CODEN:LUTEDX(TETS-5419)/1-83/(2001)&local 3

A System Dynamics Simulation Study of a Software Development Process

Carina Andersson
Lena Karlsson

March 2001

LUND INSTITUTE
OF TECHNOLOGY
Lund University

Advisors:
Martin Höst
Department of Communication Systems
Josef Nedstam
Department of Communication Systems
Bertil I Nilsson
Department of Industrial Management and Logistics

ERICSSON

Advisors:
Wladyslaw Bolanowski
Susanne S Nilsson

Abstract

In order to define software quality and set up quality goals, it is important to have a well-defined set of quality metrics on the software product. One necessary step to identify a relevant set of metrics is to understand the mechanisms in the software projects. Such an understanding can be achieved by building and simulating models of the software development processes.

The purpose of this thesis is to build and simulate a model of a software development process at Ericsson Mobile Communications AB to visualise the process mechanisms. The thesis aims at building up their competence regarding modelling and simulation of software development processes.

The study began with an analysis of the existing software processes at Ericsson Mobile Communications AB and continued with the development of a basic model of one of the processes. The model showed which variables that need to be more carefully regarded and measured. The model was finally simulated in a commercial simulation tool in order to visualise the mechanisms in the process. The simulations showed that allocating more resources to requirement and specification tasks would result in a shorter lead-time and a higher quality of the developed software than what is currently accomplished at Ericsson Mobile Communications AB.

Other results in the report are a theory section about modelling and simulation to enhance the reader's competence, and a description of possible future development of the model.

Table of Contents

1 Introduction

1.1 Background

Currently, a metrics programme is being launched at the software development organisation at Ericsson Mobile Communications AB (ECS). The objective of this metrics programme is to gradually define software quality and set up quality goals. In this process the understanding of mechanisms in the software project is a necessary step to identify a relevant set of metrics. Such an understanding has to be built up by monitoring parameters and results of the software projects – a procedure that takes a long time. Building models and simulating these models can speed up this procedure.

The constantly ongoing work with quality improvements at ECS aroused the interest to measure parameters of the software development processes. Thus, this report describes the model building and simulation in order to visualise which parameters that should be measured.

There are several other advantages of building models of software processes. By simulation some new knowledge might be gained, that can help ECS to improve the current processes. Building and simulating models also opens a new opportunity of visualisation of the mechanisms of the processes – it can be used for training and to enforce motivation for changes.

1.2 Objective

The overall goal is to build and simulate a software development process model. This will visualise different connections and relations that affect the quality in ECS's development projects. The thesis aims at increasing the competence at ECS on how to build models of software processes and how to simulate the models using a commercial tool available on the market.

In order to achieve this purpose a number of other tasks have to be carried out, which are included in the report. An overview of the area regarding process modelling and simulation and an investigation of the tools available on the market are presented. The tool used in the study is evaluated to survey its advantages in forthcoming work. The report includes guidelines for the forthcoming work on how to further develop the process model.

1.3 Outline

This report is divided into three parts:

Part I

This part contains the first four chapters, which gives an introduction to this report, and the necessary knowledge to understand the simulation study.

Chapter 1 gives the background to this simulation study. This chapter also contains the objective and purpose of this thesis and the outline of this report.

Chapter 2 gives an introduction to the discipline of Software engineering, and a brief description of the terminology that is used in this thesis.

Chapter 3 consists of an overview of modelling and simulation theory.

Chapter 4 describes the two processes that have been investigated during this simulation study, the product process and the platform process at ECS.

Part II

The second part consists of the two chapters that describe the main task of this thesis, namely to build and simulate a process model.

Chapter 5 includes the method used to perform the simulation study and the threats against the validity of the model.

Chapter 6 explains the execution of the simulation study.

Part III

The last part of the report describes and discusses the results of the simulation study.

Chapter 7 contains the results and analysis from the simulation.

Chapter 8 consists of the evaluation of the used simulation tool.

Chapter 9 is a discussion that includes a summary and recommendations for future work.

2 Software Engineering

This chapter is an introduction to the discipline of Software engineering, and gives a brief description of the terminology that is used in this thesis. The first section describes the goal, use, and purpose of the discipline, while the next section continues with the basic process activities used in software engineering. Then the waterfall model, and its process phases, is described, since it is used throughout this thesis. In order to get a well-functioning software engineering process it is necessary to continuously measure and evaluate the results of projects to improve the software quality and reduce the costs. In the third part of this chapter, a description of two metrics is included. These metrics are used to achieve the quantification of the software quality, which is necessary for the software quality improvement.

2.1 The Goal of Software Engineering

In the industrialised countries more and more products incorporate computers in some form. Educational, administrative, and health care systems are dependent on large computer systems. Software engineering is concerned with the theories, methods, and tools, which are needed to develop the software for these computers. In most cases, the software systems that must be developed are large, complex and always abstract, in that they do not have any physical form. Hence, they must be thoroughly documented in for example system designs and user manuals. In the last 30 years methods of software specification, design, and implementation have

been developed. New notations and tools have reduced the effort required to produce large and complex systems. However, many large software projects are still late, over-budget and result in poor software quality. Therefore software engineering is constantly of great importance. [1]

The goal of software engineering is to produce maintainable, dependable, efficient, and usable software. To achieve this goal it is necessary to have a well functioning *software process*.

2.2 The Software Process

Common to all software processes, there are four basic process activities [1]:

Software specification includes defining the functionality and constraints of the software.

Software development means producing the software according to the specification.

Software validation is performed to make sure that the market requirements are fulfilled.

Software evolution contains evolving the software to meet changing customer needs.

These four process activities are included in four general models: *evolutionary development, formal transformation*, s*ystem assembly from reusable components* and the *waterfall approach.*

In *evolutionary development* an initial system is rapidly developed from very abstract specifications. This system is then refined with customer input to produce a system, which satisfies the customers needs. *Formal transformation* is based on producing a formal mathematical system specification and transforming this specification to a program. *System assembly from reusable components* focuses on integrating the parts of the system that already exists rather than developing them from scratch. The *waterfall approach*, is used in the software development processes at ECS, and will be described below.

2.2.1 The Waterfall Model

The waterfall model takes the four basic process activities and represents them as separate process phases: requirement specification, software design, implementation, testing, and maintenance, see figure 2.1. After each stage is completed it is “signed-off” and development goes on to the following stage.

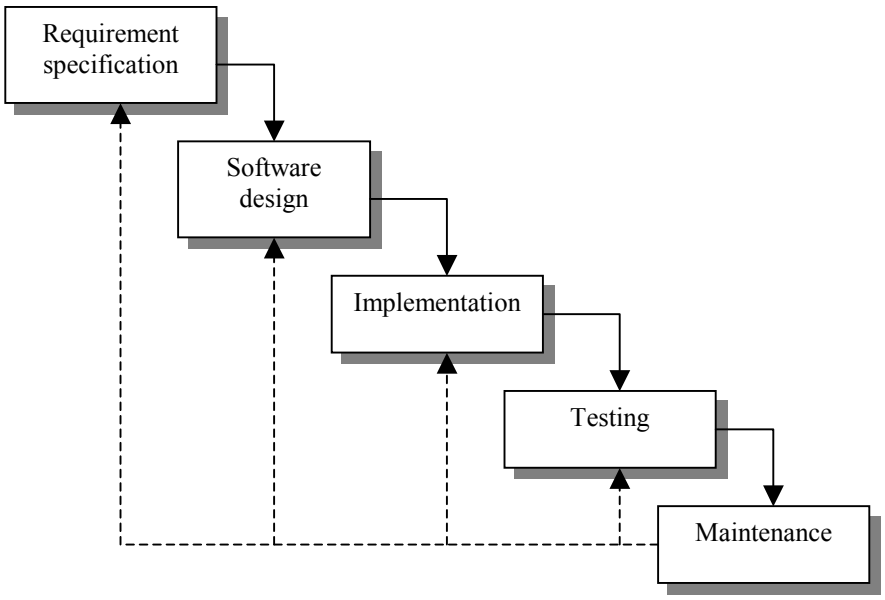


Figure 2.1 The process phases of the waterfall model

Requirement specification. The system's services, constraints and goals are established by consultation with system users, benchmarking and surveys of technology restraints.

Software design. The software design involves representing the software system functions in a form that may be transformed into one or more executable programs.

Implementation. During this phase, the software design is realised as a set of programs or program units.

Testing. The software is tested to ensure that the software requirements have been met.

Maintenance. This phase involves correcting errors, which were not discovered in earlier stages of the life cycle, improving the implementation and enhancing the system services as new requirements are discovered.

This software process is not a simple linear model but involves sequences of iteration of the development activities. Unfortunately, these iterations make it difficult to identify definite management checkpoints for planning and reporting. Therefore, after a small number of iterations, it is normal to freeze parts of the development, such as the specification, and to continue with the later development stages. This premature freezing of requirements, where problems are left for later resolution, may mean that the system will not do what the user requires. [1]

2.3 Software Metrics

A software metric is a term that embraces many activities, all of which involves some degree of software measurement. These measurements relate to a software system, process or related documentation. Examples of metrics are cost and effort estimation, reliability models, and performance evaluation. Examples of measures are size of a product in lines of code, the number of reported faults in a delivered software product and the number of person-days required to develop a system component.

Up till now metrics have been relatively little used in the software industry. Nevertheless, there is an increasing awareness that metrics have an important role in quality improvement, which in these days is leading to an increase in the industrial use of metrics.

Many of the metrics are used to estimate software costs. To do this a measurement for productivity, the number of units output divided by the number of hours input, is needed. The most commonly used measure of productivity is lines of source code per programmer-month. It is meaningless to compare productivity in this way across different programming languages, since the functionality, in the same number of lines of code, varies between languages. Thus, instead of measuring the number of lines of code, it is better to use a measure of the functionality. Several different function-based measures exist, but the best known is the *function point count.* [1]

2.3.1 Function Point Count

The total number of function points in a system or specification is computed by measuring or estimating the following program features: External inputs and outputs, User interactions, External interfaces, and Files used by the system [1].

In the simulation performed in this thesis the productivity is measured in function points, to be able to use the rules of thumb in [2].

At ECS one of the used programming languages is C, and according to [2] the number of function points (FP) is approximately equal to the number of C-statements (LOC) divided by 128. The rules of thumb are based on logical statements rather than physical lines [2].

$$FP \approx \frac{LOC}{128} \qquad \text{(Equation 2.1)}$$

2.3.2 COCOMO

One of the most widely used software cost and effort estimation models is the constructive cost model COCOMO. The effort will in COCOMO be expressed as follows:

$$E = a * S^b \qquad \text{(Equation 2.2)}$$

E is effort in person-months and S is size measured in thousands of lines of code (KLOC). The values of a and b, listed in table 2.1, depend on the type of software being constructed.

Table 2.1 Effort parameters for three modes of COCOMO

Mode	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

An organic system tends to use databases and focus on transactions and data retrieval, and can for example be a banking or accounting system. An embedded system contains real-time software that is an integral part of a larger, hardware-based system, and can for example be a missile guidance system. A semi-detached system is somewhere between organic and embedded.

With a similar COCOMO formula it is also possible to predict the duration of the project in calendar months. For further information see [3].

2.4 Conclusion

To be able to identify a relevant set of metrics it is necessary to understand the mechanisms of the software processes. Building models, and simulating these models, can make it easier to get this understanding. The next chapter describes the theory of model building and simulation.

3 Modelling and Simulation

The purpose of this chapter is to give an overview of modelling and simulation theory. It starts with a description of how to construct a model and the existing modelling methods. The chapter continues with the advantages of simulating these models. From chapter 3.5 only the concepts of *system dynamics* is described as this is the method used in this thesis, see chapter 6.2. The rules and schematic convention of system dynamics are described. In the end of the chapter there is a review of some simulation tools used for system dynamics.

3.1 Modelling Concepts

A *model* is an abstraction of a real object or system, and modelling a system means capturing and abstracting the system's components, relationships and behaviour, according to the model's goal. [4]

Simulation can help to gain insight into the behaviour of software development processes. Creating a model of the process and then simulating the behaviour of the process over time allows us to understand and predict process behaviour. Modelling and simulation can produce tools tailored to the profile of an organisation to manage the live process and to predict and improve performance.

3.2 Constructing a Model

The first step in a simulation study is to develop a model representing the system to be investigated. A suggested way is to reduce the real system to a logical flow diagram, where the components themselves may be broken down into sub-components, and so on. The system is broken down into a set of elements for which operating rules may be given. After specifying these rules and logical linkages it is necessary to test the model. This test can be done by performing a gross version of the simulation on a calculator, and checking whether each input is received from the right source and whether each output is acceptable. This is done stepwise, where the time step is a value specifying the time increment for each step of the simulation. [5]

The typical simulation model consists of a high number of elements, rules and logical linkages. Therefore, after the individual components have been tested, it is important to test the validity of the model to reasonably predict the behaviour of the system being simulated. The output data from the model should be compared with the corresponding performance data from the real system. If the real system has not already been in operation and the model is intended to simulate operating policies for a proposed system, for which no available data exist, the only way to validate the overall model is to have knowledgeable people carefully check the credibility of output data for a variety of situations.

3.3 Modelling Methods

There are three basic kinds of modelling methods: *analytical*, *continuous* and *discrete* modelling.

The analytical model provides average data on process behaviour and is often used in the software community. An example is the COCOMO model that is used for estimating schedule and effort for a given software product. These analytical models do not consider dynamic interaction between factors inherent in the process, neither are they possible to simulate. [6]

More detailed and realistic predictions of the process behaviour require more sophisticated models, generally based on simulation techniques. Such techniques are either discrete or continuous or a combination of these, which result in a *hybrid* model. [7]

Continuous model
The continuous type of simulation technique is based on *system dynamics* and is mostly used to model the project environment. This technique is useful when controlling systems, with dynamic variables, that change over time. Examples of these variables are productivity and defect detection rates. The continuous model, which is a *qualitative* model, represents the interaction between project factors as a set of differential equations. Integrating these equations over time describes the behaviour of project variables such as staff levels, motivation, resource consumption and the number of detected errors. Both the *qualitative* and the *quantitative* models

result in numerical data. But while the results from the qualitative model only should be interpreted as tendencies of an increase or a decrease, the results from the quantitative model can be interpreted numerically.

