LUND
UNIVERSITY

Simulation of Market-Driven Requirements Engineering Processes

Master thesis at Lund University
Faculty of Engineering
Department of Computer Science
Author: Christofer Tingström
Supervisor: Martin Höst
Karlshamn 2008-02-21

Abstract

Market-Driven Requirements Engineering (MDRE) is more and more used in the software industry. MDRE differs to the traditional Requirements Engineering as no specific customer does exist. The producer of the software is interested in releasing a product to the open market and there are several aspects to consider. One of the major challenges in MDRE is the time to market parameter. The trade off between the release date and the features in the product is difficult to determine. The producer wants the most important features for the customer to be included in the product and as early as possible. The producer would like to optimize the MDRE process and due to that also optimising the time to market parameter. Simulation modelling is a good technique to investigate and measure the impact of the new processes.

The objective of this thesis is to design, model and develop a framework for MDRE processes. The purpose of the framework is to be possible to model a MDRE process. In addition, it should also be possible to simulate the model and obtain measurements of the models. One of the major challenges of the thesis has been to model the framework with the right level of abstraction. In addition, the framework should be easily extendable to fit another software process. The framework should also be able to implement the REPEAT and RSQ model.

The methodology of the thesis can be divided into three parts. The first part of the methodology was the literature study where the domain knowledge of Market-Driven Requirements Engineering and simulation models was gained. Secondly, an evolutionary development approach was used to model, design and develop the framework. Finally, the framework was validated and verified. The validation and verification was carried out by develop the REPEAT and RSQ model. Then the measurements of the models were compared with already published results.

The framework was developed with the objectives in mind and the framework was divided into three different layers. The first layer was the simulation layer which contains components for any simulation model. The second layer was specific for software processes and the third layer was specific for Market-Driven Requirements Engineering. It was found out during the thesis that the REPEAT and RSQ model could be implemented with the framework. The REPEAT model required more effort than the RSQ model due to the reason that the REPEAT model was more complex. The REPEAT model could be implemented when an additional layer was developed and added.

Acknowledgements

My warmest thanks go to my supervisor Martin Höst for his inspiring guidance and support of this thesis. I also would like to thank Björn Regnell which introduced me to the Market-Driven Requirements Engineering field and his support and advice during the thesis.

Contents:

1. Introduction

1.1 Background

Software engineering is not an old discipline, actually only a decade ago it was not considered as an engineering discipline [19]. Software engineering is complex and differs from other engineering disciplines.

A classic example is to compare a software project with constructing a bridge. Firstly, bridges have been built for centuries and the processes are very stable and have been verified over a long period of time. However, there are few other significant differences. The bridge is more concrete artefact than a software artefact. A late new requirement for a bridge is easier to estimate the time required to complete compare to add a new function to the software product. In the software product case the estimated time required depends on the flexibility of the architecture and can be more difficult to measure. Furthermore it is difficult to fully understand the impact of a new requirement on the architecture before implementation is in progress.

In addition estimating the completeness of the bridge is easier than for the software product. In general you can visually see how far you have reached in building a bridge in contrast to software, where the developers see the number of requirements implemented. However, these are not reliable to use to estimate the amount of time to finish the product.

In every engineering discipline a process is essential and benefits the organization instead of using an ad-hoc process. In software engineering several software processes are used, e.g. the waterfall model, the incremental model and the evolutionary model [20].

Today in the software engineering industry Requirements Engineering (RE) is a topic that is widely discussed. Requirements engineering is normally one of the first stages in a software process and is the foundation of the project. The tasks that have to be carried out in RE are to elicit requirements, prioritise the requirements, write a specification and validate the requirements [9]. The elicitation process is to find and formulate requirements. There are different techniques, which are described in 3.1.1. The validation of the specification is important as the specification shall reflect the customer's expectation. In addition to this the developer shall understand it and the stakeholders shall receive what they expect. There are a few quality criteria the specification shall fulfil, described in 3.1.2. The specification is the output in the RE stage and will serve as an input to the designers and developers of the software. It has come to attention that the amount of time spent on Requirements Engineering can save time and money later in the development process [20].

Market-Driven Requirements Engineering (MDRE) is an interesting topic which is different compared to the traditional Requirements Engineering. MDRE is to develop

software for an open market. In MDRE there is no customer compared to bespoken development and due to that there is no feedback to receive from the customer. One major challenge is to optimize the time to market parameter [7]. It can be the difference between success and disaster to release the software at the open market at the right time. The time to market parameter can in best case generate large revenue for the company and add additional market shares.

In addition, the success of the developing organisations depends on several measurements [4] as customer satisfaction, product reviews and the effort from the marketing department. These measurements can in best case generate increasing market shares and large revenues. The customer satisfaction and the product reviews are in general connected to the product quality and the features of the product. Due to that another important and difficult challenge is to be able to select and prioritise the right set of requirements for the product. Furthermore it is difficult for the producer to know which requirements customers are interested in as well as which requirements generate the most possible customer satisfaction. The producer is normally performing a market analysis to establish which needs the customer has or can be interested in. The results of the analysis-stage together with new possibilities from new technology establish a good platform to elicit requirements.

The producer is also interested in the effort required for a requirement and which value the requirement has in the end to the potential customer. In the best case a producer would like to select the requirements which has low effort but generates a high value for the customer. The more effort that is required for a requirement the more resources and funding the requirement require. However, to be able to select the right requirements in an early stage in the MDRE process has shown to be difficult [14]. The producer cannot receive any feedback on the value of a requirement until the product has been on the market for a longer period. The effort is usual difficult to estimate and cannot be determined until the construction of the requirement is completed.

The uncertainties to implement the correct requirements cause the producer to rely on their MDRE process and their resources, skills and experience. There exists a few metrics which can be useful. The producer can, for example, measure product quality and decision quality. The process itself is important for the producer and it is important to be able to allocate the resources in the best way to generate high product and decision quality.

There are several ways to improve the process e.g. through pilot studies and controlled experiments [22]. However, these methods require many resources. Furthermore there is a less expensive solution in terms of funding and effort and that is the use of simulation modelling. Simulation models reduce the risk of implementing processes which will not improve the process quality. In MDRE simulation models are interesting as the research field is new. Simulation models can be a good idea to demonstrate the strength with MDRE process for stakeholders which have doubts about new MDRE processes. In addition, a process which is not improving the quality

can be of great risk especially in a large organisation with many people and this can in the worst case cause serious problem which will deteriorate the process.

Simulation modelling is cost effective in terms of effort and resources compared to controlled experiments and pilot studies. However, it does require resources and effort and can be complex and difficult. The idea with this thesis and its belonging framework was to establish a platform for simulation modelling of MDRE processes. One of the keys in software development is to reuse code and the goal of the framework is that it will be used instead of simulation models created from scratch.

The disadvantage with simulation models is that it might be problematic to present an accurate picture of the reality. Simulation models are based on a set of assumptions which are important to consider before a transition from simulation model to the real process.

1.2 Purpose and Scope

The purpose of this master thesis is to create a framework for modelling of Market-driven Requirements Engineering processes. In addition these models should also be possible to simulate. Hence is should be possible to model any MDRE process and then simulate the process by using the components in the framework and if necessary extend it with special components that easily are integrated in the framework. The usability and support of MDRE framework is tested by implementing two models, the RSQ model [17] and the REPEAT model [15]. The purpose is also to establish a good platform for MDRE processes. In addition it should be flexible, and easily extended.

The framework and the simulation are very close connected with each other and a distinct line between the two is not possible to set. The simulation model is close to the MDRE modelling and a few MDRE model components contain simulation model components.

The scope of this master thesis is to develop a framework and the models created from the framework should be possible to simulate. Input data as well as output data from the framework shall be easy to configure. The framework shall also establish a good platform which can be extended to fit other use cases and thereby it should be possible to model an arbitrary MDRE process.

1.3 Target Group

The principal target group of the thesis consists of senior students with knowledge in development processes in software engineering, researchers from the Departments of Computer Science, Faculty of Engineering, Lund University and other researchers in software engineering.

1.4 Glossary

Bespoken development	Traditional software development with a customer

MDRE	Market-Driven Requirements Engineering
MTTM	Mean time to Market
REPEAT	Requirements Engineering ProcEss At Telelogic
RSQ	Requirements Selection Quality
Stakeholders	Groups or individuals which have an interest or are affected bye the project
UML	Unified Modelling Language
User	The user of the framework

1.5 Outline of the Thesis

Chapter 1 is a background chapter, including introduction, scope and purpose of the framework, target group, glossary and outline of the thesis. The purpose of the chapter is to give the reader and introduction to MDRE and simulation modelling.

Chapter 2 is a description of the methodology that has been used in this thesis. In addition the objectives of the framework are also presented here.

Chapter 3 contains the theoretical background of this thesis as well as related work. The focus will be on MDRE and simulation modelling. Requirements Engineering is also presented.

Chapter 4 presents the framework and the simulation models. The modelled, designed and implemented framework will be presented. A UML diagram will be displayed as well as the details of each component in the framework. In addition, the theory of each simulation model will also be presented.

Chapter 5 presents the results of the implemented simulation models. The verification as well as the validation of the RSQ model and the REPEAT model will be presented.

Chapter 6 presents the conclusion for the work of this thesis. It also describes a few design issues. Finally, the chapter raises question for future modelling of other software processes and extension of the framework.

2 Objectives and Methods

2.1 Objectives

This thesis has been investigated around three questions:

Q1: Which level of abstraction shall the framework have?
This question is difficult and important. The abstraction level determines several issues in the framework. Firstly, if the abstraction level is high the user might find it impossible to create a simulation model with reasonable assumptions. However, if the abstraction level is low the user might find it difficult to create a simulation model, as there are many details to consider. The abstraction of the framework is a trade-off between the levels of assumptions for a simulation model.

Q2: Which architecture shall the framework have to fit general software processes?
The framework is used for MDRE process. However, the idea was not to model it explicit for MDRE process. Hence, other software processes should be able to model with reasonable extension. The idea was to have different layers in the framework to fit other processes, e.g. a test process.

Q3: Can the REPEAT model and RSQ model be implemented with help of the framework and the simulator engine?
The initial thought with Q3 is to gain knowledge of the flexibility and ease of use with the framework. The REPEAT model [15] and RSQ model [17] is described later in this thesis. If these models are possible to implement, the framework can be considered to be useful for simulation models.

2.2 Methodology

This master thesis has been carried out with three different methods.

The first part of the methodology was the literature studies [6]. The literature study is essential to gain domain knowledge for the thesis. The literature was focusing on two fields, simulation modelling and market-driven requirements engineering. The major part of the literature was research papers.

The second part of the methodology was the evolutionary development approach [6] of the framework. The first task is the initial specification. The specification contains the requirements of the framework. The implementation of the specification is the modelling, design and development of the framework.

Table 1 shows the task that has to be carried out in each step. In Table 1, the input to each step and the output after the task has been completed are also displayed.

Task	**Input to Task**	**Output from task**
1. Initial Specification	Initial ideas	Initial specification of the framework

2. Feedback from Supervisor	Specification	Changed specification according to the feedback from supervisor
3. Implement the specification	Specification	Implemented model
4.Feedback from model	Implemented model	Changed specification according to feedback from the model itself.
5. Return to point 2.		
6. Obtain a framework	Implemented model	Framework

Table 1. Evolutionary development iteration.

When the thesis subject was determined the specification was not completed. The specification has evolved for each one of the iterations.

The three first steps are self explained. The 4th step has been giving feedback in several ways. The implementation gave feedback either from use case, test cases or observations of the behaviour of the executed simulator. The test cases tested each method. The use cases verified that the different components were integrated. The execution was necessary to evaluate and verify the framework.

There have been two sources to gain knowledge and information for a new specification. The first one is the prototype itself which has generated new idea for the framework during the previous implementation. Secondly, the supervisor has been very useful with extensive feedback on the prototype.

The third part of the methodology is to validate the result, in this thesis the framework. It is important to validate that the framework suits its purpose, to model a MDRE process and then simulate the model. The framework has several purposes, see 1.2 and to validate these purposes the RSQ and the REPEAT models have been modelled and execute. Furthermore the above mentioned models also need to be validated. The validation of the RSQ and the REPEAT models are to use a known set of input data which generates a known set of output data [6]. The simulation model is the executed and the output data is compared with the known output data. The input data and output data has been collected in different publications.

3 Theoretical Background and Related Work

3.1 Software Requirements Engineering

Requirements engineering is one of the first stages in the software development process. Requirements engineering is an important process as the requirements obtained will be the foundation for the software project and it is essential that the customer and the developers understand each other. In worst case, if the developers and customer do not understand each other, the customer will obtain a product that he does not want and neither can use as it is not suited for his task.

Requirements engineering is normally a process which contains a few stages e.g. elicitation, specification and validation which are presented below.

3.1.1 Elicitation

The objective of elicitation is to find and formulate requirements. An elicitation starts with elicit the goal for the system, then present work and present problems.

Several possible solutions of handle the details of the system are investigated and finally transforming the issues and possibilities into requirements.

Elicitation can be a problematic process with several difficult issues, involving several stakeholders with different and conflicting interests. There is also a general resistance to changes. One of the major problems is that the stakeholders cannot express what they need in their software. Another issue is when the stakeholder specifies a solution instead of a demand. A few examples of elicitation techniques [9] which can be good to gain requirements are stakeholder analysis, interviewing, observation, brainstorming and prototyping. Of course, there exist several other techniques and which one to use depends on the specific case.

The requirements that are generated are normally classified into four different types [9]. The first type is the data requirements. This requirements focus on what data should the system input and output and what data shall the system store internally. The second requirements are the functional ones. These describe which functions the system has. Furthermore the quality requirements are also known as the non-functional requirements which describe how well the system performs its intended functions. Finally, there are the managerial requirements. These requirements are in an area between requirements and contractual issues e.g. the price, when to pay and legal responsibility.

3.1.2 Specification and validation

A specification is the document that contains the requirements. The specification may work as a contract between the customer and the developers. The specification is normally what a software developer use as foundation for their development of the software. A good specification fulfils the following quality criteria [9]:

- Correct: Each requirement is correct and reflects a need.

- Complete: All necessary requirements are included
- Unambiguous: All parties agree on meaning.
- Consistent: All parts match in the sense that they do not conflict.
- Modifiable Easy to change, maintaining consistency.
- Verifiable: Possible to see whether requirement is met.
- Importance: Each requirement should state its priority.
- Stability: Each requirement should state its expected frequency of changes.
- Traceable: A requirements is traceable if you can see where it comes from and where they are used in design and code.

There are a few methods to use validate the specification [9], e.g. check specification in isolation and check against surroundings. When a specifications is checked in isolation no other sources are consulted than the specification itself. The check against surroundings is the contradiction and other sources are consulted. This means in practice that customer, users and developers makes the checks [9].

Firstly, to validate the specification in isolation there are a few techniques. The contents check is one and the purpose is to see if everything is in the specification. A typical specification normally contains the following [9]

- Introduction
- System goals
- Data requirements
- Functional requirements
- Handling of special cases
- Quality requirements (non-functional requirements)
- Other deliverables
- Glossary

In addition a structure check can be a good technique to see if the pattern is correct. A good pattern for a requirement [9] is if there is a number or ID, the requirement is verifiable and the purpose is stated. There should also be examples of ways to meet the requirement and plain-text explanations of diagrams. The importance and stability should be stated and rather cross references than duplicate information. It should also be indexed and an electronic version should be available

The other technique in isolation is the consistency check. The purpose is to look for missing parts or inconsistence.

The second method is to validate the specification with its surrounding. There exist several techniques [9]. A review is one and goal-requirements tracing is another. In goal-requirements tracing checks are done. Are all the goals and critical issues are covered by requirements? Also are all requirements justified by meaningfully purposes? Furthermore is risk assessments, how risky are the specification to fulfil for the developer? Can the deliver the expected outcome?

