NTNU – Trondheim
Norwegian University of
Science and Technology

Human Reliability and Software Development

Merete Aardalsbakke

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Tor Stålhane, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Sammendrag

Begrepet "Human Reliability" har blitt viktig innen høyrisiko industrier. Interessen har vokst også innenfor programvareutvikling for å redusere menneskelige feil og deres negative innvirkning på programvareutvikling. Menneskelige feil koster IT industrien store mengder tid og penger hvert år.

SHERPA er en "Human Reliability" metode laget for å passe flere industrier. Denne masteroppgaven foreslår at noen justering er nødvendig for å gjøre metoden mer passende for programvareutvikling. For å evaluere SHERPA har det blitt utført to forskingsfaser, en fokusgruppe og et eksperiment. Fokusgruppen hadde to formål, først å utføre en hierarkisk oppgaveanalyse, det første steget av SHERPA, og deretter diskutere viktige aspekter ved en programmeringsmodell. Funnene fra fokusgruppen ble brukt for å justere SHERPA videre før eksperimentet.

Formålet med eksperimentet var å teste SHERPA på et sett med oppgaver, og å evaluere justeringene gjort med SHERPA før eksperimentet. Funnene fra eksperimentet ble brukt til å diskutere og evaluere SHERPA og justeringene. En ny versjon av SHERPA, tilpasset programvareutvikling blir presentert i denne masteroppgaven.

Etter å ha utført to faser med forsking og datainnsamling, kan det bli konkludert, basert på resultatene fra dette studiet, at SHERPA er et nyttig verktøy for å utforske menneskelige feil innenfor programvareutvikling.

Abstract

Human Reliability has been an important term within high-risk industries. The interest has emerged within software development to reduce human errors and their negative impact on software engineering. Human errors cost the software industry an enormous amount of time and money every year.

SHERPA is a Human Reliability method made to suit several domains. However, the project report suggests that a few changes are necessary to suit software development. To evaluate SHERPA two phases of research was conducted, a focus group session and an experiment. The focus group session was conducted prior to the experiment. The focus group had two agendas, firstly to conduct a hierarchical task analysis, the first step of SHERPA, and secondly to discuss important aspects of a programming behavioral model. The findings from the focus group session were used to make adjustment to SHERPA before the experiment.

The purpose of the experiment was to test SHERPA on a set of predefined tasks, and to investigate the adjustments made to SHERPA prior to the experiment. The findings from the experiment were discussed and used to evaluate SHERPA as well as the adjustments. A new version of SHERPA, more suitable for software development, is presented in this master thesis.

After conducting two phases of research and data collection, it can be concluded, based on the results from this study, that SHERPA is a useful tool in exploring human errors in software development.

Preface & Acknowledgements

This study is a master thesis conducted in the last semester of masters degree program in Computer Science at the Norwegian University of Technology and Science(NTNU). The specialization of this thesis is Software. The research conducted in this thesis is for the Department of Computer and Information Science.

I want to acknowledge the different persons that have taken part in this research. Firstly, I would like to give a big thank to my supervisor Professor Tor Stålhane for much appreciated guidance and feedback during the semester of study, in the spring of 2014. I would also like to give a special thank to Esben Aarseth, fellow student, for his valuable help and sharing of experience. Lastly I would like to give a thank to the participants who attended the focus group, for helping to conduct the Hierarchical Task Analysis and provide useful information about their experience with errors in software development.

Contents

List of Figures

List of Tables

Abbreviations

HRA	**H**uman **R**eliability **A**analysis
HR	**H**uman **R**eliability
PSF	**P**erformance **S**haping **F**actors
HEP	**H**uman **E**rror **P**robabilities
HEART	**H**uman **E**rror **A**ssessment and **R**eduction **T**echnique
THERP	**T**echnique for **H**uman **E**rror **R**ate **P**rediction
CREAM	**C**ognitive **R**eliability **E**rror **A**nalysis **M**ethod
SHERPA	**S**ystematic **H**uman **E**rror **R**eduction & **P**rediction **A**pproach
SPAR-H	**S**tandarized **P**lant **A**nalysis **R**isk-**H**uman Reliability Analysis
SRK	**S**kill- **R**ule- **K**nowledge- based approach
GEMS	**G**eneric **E**rror **M**odelling **S**ystem
HEI	**H**uman **E**rror **I**dentification
HTA	**H**ierarchical **T**ask **A**analysis

Part I

Introduction

Chapter 1

Introduction

This chapter will introduce the motivation for doing the research and the research questions formulated to drive the research in this study. The scope of the thesis and a thesis outline is also presented.