The system dynamics model describes the system in terms of *flows,* for example the error generation rate, that accumulate in various *levels,* for example the number of errors. A system dynamics simulation can model the continuous change in, for example, productivity, resource constraints and schedule pressure, as the project progresses. This kind of simulation model is a difficult way to describe discrete process steps since the nature of the simulation tools, used for continuous modelling, implies that all levels change at every time interval. If the process contains sequential activities, some mechanism must be added to prevent all activities from executing at once. This is the case when for example all design work has to be completed before coding starts. Examples of system dynamics tools are Dynamo, ithink (Stella) and Powersim. [6, 8]

Discrete model
In the discrete model, time is advanced because of a discrete event. This means that continuously changing variables only are updated at the event times. This can cause problems in the integration of the continuous variables or may create instability in the behaviour of feedback loops. It is also difficult to use dynamic relations in the simulation model. The discrete model requires a large amount of detailed information in order to give valid numerical results that are required in a *quantitative* model. Because discrete models are based on the idea of sequence of activities, it may be hard to represent simultaneous activities. To capture this in a discrete model it is necessary to model sub-components so that each component can be in only one activity at a time. Discrete event modelling can be used on a queuing network, which represents the component activities, their interactions and the exchanged artefacts. ProModel, GPSS and SIMAN are examples of discrete-event simulation tools. [6, 8]

3.4 Advantages with Simulation

Understanding a system's behaviour and the parameters that affect performance is vital to a company's management. Many of the inner mechanisms of the system are revealed during a simulation study. The studies performed before modelling often result in a detailed understanding of the system. Visualisation of the model then adds even more understanding of system behaviour. Further more, a simulation can be shown and explained to others in the organisation. [9]

Some of the advantages with simulation are that:

- simulation helps to understand the complex nature of dynamic systems and can be used for training.

- mathematical models, by themselves, can not describe most complex systems, with stochastic elements.

- experimenting with a system itself is often too expensive, lengthy or impossible.

- simulation allows studies of a system over a long time period since time is compressed.

3.5 The Rules of System Dynamics

Systems thinking is the base of all system dynamics simulations. It is a way of thinking about, and describing the forces and inter-relationships that shape the behaviour of systems. This discipline helps to see how to change systems more effectively. [10]

System dynamics simulations are based on the principle of *cause and effect*, *feedback*, and *delay*. *Cause and effect* is a simple idea, but some simulations based on methodologies other than system dynamics do not use it. The idea is that actions and decisions have consequences, for example price affects sales, and births affect the size of a population. When examining these cause and effect relationships isolated, they are usually very easy to understand. However, when they are combined into long chains of cause and effect, they can become complex. *Feedback* is the process in which an action taken by a person or thing will eventually affect that person or thing. A feedback *loop* is a closed sequence of causes and effects, a closed path of action and information. An interconnected set of feedback loops is a feedback *system.* [11]

Causal-loop diagrams can be created and are often used in system dynamics to illustrate cause and effect relationships. In such diagrams arrows are used to indicate the relationships. Sometimes, information about the way in which the relationship works is also included in the diagram. One way of showing this is by adding an “o” in the diagram, which implies a “change in the opposite direction”. The relationship between price and sales is such a relationship, where an increase in price leads to a decrease in sales. The relationship between births and population is described by another character. When births increase, so does the population. This is a situation where a change leads to a “change in the same direction”. Adding an “s” to the arrow in the diagram indicates this. Figure 3.1 shows a simple causal-loop diagram where price has a negative effect on sales, which in turn has a negative effect on unit costs, which in turn has a positive effect on price.

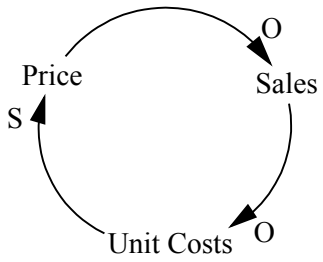


Figure 3.1 Causal-loop diagram illustrating connections between price, sales, and unit costs.

Not all cause and effect relationships occur instantaneously. Sometimes the consequences of an action or decision are not apparent until several days, months, or even years after an event has taken place. It is difficult to understand a system when the consequences can not be seen close to the behaviour. *Delays* can produce interesting and complex behaviour in systems even when those systems have no feedback and limited cause and effect complexity.

Understanding the concepts of cause and effect, feedback loops, and delays provides a good foundation when beginning to uncover the complexity of a system's nature. However, feedback loops alone do not indicate what the entire system's behaviour will be since the system is affected by many other variables. It is hard to anticipate the behaviour of a system from a causal-loop diagram alone, but such diagrams are useful when it comes to isolating the feedback structures.

3.6 System Dynamic Schematic Convention

From a system dynamics perspective all systems are, in a simulation tool, represented in terms of *level* and *rate* variables, with *auxiliary* variables used for added clarity and simplicity. A level is an accumulation, or integration, over time of flows or changes that come into and go out of the level. The term level is intended to invoke the image of the level of a liquid accumulating in a container. The flows increasing and decreasing the levels are called rates. Thus, a manpower pool would be a level of people that is increased by the rate of hiring and decreased by the rate of firing and quitting. Rates and levels are represented as stylised valves and tubs, as shown in figure 3.2. Flows will always originate somewhere and terminate somewhere. Sometimes, the origin of a flow is treated as essentially limitless, or at least outside the model-builder's concerns. In such a case the flow's origin is called a *source*. Similarly, when the destination of a flow is not of interest, it is called a *sink*. For example, for a level of workforce these symbols represent where people come from when they are hired and where they go after leaving the project. Both sources and sinks are shown as little clouds, as illustrated in figure 3.2. [12]

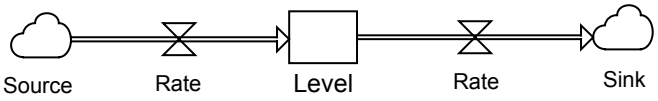


Figure 3.2 Representation of rates and levels

Tangible variables are either levels or rates, which means that they are either accumulations of previous flows or are presently flowing. Usually, however, it is difficult to write a rate equation without doing some computation, which is done with auxiliaries. Auxiliary variables are combinations of information inputs and are represented by a circular symbol, as shown in figure 3.3. A few other symbols complete the designation of items included in formal system dynamics diagrams. In addition to the variable symbols shown above, models also include constant terms, which are parameters of the model whose values are assumed to be unchanging throughout a particular computer simulation. The simple arrow symbolises an information flow while the double arrow represents a physical flow, for example people or software. The Rate in figure 3.3 can for example be a total productivity, which in this case is the product of the number of people, the Constant, and their individual productivity, the Auxiliary.

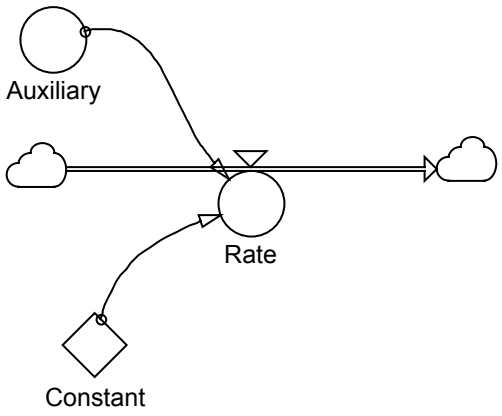


Figure 3.3 Representation of auxiliary and constant variables

In a simulation tool the variables - rates, auxiliaries, levels, flows and constants - are selected from the toolbar, positioned in the workspace, and connected with arrows. For each variable a number or equation has to be defined. In Powersim, for example, this is made in the *Definition box* in the

Property dialog box. Units for all variables are defined as can be seen in figure 3.4.

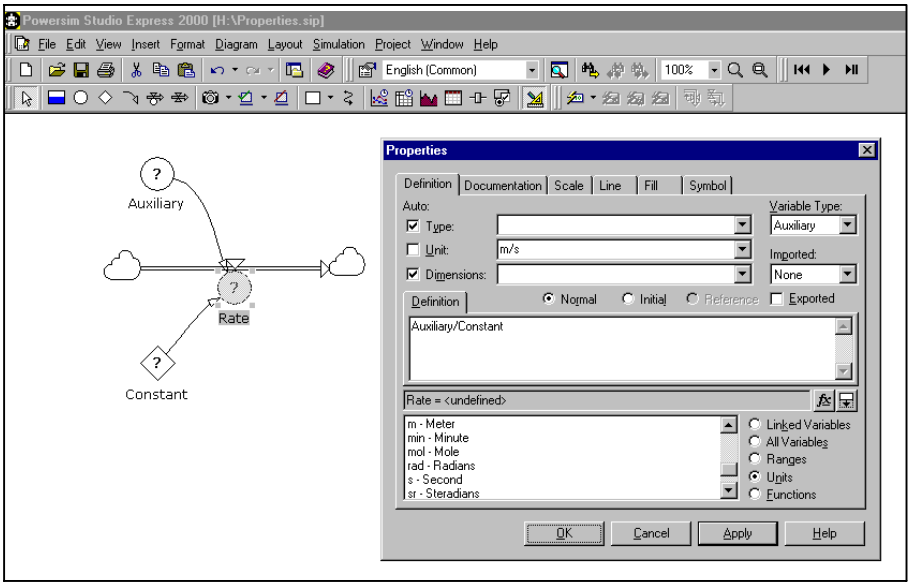


Figure 3.4 Properties window in Powersim

Finally, because complex models are often diagrammed in multiple displays, situations arise in which variables pictured on one diagram are used in another diagram. These variable cross-references are shown by including the symbol of the other diagram's variable in parentheses as shown in figure 3.5.

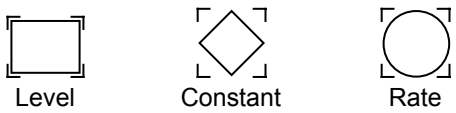


Figure 3.5 Representation of variable cross-reference

3.7 Review of Simulation Tools

The simulation tools described in this chapter are some of the most common simulation tools for system dynamics simulation. In these simulation tools it is possible to both develop the underlying model and create the *interface*, i.e. the buttons and commands that the users see. [13]

Powersim
In mid 1980s the Norwegian government sponsored research aimed at improving the quality of high school education using system dynamics models and the result was an early version of Powersim. Today, Powersim is a flow-diagram-based modelling tool, which is able to show multiple models simultaneously and connect separate models to each other. It is possible to use slide buttons to handle inputs and reports, and plots and tables for the output. It is also possible to add causal loop diagrams.

Vensim
Originally developed in mid 1980s for use in consulting projects, Vensim was made commercially available in 1992. Vensim is an extremely powerful model development language. The modelling begins with the use of a sketch tool to enter the causal loop diagrams, which will become the basis of the model. Vensim automatically documents the model and creates trees that allow tracing cause-and-effect relationships throughout the entire model. It offers sophisticated statistical and graphics features and allows the user to create menus, input screens and text screens.

Dynamo Plus
Dynamo was the first system dynamics simulation language, and for a long time the language and the field were considered synonymous. The language was made commercially available in the early 1960s. Dynamo Plus allows building extremely large models with a variety of sophisticated programming features. The programming starts by typing in equations, based on diagrams, drawn on paper. The tool is complex but has great programming power.

Stella/ithink
Originally introduced on the Macintosh in 1984, the Stella software provided a graphically oriented front for the development of system dynamics models. Stella is used for educational solutions while ithink is used for business solutions. Because of the powerful features and ease of use, Stella/ithink is one of the most popular system dynamics modelling tools. It allows drawing stock-and-flow diagrams and maps the structure of the system before entering equations. More details can be added, elements can be grouped into sub-models, and it is possible to zoom in for more detail in complex models. The manual is also a good introduction to systems modelling.

Extend
Extend was created in the late 1980s and includes graphical model building with integrated animation. Extend is a powerful and flexible simulation tool, which has possibilities for both discrete-event and continuous simulation. Extend's Specialised Extend toolkits provides solutions in specific areas, for example Extend + BPR that focuses on business process modelling, human performance studies and workflow analysis. [14]

3.8 Conclusion

In this chapter the theory of modelling and simulation has been discussed. One way of building models is based on system dynamics, which is appropriate for simulating the project environment and its dynamic and qualitative attributes. The project environment at ECS is described in the next chapter.

4 Process Description

In this chapter the two processes that has been investigated during this simulation study are described. These are the *product process* and the *platform process* at ECS. The third section of the chapter discusses the differences between, and similarities of, the two processes. [15]

4.1 The Processes at ECS

At ESC there are different kinds of software processes, for example platform, product and support processes. The result of a project that follows the platform process is delivered to the projects that follow the product process for completion of the software. The support processes are used on a regular basis when needed. Examples of support process are Configuration Management, Code Review and Document Review processes. [16]

This separation of the processes is fairly new; the platform process has been active for only a year. The platform process used to be a part of the product process.

4.2 The Product Process

The Software Product Process at ECS has been active several years and is more mature than the platform process. The product process is a waterfall process, divided into eight consecutive phases with milestones (MS) in-between, see figure 4.1. The product process follows PROPS, a project planning method developed by Ericsson [17]. The Time To Market-process (TTM), a management process, is also used to help to introduce a new product within budget and schedule.

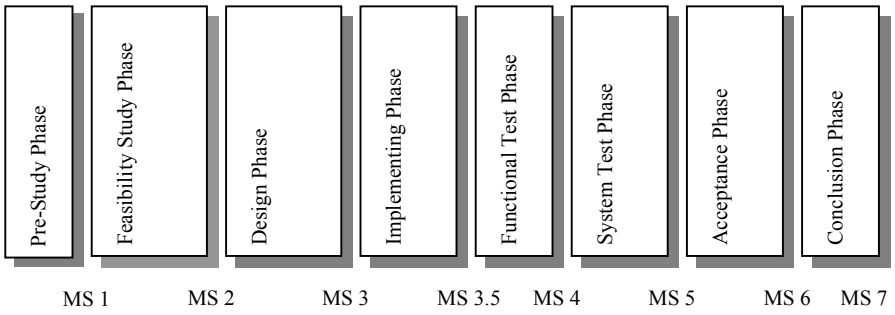


Figure 4.1 The Product Process

In the preparatory *pre-study* phase, before the project has been formally started, market requirements are documented and analysed, so that the requirements will fulfil the business idea and the customer needs. The purpose of the pre-study phase is to ensure that a business idea is technically and commercially feasible.

The second phase is the *feasibility study*. The requirements are documented and analysed in more detail. The project is outlined and the needed resources are specified. The purpose of this phase is to decide on a strategy, define the project goal and prepare project plans.

Between MS 2 and MS 4 the work is separated into different modules. At this time there are a lot of people entering the project, and the design, implementation and function tests are made in work packages in parallel.

The purpose of the *design* phase is to produce design documentation for the software modules and their interactions and internal structure. The documents are intended to support software designers engaged in changing or maintaining the source code. The technical content shall be sufficient to support the design implementation. Test cases shall also be designed.

The intention of the *implementation* phase is to implement the result from the previous phase, both in terms of source code and test instructions for the previously developed test cases. The work is done separately for each module.

The purpose of the *function test* phase is to make sure that all the functions work together properly and that all the software module requirements have been implemented, by testing the software system.