3.2 Market-Driven Requirements Engineering

Market-driven incremental product development is becoming increasingly commonplace in software industry [4]. Incremental product development is planned and executed with the goal of delivering an optimal subset of requirements in a certain release [4]. Due to market-driven development has increased, MDRE has also increased.

The MDRE process is different from traditional requirements engineering. The major difference is the stakeholding and schedule constraints [7] as well as managing the constant flow of new requirements [7]. There exist no customers or no defined set of users, only potential customers and users. Traditional Requirements Engineering assumes a customer which is ordering the product and is responsible for the financial. In MDRE the major stakeholder is the developing organisation and they decide which requirements shall be included in the next release.

According to [7] there exist a several challenges in MDRE. However, the challenges are not only of technical nature, there are also social and organisation challenges. Firstly, the organisational challenges are co-ordination and that the communication is enforced. Secondly, the social challenges are how the development department and the market department are communicating and how to encourage people to improve requirements engineering in the organisation. Finally, there are also technical issues as release planning, techniques for requirements prioritisation and effort estimation as well as requirements traceability and inter-independency problems. In addition, challenges of the simple techniques all so exist as writing understandable requirements. The problem is that many people with different backgrounds commit requirements to the product e.g. developers, users, marketing analysts and customers. They use different languages and notations.

In any MDRE process there are several keys to success. Firstly, it is important to implement requirements that are of high value to the customer. Furthermore competition and price of product are also important. In MDRE there is always a pressure on the time-to-market for the product. The major challenge and key to success in MDRE and the goal of the company is to deliver the right product at the right time [7].

The MDRE process characteristics are elicitation, specification and validation, release planning, process quality and managing the constant flow of new requirements. These characteristics will be more described in the sections below.

3.2.1 Elicitation

The elicitation in MDRE is different to the traditional elicitation with custom specific software. In the MDRE case there are no customers, only potential customers. Hence the customer does not provide any requirements. Instead requirements can be elicited from opportunities provided by new technology. The developers are also inventing requirements [14] which can be based on strategic business objectives. The domain knowledge and product vision are essential for the developers to be able to invent requirements.

Furthermore a good opportunity to elicit requirements is to collaborate with key customers. Market analysis is also important which helps the producers to understand how the customers rate certain requirements as well as to understand their selection process.

3.2.2 Specification and Validation

The specification differs from the traditional specification; the language is less formal and natural language is used [16]. During the elicitation phase a significant number of requirements are elicited, hence the prioritisation is essential and a key element in the MDRE process. The validation needs to fulfil the same criteria as the validation in 3.1.2.

3.2.3 Release Planning

Release planning is another key in MDRE. An important issue is the strategic decision of what to deliver and when [4]. Release planning involves prioritization and cost estimation. It is important for a release that the estimations are accurate due to under-estimation of effort required for implementing a requirements can cause heavy delays which may cause lost of market shares to competitors and revenues. However, over estimation will cause less requirements to be implemented and this can have the similar effect as under estimation. The release planning tries to estimate the development efforts and expected revenues and the priorities for the end user. It is of major interest for the developer to frequently deliver new releases with improved functionalities. Firstly, keep the existing customer and as well gain new customer from the competitors.

3.2.4 Process Quality

It is always important in any process to measure the quality of the process and constantly try to improve the process. There are a few key issues [6] concerning process quality and the improvement.

The first one is the effect of the improvement of the process. What will the organisations gain with the new improvement of the process? Will the quality of the process improve and can the improvement enable new things which were not possible before? Furthermore the costs of the improvement of the process are always essential and the benefits of the improved process quality need to match the cost of the improvement, the costs can be educating the employees, investing in new technology. Over a longer time period the organisations need to generate positive revenue and the purpose of the improvement of the process is in general to increase the profit. The usability is also important; the users of the process need to easily understand the improvement and easily do their tasks. In improvements of the process and to increase quality, it is interesting how much of the process can be completely automatic and without involvement of the employees. The last one is the acceptance of the new improved process from the employees in the organisation. In general people are reluctant to accept new methods of doing their work and it is important that the employee accept the improvements.

In MDRE processes the process quality is related to the quality of the artefact that is produced from the process. Hence it would be possible to measure the quality of the artefact and thereby receive the process quality. A good approach is to measure the decision quality. A good model to use is the alpha/beta model [17]. The alpha/beta model has divided the decision into four cases. The alpha requirement is a requirement that idle is selected as it is a high quality requirement, meaning providing a high value for the customer at a low effort. The beta requirement is the "unwanted" requirements which should not be selected. These requirements provide low value to the customer to a high effort.

		Decision	
		Selected	Rejected
Requirements Quality	Should be selected(α)	A Correct selection ratio	B Incorrect selection ratio
	Should not be selected(β)	C Incorrect selection ratio	D Corrent selection ratio

Table 2. Alpha/beta model.

This model enables a few possible metrics.

Product Quality $Q_P = \frac{A}{A+C}$

Decision Quality $Q_D = \frac{A+D}{A+B+C+D}$

Golden Grain Ratio $G = \frac{A}{A+B+C+D}$

The product quality is the share of alpha requirements that are implemented. The decision quality represents the share of correct decision (A and D) in relation to the total number of decisions. The golden grain ratio is the share of correct selected requirements divided with all selected and rejected requirements.

The major problem with this model is that it is very difficult to define a requirement as an alpha or beta requirement. Normally this can only be determined when the product has been on the open market for a longer period and a market analysis has been conducted.

In addition there exists another possibility to analyse the decision. It includes two parameters, effort and value. An idle situation any producer would like to prioritize all requirements with low effort and high values.

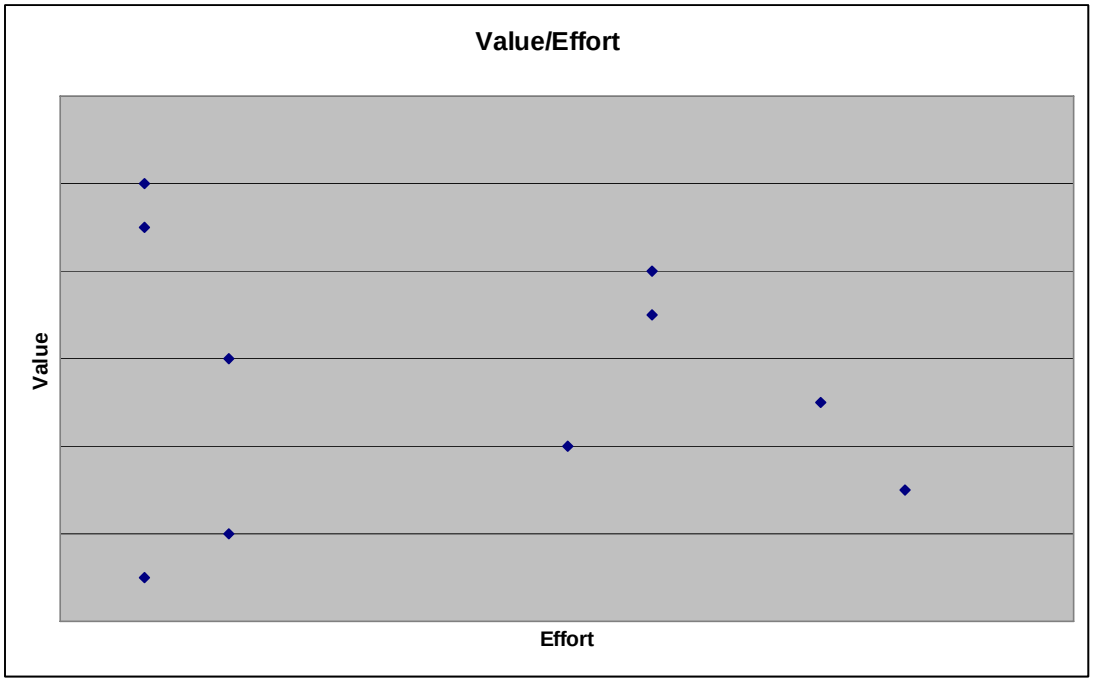

Figure 1. Value/Effort diagram.

In the Figure 1 is an example of the value/effort diagram. It is the producer that determines which requirements to implement. In a value/effort diagram it is most interesting to focus on the requirement in the top left corner and the producer shall avoid the requirements in the right bottom corner. In this sample there are three requirements which require low effort and have high values. These three requirements are of interest to the producer and further analysis is of interest

Furthermore there exists a problem with the value/effort diagram and it is not possible to have the actual value and effort at the decision stage. The value will normally be known to the producer after a longer period at the market. The effort will firstly be known after the construction phase. The possibility the decision maker has, is to estimate a value and an effort. This can cause problem as the estimations will contain errors which can cause the wrong requirements to be implemented. This is shown in Figure 2.

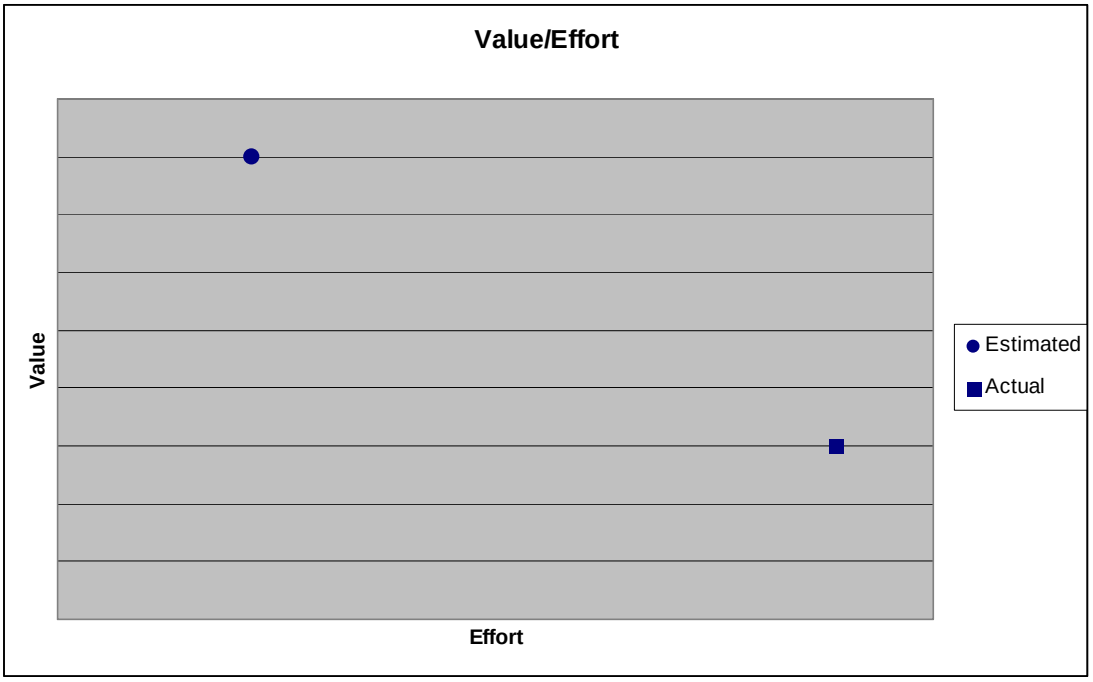

Figure 2. Value/Effort with estimated and actual value and effort.

It should also be known that value and effort does not always depend only on the requirements itself, but also its relations to other requirements

3.2.5 Managing the constant flow of new Requirements

In MDRE the product is normally not finished with the first release. It is normal that the first release will be followed of new releases in contrast to bespoken development. In addition a constant flow of requirements needs to be handled. An issue is to be able to manage the requirements in the best possible way. It is important to record new ideas from e.g. developers, customers, marketing department and users.

A repository for storing the requirements can be a good idea, where high value and low effort requirements can be selected and implemented in the next release. The selected requirements will be the foundation of the next release.

The model RAM [4] is another approach which instead of using one repository where all the requirements are stored, the RAM model has multiple levels of abstraction: There is the product level (goal), feature level, function level and component level requirements. The advantage is that it is easier to compare requirements as they are homogenous regarding the abstraction level [4]. The organisation using the RAM model is working with requirements market-driven and product oriented instead of being project oriented. The requirements are the basis for product development [4]

3.3 Software Process Simulation Modelling

3.3.1 Introduction to Simulation Modelling

Simulation modelling and analysing is a topic which has become popular [2]. The simulation and analysing technique is used to improve or investigate process performance. The technique is very cost-effective compared to carrying out controlled experiment and pilot studies. Simulation modelling requires less allocation of resources compared to the other mentioned techniques. This results in that simulation modelling and analysing is used in projects in several manufacturing and service sectors [2].

Simulation modelling and analysing is the process of creating a mathematical model of physical systems and then execute experiments on the model. Simulation models are analysed and serves as support for process or resource decisions.

The purposes of simulations are many [2]:

- Gaining insight into the operation of the system
- Developing operating or resource policies to improve system performance
- Testing new concepts and/or systems before implementation.
- Gaining information without disturbing the actual system.

Gaining insight into system when operating can be useful when the system is complex and the interaction between the components can be dynamic. An analysis conducted on a static and isolated system will be problematic or even impossible.

A system already exists and is running. The system may be improved if another resource allocation or different priorities are used. The outcome of the improvement can be developing new operating or resource polices

Testing new concepts can be the purpose for several reasons. A system might not exist and what impact will the system do on its environment. Hence, can a new system easily interact with other systems and processes and will the system deteriorate or improve the existing systems and process, then simulation modelling and analysing is a good method. The cost of modelling a system is very small compared to a capital investment.

Finally, simulations models can be the only method to gain information with a system which should not be disturbed. A few systems are very sensitive to any operation or resource change to analyse the system. A classical example is an airport security checkpoint.

3.3.2 Advantages of Simulation Modelling

There are several advantages of simulation modelling [2]. The mathematical models are executed on computers; hence the experiments can be conducted on compressed time. This is a major advantage as some systems can take months to complete but with help of the computer and simulation models the results can be obtained faster.

This gives the opportunities to experiment with different configurations in a reasonable time.

In addition, simulation models can easily be demonstrated, a few simulation software packages allow the user to dynamically animate the model which can increase the understanding of the model.

3.3.3 Disadvantages of Simulation Modelling

Although simulation modelling is useful there are disadvantages [2]. The disadvantages are not directly associated with the modelling and analysing of the system.

Firstly, a simulation model cannot give accurate output data if the input data is incorrect. It does not matter how good the simulation model is if the input data is incorrect. This seems very reasonable although it is a common source of error of the simulator modelling [2]. Many practitioners easily accept historic data with dubious quality and they are hoping on saving time on data collection. This might be the reason that many developers of simulation models rather develop the mathematical model than collecting quality input data. Furthermore there is often a lack of measurement data for proper model calibration [13].

Secondly, simulation models cannot provide simple answers to complex problems. In fact, a complex problem is more likely to require complex answers. It can be useful to simplify the system and by that save time and develop a model in a reasonable time. However, simplifications of the system are always essential to analyse. Ignoring a critical element of the system will cause the output data of the simulation model to be less reliable. The complexity of process can be difficult to model and can also be costly if the model shall adequately represent the actual behaviour of the process [13].

Furthermore a simulation model itself is not the answer to the problems. The simulator can never solve a problem, it can only show possible solution and the reliability of the solutions depends on the accuracy of the mathematical model and the input data.

3.3.4 Classifications of Simulations Models

Simulation models can be classified in several ways. The two main models is either the physical or the mathematical model. The physical model would be in real time and with real resources to conduct a controlled experiment. This might take long time but can give more accurate results as it does not require as many assumptions as the mathematical model. The physical model can be either dynamic or static. The dynamic is studied over a period of time and in the static model the time is fixed.

