Query Language

ACKNOWLEDGMENTS

I would like to thank my teacher M. Sc. Magnus Westerlund at Arcada who has given me valuable feedback and guidance during the writing process. I would also like to thank my colleagues and project members at Elisa for the help and guidance finding the right solutions for the system. Especially I would like to thank M. Sc. Jani Sammalmaa for introducing me to the project and helping me to find a suitable topic for the thesis.

I also want to thank my family who have encouraged and supported me during my time at Arcada. Finally I would like to thank my friend Farzan Yazdani for the support, ideas and advices for my thesis.

Sebastian Lönnfors
Helsinki 4.5.2012

1 INTRODUCTION

This thesis focuses on the first steps of the system development process; Requirements determination, requirements specification and architectural design. Other phases are also mentioned to get a good understanding of the whole system development process.

The requirements determination phase is where the requirements are gathered and organized into proper groups, for example functional-, operational- performance- or delivery requirements. The outcome of the step is requirement documents were all the requirement statements are gathered.

The requirement specification phase is the phase were requirement document is modeled into use cases using the UML model. The use cases are discovered by the requirements in the requirements document and by discovering the actors and their role in the system. Discovery is done by asking questions about the system. The behavior of a system is then illustrated in use case diagrams where after the behavior is documented in use case documents.

The architectural design phase is the phase were basic structure of the system that identifies the key components and the communication between these components is designed. An often used method is the multi-tier client-server architecture, were the network load is distributed between groups of servers. The deployment diagram is the diagram were the system architecture is illustrated which displays the execution architecture and relationship between the nodes, devices and artifacts of the system.

1.1 Background

Elisa is a telecommunications and ICT service company. Elisa online and ICT services provide experiences and businesses with productivity.

Elisa serves approximately 2.2 million consumers, companies and public administration organizations. As the market leader in mobile subscriptions, Elisa offer the most comprehensive and fast 3G network and 4G speeds in Finland. In 2011, Elisa's revenue was €1.53 billion and employed 3,750 people. Elisa offer international services in partnership with Vodafone and Telenor. (Elisa Corporation, 2012)

1.2 Objective

In this thesis the goal was to produce a software system design for Elisa Corporation. The design includes a requirements list, requirement specification list including use cases and a specification of the architecture of the hardware system. The purpose is to make a design using UML modeling concepts which later on could be used when the agile system development would start to increase the iterative cycles and thereby get the product delivered faster with little or no change.

The questions answered in this thesis are: How are requirements determined? How are requirements specified? How is the architecture for a system designed?

The thesis is divided into a theoretical and a practical part.

1.3 Theoretical

The theoretical part examines the theory of software system development from a system developer's point of view with the goal to know how to follow the right methods when designing the system. It includes system development approaches and system development phases were it focuses on the requirement determination, requirement specification and the architectural design phase. The design follows the UML model were use case diagrams and documentation is studied followed by the deployment diagram theory.

1.4 Practical

The practical part of the thesis consists of requirements determination, requirements specification and architectural design phases using the UML model. The requirements are determined were after they are documented. The documented requirements are turned into use case diagrams and documented as recommended by the UML model. The architectural hardware system is designed into the deployment diagram. The system development follows the iterative and incremental methods and changes are made to the system several times during the process.

1.5 Limitations

This thesis will focus on the first steps in system development process; Requirements determination, specification and architectural design. The other steps are covered in the theoretical part to get a good understanding of the complete process. The first steps cover only the parts that the customer has requested to fulfill the requirements, the UML model with all the diagrams are for example not studied in this thesis.

2 SYSTEM DEVELOPMENT THEORY

System development includes the wide and complex processes were a new software product is developed. A product developed can also be a specific part for an existing system. The process is divided into phases and consists of the following main phases: (Sommerville, 2007:64)

- Requirement determination and specification
- Architectural design
- Development
- Testing, verification and validation
- Maintenance

2.1 Software security

Software security is important to keep in mind during the system development process. It's common with software failures which appear during the maintenance phase which in most cases causes inconvenience but not serious damage. Some system failures can in some circumstances result in economic loss, physical damage, or even threat to human life. Because of these it's important for the system to be secure and by that to hold the following main features: (Sommerville, 2007: 44-49)

- Availability, were services are delivered when requested.
- Reliability, were services are delivered as specified.
- Safety, were services are separated to avoid catastrophic failure.
- Security, were the system is protected from accidental or deliberate intrusion.

Also other properties are important for security; these include reparability, were the system can be repaired in a short time when a failover situation occurs, maintainability, were the software can be modified to operate with new requirements, survivability, were the system continues to deliver service while a part of the system is disabled and error tolerance, were user input errors are avoided and tolerated.

In system development it's important to know about the security problems. The most common security problems which exist in software are: Buffer overflows, were memory is allocated so that the system crashes which leads to loss of data or change in data which changes the behavior of the system. Unverified inputs, when unverified data acts on the behavior of the system. Unclear and vulnerable functions, when commands or functions which are not intended for the system may be used. Configuration problems, when configuration problems can make the system unstable.

2.2 Development approach

When a software system becomes larger and many people work together on the development, in form of projects, it's even more important that every person involved work on the system using a specific method. The method describes how a specific problem is solved step by step and shows how to act at every single step. (Wiktorin, 2003:28)

Depending on the system the software development methods varies from case to case. One approach to system development is the sequential and transformational structured method, were the development is assumed to go straight from the top to the bottom to deliver solutions that satisfies business functions. (Maciaszek, 2001:24)

The waterfall approach which follows transformational structured method is displayed in Figure 1.

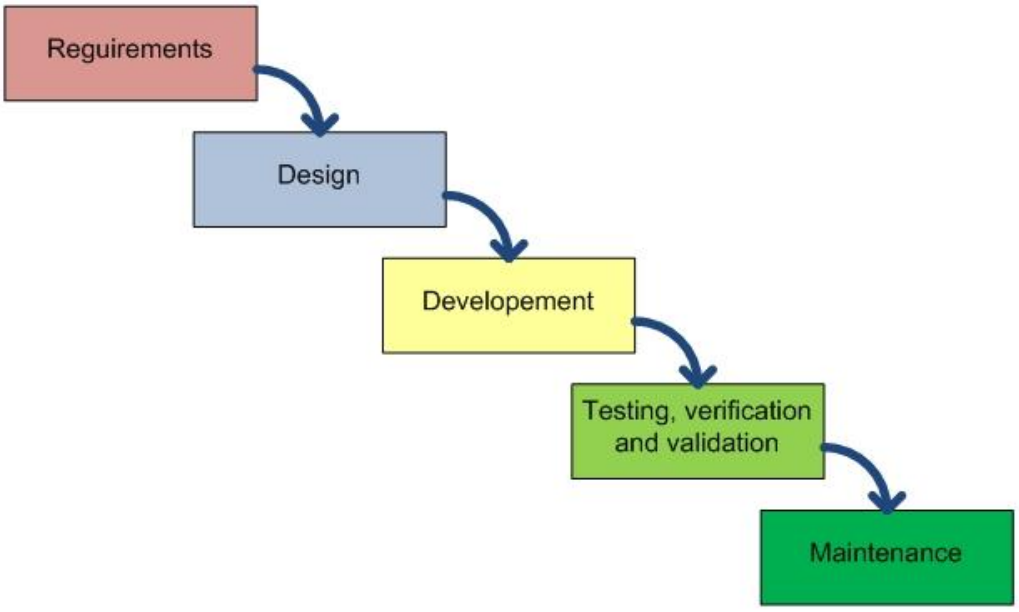

Figure 1, Waterfall model (Wikipedia, 2012)

Change during the software development process is in almost every case unavoidable. New technologies and business needs, often cause change in the system requirements (Sommerville, 2007:71). The material from the earlier phase can include errors or may be insufficient (Wiktorin, 2003:30). The need for change in design and implementation during the system development requires the process to be iterative. An example of the iterative development process is visualized in Figure 2 were the whole system development process is divided into different iterative phases.