1.1 Motivation

Errors are made in all industries, including software development. In software development these errors are referred to as bugs, and most of these bugs arise from mistakes and errors made by people in either a program's source code or in its design. Bugs are a consequence of the nature of human factors in the programming task.

These bugs cause a lot of trouble, and may in fact be extremely expensive. CEO of Undo Software, Greg Law put in this way:
"To put this in perspective, since 2008 Eurozone bailout payments to Greece, Ireland, Portugal and Spain have totaled $591bn. This is less than half the amount spent on software debugging over that same five-year period."

The statement of Greg Law [1] shows how incredibly expensive it is to correct errors made by software engineers. As correcting these errors are expensive, time-consuming and leads to bad software, we wish to find a way to help programmers avoid making these errors. Schulmeyer presented the need for a model on programmer behavior in his article about Net Negative Producing Programmers [2].

A lot of research has been made on human reliability, especially in hazardous industries like nuclear power plants. Within software development there has been done little to no significant research on this subject. The work done in human reliability concerns high risks industries and was executed to prevent the accidents attributable to human error [3].

There exists numerous HRA models, and through *TDT-4501 Specialization Project* in the fall of 2013 the Systematic Human Error And Prediction Approach, SHERPA, was evaluated to be the model best suited for software development. The motivation for this master thesis is to explore the model further, and to investigate if it may be suitable for software development.

1.2 Research Questions

With the goal of exploring whether or not it is possible to use an HRA model to prevent developers from committing human error in software development, five research questions has been formulated.

1. *RQ1. Is it possible to successfully apply HRA to software development?*

   This research question concerns whether or not it is possible to apply an HRA model to software development. The question has been transferred from the specialization project conducted in the fall semester of 2013. In this thesis the overall focus has been on one HRA model, and this question can only be answered based on the result from this particular HRA model.

2. *RQ2. What adjustments are needed for SHERPA to be better tailored to software development?*

   SHERPA is developed for the process industriy, and is thereby dominated by attributes that support this industry. To make sure that SHERPA can be useful in software development, changes need to be made. This research questions concerns what these adjustments are.

3. *RQ3. Will a set of non trained students be able to conduct SHERPA on a set of problems?*

   If the HRA-model is to be used in the field of software development it is important that it is easy to comprehend. This research question is asked to ensure that the model is possible to use with limited training.

4. *RQ4. Will the students reach similar solutions?*

   This research question concerns if this HRA method will give somewhat the same problem areas within the software development process, when analyzed by different

analysts. This issue is important when considering the usefulness of the method. The consistency of the model is an important issue to consider.

5. *RQ5. Will these solutions be useful?*

   To ensure that SHERPA is useful, it is important to consider if the results it gives are relevant relative to other research that has been conducted within the field of software development.

1.3 Thesis Scope

This master thesis will focus on the use of the Human Reliability Analysis (HRA) method SHERPA in the field of software development. The thesis aims to evaluate a HRA model suited for software development, and the necessary adjustments needed to make the model applicable to software engineering. The thesis will only evaluate the method Systematic Human Error Reduction and Prediction Approach, SHERPA.

In this master thesis we are not able to test every part of SHERPA. There will be a focus on the adjustments made, to evaluate the need and usefulness of them. Further, the focus will be on if and how the method is useful within the field of software engineering. The scope of this thesis concerns the usefulness of SHERPA, and not the quality of the results given from the students. However, the results are needed to consider whether SHERPA provides usefulness of SHERPA. But the focus is not on the quality of the results.

1.4 Thesis Outline

The master thesis has been divided in into six parts: Part 1 Introduction, Part 2 Pre Study, Part 3 Research Methods and Research Design, Part 4 Research Procedure and Results: Focus Group, Part 5 Research Procedure and Results: Experiment and Part 6 Discussion and Conclusion.

Part 2 provides the report with basic understanding of the task, and detailed information about SHERPA.

Chapter 2 Human Reliability

Chapter 3 Human Error

Chapter 4 Background Information

Chapter 5 SHERPA

Part 3 Research Methods and Research Design provides infomation about how the research of this study was designed.