The mathematical model is described with help of equations and mathematical reasons. The mathematical model can be divided into static and dynamic. The static also known as the Monte Carlo model is analysed at a specific point of time with the time as a fixed variable. The Monte Carlo model can be either numeric or analytic.

The dynamic model is studied without having the time as a fixed variable. The dynamic model is either analytic or numerical. The analytic model is mainly used for optimization and is a model which you can mathematical solve. The numeric model is a model which cannot be solved and is executed to gain results.

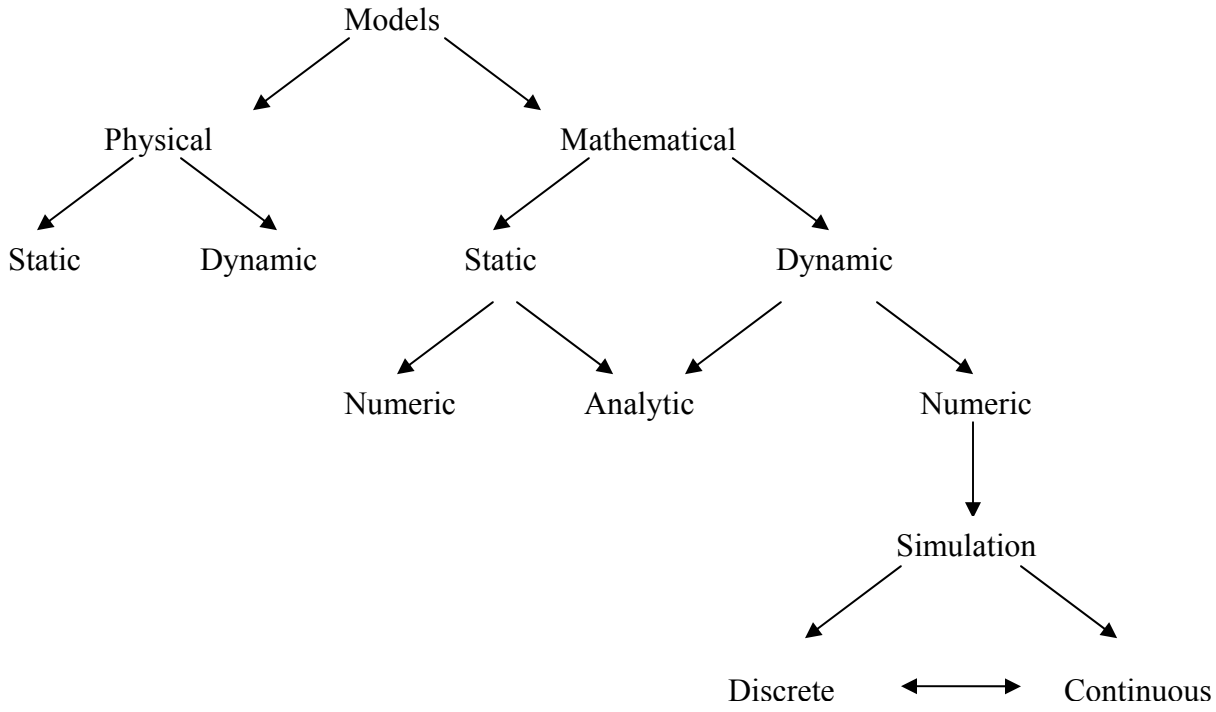

Figure 3. General classification scheme of simulation models [11].

In addition there exists another difference between the simulation models, if the model is deterministic or stochastic. The deterministic simulator contains no random variables. A known set of input generate a known set of output. The contradiction is a simulator which contains random variables and a known set of input generate a stochastic output.

The simulator model is also either discrete or continuous. The discrete model is only changed at discrete points of time and the continuous model is changed continuously. The difference is illustrated in Figure 4.

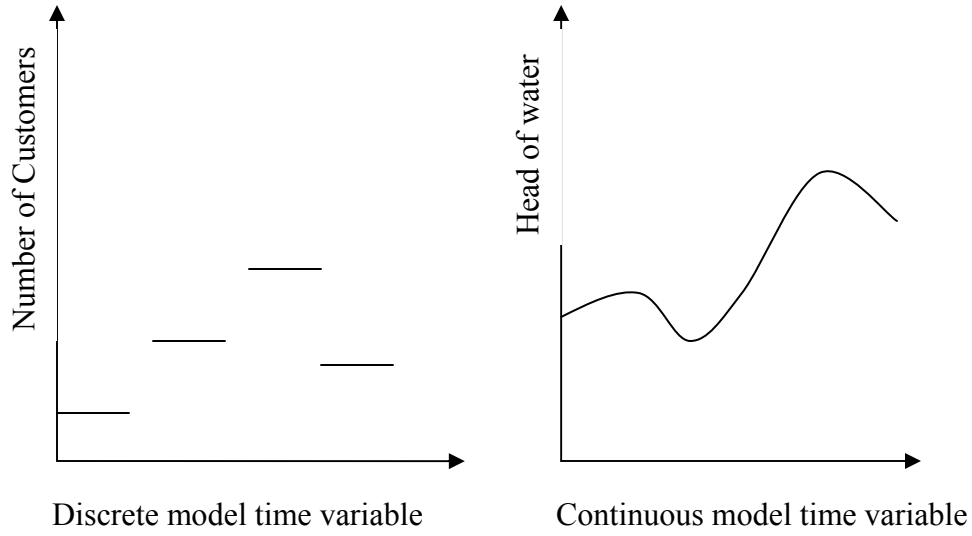

Figure 4. Difference between the discrete and the continuous simulation models [11].

The framework developed in this thesis has been simulated on a computer with a discrete, dynamic and stochastic queuing network model of MDRE processes. The queuing network enables the user of the framework to measure performance in the queuing network. A queuing network is described with the help of servers that take care of jobs and queues belonging to the servers. A simple queuing model is shown in Figure 5.

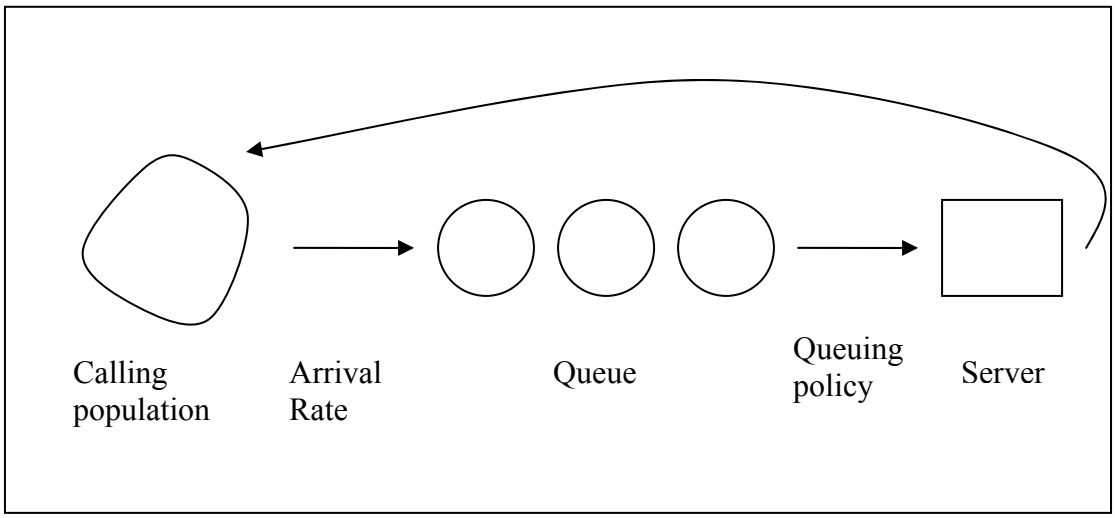

Figure 5. A simple queuing system [11].

The queuing network [8] has a population where the jobs to the servers are arriving from. If the server is idle the jobs enter the server and leave the server after the service time, hence when the server is done with the job. If the server is busy the job enters the queue and the queue is working as a first in first out (FIFO) queue in my case but it can be any queuing policy. When a job is finished, the server determines where the job shall be sent. This simple queuing network model can easily be analytical solved. However, when the queuing network tends to be more complex the simulation models are the only method to gain results from the queuing model.

4 Framework and Simulation models

4.1 Introduction

The purpose of this chapter is to present the framework which has been developed. In addition the simulation models will also be presented. The initial idea with the framework is that it should be easy to use, flexible and extendable. The purposes with implementing the simulation models are not execute them with different input data and analyse the output data. These analyses are already collected in previous work [5] and [18].The implementation of the simulation models, with the help of the framework has been conducted in order to gain knowledge of the flexibility and extendibility of the framework and its usability. The results will be presented in chapter 5.

4.2 Framework

The framework has been divided into three different layers, Simulator layer, Software Process layer and Requirements Engineering layer.

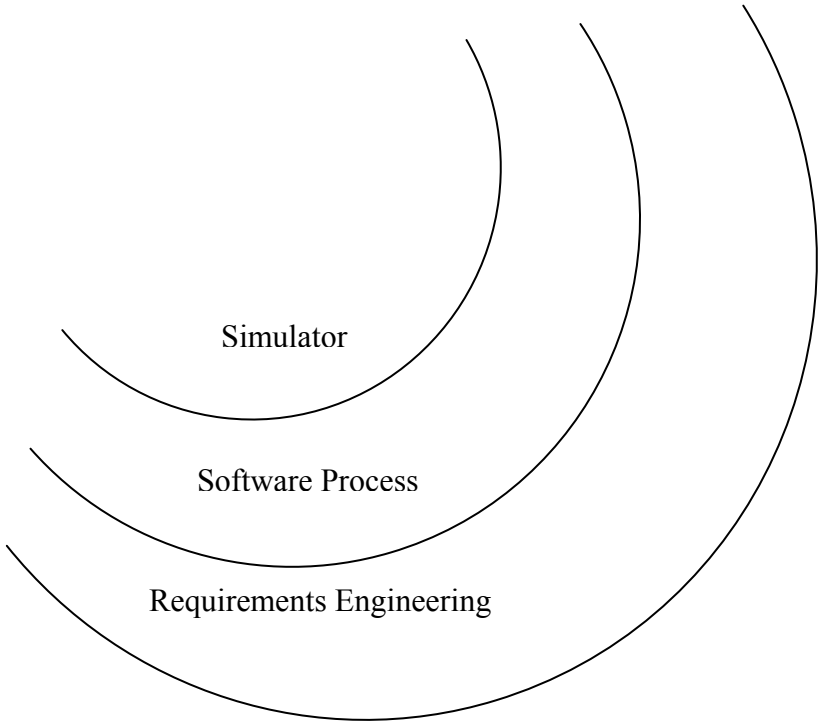

Figure 6. The different layers in the framework.

Each layer contains a few components and the purposes of the components will be presented below as well the interaction between the different layers.

The Simulator layer contains the components `DataHandler`, `Event`, `EventList`, `Global`, `Process` and `SampleFile`. The simulator layer contains components for any kind of simulation model. This layer is not specific for software engineering processes. The purpose of the components is to be a foundation for simulation models.

- `DataHandler`
  - Controls which output data will be produced.
  - The user can customize output data by inherit and override this class.
  - `DataHandler` is only called when an `Event` is leaving the system or when measurement in the `Stage` component is conducted.
  - It is not possible to override the `DataHandler` and expect to be able to control all possible measurement on the queuing network. The solution is in that case is to also override the measurement method in the `Stage` component in the Software Process layer.
- `Event`
  - The message sent between the different `Processes`.
  - The `Event` component always contains an `Assignment` if it is not a measure `Event`.
  - Each `Event` contains a type attribute and this attribute can be of three different types. The two first types contain an `Assignment`. These types tell whether an `Event` has arrived or has departed. The last type is measure which is performing the implemented measurements.
- `EventList`
  - The `EventList` is storing an `Event` in a linked list. The `Event` is sorted according to their departure time.
  - The most important methods are `addEvent` and `getFirst`. The second method returns an `Event` with the lowest departure time
- `Global`
  - Handle global variables as time and type.
  - The seed is also set here for the different distribution.
  - A few rules apply to set the seed and due to that the seed has a set and get methods.
- `Process`
  - `Process` is an interface and the purpose of the `Process` is to be the phase where an `Event` is handled and possibly modified.
  - The `Process` is implemented both by the `JobGenerator` and the `Stage` in the next layer.
- `SampleFile`
  - The `SampleFile` samples the output data.
  - There is only one parameter to consider for the `SampleFile` except the input file and the output file. The parameter to consider is the sample interval.

    - This interval is exponential distributed to prevent any unexpected pattern to exist in the sampled data.

The Software Process layer contains the components `Assignment`, `FrameworkException`, `JobGenerator`, `Resource`, `Rule`, `RuleBase` and `Stage`. This layer is specific for software process simulation models.

- `Assignment`
    - An `Assignment` is a task, can be a requirement, a design job etc.
    - An `Assignment` is always attached to an `Event`. However, an `Event` can exist without an assignment.
    - The `Assignment` was created as a base for all possible jobs in a software process.
- `FrameworkException`
    - The `FrameworkException` is thrown from the framework at several occasions.
    - The initial idea was to build an Exception tree. However, there is only `FrameworkException` and it is only containing an error message.
- `JobGenerator`
    - The `JobGenerator` is the foundation for generating a job. The `JobGenerator` implements the `Process` interface.
- `Resource`
    - The `Resource` is the component which represents the servers in the queuing network.
    - The `Resource` work as a base class and is further extended in the RE layer by the `Engineer` component.
    - The `Resource` contains attributes as availability and id.
- `Rule`
    - The `Rule` is an interface which is used if the problem to determine the next `Process` for an `Event` is complex.
- `RuleBase`
    - The `RuleBase` is an abstract base class which implements several methods of the `Rule` interface.
- `Stage`
    - The `Stage` implements the interface `Process`.
    - In the framework the `Stage` is a central component. Firstly, it is important for the modelling of a MDRE process. In addition the major part of the simulation is done here.
    - The `Stage` is also closed connected with a `Rule`, if that exists.

The RE layer contains the components `Engineer`, `REGlobal`, `Requirement` and the `RequirementGenerator`. An `Engineer` is a subclass to `Resource`. This layer is specific for Requirements Engineering simulation models.

- Engineer
  - The Engineer has four different skills. There is one skill for every type of Requirement. The skills are initiated at the creation of the Engineer and are not changed during the simulation.
- REGlobal
  - This component contains configuration data for the different distributions in the components Engineer and Requirement.
  - The lowest, highest and median value for the attributes effort and value in the Requirement component and skills in the Engineer component is stored here. There component contains default values if the user of the framework does not set any values.
- Requirement
  - The Requirement is a subclass to Assignment. The Requirement component is created from the RequirementGenerator. The Requirement will be assigned an effort and a value. In addition the type of the Requirement is also set when it is created.
- RequirementGenerator
  - The RequirementGenerator is a subclass of the JobGenerator. The purpose of this component is to generate Requirements to the simulation model.

An overview of the Framework is presented in the UML diagram below.