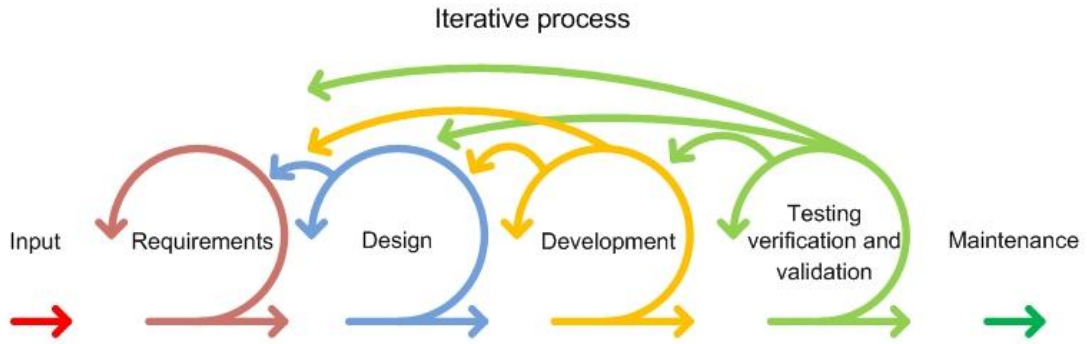

Figure 2, Iterative process

In software development it is possible to divide the software specification, design and implementation into different pieces were every piece represents a well separated increment of the system. The increments are each developed individually. (Sommerville, 2007:71)

By dividing the system into increments it is possible to deliver the product in several versions. First the most critical pieces of the system are processed and after that the other parts are processed as extensions to the original system. The advantages with incremental development are: (Witkorin, 2003:34-35)

- The increments are controllable and easier to review. It's possible in an earlier stage to determine misunderstandings and errors.
- The system can in an earlier stage be delivered in pieces to the customer to satisfy tight schedules.

An example of the incremental system development is represented in Figure 3 were the input is divided into three different pieces of increments, each developed individually and finally gathered to one final system.

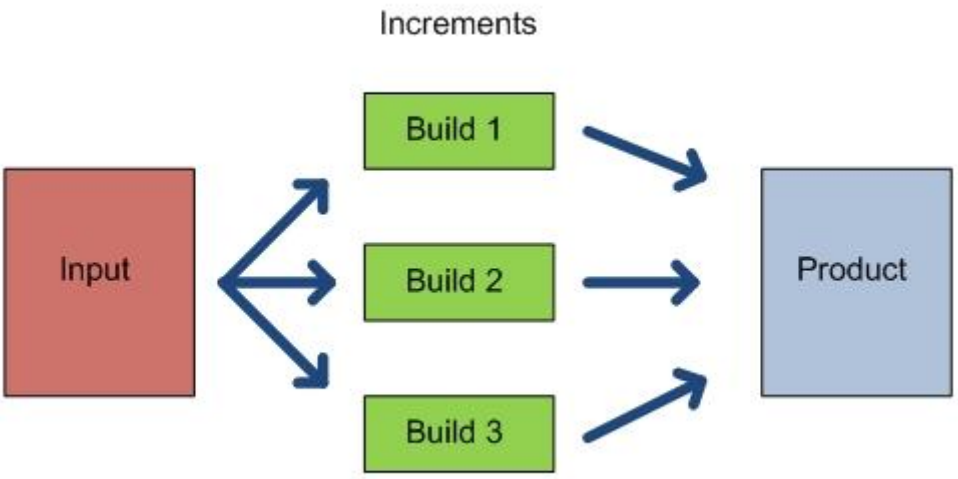

Figure 3, Incremental process

Another system development approach is the spiral model which follows the incremental and iterative design. In the spiral development model the design is following a spiral were all the system development phases are passed by several times. (Görling 2009:59-60)

2.3 Agile software development

Agile software development methods are iterative and incremental and different from traditional software development methods because they focus on the adjustment to change and the delivery of high quality systems using a simple process. Agility is to thin down the traditional software development to a lightweight process so that changing architectural environments, changes in user environments and changes in project timetables can be adjusted to the new situation. (Dingsøyr, Dybå, Moe 2010:15-16)

In agile software development, feedback is used and acquired in short loops to change the product to correspond to new needs so that a desirable product can be delivered to the customer. (Dingsøyr, Dybå, Moe 2010:15-16)

Many different agile software development methods exists which all varies from each other, some examples of these are XP, Scrum and Kanban.

2.4 UML

In system design different design models can be used to help to describe and visualize the system. A system development approach is the object oriented approach which follows the iterative and incremental process. The object oriented approach has got an approved standard known as UML (Unified Modeling language). It leads to better reusability of code and information, shorter development time, improved software quality and greater understandability. (Maciaszek, 2001:24-26)

Object-oriented modeling languages started to increase in the later 1980s and became more complex at the same time as the applications grew larger. Modeling became important, providing the blueprint of the system and being a central part of all the activities that lead up to the deployment of good software. Every system may be described in a different way using different models; each model is thereby an abstraction of the system. (Booch, Rumbaugh, and Jacobson, 1998:14-17)

The Unified Modeling Language was created by the Object Management Group as a standard to provide a modeling language for object-oriented software engineering. The UML models provide the following features: (Booch, Rumbaugh, Jacobson. 1998:17)

- Visualization, were the planned system is visually described.
- Specification, were the structure or behavior of a system is described.
- Template, were the template works as a guide for the system.
- Document, were decisions are documented.

In the UML model there are two main diagrams to provide graphical representation of a system. The diagrams contain symbols that represent the graphical elements in the UML model of system. The two main classes of diagrams are structure-and behavior diagrams. (Fakhroutdinov, 2010)

The static structure of the system and its parts on different generalizations and implementation and relation between them are presented in structure diagrams. The dynamic behavior of the objects in a system, illustrated as a series of changes to the system over time is presented in behavior diagrams. The hierarchical categorization of the UML diagrams is presented in Figure 4. (Fakhroutdinov, 2010)