Chapter 6 Research Methods

Chapter 7 Validity of Research Methods

Chapter 8 Research Design

Part 4 Research Prodecure and Results: Focus Group presents procedure and results from the focus group session in addition to an introduction to what is conducted in the focus group

Chapter 9 Herarchical Task Aanalysis

Chapter 10 Results From Focus Group

Part 5 Research Prodecure and Results: Experiment presents the adjustments made to SHERPA, and detailed information about the procedure and the results found during the experiment.

Chapter 11 Adjustments made in SHERPA

Chapter 12 Procedure

Chapter 13 Results and Findings

Part 6 Discussion and Conclusion provides a discussion on the results from the results, and a detailed discussion regaring the research question asked in this thesis. A new verison of SHERPA applied to software development, with anexample is provided. A conclusion is provided as well as Further work

Chapter 14 Discussion

Chapter 15 SHERPA

Chapter 16 Validity

Chapter 17 Conclusion

Chapter 18 Further work

Part II

Pre Study

Chapter 2

Human Reliability

This chapter contains information on Human Reliability and a detailed description of Human Reliability Analysis.

2.1 Human Reliability

Human reliability is a concept related to human factors and ergonomics on how humans perform in manufacturing, medicine and generally in all working areas. Swain and Guttman [4] define human reliability as the probability that a person (1) correctly performs an action required by the system in a required time and (2) that he does not perform any extraneous activity that can degrade the system. There are other qualitative definitions, for instance in Hacker [5], related to the human ability to adapt to changing conditions in disturbances.

There has been a lot of research on human reliability related to critical systems like nuclear plants and air traffic management [1]. The researchers found that the majority of accidents are related to either human error or bad management. For critical systems like nuclear power plants these errors pose a tremendous risk which we can not afford.

Bell and Holroyd did a study on human reliability, and found that there exist 71 HR-models [6]. The HRA model are classified as first, second and third generation. The first generation tools were developed to help risk assessors predict and quantify the likelihood of human error. A trend for these models is that they encourage the assessors to break tasks down into smaller components to consider the impact of modifying time pressure, equipment design and stress [6]. Second generation models are all models developed after the 1990s. These models attempt to consider context and errors of commission in human error prediction. New tools, based on first generation tools, are now emerging and are referred to as third generation methods [6].

2.2 Human Reliability Analysis

This section covers human reliability analysis, in addition to two sections that covers background information needed to understand the analysis.

2.2.1 Performance Shaping Factors (PSF)

Human behavior are affected by several factors, such as adaptability, flexibility and task environment [7]. These factors are what we call performance shaping factors, PSF. According to Mackieh and Cilingir [8], these factors can be divided into internal, external and stressor performance shaping factors. The external factors include the entire work environment [7], like written procedures and oral instructions. The internal factors represents a person's individual characteristics [7], his skills, motivations, and expectations that may influence performance. The stressors results from a work environment in which the demands placed on the operator by the system do not conform to his capabilities and limitations [7].

2.2.2 Human Error Probability (HEP)

Human Error Probabilities (HEPs) refers to the prediction of the likelihood or probability of human errors [9]. The definition of HEP is:

$$\text{HEP} = \frac{number\ of\ errors\ occured}{number\ of\ opportunities\ for\ error}$$

Kirwan stated in his article that HRA's central tenet is to keep the HEP estimate process as accurate or at least conservative as possible, rather than optimistic [9]. We want to keep the estimates this way to avoid underestimating the risk, or to make wrong errors highlighted for reduction. The validation of the HRA quantification techniques relies on the collection of real human error probabilities, to compare the techniques to real world data and thus make them empirically validated.

2.2.3 Human Reliability Analysis

Human reliability analysis is an analysis that helps us to better understand what are causing errors and faults in systems. Essentially, HRA aims to quantify the likelihood of human error for a given task [10]. HRA assist in identifying vulnerabilities within a task, and may also provide guidance on how to improve reliability for the specific task.

In HRA there are several steps needed to perform a complete analysis. The steps are presented in Figure 2.1 and the following sections will describe the general human reliability process.

2.2.3.1 Problem Definition

The first step is problem definition, which is used to determine the scope of the analysis, the type of analysis that will be conducted, the tasks that will be evaluated and what human actions will be assessed. According to NASA's report on HRA [11] there are mainly two factors that impact the determination of scope of the analysis: the systems vulnerability to human error and the purpose of the analysis. If a system is highly vulnerable to human error, a larger scope is needed to fully understand and mitigate the human contribution to system risk. The purpose of the analysis is important when we determine whether we need a qualitative or quantitative analysis.