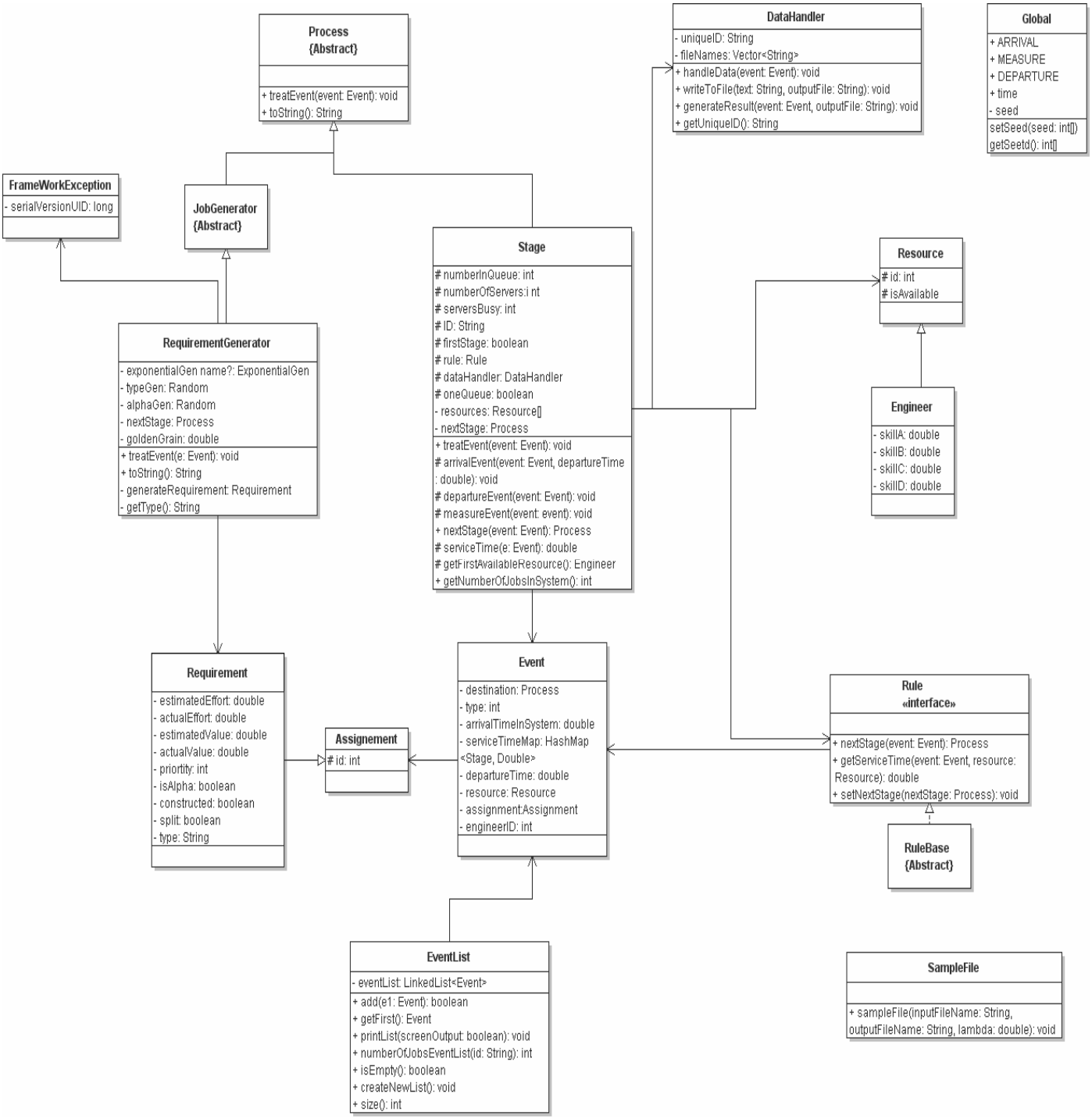

Figure 7. The UML diagram of the developed framework.

The thought is that these components should be enough to create a simulation model for MDRE processes. In addition a package SSJ [21] is used for generate exponential, triangular and uniform distributions. This package also contains several other components.

The `Stage` component can also be set with either one common queue for the servers or that each server has their own queue. This enables the user of the Framework to model n M/M/1 queuing network or one M/M/m queuing network. Of course, the arrival and service process depends on how the arrival rate and service time is implemented.

Furthermore the difference between the `DataHandler` and a measurement `Event` is important to clarify. The `DataHandler` is responsible for both writing data to a file and handle data. However, the `handleData` method is only called when an `Event` has no further destination. It can be because the `Requirement` has been implemented or the `Requirement` has not been selected. The measurement `Event` is used and handled by the `measureEvent` method in the `Stage` component. The measurement there depends how the method is implemented but in the standard component the amount of jobs in the system is measured and then the `DataHandler` is called to write the data to a file.

In addition, the service time in `Stage` is always received from the `Rule` if a `Rule` exists. The service time can depend on the `Resource` so the service time method has both an `Event` as well as a `Resource` as input parameters. When a queuing network is created the `setNextStage` method should always be set in the `Stage` component. There is a similar method in the `Rule`. However, this method is only used for measurement and should never be set by the user of the framework.

The framework which is best overviewed in the UML diagram in figure 7 contains several assumptions and the thought below is to clarify the assumptions as well to argue for the assumptions. The first assumption is concerning the `Requirement` which contains several attributes as estimated effort/value and actual effort/value. These attributes should be well understood from the chapter Process Quality. The attribute isAlpha should also be understood from the same chapter. There is also a type attribute. A requirement can be of different types, firstly the classification functional or quality requirement [9]. In addition, a requirement can be focusing on e.g. network architecture, user interface etc. The assumption in this framework is to divided the `Requirement` into four different types; A, B, C and D. The main purpose of doing this relates to another component the `Engineer` component. An `Engineer` can have skills in different areas, hence an `Engineer` has different skills for the different types of requirements; A, B, C and D. This is an easy way to model that engineers have different special competences in different fields. The skills of an `Engineer` are generated from a triangular distribution. There are already standard values for the skills. However, these values are possible to modify.

In addition an `Engineer` is never improving his skills during the project. This might not be realistic in reality and it would be more reasonable that the `Engineer` continuously is improving the skills, i.e. learning. In addition no split of requirements, merges or change request are implemented. However, splits, change request and merges should be possible to implement in the rules. This has not been implemented

as simulation models below do not require any of these tasks. The design and implementation of the framework does not see any of these tasks as an `Assignment` which would require a particular component. It should be implementation in a `Rule`.

4.3 RSQ model

The Requirements Selection Quality (RSQ) model is an analytical model and is a model for requirements selection process. The typical way of storing requirements is to use a repository where all requirements are stored. When the requirement arrives, it is evaluated based on the estimated value and effort [17]. In order to prevent the process from being overloaded [7] it can be a good idea to introduce a screening stage. The purpose of the screening stage is to assess the requirement in a fast way and determine if the requirements should be evaluated. The evaluation stage includes requirements analysis, specification, validation and prioritisation [17]. Figure 8 shows a simplification of the requirements selection process.

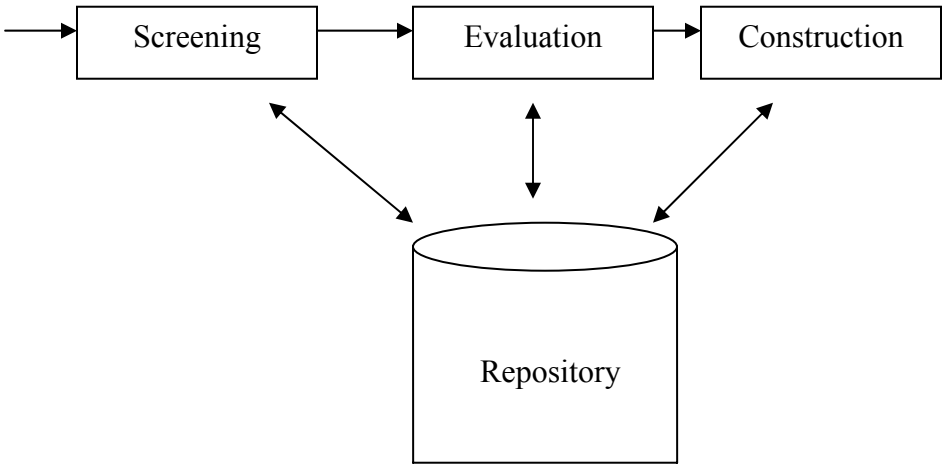

Figure 8. Simplification of a requirements selection process [17].

The screening activity is preventing the requirements to reach the evaluation stage. The screening is based on product profile and business strategy [17]. In the evaluation stage the effort and the value are estimated and in the construction stage the requirements are constructed and integrated in the product.

A refined requirements selection process is shown in Figure 9.

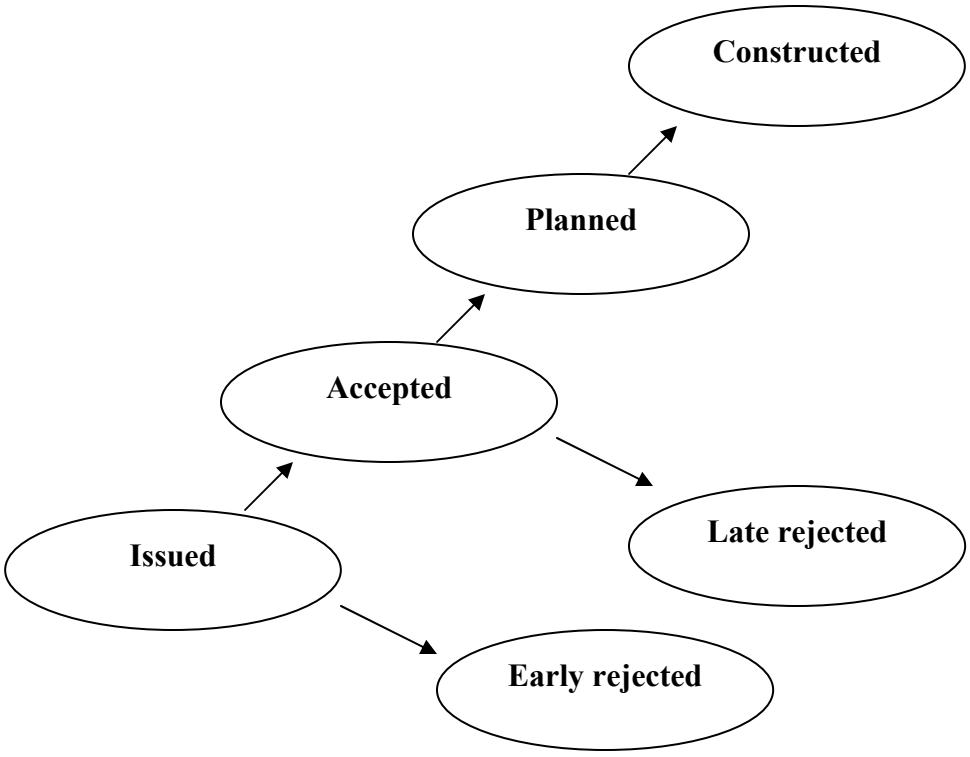

Figure 9. A state model of requirements refinement [17].

The model in Figure 9 is a state-based model for the requirement. An arriving requirement receives the status issued and is either early rejected or accepted in the screening stage. The requirement continues to evaluation stage and is either planned for construction or late rejected. If a requirement is planned then the producer is estimating that the requirement has a good potential value for the customer to a reasonable effort. When the requirement has received the state constructed, the requirement is completed by the construction activity. All requirements states are stored in the repository and the changes in the requirements states imply an update of the repository [17].

In Figure 10, a queuing network model is presented. The model is based on the refinements of Figure 9. The model has three stages, screening, evaluation and construction. Each stage is modelled as number of servers and each server has its own queue, which gives that each queue is M/M/1 [8]. M/M/1 queues are good for calculating analytical. The arrival rate is assumed to be Poisson process and all the service rates are exponentially distributed. In every stage a server represents one employee and the queue to the server is the work repository. All servers have the same service rate, to simplify calculations. The service rate is the mean time for one engineer to do the work at that stage. The work is randomly divided among the servers, with equal probabilities.

The simplification of the model does not take different skills of engineering into consideration. In addition, a requirement which has been removed the process can never re-enter the process. Furthermore decomposition of requirements is neither considered. The model does not real complex requirements selection process. The purpose of the model is to get a model which it is possible to obtain analytical results. The following aspects have not been considered [17]; deadline and budget restriction, competition between requirements analysis and construction resources, disposed requirements during construction, dependencies between requirements, requirements decomposition into sub-requirements, etc.

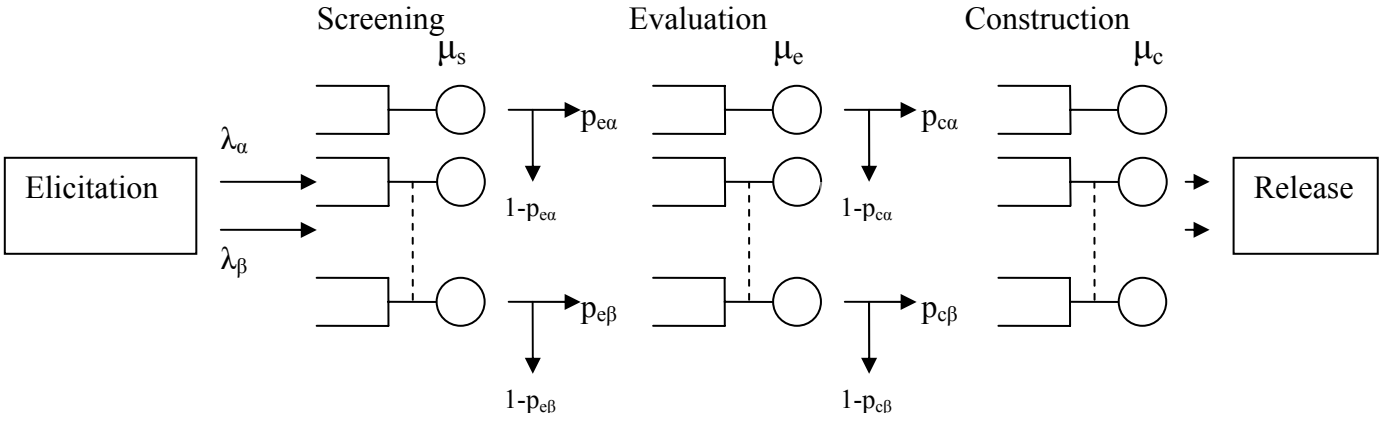

Figure 10. A queuing network model of the requirements selection process [17].

4.4 REPEAT process

The REPEAT process is similar to the RSQ process. They are similar in elicitation and both have analysing or evaluation phase. The RSQ model is a simpler model as it only has one release and no deadline. The REPEAT process manages requirements continuously. The REPEAT process is carried out in several iterations and three iterations are always carried out in parallel. The REPEAT process produce two released per year. The duration of each instance of the process is fourteen months. When the iteration is finished a product release is delivered. The REPEAT process includes the following stages; elicitation, selection, change management and construction and finally conclusion

4.4.1 Elicitation

The elicitation stage is conducted in order to collect and classify the requirements. The collection of requirements are submitted by stakeholders and stored in a database. The requirements are described in natural language and a summary is provided by the issuer. The issuer is also giving the requirement an initial priority. The purpose of the priority is to suggest in which release the requirement should be implemented

in. The priority is a subjective measure reflecting the view of the issuer. The priorities are measured on a scale with three levels shown in the Table 3.

Priority	Semantics
1	The requirement is allowed to impact on-going construction of the previous release.
2	The requirement is incorporated in the current release planning.
3	The requirement is postponed to a later release

Table 3. The scale of the priority [5].

4.4.2 Selection

The selection stage has three primary goals [5].

- select which requirements to implement in the current release
- specify the selected requirements in more detail
- validate the requirements

In this stage the output is a requirements document. The document contains a selected list and a non-selected list. All requirements on the selected-list are specified in detail and the effort in estimated hours. The non-selected list contains all requirements which are postponed to the next release. The selected requirements are divided into two lists, a must list and a wish list. The total effort of the must list is estimated to allocate 70% of the available effort. The wish list is estimated to allocate 60% of the available effort. If the estimations are correct the must list will be implemented and half the wish list will be implemented. The estimations may not be correct and thereby all requirements on the wish list are specified.