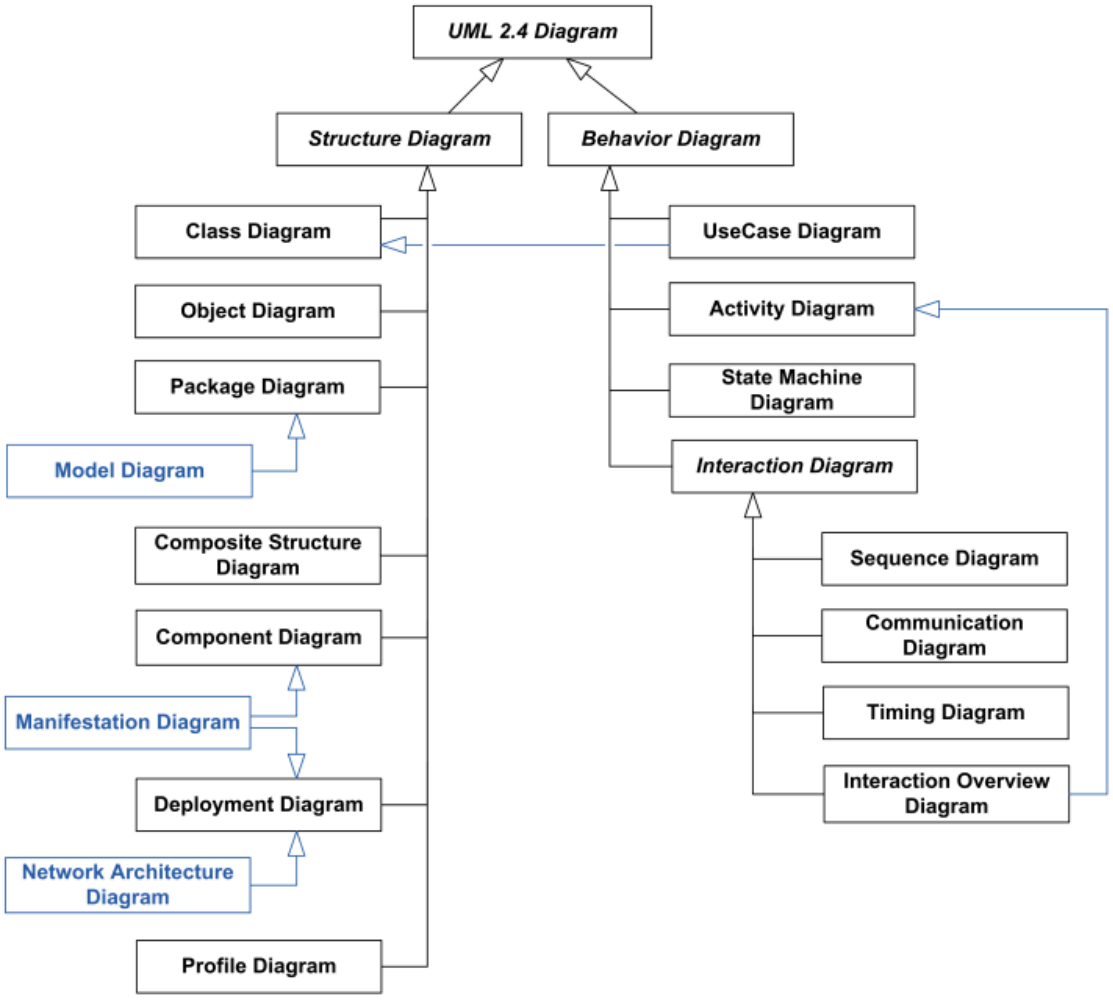

Figure 4, UML Diagrams (Fakhroutdinov, 2010)

2.5 Use case

Use cases are categorized under behavior diagrams in the UML model. The behavior of a system is described in use case diagrams. During analysis of system requirements the use cases are captured by focusing on the behavior of the system. The behavior of the system can be used to specify how it is going to be implemented during design of the use case views. (Maciaszek, 2001:133-134)

A use case can represent different things. It can represent functionality, including main flow of logic, variations (sub flows) and exceptional conditions (alternative flows), also externally visible functionality (not an internal function) can be represented. It can represent orthogonal functionality, were use cases can share objects during executions but the execution of each use case is independent of the others. It can also represent functionality that brings an identifiable value to an actor and that value is achieved in a single use case. The functionality can also be started by an actor, but once started the use case can work together with other actors. (Maciaszek, 2001:134-135)

Use cases are discovered by analysis of the requirements, which are identified in the requirements document. Actors and their part in the system are also examined from the requirements. (Maciaszek, 2001:134-135)Functional requirements are used for discovering use cases. Use cases can be identified from the analysis of tasks performed by actors. It's suggested to ask questions about actors for use case discovery, the questions can be of the type: (Maciaszek, 2001:134-135)

- What are the main tasks performed by each actor?
- Is an actor able to access or modify information in the system?
- Can the system notify an actor about any changes in other systems?
- Should an actor be informed about unexpected changes in the system?

2.5.1 Use case diagram

The use case diagrams describe the act that a system should achieve in association with other users of the system. A diagram assigns use cases to actors which is the principal visualization technique for a behavioral model of the system. A use case diagram example is displayed in Figure 5. (Maciaszek, 2001:51)

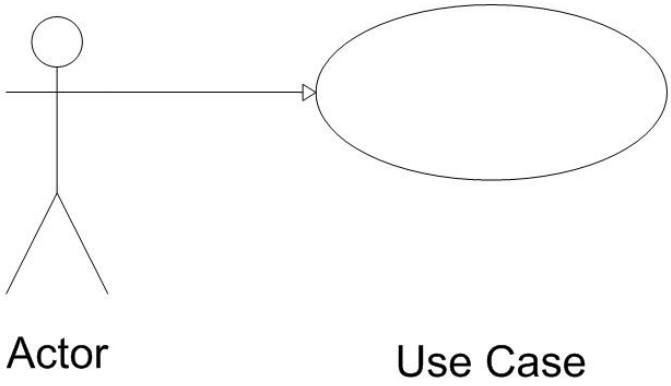

Figure 5, Use case example.

Use cases represent the interaction and the actors involved. Actors are represented as stick figures and each class of interaction (use case) is represented as a named eclipse. The use cases together represent all the possible interactions in the system requirements. (Sommerville, 2007:155)

Specification of use cases includes graphical presentation of actors, use cases and four different relationships. The association relationship establishes the communication path between an actor and a use case. The include relationship is an included use case that is necessary to complete the use case that activates it. The extend relationship extends a behavior of a use case by activating another use case at specific extension points. The use case generalization relationship can exist between two use cases or two actors that have commonalities in behavior, structure, and purpose. (Maciaszek, 2001:135-136)

Figure 6 illustrates the essential elements needed in use case notation.

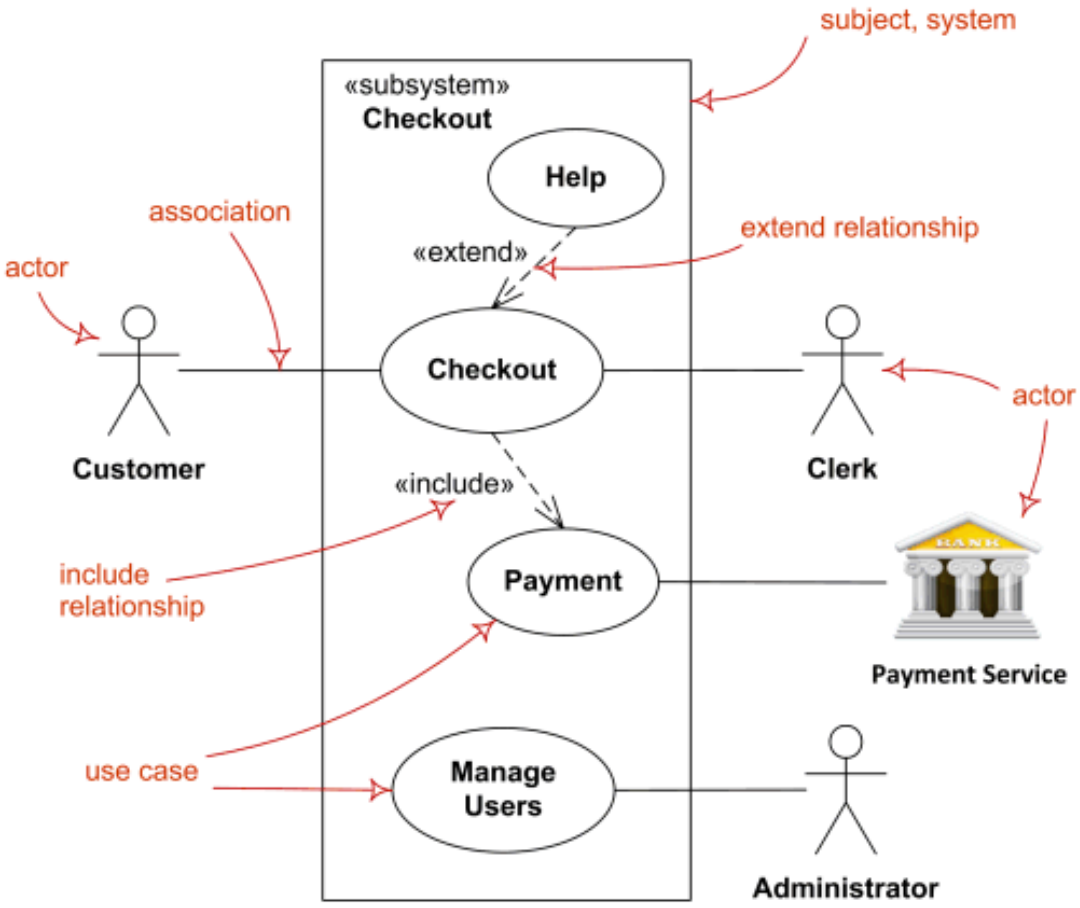

Figure 6, Major elements of the use case diagram (Fakhroutdinov, 2010)

2.5.2 Use case documentation

Graphical use case representation is only one part of the complete use case model. Each use case in the diagram has to be further described in a flow of events document. (Maciaszek, 2001:52)

The use case document describes the systems functionality when an actor triggers a use case. The structure of the document can vary, however it is recommended to contain some features. The structure should include a short description of the use case followed by participating actors and preconditions to be able start the use case. The use case document should also include a description of different types of events. These events are main course of events, sub course of events, alternative courses and post conditions. (Maciaszek, 2001:52-53)

An example of the use case document structure is displayed in Figure 7.