2.2.3.2 Task Analysis

The second step is task analysis, which is a systematic method used to identify and break down tasks into subtasks that describes the actions required by humans to achieve the systems goal. A task analysis is conducted after a functional analysis. In this analysis, functional flow diagrams are developed to accentuate the chronological sequence of functions. A comprehensive task analysis identifies all human actions and serves as a building block for understanding where human error can occur in the process.

2.2.3.3 Error Identification

Error identification is the third and most important step according to NASA [11]. By error identification, we mean human error identification where human actions are evaluated to find which human errors and violations can occur. During this step it is important to find the type of error, as well as what kind of performance factors that could contribute to the specific error. See section 2.2.1.

2.2.3.4 Error Representation

The next step is error representation, also described as modeling. In this step data, relationship, and interference is visualized. This is done to better understand situations that cannot easily be described with words alone. The human errors are modeled and represented in Master Logic Diagram, Event Sequence Diagram, Event Tree, Fault Tree or a generic error model. There are also other error modeling techniques that can be used. During this step it is important for the analyst to consider dependencies between different types of human errors in order to get a better perspective.

2.2.3.5 Quantification and Integration

Quantification and Integration into PRA (Probabilistic Risk Assessment) is the step where probabilities are assigned to the errors, and it is in this step that we decide upon which errors are the most significant errors to the overall system risk [11]. When the most dominant errors are selected and probabilities and failure estimates are assigned, the analysts may start to make decisions about the human-machine interface. The specific steps in quantification are dependent on which human reliability method are being used.

2.2.3.6 Human Error Management

Human Error Management is the last step of an HRA. The philosophy of human error management is to assume that humans will always make mistakes. Even though we are trained to a set of tasks, there will be mishaps due to human errors. In human error management the idea is to develop a system that will minimize errors, but at the same time tolerate those that are not crucial to the system and will not lead to any serious failure or lead to mishap.

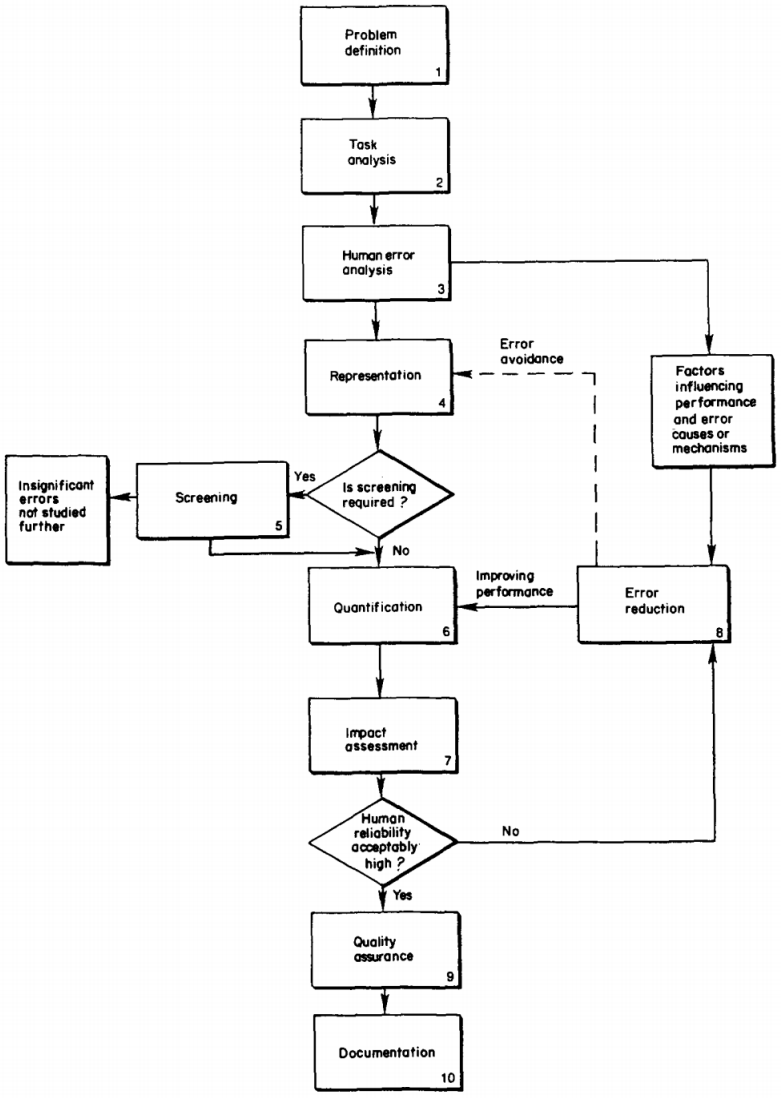

FIGURE 2.1: HRA process

Chapter 3

Human Error

This chapter contains detalied information on Human Error, and provides information on human errors made during software development.