4.4.3 Change management during construction

The change management is carried out in parallel with the design, implementation and testing of the requirements and handles changes in priorities of the requirements. This stage contains two sub stages, one before code stop and one after code stop. When the code stop stage is entered no implementation is done and the focus is on testing. However, if a requirement with high priority is issued this requirement may be allowed to affect the ongoing construction [5]. The must and wish list will be affected by this late arrival of requirement as the list must be rearranged so that the new and more important requirements can be incorporated.

4.4.4 Conclusion

The final report is written in this stage and metrics are collected. The final report includes the lesson learned in this iteration of the REPEAT process. The metrics may act as decision support when allocating resources.

4.5 REPEAT Simulation Model

The simulation model [5] is based on the REPEAT process, described above. The simulator model is a queuing network and is implemented using discrete event simulator. The simulation model contains three stages; elicitation, selection and construction. Figure 11 presents the simulator.

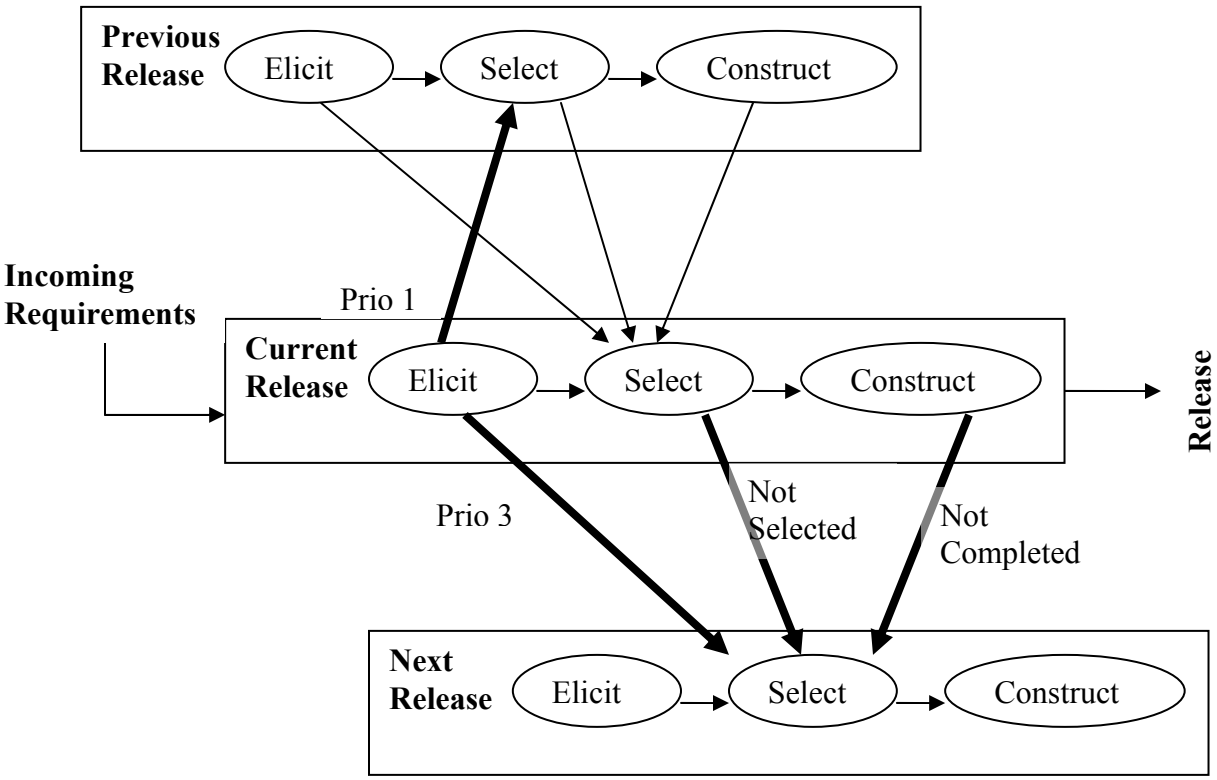

Figure 11. Simulation Model [5].

The requirements are entering the simulator from the environment. The requirements must pass the three stages to be included in a release. The conclusion stage, mentioned in 4.4.4 was not included in the model as it is independent from the other stages [5]. The stages are modelled as processes with a queue of incoming requirements and a pool of servers which represents the employees. The arrival of requirements is modelled as a Poisson process and the elicitation stage is an M/G/m queue and the two other stages are G/G/m stages. The queue is a FIFO (First In First Out). Furthermore each release has its own resources.

When a new instance is created, a number of servers in each stage are created. When the elicitation is in progress, selection stage is idle and is waiting for the elicitation to start. The construction stage is idle during elicitation and selection stage and is waiting for the selection stage to finish. The employee represented in the simulation

model is the same as represented in the previous release. This is simple approach to represent that the employee divided their time between different activities

4.5.1 Elicitation

In the elicitation the priority is set. If the priority is one (high) and the previous selection stage is still in progress the requirement is sent to previous selection stage. If the previous selection stage is no longer in progress the requirement is sent to selection stage of the current release. If the requirement receives three (low), the requirement is sent to selection stage in the next release. In any other cases the requirement is sent to selection stage in the current release.

4.5.2 Selection

During the selection stage the time a requirement will spend in construction is estimated. The selection stage is creating a must and a wish list. The must list is allocating 70% of the effort and the wish list is allocating 60% of the effort. The requirement which is selected for the any of the two lists is transferred to the construction stage of the current release. The other requirements excluded from the two lists are sent to the next selection stage.

4.5.3 Construction

The requirements arriving in the construction stage is constructed until the deadline is reached. Before a requirement is constructed the actual effort is calculated for the requirement. If it is not possible to complete the construction of a requirement before the deadline the requirement is sent to the next selection stage. Hence requirements are either 100% completed or 0 % completed. Not all requirements in the must list and half the requirements (70% and 60%) in the wish list is created as planned in the selections stage, due to that the estimation of effort contains a parameter controlled error.

4.5.4 Simplifications

In this simulation model there are two significant simplifications [5]. The first simplification concerns the use of employees. In the reality there is one pool of employee containing a number of developers. These developers work in all the stages for all the releases. The simulation model has one pool of servers for every stage of every release. To adjust this, the servers of every stage are idle when the stage is inactive. According to [5] this gives and adequate accuracy.

The second simplification was concerning how the must and wish list is created. In this simulation model the first incoming requirements are put in the lists. There is no consideration concerning priorities. This means that priority one from a later release rarely is selected due to late arrival and the lists are already full.

In addition the serving time of the construction stage was also modified [5]. The maximum serving time was changed from 170 to 91 days and the most common service time was changed from 19 to 45 to produce the same workload. Otherwise the largest job would never be implemented according to the simulation parameters, see

5.3.2 for further details concerning the simulation parameters. Another simplification is that the simulation model never rejects a requirement and this is done at a small extent in the real process [5]. However, these requirements are rejected very rarely so it is not likely that it affects the validity of the results.

5 Results

5.1 Introduction

This chapter focuses on the implementation of the simulation models with help of the framework. The results of the implementation will be presented in the following sections. In addition, the simulation models also need to be verified and validated. The verification answers the question, is the model correct implemented? The validation answers the question is the model correct? The difference between the two questions might not be completely clear. The verification state if the model meets the requirements. The validation is if the requirements are correct interpretation of the actual process [9].

5.2 Implementation of the Simulation Models

The implementation of the simulation models differed in the level of difficulty. The RSQ model was not as complex as the REPEAT model and thereby the implementation was less complicated. The RSQ model could be implemented with the components in the framework. The elicitation was done by using the `RequirementGenerator` and the three stages; screening, evaluation and construction all use the `Stage` component. The major differences between these three stages was the service rate, number of servers, minor tasks as estimation of value and effort and of course how it is determined if the requirement will be chosen for the next stage. A specific `Rule` for each `Stage` was developed with the specific characteristic as mentioned above, see figure 14 for the rules which where implemented. In addition the `DataHandler` component was extended to gain the data which will be used for validation. The Figure 12 presents an Event arriving to a `Stage`, in this case the Screening `Stage`. The verification and validation will be presented in the next section.

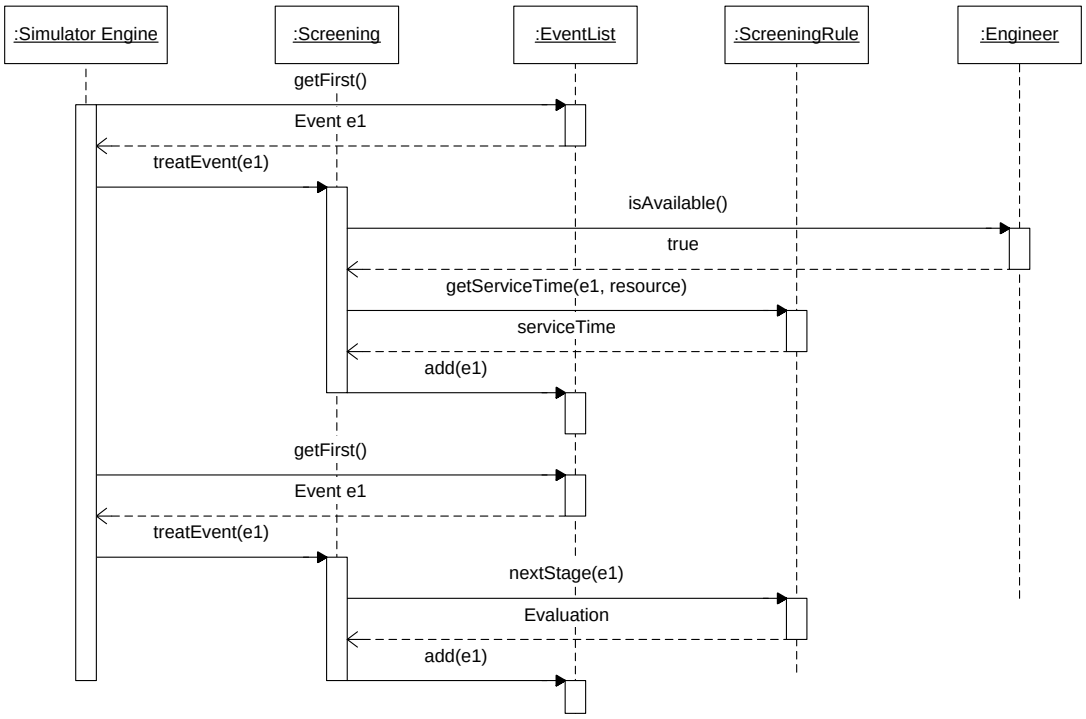

Figure 12. An Event arrives to the Screening Stage.

The REPEAT model was more complex as described in section 4.5 and the implementation was of course more complex. The first compatibility issue with the framework was that the simulation model produces *n* releases. Where will the new releases be created and when? In addition three releases were running simultaneous and requirement would not be sent the next stage until the previous stage was finished. In addition the requirements could be sent between the different releases. The RSQ model was an n M/M/1 [8] queuing network and the REPEAT model was an M/M/m [8] and G/G/m [8] queuing network. The major difference between M/M/M, G/G/M and M/M/1 queues were that the two first queues had one queue for all the servers and the n M/M/1 queues had a separate queue for each server and this cause differences in the effect of the process. The utilisation of the M/M/M and G/G/M queues was higher than for n M/M/1. However, this would not cause problem with the framework as the `Stage` component could be parameter controlled for one queue or a specific queue for each one of the servers.

The REPEAT model required an extension of the framework; an additional layer was added to the framework especially developed to fit the REPEAT model, see figure 13 for the REPEAT model specific architecture. The REPEAT layer contained two components, `RepeatStage` and `RepeatHelp`. Firstly, the stage was extended with a stage adapted completely for the REPEAT model, the `RepeatStage`. The arrival was different as a `RepeatStage` only exists for a limited period of time. The departure was also different when it checks for next job to perform and in addition the

measurement was different when it was essential to measure different `RepeatStages` in different release versions.

The REPEAT model also needs the possibility to push a requirement to the next release if it was not possible to finish the requirement in that stage and the REPEAT model should be able to create new releases when one was finished so three releases always were simultaneous running. A second component which creates new releases and contains important data as the length of elicitation, selection and construction was also created, the `REPEATHelp`. In addition new rules were also created. However, the framework always assumes new rules were created as that was a design decision. The `DataHandler` was not necessary to extend as the interesting data was not when a requirement had left the system in the validation. The validation was focusing on the number of requirements in the selection stage.

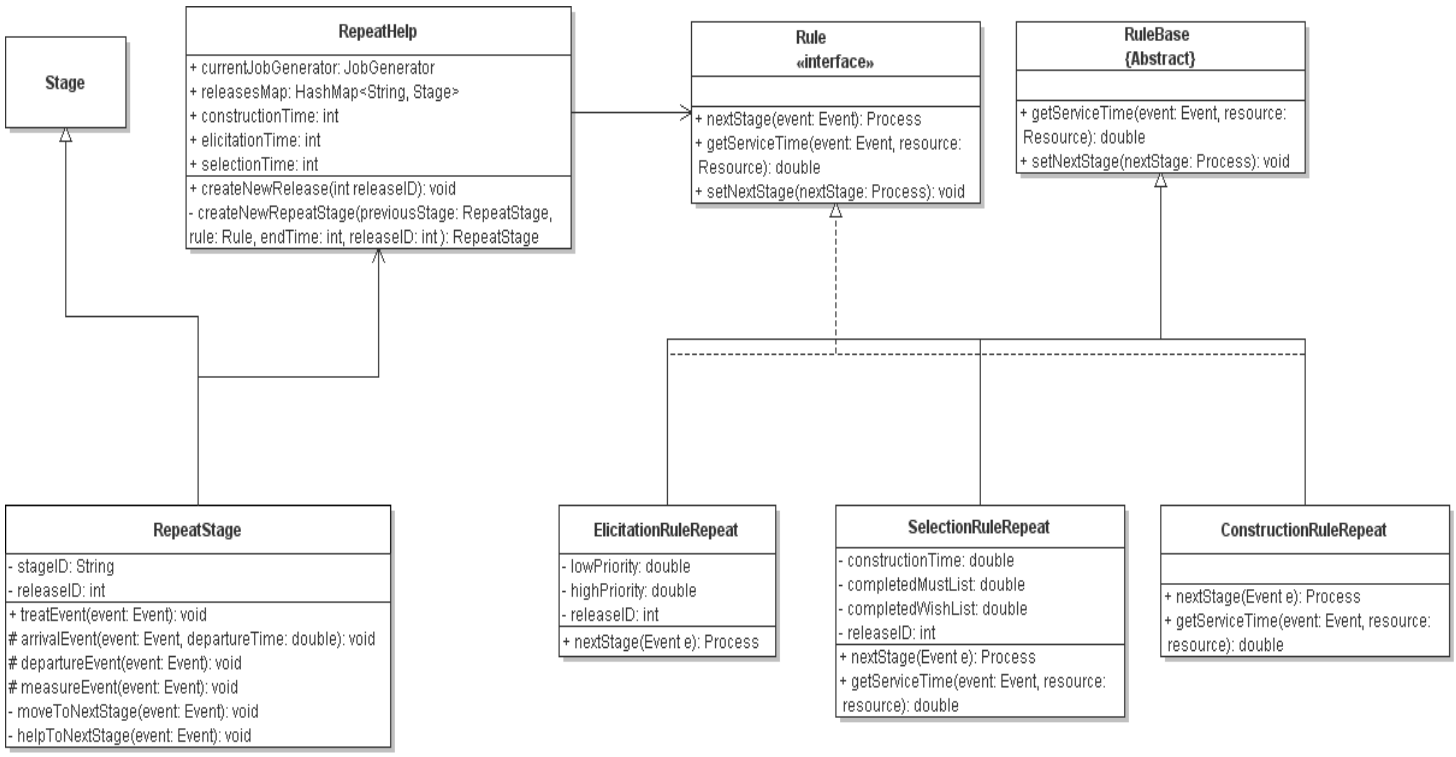

Figure 13. UML diagram of the REPEAT layer and the rules.