SYSTEM

Author (s): **Date:**
Version:

USE CASE NAME:		**USE CASE TYPE**
USE CASE ID:		Abstract: ☐
PRIORITY:		Extension: ☐
INVOKED BY:		
PARTICIPATING ACTORS:		
DESCRIPTION:		
PRE-CONDITION:		
TYPICAL AND ALTERNATE COURSES OF EVENTS:	**Step 1:** **Step 2:** **Step 3:**	
POST-CONDITION:		

Figure 7, Use case document.

Normally use cases changes several times during the development process. The first stage is to specify a short description of the requirements, after that the document is written iteratively step by step. The document is completed at the end of the requirement specification stage. At the end of the system development process the document will be used to create user documentation of the implemented system.

2.6 Architectural design

The architectural design is the design were the basic structure of the system that identifies the key components and the communication between these components is designed. Different architectural models can be used to design the architecture of the system.

The client - server architecture model presented in Figure 8 is used when designing system architecture. The model describes the relationship between programs in an application were the server offers service to one or more clients.

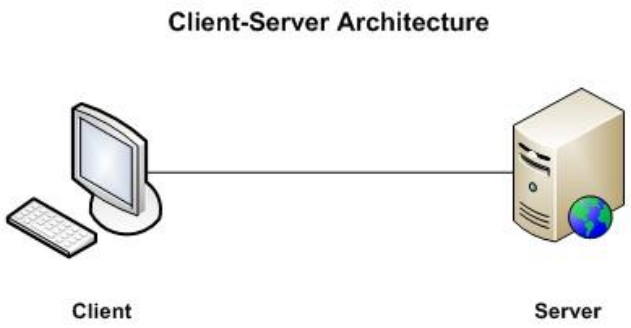

Figure 8, Client - Server architecture.

In a client - server architecture, a client browser displays the web pages delivered by the web server. The clients use HTTP to get web pages from the server. The web page can be scripted or it can include executable modules or objects. Scripted pages and applets can be downloaded and run within the browser. Additional functionality can be provided by objects such as ActiveX controls. (Sommerville, 2007:274)

The deployment design of the system must take security problems into consideration. Secure transfer and encryption methods and authentication methods ad further deployment demands to the system. The detailed design also consists of planning the network loads and backups.

The application server is often used when distributed objects are involved in the implementation. In many solutions the application server and web server is the same node.
The database server is used to store data, providing scalable storage and multiuser access.

An expanded model of the client-server architecture is the three-tier client-server architecture which is presented in Figure 9. The architecture is used were the presentation, application, processing and the data management are logically separate processes which execute on different processors. (Sommerville, 2007:273-274)

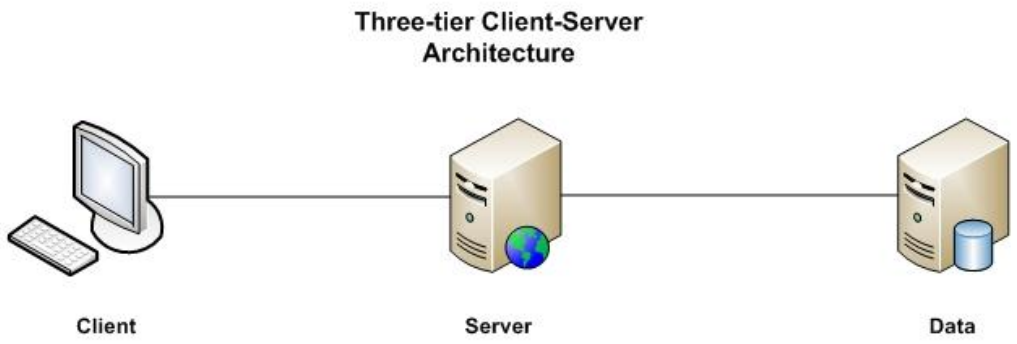

Figure 9, Three-tier Client-Server architecture.

In some systems it is appropriate to extend the three-tier server architecture model to a multi-tier alternative were additional servers are added to the system shown in Figure 10. This is done to distribute workload across multiple servers for load balancing to reduce network traffic to a single server. The architecture is normally divided into four different groups of nodes: (Sommerville, 2007:273-274)

- Client with browser
- Web server
- Application server
- Database server

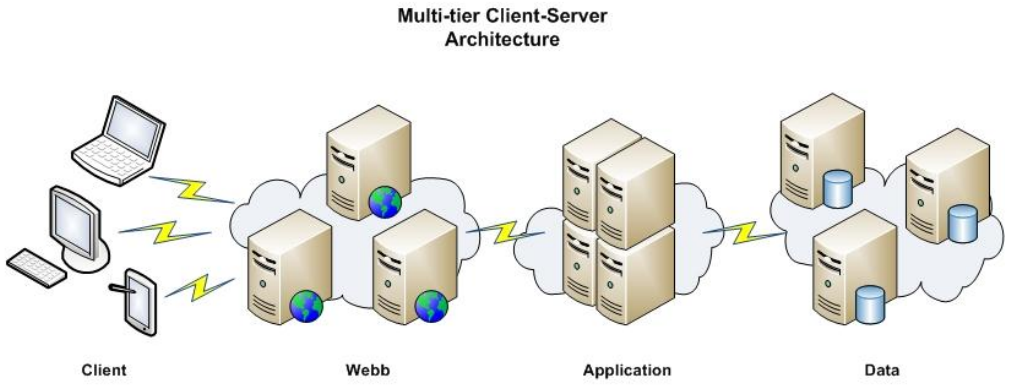

Figure 10, Multi-tier architecture.

2.6.1 Deployment diagram

In UML, system architecture is represented as deployment diagrams, which illustrate the execution architecture of the system. This includes nodes, either hardware or software execution environments, as well as the middleware connecting them. (Maciaszek, 2001:207)

The computational resource on which artifacts can be deployed for implementation is called a node and is graphically represented as a 3 dimensional cube as shown in Figure 11. The node has a memory and some computational capabilities. (Fakhroutdinov, 2010)

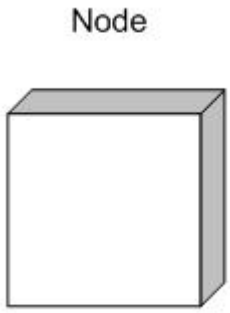

Figure 11, Node.

The software component (artifact) is a part of the implementation or a software system and is graphically represented as a class rectangle often with the keyword «artifact» as illustrated in Figure 12. (Fakhroutdinov, 2010) UML defines different types of artifacts which optionally can be displayed as an icon in the upper right corner. The artifact can be an executable, library, table, file, document, report and prototype.

Figure 12, Artifact.

A device represents a physical computational resource with processing capability and is presented as a 3-dimensional view of a cube as a node explained with keyword device, as displayed in Figure 13. On the device artifacts can be deployed. (Fakhroutdinov, 2010)

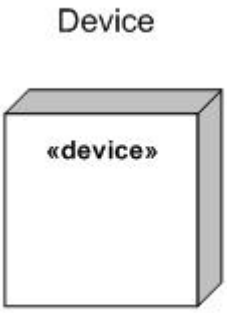

Figure 13, Device.