The purpose of the *system test* phase is to integrate and test all the systems in the mobile phone.

The purpose of the *acceptance* phase is to make the software ready for *type approval*. In order to get a type approval from the authorities, a series of acceptance tests have to be made to ensure that the net is not disturbed by the mobile phone's signalling.

The *conclusion* phase is the last phase of the project, during which the experiences made in the project are documented and lessons transferred to the organisation. After this, the project is formally closed. The purpose of the conclusion phase is to terminate the software project and to write a final report, and ensure that the organisation will have access to and be able to learn from the experiences made and the competence development achieved in the project.

4.3 The Platform Process

A product platform is a set of subsystems and interfaces developed to form a common structure from which a stream of derivative products can be efficiently developed and produced. Benefits of a product platform are for example increased speed in product development, reduced product development cost, increased product variety and functionality and potential for new products. [18]

The Software Application Platform process at ECS is used to develop a software platform for several different mobile phones. This platform process is also a waterfall process, which is divided into four phases: pre-study, feasibility study, execution, and functional test, see figure 4.2. The platform process also contains an internal sub-process, the Software Module process, which describes the activities to design, implement and test the functionality assigned to a software module.

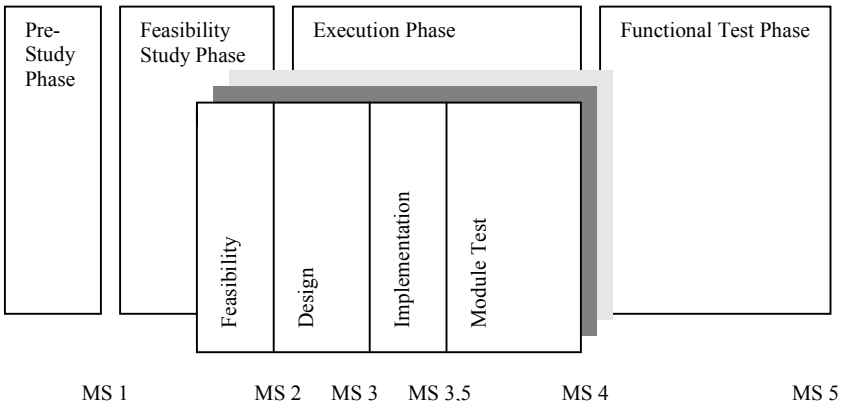


Figure 4.2 The Platform Process

The platform process is performed similarly as the product process. But the last phase, where the integration and functional test of the software platform should be made, is not used today. Instead this is performed after delivery to the product process at MS 4. The product process will in the future integrate the software platform with the software product, after MS 5, and after trouble reports from the product process the platform will be corrected.

4.4 Process Requirements

The platform process is used to develop the generic software to a number of mobile phone models with similar performance. It is important that the generic software is of high quality and reusable so that it is possible to further develop the platform. Since the platform is not developed only for one target phone the platform must be adaptable to several products. It should also be possible to add or remove features and memory capacity to more or less advanced models.

For the product process it is important that a market validation is made to ensure that the product meets the market requirements. It is also important with high delivery precision and the tool for this is the TTM-process. A type approval is done, before the mobile phone is allowed to be used in the mobile net, to avoid that a prototype, due to a manufacturing fault, disturbs a net.

4.5 Conclusion

This chapter brings Part I of this report to an end. Thus, all the theory necessary for this simulation study has been discussed. Part II, which starts with the forthcoming chapter, will use this terminology in describing the method and execution of this simulation study.

5 Method

In this chapter the method, developed to accomplish this simulation study, is described. This means that this chapter only contains *the planning* of how the assignment should be performed and not the execution itself, which is described in chapter 6. The planning of a simulation study also includes defining criteria for different choices made in the study; this is available in chapter 5.2. Chapter 5.3 describes general threats against the model's validity, credibility and usability.

5.1 The Steps in the Simulation Study

The simulation model building process is broken down into three phases: problem definition, simulation planning and simulation operation, see figure 5.1. This break down structure is influenced by [19, 20] and is adjusted to meet the expectations of this thesis. One change is made: adding a simulation step after the *model translation* to make the process able to work in an iterative manner.

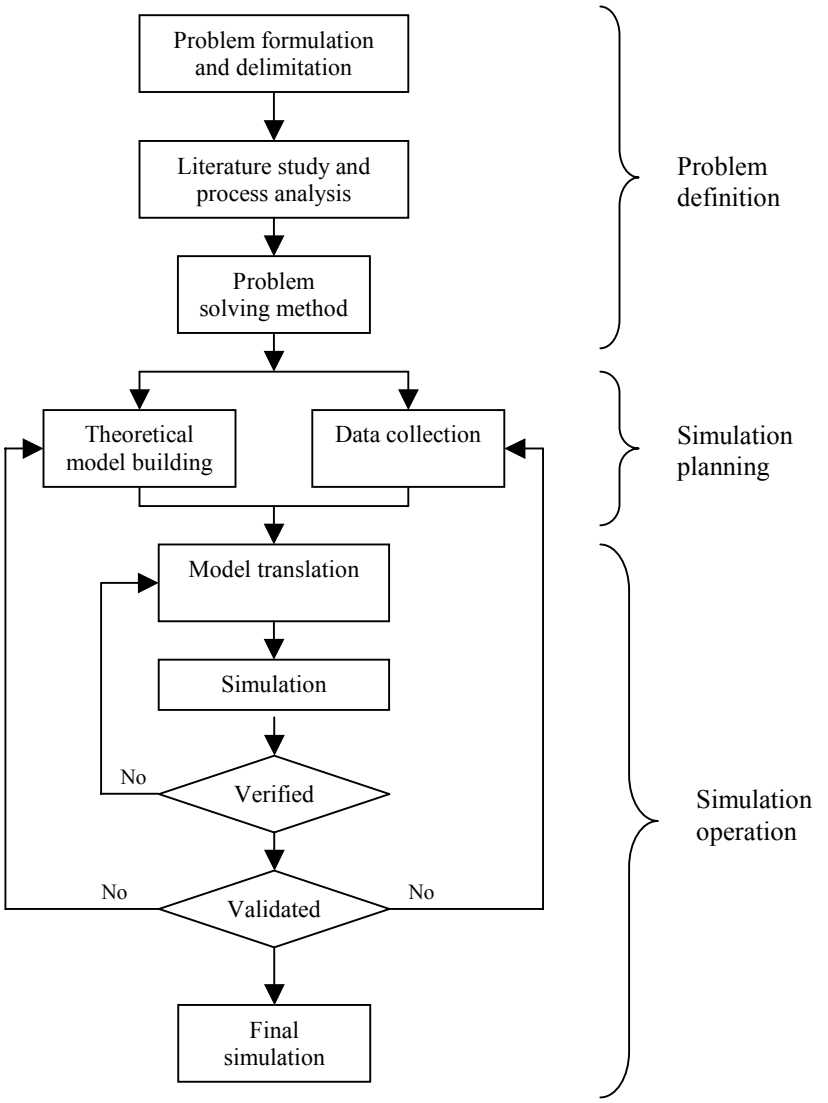


Figure 5.1 Steps in the simulation study

In the first phase, *problem definition*, the problem should be formulated, defined and delimited. The objective should be set and the goal described. Determining how much the model should include is an important issue and an iterative process. To fully consider the whole objective, the model needs to have the right approach. [21]

The problem was defined and formulated at ECS but the delimitation was made after surveying the processes. This phase also includes literature study and process analysis. Literature of software engineering, experiment methodology and systems thinking provided the necessary knowledge to accomplish this simulation study. The process analysis was concerned with studying the existing processes at ECS to understand the relationships

between different parts of the process. The problem solving method was chosen to meet ECS's goal and to answer their questions, which are described in chapter 6.1.1.

The second phase is the *simulation planning* phase. When planning and developing a simulation model, it is necessary to identify the relevant parameters, their relationships, and behaviour, for inclusion in the model. The input parameters to include in the model largely depend on the purpose of the model. [21]

This phase concerns theoretical model building and data collection. The data is the relevant parameters in the simulation model and these were identified and described. The quantitative data was collected through reports and software estimation literature [2, 12]. The theoretical model building includes putting together the relationships between the relevant parameters. This was performed through discussions with experienced employees at ECS. This phase was returned to during the remaining work of the simulation study.

The third phase, *simulation operation*, aims at finding the values of the output parameters. The simulation model was verified and validated to ensure its credibility and relevance. The model was simulated to discover the output parameters and relate them to estimates of time and effort.

This phase starts with translating a small part of the theoretical model to the simulation tool and running the simulation to ensure that it works properly. This was performed iteratively by adding more features to the model, which in this way was further developed. The verification and validation was made continuously through the model construction with help from the supervisors. After this the final model was simulated in order to receive the results needed for the following analysis.

5.2 Selection Criteria

In the second phase, choices of simulation model and simulation tool were made. The simulation model was chosen to meet ECS's desire to increase the systems thinking in the company and to point out the qualitative relations in the development processes. There is a lack of quantitative documentation that had to be manageable by the model. It also had to be possible to simulate the model.

To make a first selection of simulation tools the criterion is that the tool in question is often mentioned in the software literature and is adjusted to the chosen simulation model. From this selection, the tools that were examined had to have an easily available, free demo version, with well-equipped help functions and tutorials. The tool had to be simple to learn and use, to provide a possibility to make an easy adaptation of the model for future work.

5.3 Threats

To avoid the consequences of several possible threats against the model's credibility, these threats were carefully considered during the whole simulation model building process.

One of the problems to consider was that the processes at ECS are new and not so well documented. This means that there is a lack of measurements and it may be hard to find relevant data on an appropriate level. This implies that there is a risk that the model could become too general and therefore not visualises ECS's reality.

One threat against the credibility of the model was that values of the chosen input parameters could be impossible to find in ECS's development projects. The consequence of this would be that parts of the model had to be rebuilt, and new input parameters would have to be found, under time pressure.

Another possible problem is that the results from the developed model would not be usable enough. This can be the case if the input parameters can not be affected by ECS's managers or if the communication with the decision-maker fails. It is essential to promote model credibility and to ensure that the correct problem is solved.

A threat against the model's validity is that the validation was performed entirely through discussions with employees at ECS. The respondents might have given subjective answers and might not be competent enough. This can result in inconsistent and corrupt data. [9]

If the simulation tool chosen is not adaptable to the problem solving method, the problem solving method would have had to be revised since a change of tool would be more costly both in terms of money and effort.

For future use there is a risk that the results and analyses are misinterpreted because the user is not well acquainted with the utilisation of the simulation model.

5.4 Conclusion

With this well-defined plan of the simulation method, a stable foundation for the forthcoming work was achieved. The selection criteria and threats, discussed above, were taken into consideration throughout the study. The forthcoming work will be described in the next chapter.

6 Execution of the Simulation Study

This chapter describes the execution of the three phases of the simulation study: problem definition, simulation planning, and simulation operation. In the problem definition phase, the problem was concretised in order to build the theoretical model, which was done in the simulation planning phase. This theoretical model was transformed to a simulation model in the simulation operation phase.

6.1 Problem Definition

The first phase in the simulation study was the problem definition. This included formulation and delimitation of the problem, choice of problem solving method, literature study, and process analysis, see figure 5.1.

6.1.1 Problem Formulation

The original problem was formulated and defined by ECS at the beginning of this thesis. The overall objective for ECS was to visualise the process mechanisms and the relations between the factors that affect the product quality. One of the most important factors that affects the product quality and the lead-time of the project is the allocation of resources, i.e. the number of people in each phase. The task was during this thesis broken down and concretised to meet the objective.

To concretise the task different questions were addressed:

- What factors affect the quality of the products in ECS's software development projects?

- How do these factors affect each other?

- Can a relocation of resources, to the different process phases, affect the quality?

- Is the lead-time of the projects affected by the allocation of resources to the different phases?

The problem approach, chosen to be able to answer these questions, was to work out factors affecting the quality of the software developed at ECS, with help from the supervisors and software literature. Discussions with employees at ECS gave the most important factors and their relationships. Through these discussions *influence diagrams* were created in order to understand what affects the time in each phase [4].

To answer the last two questions it was also necessary to study the resource allocation between the requirement phase and the test phase. A problem in the recently concluded projects at ECS was that the test phase required a great proportion of the resources compared to the other process phases. Another problem was that the projects were delayed due to insufficient time planning, which lead to increased schedule pressure. This approach showed how the quality and the lead-time were affected by a relocation of the resources.

The influence diagrams, together with the resource allocation approach, created the base of the simulation model.

6.1.2 Problem Delimitation

This thesis was performed at the unit Software Application Platforms, which follows the platform process, described in chapter 4. However, the platform process was at the moment a new process and not yet fully executed and this implied that there was no data available. Therefore, the model shows the product process instead of the platform process. Since the platform process used to be a part of the product process there are many similarities between them. In the future, when there are more data available, it will be possible to rework the model to be applicable to the platform process.

The model is based on data from a final report for a software sub-project. This project was performed at ECS and concluded in 1999. This sub-project was chosen to be the base of the model since it was the project, among the projects recently finished at ECS, that had the greatest amount of data available.

The study was focused on the requirement specification phase and the test phase, see figure 6.1. The phases in between were excluded because they were not important enough in the problem approach, described in chapter 6.1.1. For the same reason the phases that follow the test phase were excluded. The requirement phase is the base of the software project and it is important to understand how the result of the test phase is affected by the requirement specification quality. The problem at ECS was, at the time of the problem formulation, that a great fraction of the human resources was required in the test phase, due to insufficient effort estimation. The test phase is important both in respect of time and effort since this reflects the software quality.

The requirement phase includes the pre-study phase and the feasibility study phase and stretches from MS 0 to MS 2, according to chapter 4.2.

The test phase in this thesis involves the functional, system and acceptance tests that are performed between MS 3.5 and MS 6. All these types of tests are included, since the data available did not separate these and they overlapped in terms of time.