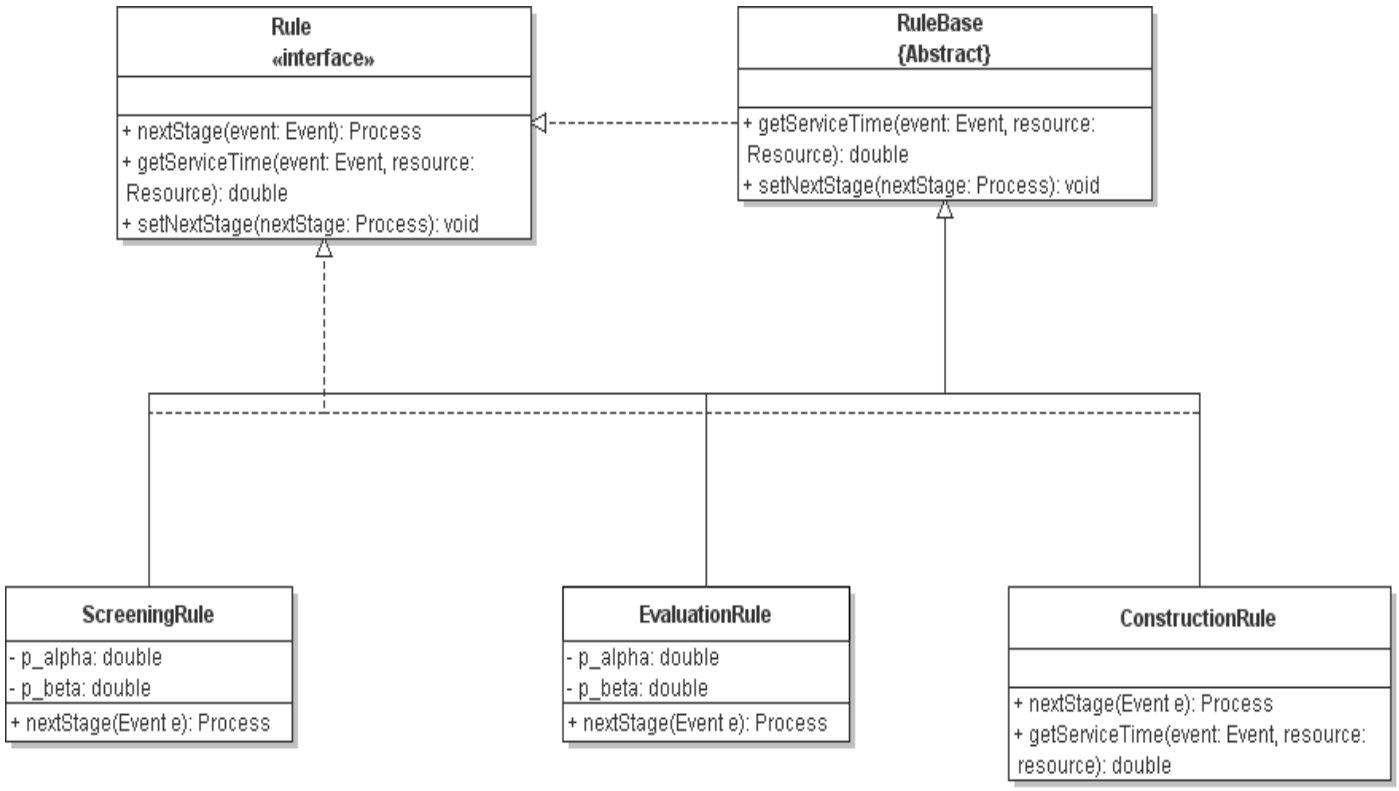

Figure 14. UML Diagram of RSQ rules.

5.3 Verification and Validation

5.3.1 RSQ

The verification of the RSQ is done with help of Little's law [8], which is $N=\lambda t$. This means that the number of customer in the system (N) is equal to the arrival rate of customer to the system (λ) multiply the time the customer spends in the system (t). The arrival rate is given as input data to the model and was compared with the measured number of customers and the time spent in the system. The tables below will show the measured data.

The verification was used with the following data. The screening and evaluation stage accepted 80% of the alpha requirements and 20% of the beta requirements. The golden grain was set to 10%. The service rate in the screening, evaluation and construction stage was 10 requirements / unit. The test was first done with one server in each stage and then with two and three server per stage. The data for the average time in system and the average number of customer in the system was sampled with help of an exponential distribution with a mean of hundred. So in average every hundred point was taken for the measurement. The reason why samplings is done with exponential distribution and not just take every hundred point is to avoid a pattern in the data. The sampled data was also checked against autocorrelation and no autocorrelation was found.

λ	1	5	10
N	0.15334	0.766698	1.599247
t	0.147116	0.151851	0.159134
Estimated N	1.042301	5.048999	10.04967
% error	4.23	0.98	0.497

Table 4. Verification of RSQ with one server in each stage.

λ	1	5	10
N	0.131021	0.67341	1.373796
T	0.131624	0.136359	0.137018
Estimated N	0.995423	4.938508	10.0264
% error	-1.2298	-0.264	0.264

Table 5. Verification of RSQ with two servers in each stage.

λ	1	5	10
N	0.127639	0.68618	1.383877
T	0.144207	0.135729	0.137032
Estimated N	0.885113	5.055524	10.09892
% error	-11.4887	1.11	0.989

Table 6. Verification of RSQ with three servers in each stage.

The verification results are in general good; see Table 4, Table 5 and Table 6. The only result that differs severe from the rest is in Table 6 with a λ of one. An extension of the time the simulation model is running would maybe improve the results. However, the author wanted the time to be same in all tests. Another interesting pattern is that the percentage error is lower for a higher lambda. This might be a low λ fewer requirements are arriving in the system compared to the measure time. The measurement is done very frequently. Furthermore it is preferred to keep the measurement interval constant for all measurement. The measurement interval should not be tested with different interval to receive a good result. Seven of the nine requirements have an estimation error lower than two percent. I think with the rest of the result a conclusion can be that the verification of the RSQ model is correct.

The validation was carried out by adapting the model to input data in an article [17]. The articles output data has been calculated analytical for the RSQ model. In addition the input data estimated after a survey with stakeholders from the industry. The input data is presented in Table 7 and Table 8.

λ	G	m_s	m_e	m_c	μ_s	μ_e	μ_c
3 req / day	10%	1	2	20	4 req / day	2 req / day	0.05 req / day

Table 7. Parameter used for RSQ, same in all tests.

Probability	Case RANDOM	Case LOW	Case HIGH	Case IDEAL
$p_{e\alpha}$	0.5	0.6	0.7	1
$p_{e\beta}$	0.5	0.4	0.3	0
$p_{c\alpha}$	0.5	0.8	0.9	1
$p_{c\beta}$	0.5	0.2	0.1	0

Table 8. The four cases in the RSQ test.

The most interesting measurement of the model was the Mean time to market parameter (MTTM) and below is the formula for calculating MTTM.

$$MTTM = T_S + T_E + T_C$$

$$T_X = m_X \times \frac{\frac{1}{C_X}}{1 - \frac{\lambda_X}{C_X}}$$

$$C_S = m_S \times \mu_S$$

$$C_E = m_E \times \mu_E$$

$$C_C = m_C \times \mu_C$$

The T is representing the time in the different stages; screening, evaluation and construction. The C is the total capacity in each stage, the number of server m multiplied with average service rate μ.

	Case RANDOM	Case LOW	CASE HIGH	Case IDEAL
Analytical MTTM	81.8	33.0	29.1	30.1
Est. MTTM	82.067	33.872	28.869	29.548
% error	0.326	2.64	-0.794	-1.83

Table 9. Results of validation.

The simulation model was executed for a longer period as it was seen in the beginning that the result improved with the length of period. The results are good as all of them are approximately in the interval -2 % to +3%. Furthermore the results were

checked against autocorrelation for the estimations and no autocorrelation could be found.

5.3.2 REPEAT

The verification of the REPEAT model caused minor problems and a simpler version was created to verify it. The simplification only including the release 1, 2, and 3 and no new release was created. In addition a requirement left the stage immediately and did not wait till next stage was ready. The verification was carried out with help of Little's law [8] as it was presented in the RSQ model. The result is presented in Table 10.

Λ	**0.01**	**0.1**	**0.5**
N	0.020	0.164	1.29
T	2.14	1.73	2.48
Estimated N	0.0097	0.095	0.522
% error	-2.07	-4.95	4.54

Table 10. Verification of the REPEAT model.

The verification of the REPEAT model is as mentioned a little more difficult, as this process easily can be instable. The result is in general good; see Table 10, with a low percentage error, approximately interval -5% to +5 %. However, to obtain these results the simulation was executed for a long period of time. The N and t are also obtained with the same method as is mentioned with the RSQ model. The N and t are sampled with a mean of hundred with the λ 0.01. The two other could not be sampled with such a low mean without autocorrelation existed. The mean was extended to every thousand point. The sample points is as previous exponential distributed over the mean. The first measure with the lowest λ has a lower estimation error and the reason is the sampled mean is lower than the two others. In addition to obtain a similar result the two others would need the same amount of sampled points. However, the author preferred to run the simulator with the same period of time for all the cases instead of trying to obtain the same amount of sampling points.

The validation was carried out with the parameter [5] in Table 11.

Parameter	**Case 1**	**Case 2**	**Case 3**
Time between two Consecutive release start-ups	126	126	126
Time from start of release to start of selection stage	126	126	126
Time from start of release to start of construction stage	168	168	168
Length of construction stage	126	126	126
Mean time between tow consecutive requirements	0.33	0.33	1.8

Number of servers in the elicitation stage	30	30	30
Elicitation time per requirement	(0.010,0.031,0.062	(0.010,0.031,0.062	(0.010,0.031,0.062)
Number of servers in the selection stage	30	30	30
Selection time pre requirements	(1,2,10)	(1,2,10)	(1,2,10)
Number of servers in the construction stage	30	30	30
Construction time per requirement	(1,45,90)	(1,45,90)	(1,45,90)
Fraction of requirement of priority 1	0%	10%	10%
Fraction of requirement of priority 3	0%	25%	25%

Table 11. Validation of the REPEAT model.

The unit of the parameters in Table 11 is days and the service times for elicitation, selection and construction are triangular distributed.

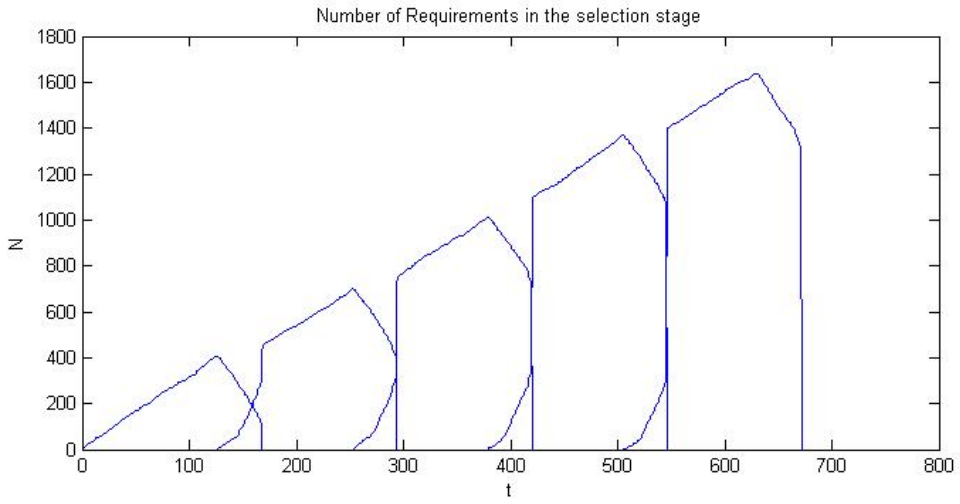

Figure 15. Number of requirements in the selection stage.

Figure 15 shows the number of requirements the selection stage in the implemented REPEAT model, which is the first case in Table 11. The first visible conclusion is that the process is overloaded and that is correct according to the article [5] where the validated data are collected. In addition the first release should have approximately 400 requirements and in Figure 15 this is similar n. Then the 5th release should have approximately 1600 requirement which also is similar in the Figure 15. However, there is one remark comparing to article of validation. The curves in Figure 15 of

every release are immediately zero when a new release start and this is not the case in the article of validation. The reason is that in this implemented version all requirements are immediately transferred to the next selection when the end time is reached for the selection stage in the current release.

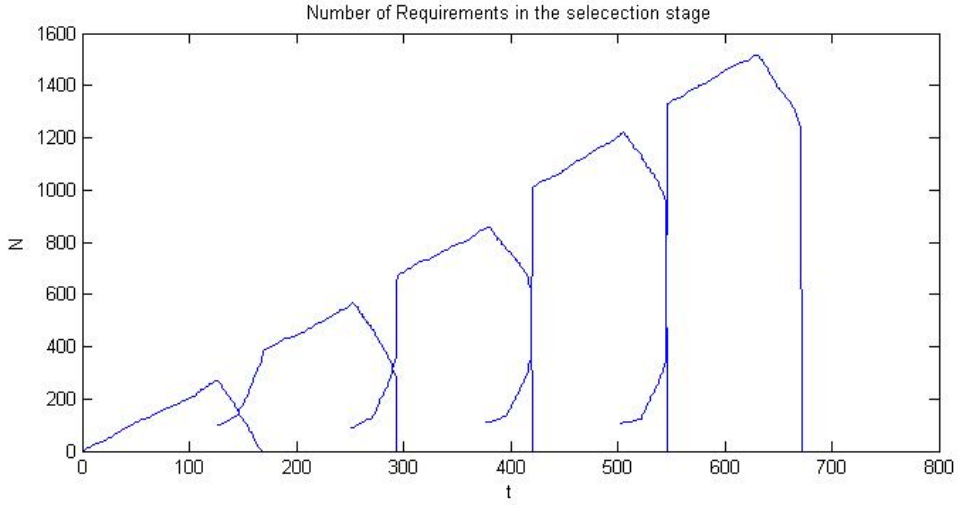

Figure 16. Number of Requirements in the selection stage.

In Figure 16 the result is presented of the second case from Table 11. The process is still overloaded, when prioritisation is introduced. The number of requirements in the selection stage is similar to the one in case 1, also according to the article [5].

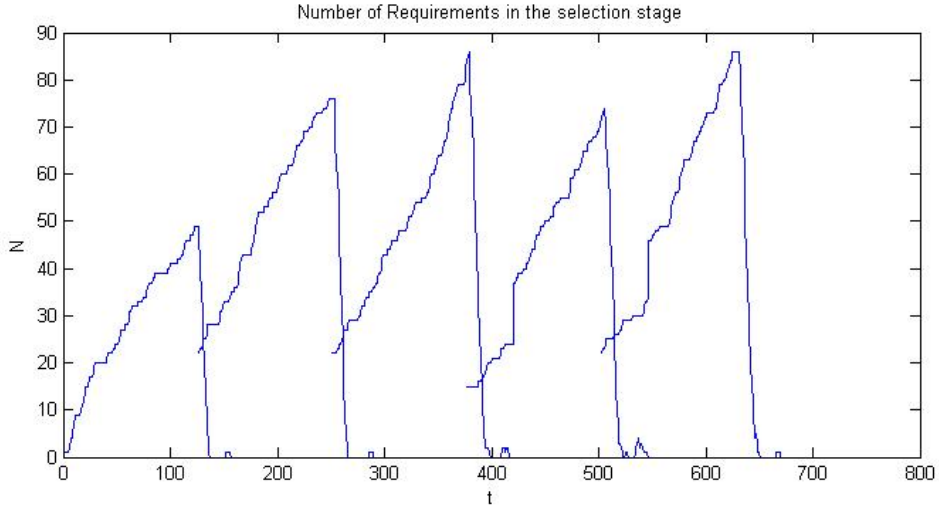

Figure 17. Number of Requirements in the selection stage

In the last case, case 3 the arrival rate has been changed from 3 requirements per day to 1.8 requirements per day. The result is shown in Figure 17. The process is obviously stable as there is no constant increase in the selection stage. According to the

article approximately 80 requirements are in the selection stage and that is similar in Figure 17.