The deployment diagram illustrates the dependency relationship between the nodes, devices and artifacts. The connection relationship represents an association between the nodes as shown in Figure 14. (Fakhroutdinov, 2010)

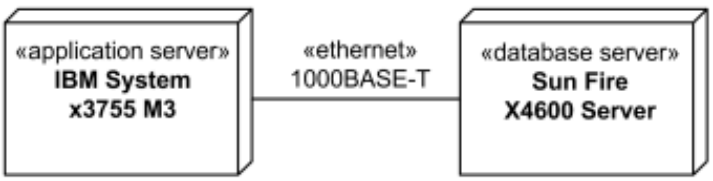

Figure 14, Communication path. (Fakhroutdinov, 2010)

A complete example of the deployment diagram is presented in figure 15.

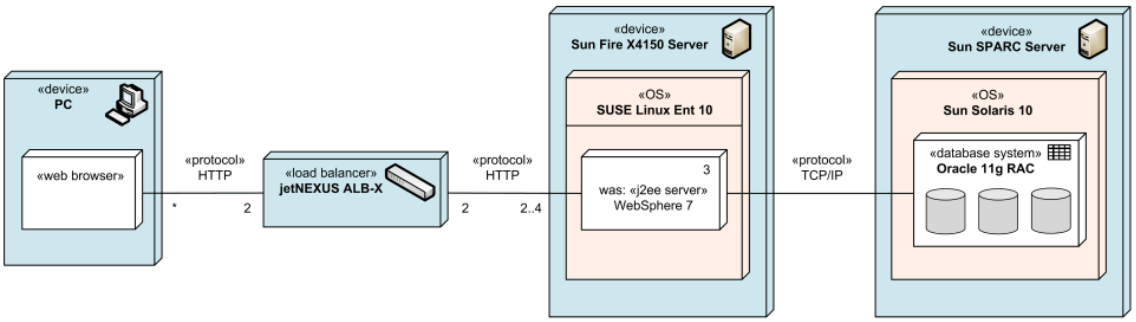

Figure 15, Deployment diagram example. (Fakhroutdinov, 2010)

3 DEVELOPMENT PROCESS

The system development activities vary depending on the method used. Some methods focuses on the practices of software development and others focus on management of the software product while some provide full coverage over the development lifecycle. Another variation between the processes is how much work is spent on the different phases.

The system development process can further be divided into the phases displayed in Figure 16. (Maciaszek, 2001:15-16)

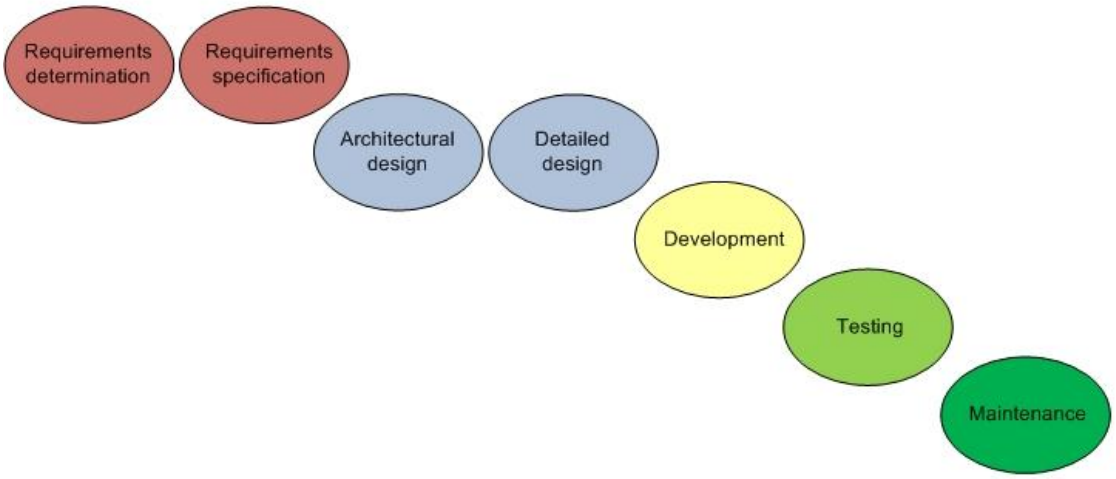

Figure 16, System development process

3.1 Requirements determination phase

Requirement determination is the first phase of the system development lifecycle. The purpose of requirement determination phase is to provide a describing definition of functional and other requirements that the customer expect to hold in the implemented and deployed system. The requirements define the expected services of the system and the rules that the services have to follow. (Maciaszek, 2001:80)

Requirements can consist of different types of requirements, which are: Functional requirements, external interfaces, functional relationship, operational requirements, quality for user experience, business requirements, performance requirements, security requirements, design requirements, competence of target group and delivery time and costs. (Wiktorin, 2003:40)

In this phase the requirements should be gathered, negotiated, analyzed and determined with the customer by a business or system analyst. (Maciaszek, 2001:80) The iterative software specification process includes four main activities. These activities are requirement discovery, requirement classification and organization, requirement prioritization and negotiation, and requirement documentation.

The techniques used for discovering requirements are interviews with the customers and other valid recourses, observations of business needs and activities, questionnaires, study of documents, forms and software systems. (Maciaszek, 2001:81) The requirements should be the best combination between manual and automatic operations. The fully or partly automated operations are the ones which should be gathered as requirements. (Wiktorin, 2003:40)

The idea to automate something usually done manually is a strategic decision that is affected by economical or resource savings considerations or then it's something that could be made just more effective or with more precision.

The requirements are organized into appropriate groups (Sommerville, 2007:148). The service statements can be grouped into those which define the scope of the system, the necessary business functions and the required data structures (Maciaszek, 2001:80).

The requirements are prioritized and conflicting requirements are found. Identification of restrictions on the system is included in the process. It also includes activities such as estimating the impact of change on other requirements and on the rest of the system. (Maciaszek, 2001:81) Errors at this stage leads to problems in later phases. (Sommerville, 2007:75)

The requirement document is the outcome of the requirements determination phase. The main body of the document consists of gathered requirements statements. The service statements can be divided into function requirements and data requirements. (Maciaszek, 2001:80) (Maciaszek, 2001:98)

3.2 Requirements specification phase

Definition of services which are required from the system takes place in the requirements specification phase. The determined requirements in the earlier phase are modeled into specification requirements by using particular methods (such as UML). (Maciaszek, 2001:17)

The requirement specification phase produces three different categories of models; state models, behavior models and state change models. Behavior specification models can be used to provide an operational view of the system. (Maciaszek, 2001:106) The most important method to use for behavior specification is use case diagrams. (Maciaszek, 2001:17)

Use cases identify the individual interactions between the actor and the system and represent a function often performed by an actor. They can be documented by text or linked to UML diagrams that develop the scenario in more detail. (Sommerville, 2007:155) Requirements need to be traced to use cases in the specification document (Maciaszek, 2001:134). The use case driven approach relies on the completeness and correctness of the use case to determine the classes. (Maciaszek, 2001:110-111)

Use case modeling is iterative and incremental. If user requirements change during the development process then the change is first made in the requirements document and then in the use case model. Changes to the use cases are then traced down to the other models. (Maciaszek, 2001:134) The outcome of the requirement specification phase is a list of all the requirements that should be fulfilled. (Maciaszek, 2001:17)

3.3 Architectural design phase

The architectural design phase is the phase where the system design begins either by planning a new system or by using an existing system. The phase includes specification of the hardware and software that the system is going to be implemented on. (Maciaszek, 2001:18)

The architectural design is the first stage in the actual design process and is an important link between the design and the requirements processes. The architectural design process is about establishing a basic structural framework that identifies the major components of a system and the communication between these components. The architecture defines the performance, robustness and maintainability of a system. (Sommerville, 2007:242)

The design may thereby depend on non-functional system requirements such as performance, security, safety, availability and maintainability. (Sommerville, 2007:242-243)
The design of the system is done by the conditions of the software and hardware platform on which the system is going to be implemented. In iterative and incremental system development the analysis models are constantly added with technical details. Once the technical details include hardware and software considerations then it becomes a design model. (Maciaszek, 2001:196)

The description of system modules is called an architectural design. The architectural design includes decisions about the solution strategies for the client and for the server components of the system. Detailed design is the description of the internal mechanism of each module (use case). In this phase complete algorithms and data structures for each module is designed.