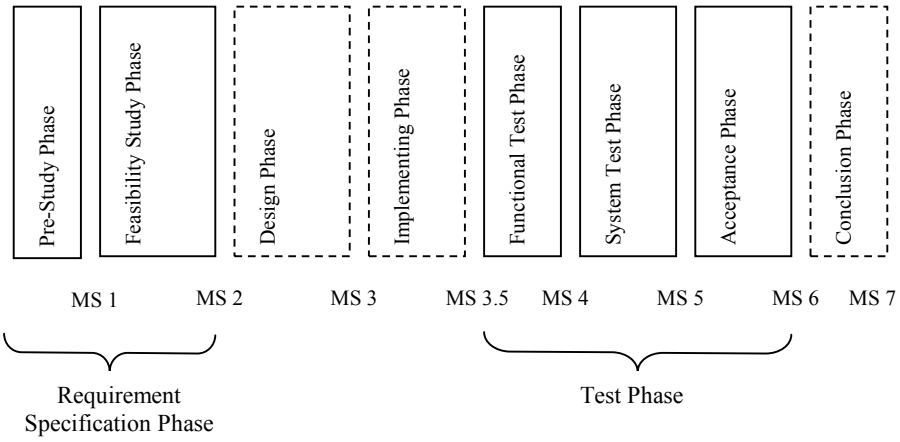


Figure 6.1 Process delimitation

6.1.3 Literature Study and Process Analysis

The literature gave insight into model building and simulation. Systems thinking and software knowledge was gained from, respectively, management and software engineering literature. A selection of important concepts of software engineering is available in chapter 2 and a summary of model building and simulation in chapter 3.

Knowledge about the processes was gained from ECS's internal homepages where the different phases were described [16]. An interview with a process developer at ECS gave a deeper understanding about the platform process [15]. General information about software development processes,

and among them the waterfall process which is used at ECS, was found in reference literature [1]. ECS's platform and product processes are described in chapter 4.

6.1.4 Problem Solving Method

In order to build and simulate a process model, the problem solving method was divided into two parts. Each part involved one comprehensive choice: choice of simulation model and choice of simulation tool.

Choice of simulation model
To choose between the discrete-event and continuos models, it was necessary to understand the relationships involved in the processes at ECS. The objective of this thesis was to visualise the process mechanisms at ECS, which implied the choice of the system dynamics model since it is better at showing qualitative relationships than the quantitative discrete-event model. It was also important to consider the amount of detailed information available. The processes at ECS were at the moment rather young so there was not much data documented. In such a case a qualitative approach is better, which reinforced that the continuous system dynamics simulation should be used in this thesis. The discrete-event model's requirement for wealth of details made it difficult to use on these processes.

Choice of simulation tool
To choose simulation tool, five tools, well known among the system dynamicists, were looked into and are described in chapter 2. Four of these programs were carefully examined: Powersim, ithink, Extend and Vensim. The fifth tool, Dynamo Plus, was difficult to examine since the developer did not have a homepage on the Internet. The consulting group that created Dynamo no longer uses it. Instead they, at the moment, used a proprietary software that is a Dynamo next generation, but it was not available to the public. Therefore, no demo version of the program was available [23]. Demo versions of the other four programs were downloaded from the Internet. Tutorials and small simulation models were built to analyse the features and the differences between the programs.

Powersim was chosen because of its great variety of features and functions and the fact that the program was already in use at other Ericsson companies. The ithink demo version had no help functions, which made it difficult to get a fair opinion compared to Powersim's demo version, which had many well-equipped help functions. Powersim had many of its functions available as icons in the window in contrast to ithink, which made it necessary to have a greater knowledge of the functions in ithink.

The Extend program could be used on both discrete-event and continuous simulation, but the interface made it hard to see the difference between discrete-event and continuous models. Many of the functions had similar characteristics, which made it difficult to choose the right one since there was no information about how to use them.

Vensim was another well-equipped tool with a demo version available on the Internet. For optimal use of this program, causal-loop diagrams should be the basis of the model. In this thesis, the basis of the model is built on influence diagrams and therefor this program was not suitable.

The difference between the simulation tools above is in the model construction process but the results from this simulation study would probably have been equivalent with any of the tools. The arguments for Powersim were however stronger than the ones for the other tools.

6.2 Simulation Planning

The second phase in the simulation study was the simulation planning. This included collecting data, on which a theoretical model was built, see figure 5.1.

6.2.1 Data Collection

The first step in the simulation planning phase was the collection of factors that affect the quality of the developed software. The factors were collected with help from the supervisors and from significant software literature [2, 3]. The factors were chosen to be suitable to software development processes. The factors are enumerated below without any relative order.

Table 6.1 Factors that affect the quality

1	Number of people in the overall project
2	Number of people in the object or team
3	Personnel education
4	Personnel experience
5	Personnel salary
6	Staff turnover
7	Communication level
8	Geographical separation of the team
9	Software and hardware resources
10	Environment, for example temperature, light and ergonomics
11	Amount of overtime and workload
12	Schedule pressure
13	Budget pressure
14	Rate of requirement change
15	Amount of program documentation
16	Level of reusable artefacts, for example code and documentation
17	Level of structure in the project organisation
18	Standards that will be adhered to, for example ISO and IEEE
19	Software size and complexity
20	Testing and correcting environment and tools
21	Requirement specification accuracy
22	Amount of review

This phase was returned to iteratively during the work with the simulation model. During the simulation work described in the next phase, simulation operation, quantitative data was collected through reports and estimations.

6.2.2 Theoretical Model Building

The theoretical model building task contained two lines of action, which were conducted simultaneously. One procedure was to construct causal-loop diagrams for some essential factors from table 6.1, in order to get a basic understanding of the feedback concepts. The included factors were chosen after discussions with experienced developers. The diagrams are comprehensive in a long-term view, concerning the development process during several projects.

The first diagram, figure 6.2, illustrates how the schedule pressure affects the rate of finished work. Schedule pressure is an important factor since it often occurs in the delayed projects at ECS. According to Abdel-Hamid [12], schedule pressure can play a significant motivational role on the productivity. An increase in schedule pressure contracts the project members' slack time. Slack time is the fraction of project time lost on non-project activities, for example coffee breaks. Unfortunately, the trade-off is that the error generation rate increases with a higher schedule pressure. Shneiderman [12] suggests that schedule pressure increase the anxiety levels of the programmers who tend to make more errors. Abdel-Hamid claims that "In the struggle to deliver any software at all, the first casualty has been consideration of the quality of the software delivered." More errors generated makes the rate of finished work lower and therefore increases the schedule pressure.

Since there is an odd amount of "o" in the right loop, it is called *balancing* and is characterised with a "balance beam". An increase in schedule pressure also increases the error generation, which in turn decreases the work rate that decreases the schedule pressure. This, on the other hand, is a *reinforcing* loop, since it contains an even amount of "o", and is characterised with a "snowball".

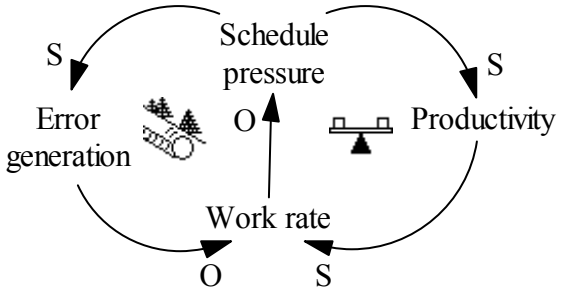


Figure 6.2 Effects of schedule pressure on productivity and error generation [12]

The next diagram, figure 6.3, shows a reinforcing loop, which covers several projects. This diagram visualises how the resources affect the product quality and the project lead-time, which is a relation that is asked for in the questions in chapter 6.1.1. A decrease in resources gives a product with decreased software quality and more rework is needed. More rework gives a longer lead-time, which in turn increases the total costs. An increase in the total costs affects the resources available for the next project.

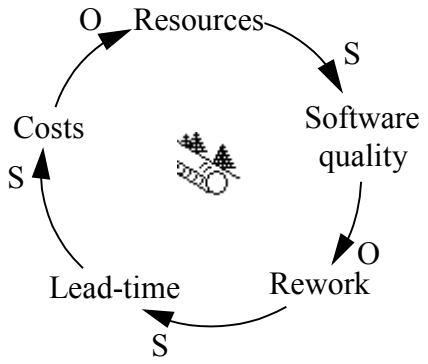


Figure 6.3 A reinforcing loop, with factors that affect the forthcoming project.

The third causal-loop diagram illustrates some factors in the requirement phase, see figure 6.4. The diagram shows, in another approach, how the schedule pressure affects the quality, but this time specifically in the requirement phase. The specification accuracy is a measure of how precise and well constructed the requirement specification is. If the specification is inaccurate, i.e. of low quality, it has to be reworked during the project, which leads to an increase in the amount of unfinished requirements. An increase in unfinished requirements and thereby an increased amount of rework leads to higher schedule pressure. This decreases the time available for review and therefore the specification accuracy gets less precise.

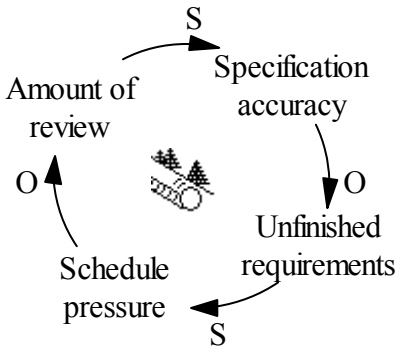


Figure 6.4 A reinforcing loop for the requirement phase

At the same time as the causal-loops were built, *influence diagrams* were constructed, see figure 6.5 and 6.6. With assistance from the supervisors at ECS, the factors in table 6.1 were regarded in respect to the quality and the time in each phase of the process. The factors that were considered to affect the lead-time the most were chosen to be included in the influence diagrams. Influence diagrams for the requirement and test phases were built to show how these factors, in some cases reworked, affect the lead-time and the software quality. Each factor's importance for each phase was considered together with the relationships between the factors.

Most of the factors in the influence diagrams were taken directly from table 6.1, but some were changed, for example number 14, Rate of Requirement Change, which was divided into Inadequate Requirements and Amount of New Market Requirements. Inadequate Requirement occurs when the Requirement Specification Accuracy is not high enough while the Amount of New Market Requirements increases because the total lead-time for the project is so long that the market requirements have time to change. Inadequate Requirements affect the amount of defect code that is produced in the design and implementation phases. In this thesis factor number 19, Software Size and Complexity, is equal to Functionality. Productivity does not affect the quality, and is thereby not seen in table 6.1, but is an important factor for the time in each phase.

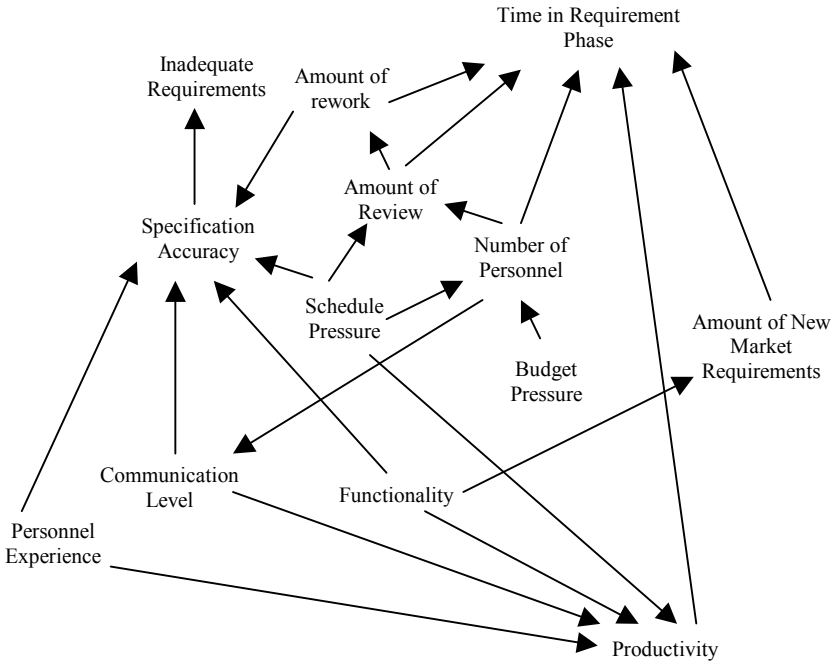


Figure 6.5 Influence diagram for the requirement phase

The test phase in the model was divided into two parts, testing and correction, since the two tasks are performed separately. The testing includes the functional, system, and acceptance tests that are performed between MS 3.5 and MS 6 in the product process.

In the model, the Amount of Defect Code originates from the design and implementation phases and therefore gives the impression that no faults are generated in the test phase. Instead there is an Errors Undiscovered-variable that is affected by the Amount of Defect Code and several different factors that affect the result in the test phase.

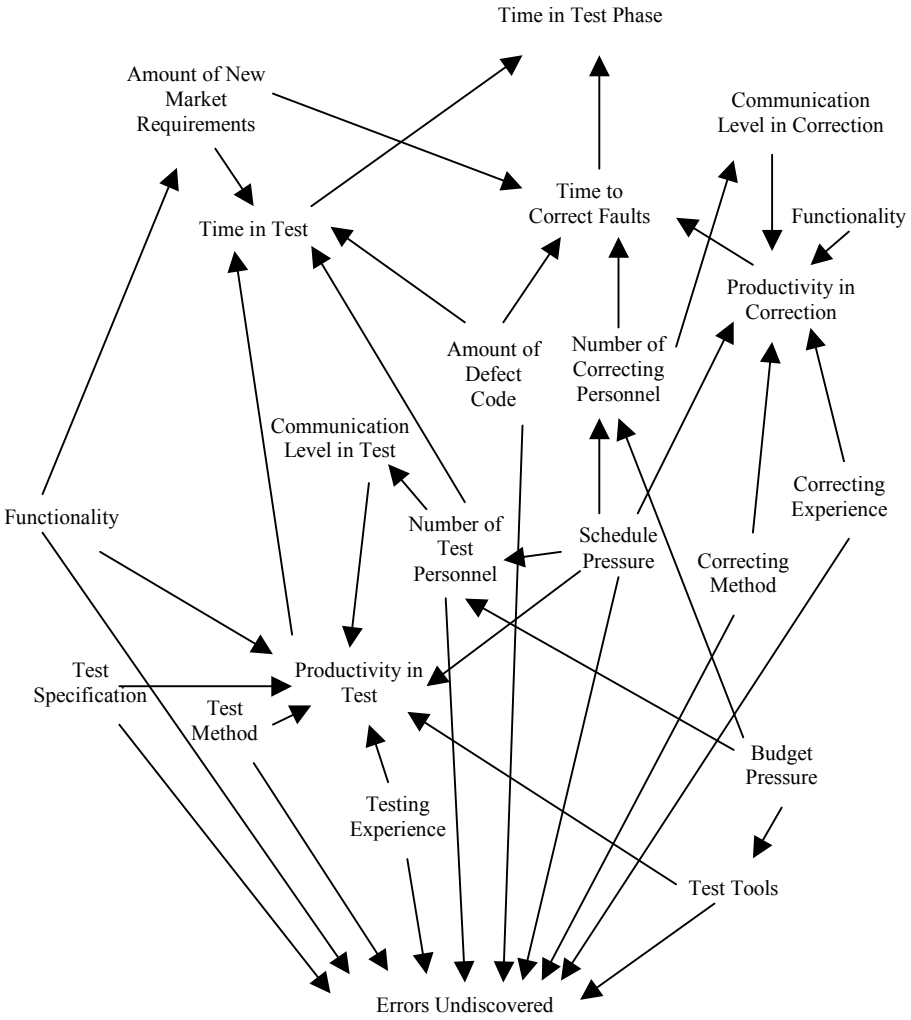


Figure 6.6 Influence diagram for the test phase

6.3 Simulation Operation

In the third phase of the simulation study, the theoretical model was built in the simulation tool, see figure 5.1. The construction started with a draft of the process model, which was then increased to include the two process phases and their relevant parameters. The final model is available in appendix A.