The RSQ model and the REPEAT model required different amount of effort to implement. The RSQ model was easier than the REPEAT model and also went faster to implement. The main reason was that the RSQ model is less complicated than the REPEAT model. The framework components were useful in both implementations. The basic components could be used to implement the RSQ model, only a rule for each stage was necessary to develop. The REPEAT model was more complex and required an extension of the `Stage` component. The correction of the both models or if it is more likely to implement more correct models with the framework is difficult to determine. However the basic components which was used for each model had been tested so probably it is likely to do less errors when line of code for the implementation decrease. The main logic for the RSQ and the REPEAT models are in the `Stage` and `RepeatStage` component. In the framework these two components are the most complex and most difficult to implement.

In addition it might have been interesting when another user would implement a model with the help of the framework and evaluate the framework and test the usability and ease of use. The framework has not been tested on any user and the author has tried to model and design a framework which should be ease of use.

6 Conclusions

This thesis presents a framework which use discrete event simulation in order to model an MDRE process. The framework shall be on a balanced abstraction level. The modelling of a MDRE framework is difficult as the abstraction level is difficult to determining. The implemented simulation models have been guiding in terms of abstraction and in addition the literature study has also served as input data. It is essential to find the right level of abstraction as it should not be too low and detailed. The user of the framework shall be able to model an MDRE process without the need to implement all details of the process in the model and have all input data available. In addition a high level of abstraction will cause the user to not be able to gain any relevant output. However, if any MDRE process can be modelled with help of the framework it will valuable for the user.

The framework should also be flexible and extensible and if the level of abstraction is not suited for the user the framework can be extended to fit the user's abstraction level. The flexibility was tested with help of implementing two simulation models. The RSQ model was not complex and could easily be implemented. The REPEAT model was more difficult and required an additional layer. However, with the extra layer the REPEAT modelled was also implemented

The architecture of the framework was one objective. It should be possible to easily extend to model and simulate other software processes. The idea of the framework was to develop it with several layers. The several layers solution should make it easily extendable, not only for MDRE processes. The first layer contains simulation model components. The second layer is the software process layers. This layer and the simulation model layer would be useful to use if someone would want to implement another software process e.g. a test process or a framework for test processes. The `JobGeneator` could be extended to generate test jobs. The `Assignment` would also be needed to extend to TestJob and maybe `Resource` also would be extended to have special test skills.

Furthermore the framework was then validated with help of implemented simulation models. The simulation models input data and expected output data was obtained from published articles. The first article [17] which validated the RSQ model the output data was analytical obtained with realistic parameters obtained after a survey with the industry. The REPEAT process had been modelled in the other article [5] and the output data from the implemented model was compared with the output data from the article. The simulation model is implemented as queuing network and then the framework was verified with help of Little's law. The verification error of the RSQ model was between -1.2% to +4.2% if the worst point is excluded and the REPEAT model was between -5% to +5%. The second result is not low but it is not likely that a fault is in the framework as both errors from the two models are low. The RSQ and REPEAT model implemented by the framework is considered to be verified and validated according to chapter 5.

In addition there is one design issue that is interesting to discuss and it concerns the design decision to use rules. The idea behind the framework is to use rules at anytime when the decision to move to next stage is more complex than only push it to the next stage. However, there is theoretical another design which could be used and that is to extend the `Stage` for each specific case. Hence the analytical model RSQ would have three unique stages, one screening stage, one evaluation stage and one construction stage. This approach has not been used but it should be possible to inherit `Stage` and override the `nextStage` method. Which approach is preferable depends on the user of the framework. However, no investigation has been conducted with the second approach and the author favours the design with the rule approach.

Finally, further work is to extend the framework. One issue which could be interesting is the skills of an `Engineer`. It is likely in any MDRE process that an `Engineer` will improve their skills the further they work in a process and there is actually an interesting idea in [10]. The idea is that each `Engineer` has a certain skill, a max skill and an experience. The time it takes to complete a task is based on the skill and the experience. In addition the skill is increasing during the process and the max skill is the highest possible skill an engineer can obtain and this is different from `Engineer` to `Engineer`.

In addition, further work could also be to extend the framework to fit another software process instead of a MDRE process, e.g. a test process. The architecture has been considering this and has tried to make it as flexible as possible during the development of the framework and the first two layers should be possible to use. In addition if the framework will be extended with a different layer or additional components in any of the layers it would also be a good idea to maybe extend the Exception component and maybe build an exception tree.

Reference

[1] Banks, J., Carson, J. S., & Nelson, B., 1996. Discrete-Event System Simulation (2nd ed.), Prentice-Hall.

[2] Chung, C. A., 2004, Simulation Modelling Handbook, A Practical Approach. CRC Press.

[3] Gordon, G., 1969. System Simulation, Prentice-Hall.

[4] Gorschek, T., Wohlin, C., 2005. Requirements Abstraction Model Requirements Engineering Journal, Vol. 11, No. 1, pp. 79-101, 2006.

[5] Höst, M., Regnell, B., Natt och Dag, J., Nedstam, J., Nyberg, C., 2001. Exploring bottlenecks in market-driven requirements management processes with discrete event simulation. The journal of System and Software 59, pp. 323-332.

[6] Höst, M., Regnell, B., Runesson, P., 2006. Att genomföra examensarbete. Studentlitteratur.

[7] Karlsson, L., Dahlstedt, Å. G., Regnell, B., Natt och Dag, J., Persson, A., 2007. Requirements engineering challenges in market-driven software development - An interview study with practitioners. Information and Software technology 49 (6), pp. 588-604.

[8] Körner, U., 2003. Köteori. Studentlitteratur.

[9] Lauesen, S., 2002. Software Requirements Styles and techniques, Addison-Wesley.

[10] Melis, M., Turnu, I., Cau, A., Concas, G., 2006. Evaluating the Impact of Test-First Programming and Pair Programming through Software Process Simulation. Wiley InterScience

[11] Natt och Dag, J, 2002. Elicitation and Management of User Requirements in Market-Driven Software Development, Licentiate Thesis, Department of Communication Systems, Lund Institute of Technology.

[12] Pedgen, C.D., R.E. Shannon and R.P. Sadowski, 1995. Introduction to simulation Using SIMAN, 2nd Ed., New York McGraw-Hill.

[13] Pfhal, D., 2006. A Software Process Simulation Model Kit in Support of Empirical Research. Proceedings of the 5th ACM-IEEE International Symposium on Empirical Software Engineering. Volume II: Short Papers and Posters

[14] Potts, C., 1995. Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software, Proceeding of the Second IEEE International Symposium on Requirements Engineering (RE'95), pp.128-130.

[15] Regnell, B., Beremark, P., Eklundh, O., 1998. A market-driven requirements engineering process – results from an industrial process improvement programme. Journal of Requirements Engineering 3 (2), pp.121-129.

[16] Regnell, B., Brinkkemper, S., 2005. Market-Driven Requirements Engineering for Software Products, Chapter in Engineering and Managing Software Requirements, pp. 287-304.

[17] Regnell, B., Karlsson, L., Höst, M., 2003. An Analytical Model for Requirements Selection Quality Evaluation in Product Software Development.

[18] Regnell, B., Ljungquist, B., Thelin, T., Karlsson, L., 2004. Investigation of Requirements Selection Quality in Market-Driven Software Processes using Open Source Discrete Event Simulation Framework.
[19] Shaw, M., 2002. What makes good research in Software Engineering? Paper presented at the Fifth European Joint Conferences on Theory and Practice of Software, Grenoble, France.
[20] Sommerville, I., 2001. Software Engineering, 6th Edition, Pearson Education.
[21] SSJ: Stochastic Simulation in Java. http://www.iro.umontreal.ca/~simardr/ssj/indexe.html
[22] Wohlin, C., Runesson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2000. Experimentation in Software Engineering – An Introduction. Kluwer Academic Publishers, Dordrecht.

Appendix A

Class Engineer

```
java.lang.Object
     |
     +--Resource
          |
          +--se.lth.re.Engineer
```

< Constructors > < Methods >

public class **Engineer**
extends Resource

The class represent an Engineer. The Engineer extends the Resource. In addition the Engineer has four skills which are all generated when an Engineer is created. The skills are generated from a triangular distribution. The parameters to the distribution (low, mean, high) can be configured in the REGlobal.

Author:
Christofer Tingström

Constructors

Engineer

`public` **`Engineer`**`()`

Default constructor for Engineer, generates all skills.

Engineer

`public` **`Engineer`**`(int id)`

Constructor for Engineer, generates all skills and sets the id of the Engineer object.

Methods

getSkillA

`public double` **`getSkillA`**`()`

getSkillB

```
public double getSkillB()
```

---

getSkillC

```
public double getSkillC()
```

---

getSkillD

```
public double getSkillD()
```

Class REGlobal

```
java.lang.Object
    |
    +--se.lth.re.REGlobal
```

< Fields > < Constructors >

---

public class **REGlobal**
extends java.lang.Object

This class contains global variables for the RE layer. The RE layer use this variables as input data for the different distribution which are used to generate data.

Author:

Christofer Tingström

Fields

engineerSkillHigh

```
public static int engineerSkillHigh
```

The highest possible value of an Engineer's skill

---

engineerSkillLow

```
public static int engineerSkillLow
```

The lowest possible value of an Engineer's skill

engineerSkillMean

`public static int` **`engineerSkillMean`**
The mean value of an Engineer's skill

requirementEffortHigh

`public static int` **`requirementEffortHigh`**
The highest possible value for Effort in a Requirement.

requirementEffortLow

`public static int` **`requirementEffortLow`**
The lowest possible value for Effort in a Requirement.

requirementEffortMean

`public static int` **`requirementEffortMean`**
The mean value for Effort in a Requirement.

requirementValueHigh

`public static int` **`requirementValueHigh`**
The highest possible value for Value in a Requirement.

requirementValueLow

`public static int` **`requirementValueLow`**
The lowest possible value for Value in a Requirement.

requirementValueMean

`public static int` **`requirementValueMean`**
The mean value for Value in a Requirement.

Constructors

REGlobal

```
public  REGlobal()
```

Class Requirement

```
java.lang.Object
    |
    +--Assignment
        |
        +--se.lth.re.Requirement
```

< Constructors > < Methods >

---

public class **Requirement**
extends Assignment

This class represent a Requirement which has several attributes. The Requirement extends the Assignment. A Requirement is generated in the RequirementGenerator.

Author:

Christofer Tingström

Constructors

Requirement

```
public  Requirement(double actualEffort,
                    double actualValue,
                    java.lang.String type)
```

The default constructor where the effort, value and type is set. An id is also set for the Requirement.

Methods

getActualEffort

```
public double getActualEffort()
```

---

getActualValue

public double **getActualValue**()

---

getEstimatedEffort

public double **getEstimatedEffort**()

---

getEstimatedValue

public double **getEstimatedValue**()

---

getPriortity

public int **getPriortity**()

---

getType

public java.lang.String **getType**()

---

isAlpha

public boolean **isAlpha**()

---

isConstructed

public boolean **isConstructed**()

---

isSplit

public boolean **isSplit**()

setActualEffort

public void **setActualEffort**(double actualEffort)

setActualValue

public void **setActualValue**(double actualValue)

setAlpha

public void **setAlpha**(boolean isAlpha)

setConstructed

public void **setConstructed**(boolean constructed)

setEstimatedEffort

public void **setEstimatedEffort**(double estimatedEffort)

setEstimatedValue

public void **setEstimatedValue**(double estimatedValue)

setPriortity

public void **setPriortity**(int priortity)

setSplit

```
public void setSplit(boolean split)
```

---

setType

```
public void setType(java.lang.String type)
```

Class RequirementGenerator

```
java.lang.Object
    |
    +--Process
        |
        +--JobGenerator
            |
            +--se.lth.re.RequirementGenerator
```

< Constructors > < Methods >

---

public class **RequirementGenerator**
extends JobGenerator

This class is frequently generating new Requirements. An RequirementGenerator object is created with a belonging lambda which is the number of requirement created per time unit. Each time when an Requirement is generated it is included in an Event which is inserted into the EventList, with a specific departure time. It is also possible to specify the probability of the type attribute belonging to a Requirement. The golden grain can also be set.

Author:

Christofer Tingström

Constructors

RequirementGenerator

```
public  RequirementGenerator(double lambda)
```

Constructs a RequirementGenerator job and the mean of the arrival intensity is attached.

Methods

setGoldenGrain

```
public void setGoldenGrain(double goldenGrainProb)
```

---

setNextStage

```
public void setNextStage(Process nextStage)
```

---

setTypeProbability

```
public void setTypeProbability(double typeAprob,
                               double typeBprob,
                               double typeCprob,
                               double typeDprob)
```

---

toString

```
public java.lang.String toString()
```

Overrides:

toString in class Process

---

treatEvent

```
public void treatEvent(Event e)
```

This class has an Event as input parameter and the destination of the event is set to the next stage in the process and the type attribute of the Event is changed to arrival. In addition a new Event with a belonging Requirement is created and the destination is set to this object with an exponential generated departure time. This Event will later be treated and then changed as the previous Event.

Parameters:

e - The Event which should be treated.

Overrides:

treatEvent in class Process

Class RepeatHelp

```
java.lang.Object
    |
    +--se.lth.repeat.RepeatHelp
```

< Fields > < Constructors > < Methods >

---

public class **RepeatHelp**
extends java.lang.Object

This class is a help class to RepeatStage, the purpose is to continuously create new releases and it also contains four global variables. The variables is the current JobGenerator, and the length of elicitation, selection and construction of the REPEAT process.

Author:

Christofer Tingström

Fields

constructionTime

`public static int` **`constructionTime`**

The length of the construction stage time in the REPEAT process. This is used both when new releases are created and as well as long as a construction object will be active.

---

currentJobGenerator

`public static JobGenerator` **`currentJobGenerator`**

The current JobGenerator. This is the JobGenerator that is generating Job. It is accessed several times to change the next stage attribute.

---

elicitationTime

`public static int` **`elicitationTime`**

The length of the elicitation stage time in the REPEAT process. This is used both when new releases are created and as well as long as an elicitation object will be active.

---

releasesMap

`public static java.util.HashMap` **`releasesMap`**

The HashMap contains all id and Stages that have been created. The id is the key.

---

selectionTime

`public static int` **`selectionTime`**

The length of the selection stage time in the REPEAT process. This is used both when new releases are created and as well as long as a selection object will be active.

Constructors

RepeatHelp

`public` **`RepeatHelp`**`()`

Methods

createNewRelease

`public static void` **`createNewRelease`**`(int releaseID)`

The method creates a new release. As three releases always are running simultaneous the next release will be the release id plus three

Parameters:

releaseID - the release which is used to create new released with the correct id.