The architectural design consists of choosing a solution strategy and the modularization of the system. The solution strategy needs to resolve the client and server issues as well as any middleware needed to combine the client and the server.
The client server architecture is a computing process were the client makes requests to the server process which services the client request. The client can access any number of servers in a system called distributed system.

Strategies of software reuse are defined in UML. The class, component or solution idea can be reused. The solution idea can be reused by using analysis and design patterns. It is used when objects or some specific parts represent good development practices and has been shown to work well in a number of situations. (Maciaszek, 2001:201-203)

3.4 Detailed design phase

In this phase detailed algorithms and data structures are developed for each module which were designed in the earlier architectural design phase. (Maciaszek, 2001:18-19)
The design provides a detailed specification for each component, describing interfaces and functions provided by each component. This detailed design will serve as the basis for the development phase.

3.5 Development phase

This phase is also known as the implementation phase and describes the actual development of the software system. The system is often developed in different increments were the system is divided into different modules., This is so that the system can for example be released in different revisions, with the most important functionality for the system implemented in the first version. Finally the different modules are integrated with each other. (Maciaszek, 2001:19-20)

3.6 Testing phase

Testing occurs continuously, but more effort is done at the end of the process. The purpose with testing is to find problems in the program so early in the system development process as possible.

Tests can be divided into two main groups, the manual tests and the automatically performed tests. An example of automatic testing is so called smoke testing were the tests are performed when the versioning takes place. Manual tests which are designed, analyzed, documented and maintained manually can be automated after they are created.

There are different types of tests, these are: Test of specification, were the specification is tested. Exploratory testing, were the tests are planned at the same time as they are accomplished. Unit tests were automatic code testing guarantee a consistent behavior. Stress testing, were the system is tested to complete its task without overloading. Acceptance testing, were the system is checked if it meets the customer's needs. Static code testing, were unused functions and loops are found. Domain testing, were different values are tested and were the output is verified. Usability testing, were it's tested how difficult it is for a common user to use the system.

3.7 Maintenance phase

The final phase is when the solution and system is handed over to the customer and maintenance of the system begins and continues until the software reaches the end of the lifecycle. (Maciaszek, 2001:20) The product is often developed in increments and features are thereby added later in new versions of the system.

4 PRACTICAL SYSTEM DEVELOPMENT

The goal of this thesis is to follow the first phases of the software development process, which are:

- Requirements determination
- Requirements specification
- Architectural design

By following these steps the goal is to fulfill the requirements of this thesis and provide Elisa Corporation with a documentation of all possible requirements and from the documentation create a requirement specification using the use case approach. The use cases would include documentation and should follow the UML modeling system to visualize the use case diagrams. Elisa Corporation also requires a documentation of the architecture of the hardware system. The architecture is required to be documented and visually presented using the UML deployment diagram.

The purpose is to make a design with the acquired models which later on can be used when the agile system development would start. By doing this the iterative cycles would be more effective and thereby get the product delivered faster with so little change as possible.

The method used for system development in this work is an implementation of the waterfall approach, the iterative approach and the incremental approach. The UML model is used to support the visualization, specification and documentation of the system.

4.1 Determination of requirements

Determination of the requirements is the first phase of the system development lifecycle. The project started by getting the members together for a meeting to set the guidelines for the system.

Observations of business needs in an earlier phase had proven that the market had a clear need of a system that could deliver all the requirements which was to be discovered. Discussions are carried out with the project group members, system administrators, system developers and others to discover all the requirements for the system.

The project group decides that the following requirements should be determined:

- Functional requirements
- Operational requirements
- Security requirements
- Performance requirements
- Delivery time requirements
- Architectural requirements

The requirements are prioritized so that the most critical and important functions have the highest priority and additional functions have a lower priority so that they could be added into a later version of the system.

The functional requirements includes management of different user levels, listing of options in the system and reports for example of changes made to the system. Operational requirements include maintainability and reliability. Performance requirements include availability for dependency. Architectural requirements include the system architecture requirements which affected on the architecture of the system.

Then the requirements are organized into groups by the functionality that they are providing. The restrictions on the system are also identified and considered through by the project group. The restrictions are made on the system because of the underlying

architecture. Also functional restrictions are identified which would provide unnecessary features that could be used for purposes that would not be intended for the system. Finally all the requirements are documented into the requirement specification document.

The requirement determination phase follows the iterative and incremental model and there are some changes made during the process. Several meetings occurred were the requirements are viewed and determined. The changes made during this phase are additions to the original requirements. Some additions acquired change in the later requirement specification.

Determined requirements from a login example are displayed bellow.

Functional requirements

The user must be able to login.
The user must be able to logout.
The user should be authenticated.
The user login information should be verified.

Operational requirements

The system should be able to be used from anywhere.
The system should be able to be used at any time.

Security requirements

The system should follow security regulations.

Performance requirements

The system should be able to perform with 500 simultaneous users.

Delivery time requirements

The system should be delivered in ten months.

Architectural requirements

The system architecture should be scalable.

4.2 Specification of requirements

When the requirements was determined and gathered into the requirements document, the next phase in the process was to specify the requirements. The phase included the modeling of determined requirements into specification requirements by using UML and to define the required system services. This is done using the customer requirements and constructing a specified list of the requirements to fulfill.

The use case approach is chosen from the UML model to provide the behavior specification of the system. The use cases are determined by the analysis of the functional requirements. The first step is to create use case diagrams and to document them. The use cases would then in the later development phase specify the system behavior development process.

4.2.1 Discovering use cases

To discover use cases the requirements document is used were the requirements are arranged into proper groups with keeping in consideration the use case approach already in this stage. The actors involved in the system are also taken into consideration so that the different groups of actors and their roles in the system are possible to identify.

Different user groups with different actor types were discovered. These actor groups have different roles in the system. By examining the requirements it became clear that there is a need to have the different user groups to restrict access to different functions of the system. This would add security and provide the possibility to display information in a way that was appropriate for the specific user group. Figure 17 is an example of two different actors as different user groups with a login use case example.

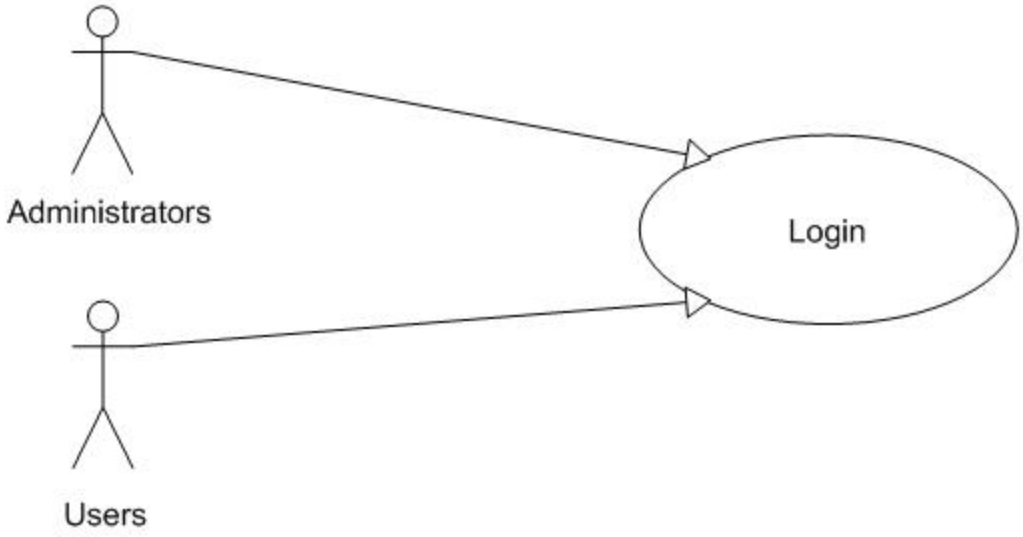

Figure 17, Actors.