6.3.1 Model Translation

The whole idea behind the model of the requirement phase is based on a flow of tasks, from customer requirements to finished specifications, see figure 6.7.

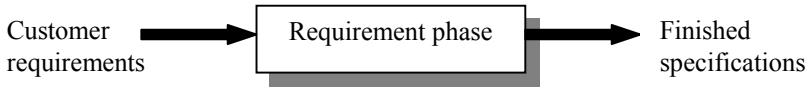


Figure 6.7 The requirement phase as a black box

Inside the requirement phase there is a transformation from uncompleted to completed tasks by the production of specifications. A fraction of the specifications are not acceptable and needs to be taken care of in the rework loop, see figure 6.8.

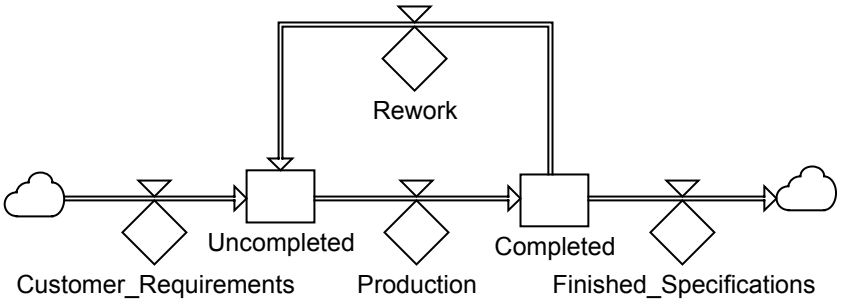


Figure 6.8 The basic flows in the requirement phase

The test phase in the model is based on the same idea as the requirement phase and is built in a similar way. A flow of test cases is performed, a certain percentage has to be corrected, and the rest is supposed to be acceptable.

This basic model was built in Powersim and further developed with help from the factors in the influence diagrams. The causal-loop diagrams were also considered during the development, to ensure that the model was adapted to systems thinking, but the parameters were chosen from the influence diagrams.

The factors from the influence diagrams were added as auxiliaries and constants to control the flows. To avoid getting a too complex model, all of the factors in the influence diagrams were not included in the simulation model. The factors included were regarded to be easier for the management to affect than the ones excluded. Some factors were included indirectly in the parameters in the model. These can be extracted from the parameters and thereby possible to affect from the user interface.

The construction was made step by step, by adding a few factors at a time and then running the simulation. The values of most of the auxiliaries and constants are taken from [22] and software literature [2]. Some values were estimated by iteration and verified by discussions with the supervisors at ECS.

6.3.2 Verification and Validation

During the development of the model and its parameters, the model was continuously verified and validated.

The verification is concerned with building the model right, i.e. confirming that the right parameters have been used. A comparison of the theoretical model to the computer representation was performed. This was made by comparing the influence diagrams in chapter 6.2.2 with the simulation model built in Powersim. The verification also included comparing the time in the simulation to the time according to the report [22] to ensure that the estimations were correct.

The validation aims at building the right model, i.e. determining that the model is an accurate representation of the reality. Validation involves checking that the model meets the expectations of the assignor. This was made through discussions with the supervisors at ECS.

6.3.3 Simulation

The final model was run in three sets of simulations to answer different kinds of questions, mentioned in chapter 6.1.1. These three sets of simulations showed: how the lead-time was affected by the amount of effort spent in the requirement phase, if the human resources in the test phase were fully used, and how a change in human resources in the test phase would affect the lead-time.

A detailed description of all the parameters and the model can be seen in appendix A. The values taken from [22] are summarised in appendix B.

6.4 Conclusion

This chapter has described the execution of the three phases in the simulation study: problem definition, simulation planning and simulation operation. The next chapter presents the results from the simulations of the final model.

7 Results and Analysis of the Simulation

The final model was simulated in two sets, called Case 1 and 2. This chapter presents the results of these cases and the analysis of them. The results are given in precise figures but since this is a system dynamics model, the results shall be interpreted qualitatively. It is the tendencies in the graphs that are important and not the exact percentages.

7.1 Case 1

Case 1 was performed to show how a relocation of resources to the different process phases affects the quality of the software product and the lead-time of the project, which was discussed in the two last questions in chapter 6.1.1.

7.1.1 Simulation of Case 1

The final model was run several times with different values of the percentage of the planned effort in the whole project spent on the requirement phase, see table 7.1. The values ranged from 8 to 13 % and the value from this particular project was 9 %.

Table 7.1 Results from Case 1

% in Req	ReqEffort	TestTime	TestEffort	TotalTime	Errors
8%	23	267	174	623	252
9%	25	236	153	585	222
10%	28	211	137	558	179
11%	31	198	129	547	158
12%	33	193	125	546	153
13%	37	192	125	551	156

The outputs, in case 1, were the real effort spent on the requirement phase, the time in the test phase, the effort in the test phase, the total lead-time of the project and the number of errors that are not discovered. The test effort is based on 3 testers and 10 correctors working full time. The total time is the sum of the time consumption in each phase, which is possible since the design and implementation phases are modelled as a delay, according to appendix A. The units of the efforts are man-months, the units of the times are working days and the errors are a relative measure of the quality.

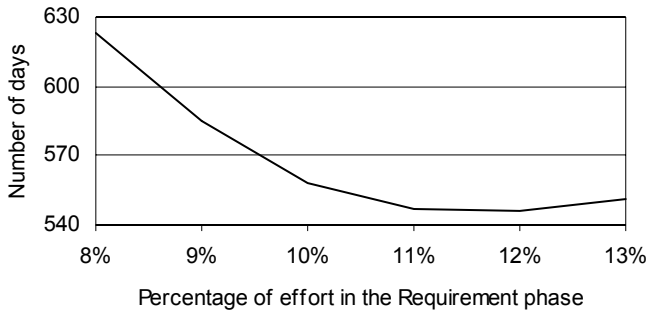


Figure 7.1 The total lead-time for different percentages of the effort spent in the requirement phase

The simulation runs indicate that the effort spent in the requirement phase has a pronounced effect on the lead-time of the project, see figure 7.1. The decrease in days, when increasing the effort in the requirement phase, arises from the increased specification accuracy. A more accurate specification facilitates the implementation and decreases the error generation and will result in a higher quality product from the start. This decreases the amount of necessary correction work and thereby shortens the time spent in the test phase. At a certain point the total lead-time will start to increase again because the time in the test phase levels while the time in the requirement phase continues to increase. The time in the test phase levels because there is always a certain amount of functionality that shall be tested at a predetermined productivity.

The errors, which are seen as a measure of the quality, were also considered during the simulation in case 1.

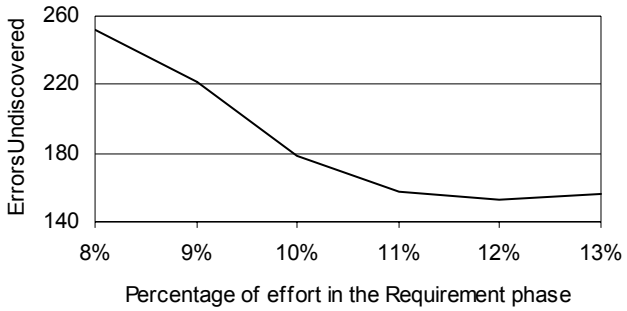


Figure 7.2 The number of undiscovered errors for different percentages of the effort spent in the requirement phase

The amount of errors that are not discovered in the test phase decreases when the effort in the requirement phase increases because the error generation is lower when the specification accuracy is higher, see figure 7.2. When the total lead-time is increased again, according to figure 7.1, the amount of new market requirements, that has been considered, also increases. Therefore there is a slight increase in the curve since these new market requirements generate more errors.

7.1.2 Comparison with Other Estimating Methods

The COCOMO model, described in chapter 2.3.2, can also be used to estimate the total effort to develop software. In this case the produced code volume is 115,5 KLOC and the mode most suitable in this case is Organic.

$$E = a * KLOC^{b} = 2.4 * 115.5^{1.05} = 352 \text{ man-months}$$

If the effort in this project [22] had been estimated according to above and the effort in the requirement phase had been planned to be 9 % as was the case in this particular project, then the effort would have been 9%*352=32 man-months. This is comparable to the ReqEffort in table 7.1, where the minimum for the errors occurs at approximately 33 man-months.

According to software literature [2] the percentage of the total effort spent on testing, in a project of this size, shall be 30. Since the simulation model only includes the requirement and test phases, it can not predict the total effort. But assuming that the total effort on this project was 352 as COCOMO suggests the test effort would be 125/352=35% of the total effort.

7.2 Case 2

This case was performed to show that it is possible to decrease the test time, and thereby the total lead-time, by increasing the human resources in the test phase.

A series of simulations were run to see how the test time was affected by a change in the number of correcting personnel, while the number of testers was unlimited. The graph is based on values presented in table C.1.

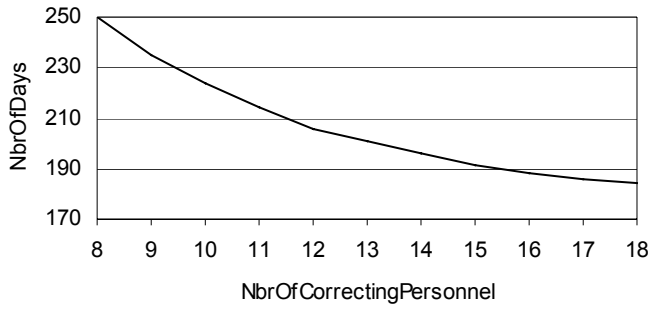


Figure 7.3 The test time when the number of test personnel is not a limiting factor

Figure 7.3 shows that the lead-time is not decreased linearly when more correcting personnel is added. The reason for this is that the communication problems increase when the number of correcting personnel increases.

A similar series of simulation runs was performed with a constant, not limiting, number of correcting personnel. The graph is based on values that are presented in table C.2.

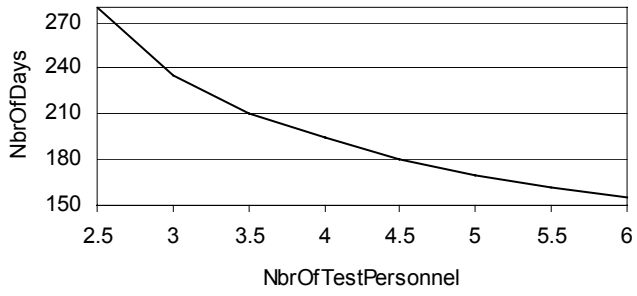


Figure 7.4 The test time when the number of correcting personnel is not a limiting factor.

Figure 7.4 shows that the lead-time is decreased, but not linearly, when more test personnel is added. The curve in figure 7.4 does not level as much as the curve in figure 7.3. This is because the communication problems among the testers are not as noticeable as among the correctors since the testers are fewer.

7.3 Conclusion

The simulations increase the user's understanding about the process mechanisms since the simulations visualise the different parameters' effect on each other and on the system.

The results from the different cases have been presented and analysed in this chapter but shall be interpreted qualitatively. The next chapter consists of an evaluation of the simulation tool that was used throughout these simulations.

8 Evaluation of Powersim Studio 2000

The first version of Powersim was developed in the 1980s in Norway. It has been developed to become an integrated environment for working with simulation models, and in these days many multinational companies all over the world are using this program to make strategic decisions.

At the stage when the choice of simulation tool was made, the available tool was an earlier version, Powersim Constructor Lite. The choice of simulation tool, for this simulation study, was therefore based on the features and functions of this old version. When it was time for the purchase, in November 2000, a new version was released, the Powersim Studio 2000.

There are three different versions of the Powersim Studio 2000: Enterprise, Standard, and Express. According to [24] the Enterprise is the full-scale platform for commercial business users and includes cross application communication. Standard offers full simulation capabilities but does not include cross application communication, while Express is a free platform that limits the number of variables to 150 and is ideal for students and academics with less complex modelling needs. The one most suitable for the assignment at ECS was the Standard version.

Three major disadvantages with Powersim Studio 2000 Standard were found during the simulation study.

One of the selection criterions for the choice of simulation tool in this thesis was that Powersim already was in use at another company at Ericsson. However, at the time of the delivering of the simulation tool it appeared that it was not very easy to convert models, created in the old versions of Powersim, to the new one, Powersim Studio 2000. This is because there is a big difference between the old and new tool so the converting results in a number of errors. It may be possible to adjust the converted model if the modeller has knowledge about the appearance and performance of the old model. The graphics are deformed at the conversion and some functions are defined differently in the two tools and therefore it takes a lot of effort to make the old model work in the new tool.

There is no easy and straightforward possibility to present the simulation results from Powersim Studio 2000 in another type of document, for example Microsoft Word. To convert a sheet from Powersim Studio 2000 it is necessary to copy a bitmap of the active window to the clipboard and then paste it into Word. Thus, a bitmap is made, instead of a Windows metafile. This will result in a low quality of the picture.

The Standard version, unlike the Enterprise version, does not include any possibilities of connecting data from external files or data sources to or from the simulation. This can be useful when a number of simulation runs, with different inputs, are to be performed and compared.

Some minor disadvantages were also found during the simulation study, which made it difficult to interpret the results. The tool transforms the calendar units automatically to a unit without decimals when it is possible. In other cases the tool chooses a unit so that the value is between 1 and 10. This can result in an output given in the unit quarter for one simulation run but the same output can be given in the unit weeks in another run. There is no possible way to control this transformation which complicates the comparison of different simulation runs.

In Powersim Studio 2000 three different calendar types exist: with leap year, without leap year and an alternative consisting of 360 days. Thus, there is no calendar type that considers weekends, i.e. using a working week of five days. The consequence of this is that the finishing date delivered by the tool is of no relevance, since it is not comparable to the real date.

When presenting output values from the simulation in a table control, the tool shows the exact value, even if it means including ten decimals. There is no possibility to change the number of decimals and this

makes the appearance of the presentation of the output parameters unpleasant.

Nevertheless, a number of improvements have been made in Powersim Studio 2000, compared to the old version. These improvements are not of general nature, but give a more professional impression.