Class RepeatStage

```
java.lang.Object
    |
    +--Process
        |
        +--Stage
            |
            +--se.lth.repeat.RepeatStage
```

< Constructors > < Methods >

public class **RepeatStage**
extends Stage

The RepeatStage is an extension of the Stage component. This class override a few methods in the Stage class to suit the REPEAT process. The REPEAT process is special as an RepeatStage object only exist for a limited period of time. In addition three releases are simultaneous running.

Author:

Christofer Tingström

Constructors

RepeatStage

```
public  RepeatStage(java.lang.String ID,
                    Rule rule,
                    se.lth.swprocess.Resource[] resources,
                    boolean firstStage,
                    int endTime,
                    DataHandler dataHandler)
```

Creates a RepeatStage object and the ID, rule, resources, firstStage, endTime and DataHandler attributes are set. This RepeatStage is also added to the release map in the RepeatHelp class.

Methods

getEndTime

```
public int getEndTime()
```

---

getReleaseID

```
public int getReleaseID()
```

---

getStageID

```
public java.lang.String getStageID()
```

---

treatEvent

```
public void treatEvent(Event event)
```

The Event is handled in this method. Firstly the service time is determined. Either is already stored in the Event, otherwise it is obtained from the specific rule connected to this object. Then the Event is treated, depending on the type attribute of the Event. Basically, this method determines the service time of an Event from the rule connected to the object and as well the next stage for the Event.

Parameters:

event - the Event which should be treated.

Overrides:

treatEvent in class Stage

Class DataHandler

```
java.lang.Object
    |
    +--se.lth.simulator.DataHandler
```

< Constructors > < Methods >

public class **DataHandler**
extends java.lang.Object

This class produces all output data. The output data is always produced from the method handleData. This method is always called when an Event is removed from the system. This method could be overridden to enable the user to collect and store output data of an Event when it is removed the system. However, the measurement conducted in the method in the Stage class is collected in the Stage class and the writeToFile is called. This class contains a unique id which is attached to every file to avoid overwritten files.

Author:

Christofer Tingström

Constructors

DataHandler

```
public  DataHandler()
```

Default constructor, sets the unique id.

Methods

generateResult

```
public void generateResult(Event event,
                           java.lang.String outputFile)
```

Generate Results from an event, the value and the effort is collected

Parameters:

event - the Event where the data is collected
outputFile - the file to store the collected data

getUniqueID

```
public java.lang.String getUniqueID()
```

---

handleData

```
public void handleData(Event event)
```

Handle data when an event has been removed from the system. This class should be overridden if the user of the framework wants to control the output data. This method is implemented to only measure the total time an Event spend in the system.

Parameters:

event - the event which has been removed from the system.

---

writeToFile

```
public void writeToFile(java.lang.String text,
                        java.lang.String outputFile)
```

This method write data to an output file

Parameters:

text - the text to write to the output file
outputFile - the file where the text should is written.

Class Event

```
java.lang.Object
    |
    +--se.lth.simulator.Event
```

< Constructors > < Methods >

---

public class **Event**
extends java.lang.Object

An Event is the message sent between the processes. The Event does in general contain an Assignment. However, there are also measure event which is specified by there type attribute. The Event contains several attributes as the destination of the Event, arrival time in the system, departure time and Resource which is handling the Event.

Author:

Christofer Tingström

Constructors

Event

```
public  Event()
```

Construct an Event object. A HashMap to store different service times is also created.

Event

```
public  Event(double departureTime,
              Process destination,
              int type,
              Assignment assignment)
```

Construction of an Event object and departureTime, destination, type and assignment is also set in this constructor. A HashMap to store different service times is also created.

Methods

getArrivalTimeInSystem

```
public double getArrivalTimeInSystem()
```

---

getAssignment

```
public Assignment getAssignment()
```

---

getDepartureTime

```
public double getDepartureTime()
```

---

getDestination

```
public Process getDestination()
```

---

getResource

public Resource **getResource**()

---

getResourceID

public int **getResourceID**()

---

getServiceTime

public java.lang.Double **getServiceTime**(Stage key)

---

getType

public int **getType**()

---

setArrivalTimeInSystem

public void **setArrivalTimeInSystem**(double arrivalTimeInSystem)

---

setAssignment

public void **setAssignment**(Assignment assignment)

---

setDepartureTime

public void **setDepartureTime**(double departureTime)

---

setDestination

public void **setDestination**(Process destination)

setResource

```
public void setResource(Resource resource)
```

setResourceID

```
public void setResourceID(int resourceID)
```

setServiceTime

```
public void setServiceTime(Stage key,
                           java.lang.Double value)
```

setType

```
public void setType(int type)
```

Class EventList

```
java.lang.Object
    |
    +--se.lth.simulator.EventList
```

< Constructors > < Methods >

public class **EventList**
extends java.lang.Object

This class is a List which acts as a Linked List and the Event stored in this List is ordered in an ascending order according to their departure time.

Author:

Christofer Tingström

Constructors

EventList

```
public  EventList()
```

Methods

add

```
public static boolean add(Event e1)
```

This method attaches and Event to the LinkedList The Event is sorted in an ascending order according to the departure time of the Event

Parameters:

e1 - the Event which should be inserted in the Linked List

Returns:

true is returned if the insertion was successful

---

createNewList

```
public static void createNewList()
```

This method creates a new LinkedList and removes the old one.

---

getFirst

```
public static Event getFirst()
```

This method return the first Event in the Linked list.

Returns:

the Event with the lowest departure time

---

isEmpty

```
public static boolean isEmpty()
```

This method checks if the Linked list is empty.

Returns:

true if the List is empty

---

numberOfJobsEventList

```
public static int numberOfJobsEventList(java.lang.String id)
```

A method to count the number of specific Events in a Linked List. The Events are grouped after their destination attribute.

Parameters:

id - the id to check, it is the id of the destination attribute

Returns:

the number of Events which matches the id

---

size

```
public static int size()
```

This method checks the size of the Linked list

Returns:

the size of the Linked List

Class Global

```
java.lang.Object
    |
    +--se.lth.simulator.Global
```

< Fields > < Constructors > < Methods >

---

public class **Global**
extends java.lang.Object

This class contains global variables and constants. Two methods for the seed is also implemented here, get and set. The setting of the seed needs to follow a few rules.

Author:

Christofer Tingström

Fields

ARRIVAL

```
public static final int ARRIVAL
```

Each Event can be in three states, either arrival, measure or departure

---

DEPARTURE

`public static final int` **`DEPARTURE`**
Each Event can be in three states, either arrival, measure or departure

---

MEASURE

`public static final int` **`MEASURE`**
Each Event can be in three states, either arrival, measure or departure

---

isSeed

`public static boolean` **`isSeed`**
if a seed is set.

---

time

`public static double` **`time`**
The time in the framework

Constructors

Global

`public` **`Global`**`()`

Methods

getSeed

`public static int[]` **`getSeed`**`()`

---

setSeed

```
public static void setSeed(int[] newSeed)
                   throws FrameworkException
```

This method sets the seed array. The first element in the array need to be greater than 1, the second greater than 7, the third greater than 15 and the last greater than 127

Parameters:

seed - the seed vector

Throws:

se.lth.swprocess.FrameworkException - is thrown if the seed is incorrect set.

Class Process

```
java.lang.Object
    |
    +--se.lth.simulator.Process
```

Direct Known Subclasses:
JobGenerator, Stage

< Constructors > < Methods >

---

public abstract class **Process**
extends java.lang.Object

This class determines the methods which need to be implemented to be able to handle Events. This class has two known subclasses, JobGenerator and Stage.

Author:

Christofer Tingström

Constructors

Process

```
public  Process()
```

Methods

toString

`public abstract java.lang.String **toString**()`

Overrides:

toString in class java.lang.Object

---

treatEvent

`public abstract void **treatEvent**(Event event)`

This method treats an Event and is implemented in the subclasses.

Parameters:

event - the Event which is treated

Class SampleFile

```
java.lang.Object
    |
    +--se.lth.simulator.SampleFile
```

< Constructors > < Methods >

---

public class **SampleFile**
extends java.lang.Object

The purpose of this class is to sample a file with a specific interval. The sampling is simple; reading an input file and producing an output file. The class also calculate a mean and a sampled mean. The sampling can also be conducted with an exponential interval.

Author:

Christofer Tingström

Constructors

SampleFile

`public **SampleFile**()`

Methods

getMean

```
public double getMean()
```

---

getMeanSampled

```
public double getMeanSampled()
```

---

sampleFile

```
public void sampleFile(java.lang.String inputFileName,
                       java.lang.String outputFileName,
                       double lambda)
            throws FrameworkException
```

This method samples a file according to the specified lambda.

Parameters:

inputFileName - the file to sample.
outputFileName - the file where the sampled points are stored.
lambda - the sample interval.

Throws:

se.lth.swprocess.FrameworkException - an exception is thrown if lambda is less than 1.

---

setExponential

```
public void setExponential(boolean exponential)
```

Class Assignment

```
java.lang.Object
    |
    +--se.lth.swprocess.Assignment
```

Direct Known Subclasses:
Requirement

< Constructors > < Methods >

---

public class **Assignment**
extends java.lang.Object

This class represents an Assignment which is always included in an Event and sent between the Processes. The Assignment is a base class and has one known subclass, Requirement. The Assignment is closely connected with the JobGenerator as the JobGenerator is extended with an RequirementGenerator and Assignment is extended with a Requirement.

Author:

Christofer Tingström

Constructors

Assignment

`public` **`Assignment`**`()`

Constructs an Assignment object.

Assignment

`public` **`Assignment`**`(int id)`

Constructs an Assignment object connected to an id.

Methods

getId

`public int` **`getId`**`()`

Class FrameworkException

```
java.lang.Object
    |
    +--java.lang.Throwable
        |
        +--java.lang.Exception
            |
            +--se.lth.swprocess.FrameworkException
```

All Implemented Interfaces:

java.io.Serializable

< Constructors >

---

public class **FrameworkException**
extends java.lang.Exception

The FrameworkException class purpose is to be a base class for the exceptions. The exception, which is thrown from the framework contains a specific message. At this time no Exception tree has been built, but if extensions are developed this should be done.

Author:

Christofer Tingström

Constructors

FrameworkException

`public` **`FrameworkException`**`()`

A FrameworkException object is created.

FrameworkException

`public` **`FrameworkException`**`(java.lang.String message)`

A FrameworkException object is created associated with a message.

Class JobGenerator

```
java.lang.Object
    |
    +--Process
        |
        +--se.lth.swprocess.JobGenerator
```

Direct Known Subclasses:

RequirementGenerator

< Constructors >

---

public abstract class **JobGenerator**
extends Process

This class extends the Process class and is a base for generating jobs. This class is abstract and contains no methods. This class is created to be a base, which should be extended to fit any software process. JobGenerator has one known subclass, RequirementGenerator.

Author:

Christofer Tingström

Constructors

JobGenerator

`public` **`JobGenerator`**`()`

Class Resource

```
java.lang.Object
    |
    +--se.lth.swprocess.Resource
```

Direct Known Subclasses:
Engineer

< Constructors > < Methods >

---

public class **Resource**
extends java.lang.Object

This class represent a Resource in the Framework. The purpose is that the Resource should be an base class. The Resource has one known subclass, Engineer. The Resource represent a server in the queuing network.

Author:
Christofer Tingström

Constructors

Resource

`public` **`Resource`**`()`

Construct a Resource object.

Resource

`public` **`Resource`**`(int id)`

Construct a Resource object and an id is attached.

Methods

getId

```
public int getId()
```

---

isAvailable

```
public boolean isAvailable()
```

---

setAvailable

```
public void setAvailable(boolean isAvailable)
```

Interface Rule

< Methods >

---

public interface **Rule**

The Rule interface enables complex rules for a Stage to be implemented. The Rule interface can always be used if the next stage of an Event is not simple to determine. In addition the Rule interface shall also be used if the service time for an Event depend on any attributes of the Assignment included in the Event or the Resource which is handling the Event.

Author:

Christofer Tingström

Methods

getServiceTime

```
public double getServiceTime(Event event,
                             Resource resource)
```

The service time is calculated and returned.

Parameters:

event - the event which service time is calculated for
resource - the resource which handles the event

Returns:

the service time for the event

nextStage

`public Process nextStage(Event event)`

Determining which is the next Stage, the logic of the rule is implemented in this method.

Parameters:

event - the next Stage can depend on the event

Returns:

the next Stage

---

setNextStage

`public void setNextStage(Process nextStage)`

This class can be of support to the measurements, if the queuing network is not complex. It is used for the RSQ model.

Class RuleBase

```
java.lang.Object
    |
    +--se.lth.swprocess.RuleBase
```

All Implemented Interfaces:

Rule

< Constructors > < Methods >

---

public abstract class **RuleBase**
extends java.lang.Object
implements Rule

This class is a base class which implements the interface Rule and implement the most basic methods as service time and the next stage.

Author:

Christofer Tingström

Constructors

RuleBase

`public RuleBase()`

Construct a RuleBase object

RuleBase

```
public  RuleBase(double serviceRate)
```

Construct a RuleBase object with a specific service rate

RuleBase

```
public  RuleBase(double serviceRate,
                 boolean exponential)
```

Construct a RuleBase object with a specific service rate which can be exponential

Methods

getServiceRate

```
public double getServiceRate()
```

getServiceTime

```
public double getServiceTime(Event e,
                             Resource resource)
```

The service time is calculated and the Event nor the Resource is considered. The service time can be exponential or 1/(service rate). This is parameter controlled.

Returns:

the service time as a double

isExponential

```
public boolean isExponential()
```

setNextStage

```
public void setNextStage(Process nextStage)
```

Class Stage

```
java.lang.Object
    |
    +--Process
        |
        +--se.lth.swprocess.Stage
```

Direct Known Subclasses:
RepeatStage

< Constructors > < Methods >

---

public class **Stage**
extends Process

The Stage class handles an Event when it arrives and when it departures. The Stage class can also perform measurement on the System. The Stage class contains the logic of the Resources allocation. If one queue is used or a separate one for each Resource.

Author:
Christofer Tingström

Constructors

Stage

```
public  Stage(java.lang.String ID,
              se.lth.swprocess.Resource[] resources,
              double serviceTime,
              boolean firstStage,
              boolean exponential,
              DataHandler dataHandler)
```

A Stage object is created without a Rule. The ID, the Resources, service time, firstStage, exponential and DataHandler attributes are set.

Stage

```
public  Stage(java.lang.String ID,
              Rule rule,
              se.lth.swprocess.Resource[] resources,
              boolean firstStage,
              DataHandler dataHandler)
```

A Stage object is created with a Rule. The ID, the Resources, service, firstStage, and DataHandler attributes are set.

Methods

getDataHandler

`public DataHandler getDataHandler()`

---

getNumberOfJobsInSystem

`public int getNumberOfJobsInSystem()`

This method recursively determining the amount of Event in this Stage and the other Stages in the system.

Returns:

the number of Jobs in the system.

---

getNumberOfServers

`public int getNumberOfServers()`

---

getRule

`public Rule getRule()`

---

isFirstStage

`public boolean isFirstStage()`

---

setMeasureInterval

`public void setMeasureInterval(double measureInterval)`

---

setNextStage

`public void setNextStage(Process nextStage)`

---

setOneQueue

```
public void setOneQueue(boolean oneQueue)
```

---

toString

```
public java.lang.String toString()
```

Overrides:

toString in class Process

---

treatEvent

```
public void treatEvent(Event event)
```

This method handles an Event and is generating a new departure time for the Event by adding a service time. In addition the destination is also set here. If the servers already are busy the Event is stored in a queue and is handled when a server is available. The choice of the server depend if all servers have one common queue or a separate one.

Parameters:

event - the Event which shall be handled.

Overrides:

treatEvent in class Process