The next step is to discover the actual use cases from the requirements specification document, questions to be asked to discover and to gather the use cases are:

- What task should the different groups of actors be able to perform?
- What information should the actors be able to view in the system?
- What information should the actors be able to modify in the system?
- What would be informed to the actor by the system?

By asking these questions the use cases started to get significance in the system and the first illustrations of the use case diagram emerged. In Figure 18, an example of all the use cases possible for the system is displayed.

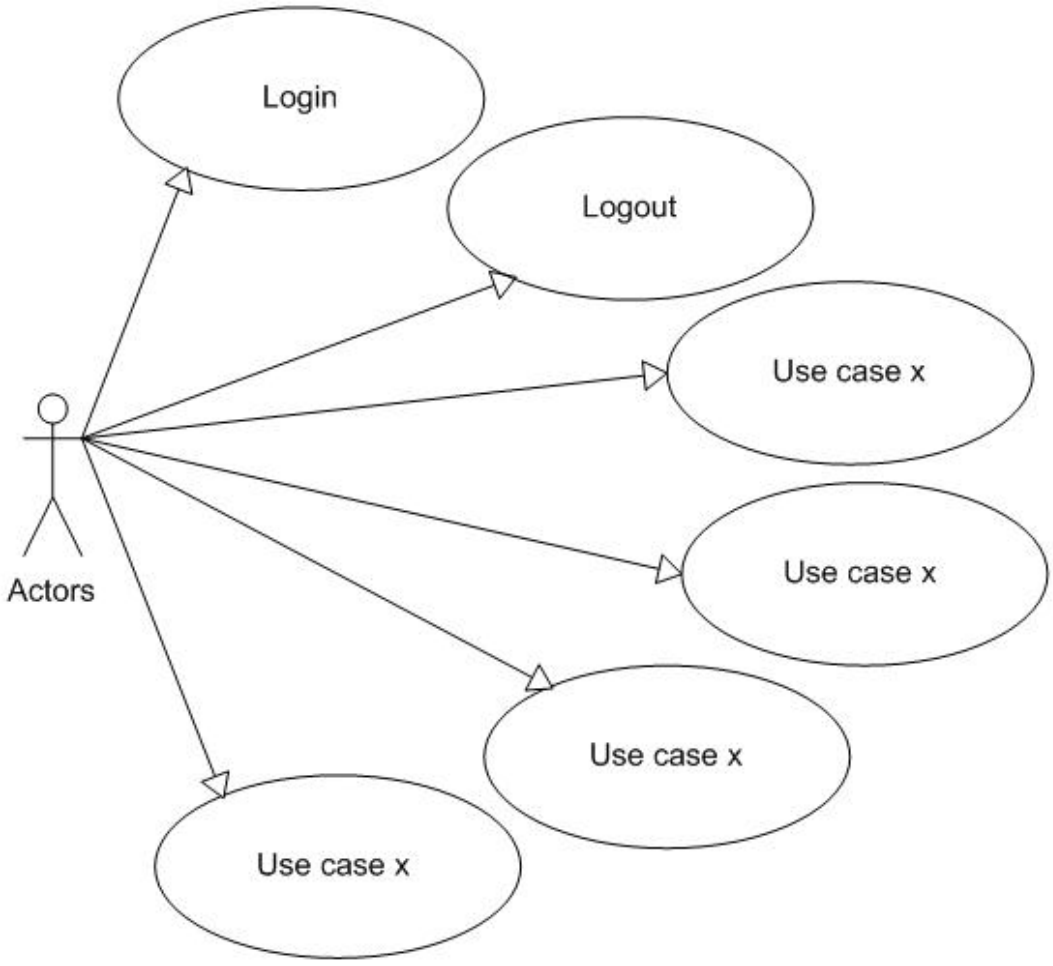

Figure 18, Use case illustration.

In the first version there are so many different use cases that they had to be merged so that the system would be understandable for a developer viewing it.

Later in the process there are changes made to the actor groups because of architectural restrictions. This impacted all the use cases created so much that changes had to be made.

4.2.2 Documenting use cases

In addition to the graphical representation of the use cases, the documentation is also needed to give a detailed step by step description of the use case.

A short description of the requirements is written, after that the document is written iteratively step by step. In Figure 19 a login process example of the use case documentation is displayed.

Example System

Author (s): Sebastian Lönnfors **Date: 14.03.2012**
Version: 0.1

USE CASE NAME:	Login	USE CASE TYPE
USE CASE ID:	1	Abstract: ☑
PRIORITY:	1	Extension: ☐
INVOKED BY:	-	
PARTICIPATING ACTORS:	Administrator User	
DESCRIPTION:	The example shows a normal login process.	
PRE-CONDITION:		
TYPICAL AND ALTERNATE COURSES OF EVENTS:	**Step 1**: The actor moves to the login page. The page welcomes the actor and displays a username and password field. A login button is also displayed. **Step 2**: The actor fills in the username and password and chooses login. **Step 3:** The username and password is verified. **Step 3a:** If the verfification is ok. The actor is moved to the system. **Step 3b:** If the verification fails. The actor is moved to Step1 with a error message "Wrong username or password".	
POST-CONDITION:	Use case x	

Figure 19, Use case example.

4.3 Designing the architecture

The next phase in the process is to create a design of the system. The plan is to create the solution for an existing hardware and software system of how the new system should be implemented. The architectural design of the system identified the major components of the system and the communication between the components.

The design of the system is done by the conditions of the software and hardware platform on which the system is going to be implemented. The architectural design include decisions about the solution strategies to resolve the client and server issues as well as middleware to combine the client and the server components to the system.

The solution chosen for the system is the multi-tier architecture, were the distributed client-server architecture is extended so that the presentation, application, processing and the data management are logically separate nodes. The architecture also provided the possibility to add additional servers to the system. The solution for the system architecture is decided to be used because of the need to reduce network traffic to a single server. The example in Figure 20 shows the multi-tier client server architecture for the example login system.

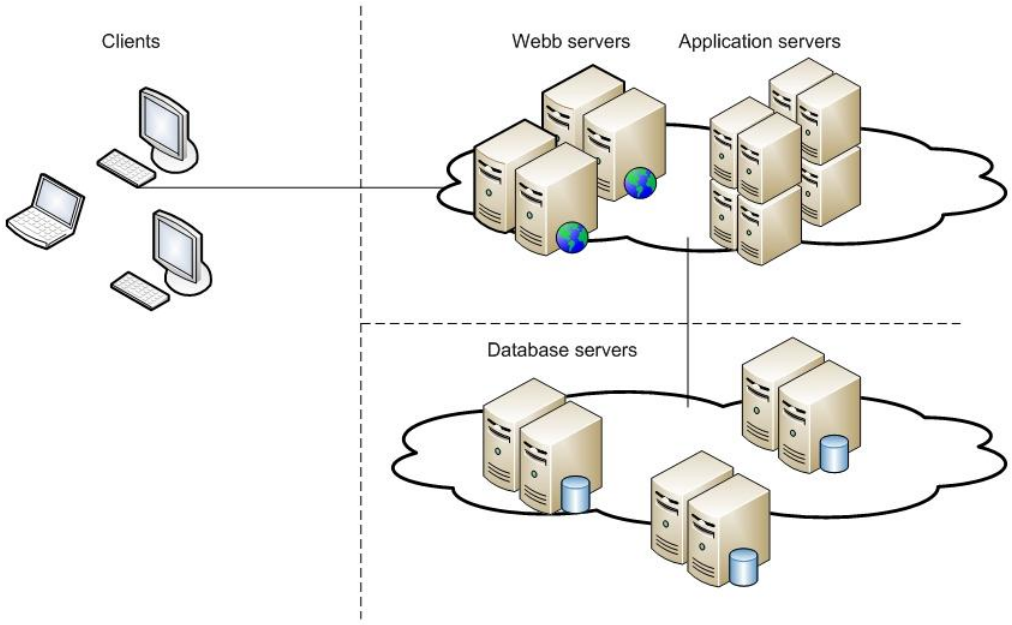

Figure 20, Multi-tier client-server architecture using the login example.