The graphics have been noticeably enhanced. The toolbar has been extended and the symbols are clarified. In the new version it is possible to add more than four flows from each level without overlapping, which makes it easier to read the sheets that illustrate the models. It is also possible to change the size of the details, for example the levels and flows, which increases the visibility in complex models.

There are several ways to view the value of the variables; an easy way is to add a *number auto report* or *time graph auto report* in the sheet. This new function is a small box, located close to the variable, in which the value is viewed during the simulation run.

In the new version of Powersim it is necessary to define units in the variable's property dialog box. This was at the start of the simulation both annoying and time consuming to consider, when the only intention was to make small drafts to test different ideas. However, this feature forces the modeller to carefully reflect on the model's construction a second time, to ensure that the units are correctly calculated through the whole model.

Powersim Studio 2000 is to a great extent a satisfactory simulation tool with good graphics and thoroughly worked through construction. But unfortunately it seems like the new Powersim version was released under stress and therefore contains some major defects, which is a pity since it in other respects is a great tool. These defects will hopefully be corrected when the new version of the tool is released in May 2001.

9 Discussion

This chapter summarises the whole simulation study and discusses thoughts that turned up during this study. The chapter also includes ideas for future development of the simulation model to increase its reliability.

9.1 Summary

This thesis was performed at Ericsson Mobile Communications AB in Lund from October 2000 to March 2001. There was a desire to increase the quality of the developed software and to improve the processes that are used for software development. To be able to make these improvements, the knowledge about the process behaviour had to be expanded. This awoke the interest to build models of the processes and to simulate these models. These models could help to visualise the mechanisms of the software processes that affect the quality.

The result from this thesis is a simulation model that visualises different relations in ECS's software development process. The work has also resulted in a collection of quality factors and their connections, which are illustrated in the influence diagrams and causal-loop diagrams. The report can be used to increase the competence on modelling and simulation of software processes.

The developed model is limited to include the requirement phase and the test phase. One of the problems at ECS is the allocation of resources in the projects. The model illustrates how a changed amount of resources in the requirement phase affects the time in the test phase, the total lead-time and the quality of the final product.

The model was used for two different types of simulations. The first series of simulations considered the resource allocation between the process phases, while the second one handled the resource allocation within the test phase.

The results from the first type of simulation were compared to the outcome of the project [22]. These showed that an increased amount of effort spent in the requirement phase would have shortened the time in the test phase and thereby the total lead-time. The simulations also showed that the quality would have been improved if the amount of effort spent in the requirement phase had been increased.

The second type of simulation concerned resource allocation between the testers and the correctors in the test phase. The simulation showed that the time in the test phase could not be decreased by only adding human resources to either the testing or the correcting group, but the resources had to be increased to both of them simultaneously.

The results from the simulations shall not be interpreted quantitatively since it is a system dynamics model. Therefore the values of the output parameters are not the essential part of chapter 7 where the results of the simulations are presented. Instead it is important to notice the tendencies of the increase and decrease that can be seen in the figures 7.1-7.4. This qualitative view also implies that the input parameters in the simulation model do not have to be precise but it is whether the parameters have an increasing or decreasing effect that is important.

One way of defining the input parameters is to let the model user control them from the user interface. However, the pronounced direction of this thesis was to build a specific model of a process at the department of Software Applications at ECS. To avoid generalising the model the parameters were set at specific values, instead of using the parameters as inputs that are changeable from the user interface. Assumptions were made about these specific values and to further improve the reliability of the model, these values need to be more carefully monitored and measured.

In order to be able to develop the model further, the model is built to be basic and easy to understand. Therefore many of the factors that affect the quality are excluded in the current model. These factors and their relationships are available in the influence diagrams for the requirement and the test phase, and can in the future be included in the model.

The major part of the knowledge gained from simulations is brought about in the model building process. The system dynamics model itself can be thought of as a by-product. The procedure to build a model forces the participants to communicate their mental models to create a common image of the organisations new direction. This implies that the decision-makers themselves should build their own models, with help from this report, to increase their awareness of the process mechanisms.

The supervisor at ECS wished to examine if a System Dynamics model is useful for modelling their software development process. We think that simulations of this kind are valuable when the purpose is to visualise process mechanisms and increase the awareness of how different factors affect the software quality. It can also be used to increase the motivation of the staff members to work with quality issues and to increase the product quality early in the project. But this kind of model should not be used for optimisation since the basic idea with System Dynamics models is to interpret them qualitatively.

Though, if the purpose is to calculate a reliable optimum it is necessary to create a discrete model instead of a continuous. This would require more detailed data than what is available at ECS at this moment. The base of our model could have been created as a discrete model since it is a sequence of activities. But a discrete model could not have taken into consideration the dynamic variables that are used, like for example SchedulePressure.

9.2 Further Development of the Model

There are a number of possibilities to further develop the model in order to increase the reliability and to make the model provide a better image of the reality. To be able to understand all the proposed changes in this chapter it is necessary to have knowledge of how the model works and is constructed.

During the building process, several assumptions of the values of the parameters had to be made, since there were no existing data of the parameters. The parameters that were found to be inadequately measured are presented below with suggestions of further development.

The value of NewMarketReqRate describes how many of the changed market requirements that are taken into consideration. It was taken from [2] but if the value had been measured, the model would be more specific for the processes at ECS. An even more accurate description of reality would be accomplished by letting the NewMarketReqRate depend on the project deadline. The intention is to take fewer market requirements into account when closing in on the deadline.

In the current model the parameter ChangeManagement considers all the work that is needed for the new market requirements, from requirement specification to design and implementation. This value is calculated iteratively and should require a more accurate value. Another alternative is to put the inflow of the new market requirements in the requirement phase in the model instead. In that way the new market requirements would be taken into account by the earlier phases so that the model better resembles the reality. However, this requires a major change in the model since in that case the phases have to work simultaneously.

To get a more accurate value of the amount of effort that is spent on each phase it is necessary to improve the time reports at ECS so that all the activities can be connected to the right process phase. This might result in a better estimation of the number of personnel in each phase. It would also give a better estimation of the productivity in each phase that can be used instead of the current values. The communication problems that affect the productivity in larger teams can be modelled separately with assistance from [12].

The productivity in the requirement phase is not modelled in the same manner as the test productivity and the correction productivity. It is possible to let the requirement productivity depend on the number of personnel in the requirement phase in the same way as the test productivity. Since the requirement productivity in its present appearance takes the planned project time into consideration, it would require great restructuring effort to the model. If this change is made it is necessary to add a parameter that indirectly affects the time consumption in the phase, by affecting the ReqProductivity. One way of solving this problem is to add a parameter that considers the amount of review work. This parameter would be affected by ReqPart. The amount of review would also affect how much of the specifications that would be reworked.

The parameter ReworkPart could be estimated more accurately if ReqProductivity had received a more proper value. This could be made by iterating ReworkPart until the time spent in the requirement phase in the model is equal to the actual time according to the modelled project. By comparing different versions of the specification documents, a value of how much of the document that has been rewritten could be estimated. Another alternative is to have the parameter as a changeable input controlled from the user interface. In that way the user can change the amount of rework and thereby the specification quality. A third possibility is to add more variables, for example the amount of review and the requested specification quality, that affect the amount of rework.

The output quality of the finished specifications, InadequateReq, is given an estimated normal value in the current model. This value could be estimated more carefully by letting the personnel in the forthcoming phases register which errors that originate from incorrect

specifications. It is also possible to add more factors that affect this parameter, for example schedule pressure and personnel experience.

The parameter InadequateReq affects the CorrectionPart in the test phase since an inadequate specification quality requires more correction. The value of the CorrectionPart can be better estimated by measuring how much of the code that has to be corrected. In a further developed model the CorrectionPart could be affected by more parameters. These additional variables can for example be SchedulePressure that is compared to the approaching deadline.

The parameter SchedulePressure in the current model can be divided into two different factors. One part only considers the SchedulePressure, which will increase with the time and another that controls the nominal number of errors committed per function point. In this way the ErrorRate will be affected by these separate parameters and possibly additional parameters of an organisational type, for example personnel experience and degree of structured techniques.

The current model only represents the requirement phase and the test phase but the phases in between are considered in the total lead-time and the test phase does not start until they are finished. This time consumption is currently affected by the specification quality but it could be possible to measure how the specification affects the time in each phase in order to get a better estimation of this value. The time consumption is in reality affected by several other factors and to model the phases in between should give a more realistic image of the real process. These phases can be modelled in a similar way as the requirement and test phases.

In the current model the time consumption in the requirement phase is controlled by the planned requirement effort while the time in the test phase goes by until the test tasks are completed without any concern of the planned test effort. In a future model the test phase can be controlled similarly to the requirement phase but this kind of model would be answering a different kind of questions. A model of this kind can only predict the software quality for a given amount of effort.

In the future it is possible to let the model illustrate the platform process instead of the product process, which was the intention at the beginning. The current model points out which parameters that need to be monitored in order to get a more reliable model. The same parameters need to be monitored when adjusting the model to the platform process, which will be possible when more platform projects have been concluded.

9.3 Conclusion

After the summary of this simulation study several different ideas about future development of the model were discussed. There are other

development possibilities that have not been considered in this chapter but there is not room for including all of them within the limits of this thesis. However, the developed model is a good foundation for further development in suitable directions.

References

[1] Ian Sommerville, *Software Engineering,* Addison-Wesley, 1996

[2] T. Capers Jones, *Estimating Software Cost,* McGraw-Hill, 1998

[3] Norman E. Fenton, Shari Lawrence Pfleeger, *Software Metrics: A Rigorous & Practical Approach,* International Thomson Computer Press, 1996

[4] Ioana Rus, James S. Collofello, *Assessing the Impact of Defect Reduction Practices on Quality, Cost and Schedule*, ProSim 2000, London UK

[5] Frederick S. Hillier, Gerald J. Lieberman, *Introduction to Operations Research,* McGraw-Hill, 1995

[6] Robert Martin, David Raffo, *A Model of the Software Development Process Using both Continuous and Discrete Models,* International Journal of Software Process Improvement and Practice, 5:2/3 June/July 2000

[7] Paolo Donzelli, Giuseppe Iazeolla, *Hybrid Simulation Modelling of the Software Process*, ProSim 2000 London, UK

[8] Robert Martin, David Raffo, *Application of a Hybrid Process Simulation Model to a Software Development Project*, ProSim 2000, London, UK

[9] Lars Randell, *A Methodology to Reduce Time Consumption in Discrete-Event Simulation Projects*, Lund University, 2000

[10] Peter M. Senge, *Den femte disciplinen,* Nerenius & Santérus förlag, 1999, (In Swedish)

[11] *Introduction to Simulation*, Powersim Corporation 2000

[12] Tarek Abdel-Hamid, Stuart E. Madnick, *Software Project Dynamics: An Integrated Approach,* Prentice Hall, 1991

[13] Peter M. Senge, Art Kleiner, Richard Ross, Bryan Smith, Charlotte Roberts, *The Fifth Discipline: Fieldbook,* Nicholas Brealey Publishing, 1999

[14] "Extend introduction", www.imaginethatinc.com, visited 001204

[15] Personal commentary: Jennie Nilsson, Ericsson Mobile Communications AB, Lund, Sweden, 001017

[16] "ECS homepage", ecs-lund.ericsson.se/lund, visited 001020

[17] PROPS - *A General Model for Project Management in a Multiproject Organization*, Ericsson Infotech AB, 1999

[18] Niklas Sundgren, *Product Platform Development-Managerial Issues in Manufactural Firms*, Chalmers University of Technology, 1998

[19] Banks, Carson II, Nelson, *Discrete-Event System Simulation,* Prentice Hall, 1996

[20] Claes Wohlin, Per Runesson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, Anders Wesslén, *Experimentation in Software Engineering: An Introduction,* Kluwer Academic Publisher, 2000

[21] Marc I. Kellner, Raymond J. Madachy, David Raffo, *Software Process Simulation Modelling: Why? What? How?,* Journal of Systems and Software, vol. 46 no.2/3, April 1999

[22] Wladyslaw Bolanowski, *Final report for <a sub-project>,* Ericsson Mobile Communications AB, 2000

[23] Personal commentary: Sarah S. Apsey, PA Consulting group, Cambridge, MA, USA, 001213

[24] "Powersim introduction", www.powersim.com/product, visited 010105

Bibliography:

Banks, Carson II, Nelson, *Discrete-Event System Simulation,* Prentice Hall, 1996

Peter Henderson, Yvonne Howard, *Simulating a Process Strategy for Large Scale Software Development using Systems Dynamics*, ProSim 1999, Silver Falls, USA

G. Kahen, M.M. Lehman, J.F. Ramil, P. Wernick, *Dynamic Modelling in the Investigation of Policies for E-type Software Evolution*, ProSim 2000, London UK

Peter Henderson, Yvonne Howard, Robert John Walters, *A Tool for Evaluation of the Software Development Process,* ProSim 2000*,* London UK

David I. Cleland, *Project Management, Strategic Design and Implementation*, McGraw Hill, 1995

Bengt Savén, *Produktionssimulering,* Mekanförbundets förlag, 1998 (In Swedish)

Per Darmer, Per V. Freytag, *Företagsekonomisk undersökningsmetodik,* Studentlitteratur, 1995 (In Swedish)

Anders Sixtensson, *System Dynamics – A Way to Improved Project Performance*, Kipling

Bo Bergman, Bengt Klefsjö, *Kvalitet från behov till användning,* Studentlitteratur, 1995 (In Swedish)

Raymond J. Madachy, Barry W. Boehm, *IEEE Computer Society Press,* www-rcf.usc.edu/~madachy/spd, visited 001025

David N. Ford, John D. Sterman, *Dynamic Modelling of Product Development Processes,* System Dynamics review, vol. 14 (1), 1998

A. Drappa, J.Ludewig, *Quantitative Modelling for the Interactive Simulation of Software Projects*, Journal of Systems and Software, vol. 46, 1999

Barry Richmond, *Systems Thinking: Critical Thinking Skills for the 1990s and Beyond*, System Dynamics Review, vol. 9, no 2, 1993

Jay W. Forrester, *System Dynamics, Systems Thinking, and Soft OR,* System Dynamics Review, vol. 10, no 2, 1994

John D. Sterman, *A Sceptic's Guide to Computer Models,* Managing a Nation: the Micro Computer Software Category, Westview Press, 1991

George P. Richardson, *Problems with Causal-Loop Diagrams,* System Dynamics Review, vol. 2, no 2, 1986