3.1 Human Error

Human error is defined as an action that is not intended or desired by the human, or a failure on the part of the human to perform a prescribed action within specified limits, accuracy, sequence, or time such that the action or inactions fails to produce the expected result, and led to or has the potential to lead to an unwanted consequence [11]. Human errors are those errors that occur due to a human mistake. It basically means that what was to be done, was either not done, done wrong or out of its scope.

3.2 Slips and Mistakes

Human errors can, according to J.Rasmussen, be broken down into slips and mistakes [12]. The models made by Reason and Rasmussen are said to have questionable relevance to software development, as they were developed to minimize mistakes in hazardous sectors like nuclear, chemical and offshore. However, software development is a creative process equally as the processes intended for the models. In addition, the terms of skill, rule and knowledge based errors are suitable for mistakes made in software development as well.

The terms skill, rule and knowledge based information processing refer to the degree of conscious control exercised by the individual over his or her activities [12]. It provides a useful framework for identifying the types of error that are likely to occur at different situations. The skill, rule and knowledge based approach is a classification developed by Reason and Rasmussen [12]. Rasmussen concluded that an individual would use a

skill to deal with a problem-free task, use rule-based behavior for handling a routine problem, and resort to first principles to deal with a novel problem [13].

James Reason has analyzed human errors and categorized them into mistakes and slips. Mistakes are errors in choosing an objective or specifying a method of achieving this objective whereas slips are errors in carrying out intended method for reaching an objective. Norman explains that the division occurs at the level of the intention: a person establishes an intention to act. If the intention is not appropriate, this is a mistake. If the action is not what was intended, it is a slip [14]. Figure 3.1 shows Reason's distinction of human behavior. The project report will look at human errors and not consider violations.