4.3.1 Deployment diagram

The system architecture is represented as a deployment diagram illustrating the execution architecture and relationship between the nodes, devices and artifacts of the system. (Maciaszek, 2001:207) The connection relationship between the nodes represents an association. (Fakhroutdinov, 2010)

Figure 21 displays the architecture for the login use case example. In the example the actor uses an optional device were a Web browser using the HTTPS protocol retrieves the login page from a server cluster device running the Linux operative system and Apache Webb server software. The server cluster communicates using the TCP/IP protocol with the database cluster to retrieve login information about the user.

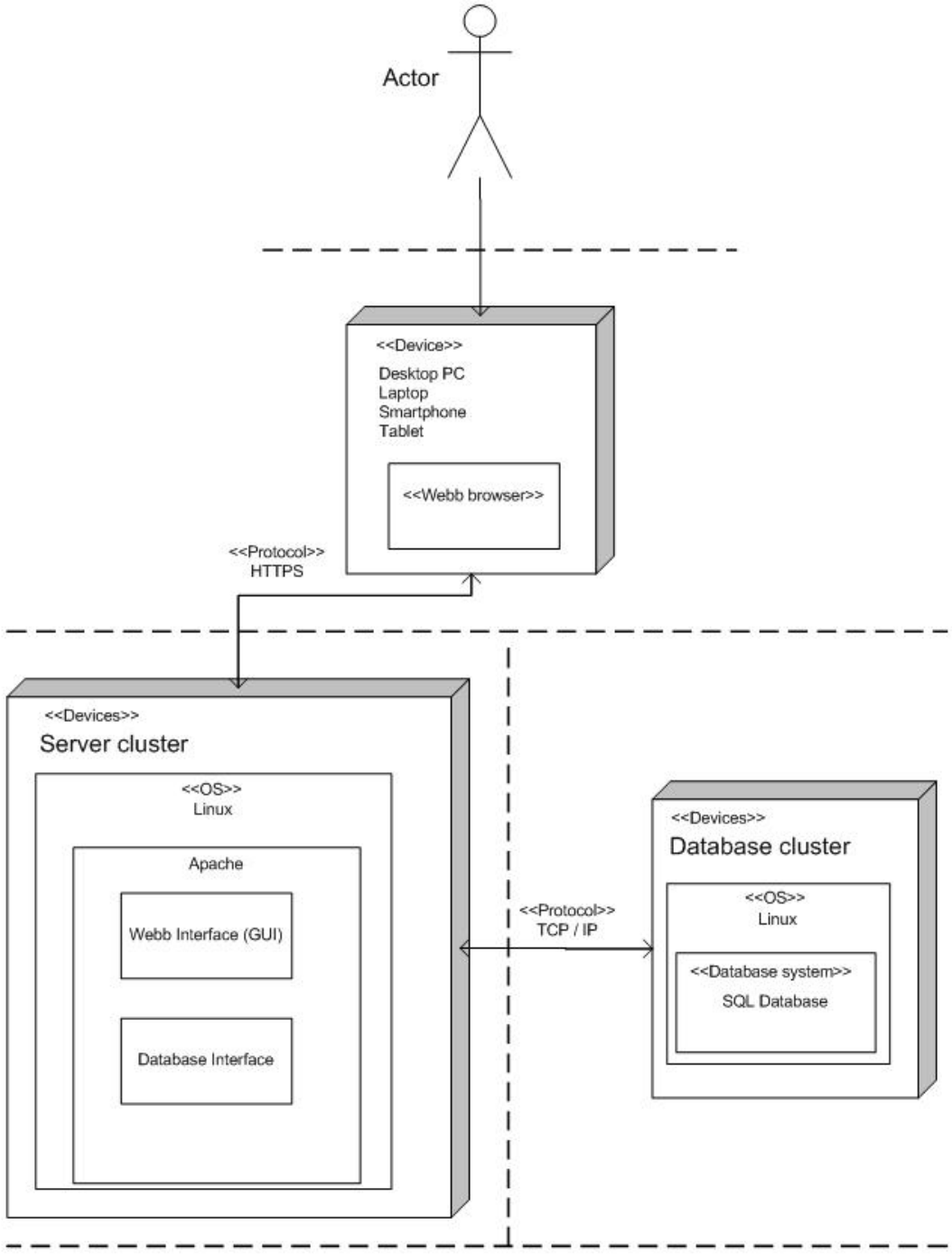

Figure 21, Deployment diagram using the login example.

By following the iterative and incremental system development the analysis models were constantly added with technical details.

5 DISCUSSION

5.1 Further development

The system provides good opportunities to be developed further. Some functions were excluded from the system due to a required integration with other systems. These integrations could be done at a later stage so that the functionality they were providing could be used by the system.

Some functionality was also excluded due to other systems which already were providing the same functionality to the underlying systems. It would be possible to integrate almost all of the functionality to the same system to make the management easier for a system administrator.

5.2 Conclusion

The development of the system went through many phases. In the beginning of the project the objective of the system was a little bit fuzzy. After a few meetings the goal started to get clearer when some of the functionality of the system was discovered and when the main architecture of the system was planned. There were many changes made to the system during the process. Almost every change made was an improvement in functionality but also some functionality had to be removed because of integration to other systems which had to be done at a later stage.

In the beginning of the process there was a challenge to find the right recourses and the material which was needed to design the system. This also brought many changes later which could have been avoided. In the end all the material necessary to complete the first steps in the system development process was found and the requirements set by Elisa Corporation fulfilled.

The outcome of the thesis was a requirement document, requirement specification and deployment diagram which where all the material that Elisa Corporation required. The material was created and handed over to Elisa as required.

By taking part in the project I now have a great understanding of different system development processes and how to design a system using the UML model. I also now have knowledge in agile system development methods and how a system is designed in a large company. Besides system development, I also now have knowledge in productification.

REFERENCES

Booch, Grady; Rumbaugh, James & Jacobson, Ivar. 1998: The Unified Modeling Language User Guide. Addison-Wesley. 512 p. ISBN 0-201-57168-4.

Dingsøyr, Torgeir; Dybå, Tore & Moe, Nils Brede. 2010: Agile Software Development: Current Research and Future Directions. 1st ed. Springer. 240 p. ISBN 978-3-642-12574-4

Elisa Corporation, 2012: On Elisa. [www] http://www.elisa.com/on-elisa/

Fakhroutdinov, Kirill 2010: The Unified Modeling Language (UML). Retreived 20.3.2012. [www] http://www.uml-diagrams.org/uml-24-diagrams.html

Görling, Stefan. 2009: Att arbeta med IT-projekt, Lund: Studentlitteratur AB, 308 p. ISBN 978-91-44-05223-6

Maciaszek, Leszek A. 2001: Requirements Analysis and System Design: Developing Information Systems with UML. Harlow: Addison-Wesley. 378 p. ISBN 0 201 70944 9

Sommerville, Ian. 2007: Software Engineering 8. 8 ed. Harlow: Pearson Education. 840 p. ISBN 10: 0-321-31379-8

Sparx Systems, 2012: UML 2 Tutorial. Retreived 10.3.2012. [www] http://www.sparxsystems.com.au/resources/uml2_tutorial/

Wikipedia, 2012: Waterfall model. Retreived 21.3.2012. [www] http://en.wikipedia.org/wiki/Waterfall_model

Wiktroin, Lars. 2003: Systemutveckling på 2000-talet, Sverige: Studentlitteratur AB, 276 p. ISBN 9789144031132.