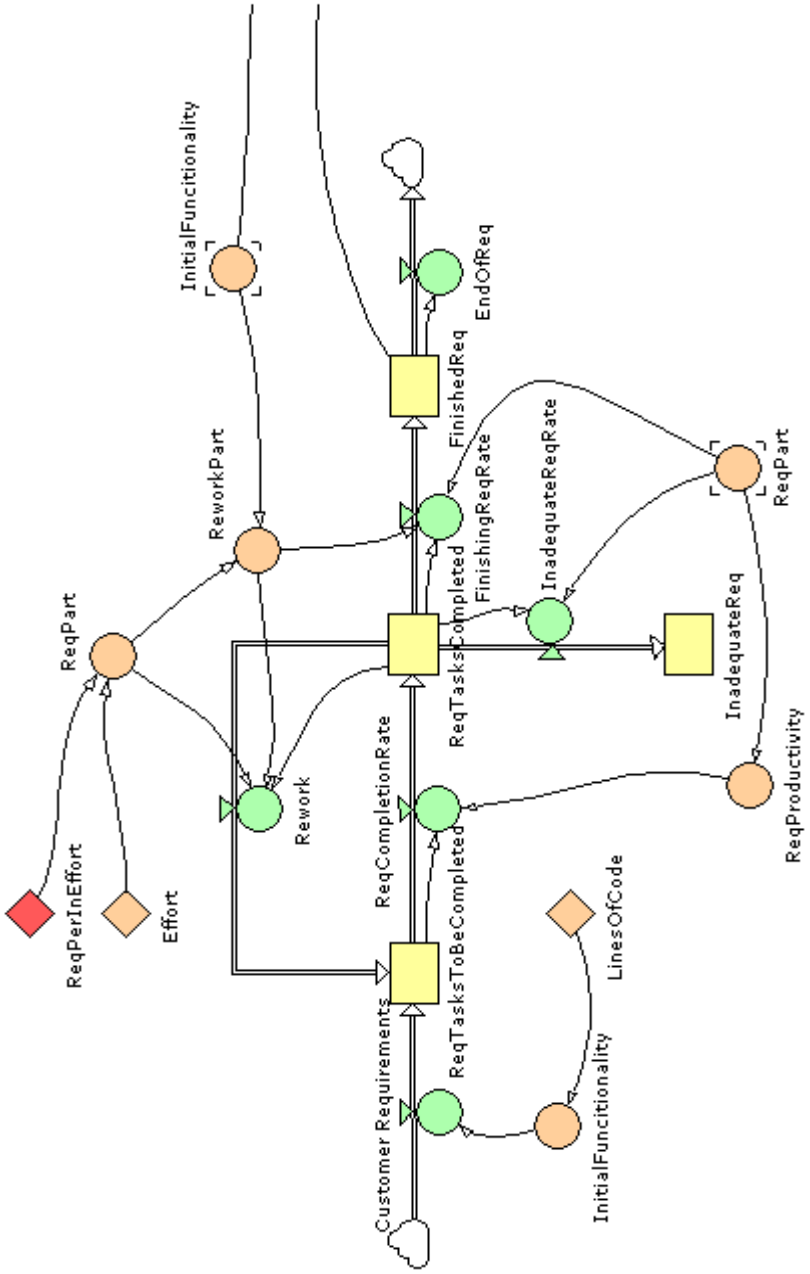
Derek Merrill, James S. Collofello, *Improving Software Project Management Skills Using a Software Project Simulator*, www.eas.asu.edu/~sdm, visited 001102

James S. Collofello, Ioana Rus, VishnuDev Ramakrishnan, Charles Musco, Kevin Dooley, *Using Software Process Simulation to Assist Software Development Organisations in Making Good Enough Quality Decisions*, www.eas.asu.edu/~sdm, visited 001102

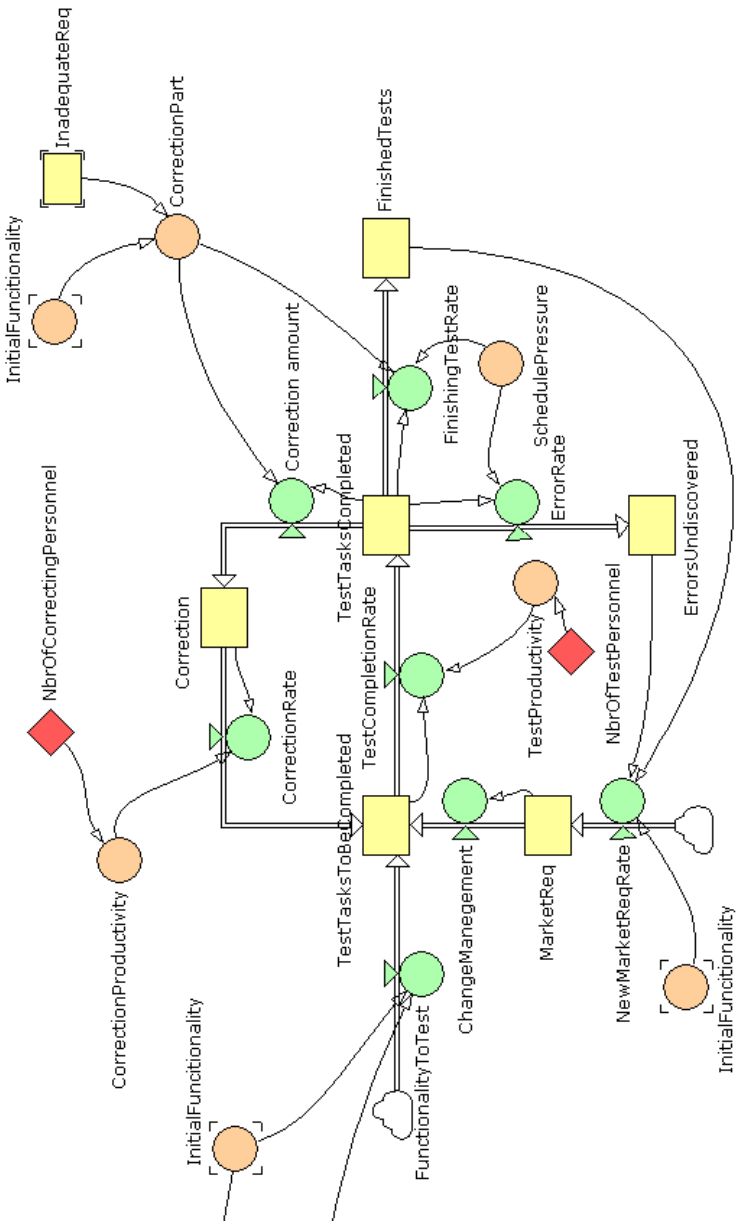
Appendix A
The Simulation Model

This appendix describes the model, both the connections between the parameters and the definition of the each parameter. The first two figures show the requirement phase and the test phase, while the third one, the Helpdesk, shows complementing parameters that is required to get a well-functioning model. This includes parameters that survey when the phases are in action and parameters that calculate the time consumption in each phase.

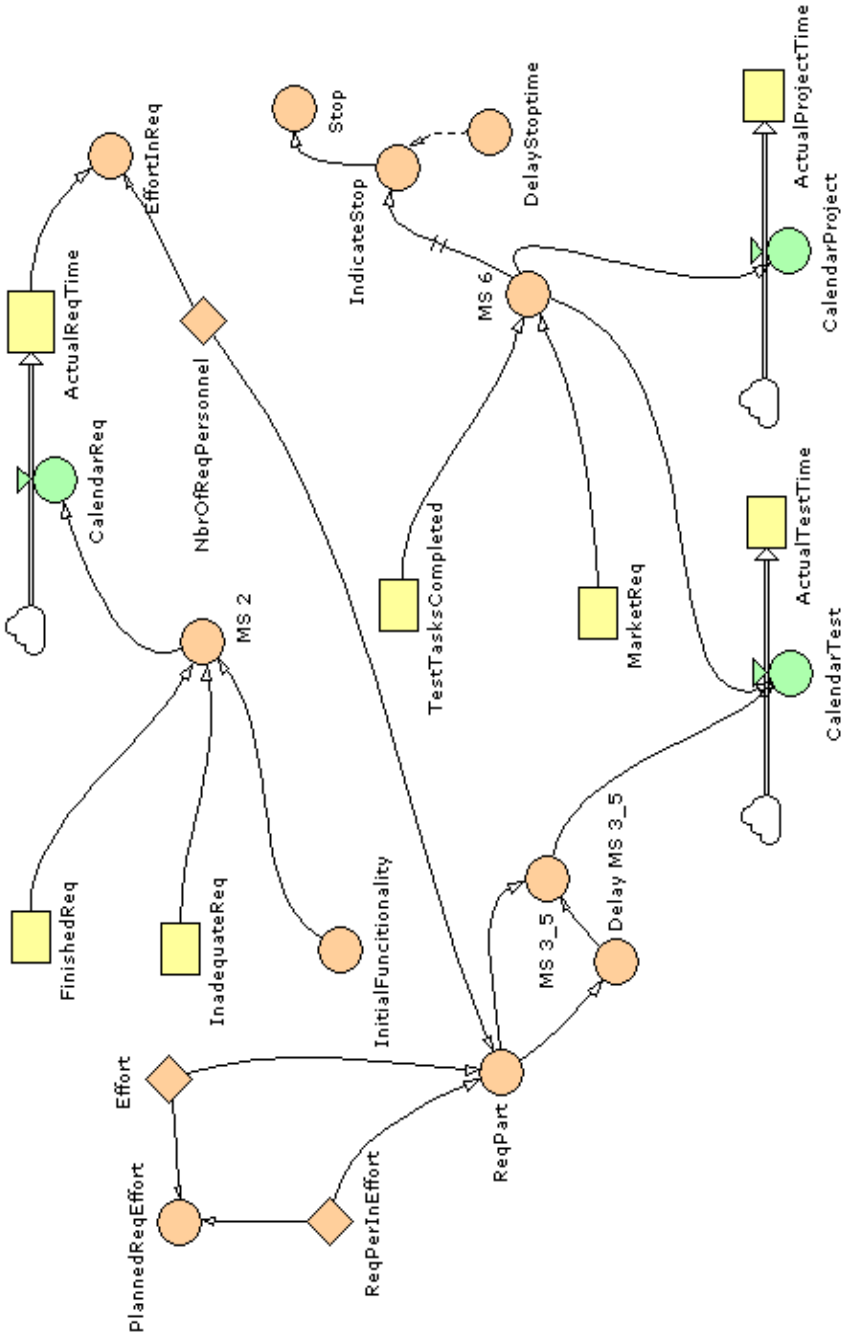
The Requirement Phase



The Test Phase



The Helpdesk



Definition of Parameters

The units used in the model are either function points, days or a combination of these. In the following text function points are abbreviated to FP.

The unit day means in the definition below calendar day, but the unit man-day is also used as an explanation of how many days it would take for one person to finish the task. This is often used in the model as the effort necessary to accomplish the task. The unit working day does not consider weekends. In some of the definitions below the constant 20 is included to transform the parameter from calendar days to working months or vice versa. In Powersim one month consists of 30 working days, there are no possibilities to change it to the normal 20 working days per month.

ActualProjectTime is an output that shows the number of days required to finish the project. The value of this level is, with help from CalendarProject, increased by one every day that MS 6=0.

ActualReqTime is an output that shows the number of days required to finish the requirement phase. The value of this level is, with help from CalendarReq, increased by one every day that MS 2=0.

ActualTestTime is an output that shows the number of days required in the test phase. The value of this level is, with help from CalendarTest, increased by one every day that MS 3.5=1 and MS 6=0.

CalendarReq keeps track of when MS 2 is reached.

```
If MS 2<1 then
     CalendarReq=1
else
     CalendarReq=0
```

CalendarProject keeps track of when MS 6 is reached.

```
If MS 6<1 then
     CalendarTest=1
else
     CalendarTest=0
```

CalendarTest keeps track of when the test phase is in progress.

```
If MS 6=0 and MS 3.5=1 then
     CalendarTest=1
else
     CalendarTest=0
```

ChangeManagement $= Min(0.8, MarketReq)$

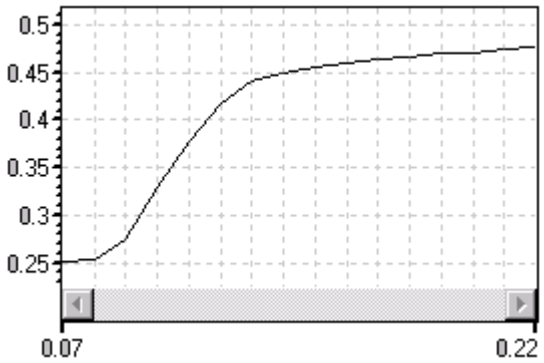
This variable considers the new market requirements that shall be specified, designed and implemented at a rate of 0.8 FP/day. The rate is iteratively calculated to get a slight effect on the time in the test phase.

Correction is a level with the initial value 0. During the simulation it contains the tasks that shall be corrected and is emptied by the flow controlled by CorrectionRate.

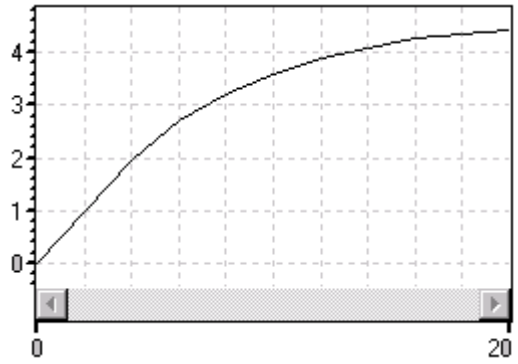
CorrectionAmount regulates the flow of tasks to be corrected. While the project is running, i.e. MS 6 is not yet reached, the flow continues.

```
If MS 6=0 then
      CorrectionAmount=CorrectionPart*
      TestTasksCompleted
else
      CorrectionAmount=0
```

CorrectionPart decides how much of the test tasks that needs to be corrected and is assumed to be an S-shaped graph with values between 25 and 47 % depending on InadequateReq/InitialFunctionality. The value 25 % is assumed to be a minimal value of the amount of defect code delivered to the test phase when InadequateReq/Initial Functionality reaches its lowest possible value. In addition to this, a bad requirement specification will increase the CorrectionPart up to 47 % as the amount of InadequateReq increases.



CorrectionProductivity decides the amount of correction that is made in one day. It is a graph depending on NbrOfCorrectingPersonnel, where one person corrects 0.5 FP/day, which is a value that has been calculated iteratively. The productivity/person decreases when the amount of personnel increases, because the communication level decreases [12].



CorrectionRate $= Min(CorrectionProductivity, Correction)$

Controls the rate of the flow of tasks from correction and is either equal to CorrectionProductivity or the remaining part of the tasks to be corrected. The unit is FP/day.

CustomerRequirements

$= Pulse(InitialFunctionality, StartTime, 3000)$

Controls the inflow of customer requirements into the requirement phase. The amount InitialFunctionality arrives at StartTime and the interval to the next pulse is 3000 days, which occurs after the project is finished since the intention is to simulate only one project at a time.

DelayMS 3.5 $= 195 * \frac{155}{ReqPart}$

Describes the time that is left for the Design- and Implementation phases, i.e. the time between MS 2 and MS 3.5. In the normal case it is 195 days, which is calculated in appendix B. The constant 155/ReqPart increases the Delay MS 3.5 when ReqPart decreases since the requirement specifications are less accurate and therefore the design and coding takes more time.

DelayStopTime is an input parameter set to an appropriate value so that the test loop has time to finish and MS 6 has not yet changed its value 1. In this case it is 70 days.

Effort is a constant that describes how much effort that was used in this particular project and has the value 278 man-months [22].

EffortInReq $= \frac{ActualReqTime}{20} * NbrOfReqPersonnel$

Indicates the actual effort spent in the requirement phase. The unit is man-days.

EndOfReq empties FinishedReq in order to transfer the functionality to the test phase.

```
If MS 3.5=1 then
      If FinishedReq>0 then
            EndOfReq=FinishedReq
      else
            EndOfReq=0
else
      EndOfReq=0
```

ErrorRate $= SchedulePressure * TestTasksCompleted$

ErrorRate controls the flow of errors and is increased by an increased ShedulePressure. When there is no schedule pressure, 5 % of the tested functionality is assumed to be incorrect but not discovered. The unit is FP/day.

ErrorsUndiscovered is an output parameter that counts the errors remaining in the software after the test phase. The unit is FP.

FinishedReq is a level where the specifications are stored when they are specified.

FinishedTests is a level where the tested and approved functionality is stored.

FinishingReqRate controls the amount of the finished specifications, which does not need to be reworked or is assumed to flow directly to InadequateReq. The unit is FP/day.

```
If Time<(StartTime+ReqPart) then
      FinishingReqRate=(1-ReworkPart-
      0.1*155/ReqPart)*ReqTasksCompleted
else
      FinishingReqRate=0
```

FinishingTestRate controls the flow of tested functionality, which does not need to be corrected, or is assumed to flow directly to ErrorsUndiscovered. The unit is FP/day.

```
If MS 6=0 then
      FinishingTestRate=(1-CorrectionPart-
      ShedulePressure)*TestTasksCompleted
else
      FinishingTestRate=(1-ShedulePressure)
      *TestTasksCompleted
```

FunctionalityToTest controls the amount of functionality to be tested. When the requirement-, design- and implementation phases are finished, all the functionality arrives to the test phase as a pulse.

```
If MS 3.5=1 then
      If FinishedReq>0 then
            FunctionalityToTest=
            InitialFunctionality
      else
            FunctionaliyToTest=0
 else
      FunctionalityToTest=0
```

InadequateReq is a level that counts the inadequate specifications in FP.

InadequateReqRate controls the amount of specifications that is incorrect. 10 % of the specifications made during the requirement phase are incorrect in the normal case. But when ReqPart is decreased from 155, the amount of incorrect specifications increases and when ReqPart increases, the amount of incorrect specifications decreases. After ReqPart has passed, the remaining functionality that is not specified is forwarded to InadequateReq since the design phase starts at this time.

```
If Time>=(StartTime+ReqPart) then
      InadequateReqRate=ReqTasksCompleted
else
      InadequateReqRate=0.1*155/ReqPart*
      ReqTasksCompleted
```

IndicateStop $= Delayppl(MS6, DelayStopTime)$

Gets the same value as MS 6 after a delay of 70 days.

$$\textbf{InitialFunctionality} = \frac{LinesOfCode}{128} = \frac{115558}{128} = 902.8$$

Transforms the amount of code into the unit function points (FP) according to equation 2.1.

LinesOfCode is a value taken from [22] and equals 115 558 LOC.

MarketReq is a level that receives the new market requirements that arrives each day. The level is decreased by the change management that completes the requirements before testing.

MS 2 indicates when the requirement phase is finished.

```
If    Runmax(FinishedReq+InadequateReq)>    0.99*
InitialFunctionality then
      MS 2=1
else
      MS 2=0
```

MS 3.5 indicates when the design and implementation phases are finished. The design and implementation starts after the planned requirement time, ReqPart.

```
If Time>(StartTime+ReqPart+DelayMS 3.5) then
      MS 3.5=1
else
      MS 3.5=0
```

MS 6 indicates when the test phase is finished, which occurs when the market requirements are frozen and the test loop is emptied.

```
If MarketReq=0 then
      If TestTasksCompleted<1 then
            MS 6=1
      else
            MS 6=0
else
      MS 6=0
```

NbrOfCorrectingPersonnel is controlled from the user interface. The initial number of correcting personnel in the test phase is calculated in appendix B, and is equal to 10 [22].

NbrOfReqPersonnel
The estimated number of requirement personnel is 3.2 and is calculated in appendix B. This value is assumed to be constant since during the requirement phase the project delay is not yet discovered and no extra personnel is needed.

NbrOfTestPersonnel is controlled from the user interface and is the amount of people that performs the software testing. The initial value is calculated in appendix B and is equal to 3 [22].

NewMarketReqRate controls the inflow of new market requirements, which is 2% per month and is based on information from [2]. New market requirements are arriving from the start of the project.

```
If   (FinishedTests+ErrorsUndiscovered)<((1+0.3*
0.02)*InitialFunctionality*(Time-StartTime)/20)
then
      NewMarketReqRate=0.02*
      InitialFunctionality/20
else
      NewMarketReqRate=0
```

The constant 0.3 is iteratively calculated in order to freeze the new market requirements at a suitable time, at approximately MS 4.

$$\textbf{PlannedReqEffort} = Effort * \frac{ReqPerInEffort}{100}$$

The planned amount of man-months in the requirement phase is shown in the user interface.

ReqCompletionRate

$$= Min(ReqProductivity, ReqTasksToBeCompleted)$$

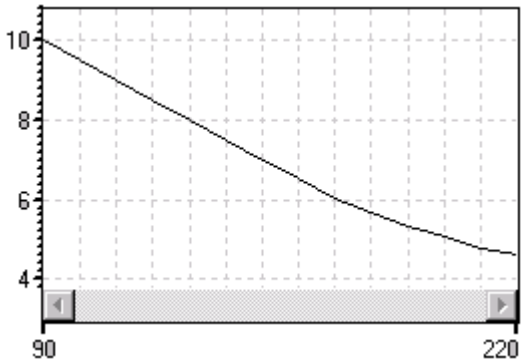
Controls the completion rate of the specifications and is either equal to the productivity or the remaining part of the uncompleted tasks. This variable solely regulates the time spent in the requirement phase. The unit is FP/day.

$$\textbf{ReqPart} = \frac{ReqPerInEffort/100 * Effort * 20}{NbrOfReqPersonnel}$$

Calculates the number of calendar days in the requirement phase.

ReqPerInEffort is controlled from the user interface and has an initial value of $\frac{25}{278} = 9\%$ where 25 is the approximate number of man-months spent in the requirement phase and 278 is the total number of man-months in the project taken from the report [22].

ReqProductivity decides the amount of specifications that is made in one day. It is a graph depending on ReqPart, the number of personnel in the requirement phase is assumed to be constant. This graph is iteratively calculated from the length of the phase and the functionality to be specified. The graph is not linear since the time in the requirement phase depends on the amount of rework, which does not depend linearly on ReqPart. The steeper part of the curve depends on the higher amount of rework necessary when the planned time in the requirement phase is shorter than the normal value. When the time is increased there is less rework made and therefore the curve levels.



ReqTasksCompleted is a level where the specifications are gathered before forwarded as finished, inadequate, or incorrect functionality.

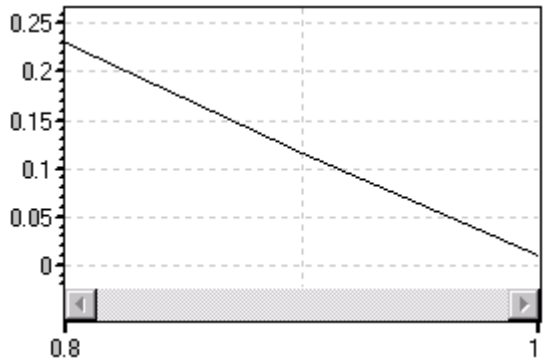
ReqTasksToBeCompleted is a level that gathers the customer requirements.

Rework controls the amount of specifications that need to be reworked. The unit is FP/day.

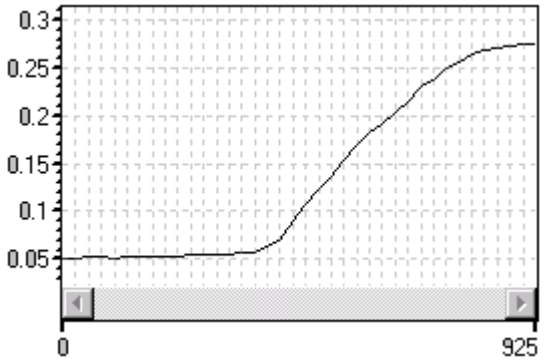
```
If Time<(StartTime+ReqPart) then
      Rework=ReworkPart*ReqTaskCompleted
else
      Rework=0
```

ReworkPart is a graph that depends on the equation for the specification accuracy, i.e. the x-axis is equal to $1 - \frac{InitialFunctionality}{ReqPart} * 0{,}02$

The constant 0.02 is empirically calculated. The fraction decides the amount of time available for each function point that shall be specified, which can be seen as a measure of the quality. The ReworkPart is the percentage that shows how much of the specifications that needs to be reworked. This value is decreased when the specification accuracy is increased.



SchedulePressure is a graph, which affects ErrorRate to increase when the deadline is approaching. The initial value is 0.05.

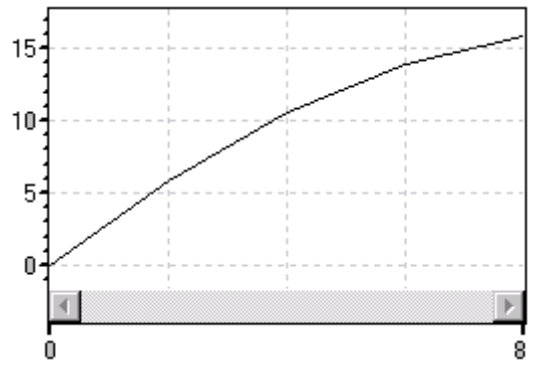


Stop $= StopIf\,(IndicateStop = 1)$

Stops the simulation when IndicateStop reaches 1. This mechanism is used to avoid MS 6 from changing its value when it has reached 1. This mechanism is necessary since MS 6 depends on Time, which increases throughout the simulation.

TestCompletionRate
$= Min(TestProductivity, TestTasksToBeCompleted)$
Controls the flow of functionality to be tested. The unit is FP/day.

TestProductivity is a graph that depends on the number of test personnel, in which one person tests 2.9 FP/day. The productivity/person decreases when the amount of personnel increases, because the communication level decreases.



TestTasksCompleted is a level that gathers the tested functionality and forwards it as finished, inadequate, or incorrect functionality.

TestTasksToBeCompleted is a level that gathers the functionality that will be tested.

Appendix B
Values Used in the Model

This appendix presents data from [22] considered being important in this thesis. Some units of the data have been changed to suit the units in the simulation model. Assumptions and calculations based on the report are also included in this context.

Table B.1 presents the planned and actual milestone passages for this particular project. The table only presents the process phases until MS 6 since the following phases are excluded from the model.

Table B.1 Project time in working days

Period	Planned	Actual
MS 0-MS 2	80	80
MS 2-MS 3	125	150
MS 3-MS 3.5	135	200
MS 3.5-MS 4	20	20
MS 4-MS 5	85	65
MS 5-MS 6	40	70
Total time MS 0-MS 6	**485**	**585**

The milestone passages according to table B.1 are not used in the model since they do not indicate when the specific tasks are completed. Instead assumptions have been made to get a model that is closer to reality.

The requirement specification documents were not fully completed at the milestone passage MS 2, but was assumed to be finished at approximately 155 days after the project started. The test phase started, in reality before MS 3.5 was reached, at approximately 350 days from the date the project started, which means that the phase stretched over 585-350=235 days.

The model can not consider over-lapping phases. This implies that the time spent in the design and implementation phases that were not over-lapped with other phases stretched over 585-155-235=195 days.

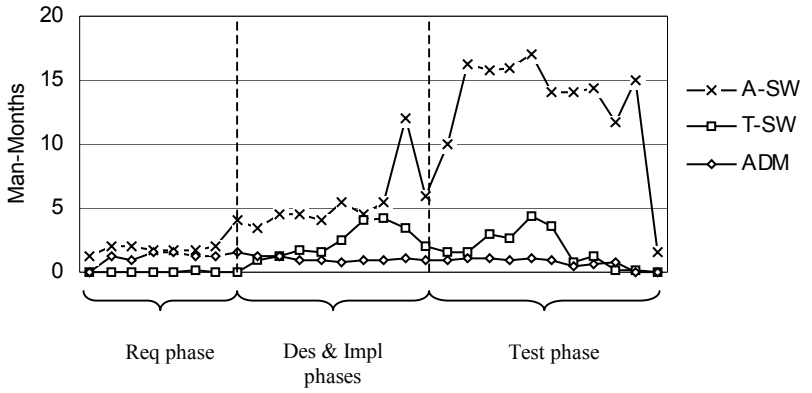


Figure B.1 Work distribution in time

The work distribution in time is presented in figure B .1. Three categories are shown: Application Software (A-SW), Test Software (T-SW) and Administration (ADM). The A-SW includes all the effort spent on specification, design, implementation and fault correction. The T-SW contains writing test specifications and performing tests. The ADM is the object leader effort.

From figure B.1 an estimate of the amount of personnel in each phase is made by counting the amount of effort spent in the phase and divide it by the number of months. For the requirement phase the effort is approximately 25 man-months divided on 155 working days, which is equal to 3.2 persons working full-time in this phase. In the test phase the effort is approximately 120 man-months spent on correction, during 235 working days, which equals 10 persons. The effort spent on testing is approximately 36 man-months divided on 235 working days, which equals 3 persons.

Appendix C Tables of Results

This appendix includes the results from the series of simulations in case 2.

Table C.1 The test time when the number of testers is not a limiting factor

NbrOfCorrectors	8	9	10	11	12	13	14	15	16	17	18
TestTime	250	235	224	214	206	201	196	191	188	186	184

Table C.2 The test time when the number of correctors is not a limiting factor

NbrOfTesters	2.5	3	3.5	4	4.5	5	5.5	6
TestTime	280	236	211	194	180	170	162	155