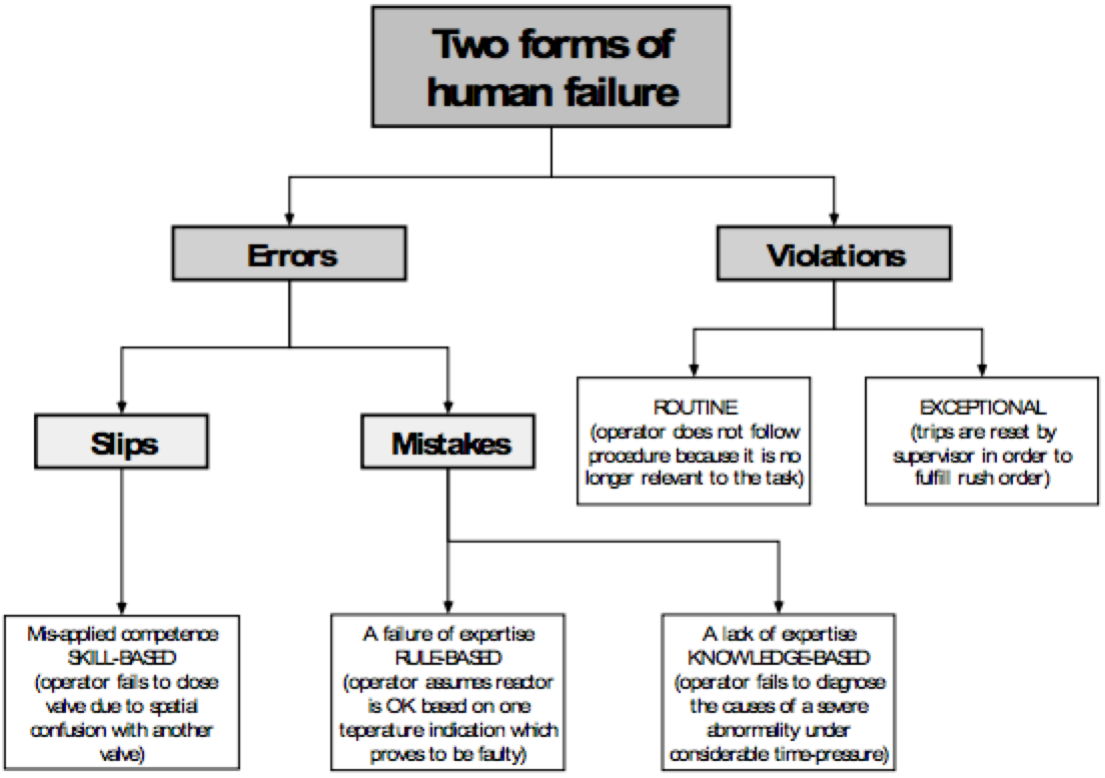

FIGURE 3.1: Classification of Human Error (Reason 1990)

3.2.1 Skill-based Error

Skill-based error refers to slips which is misapplied competence, as we see from Figure 3.1. In this behavior the individual is able to function effectively by using pre-programmed sequences of behavior, which do not require much conscious control [12] (see figure 3.2). The situations of skill-based behavior requires highly practiced and essentially automatic behavior with only minimal conscious controls [3]. An example of skill-based behavior could be driving along a familiar route in a car.

3.2.2 Rule-based Error

In rule-based behavior, an error of intention can arise if an incorrect diagnostic rule is used. Rule-based error occur when the situation deviate from normal, but can be dealt with by the operator consciously applying rules which are either stored in memory or are otherwise available in the situation. [3]. An example of rule-based mistake could be that a developer used the syntax in java when writing code in C.

3.2.3 Knowledge-based Error

In the case of knowledge-based mistakes there are other important factors, as knowledge-based error occurs when there are no predefined behavior. Most of these factors arise from the considerable demands on information processing capabilities of the individual that are necessary when a situation has to be evaluated from first principles [12]. Human does not perform well in highly stressed and unfamiliar situations where they need to act quickly without any known rules. There are described a wide range of failure modes in these conditions:

Out of sight, out of mind- effect:
only the information, which is readily available, will be used to evaluate the situation.

I know I'm right-effect:
problem solvers become over-confident in the correctness of their knowledge.

Encystment:
when the individual or operating team focuses in one aspect of the problem, and exclude all other considerations.

Vagabonding:
an overloaded worker gives his/her attention superficially to one problem after another without solving any of them.

In Figure 3.2 the relationship between human errors and the consciousness is depicted. We can see that little consciousness is necessary in skill-based behavior, but it increases with each level of human behavior.

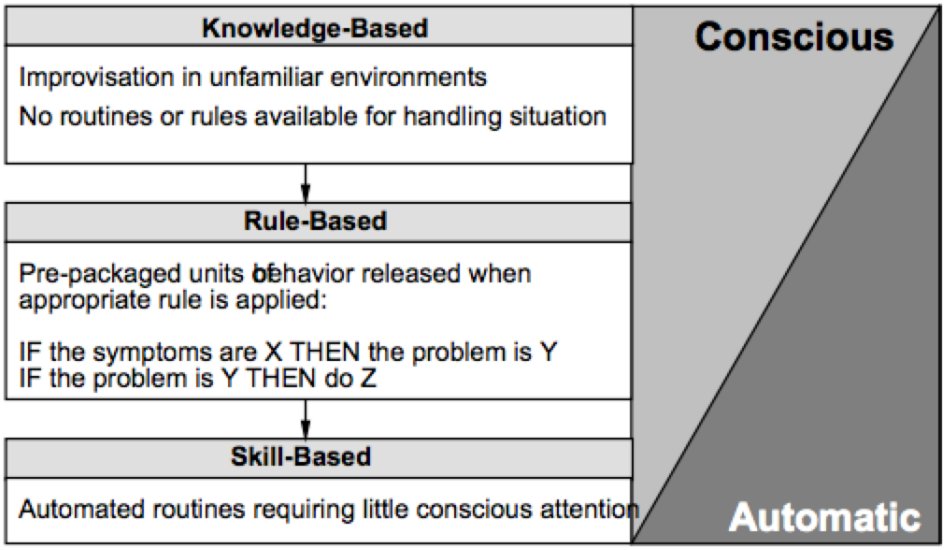

FIGURE 3.2: The Continuum between Conscious and Automatic Behavior (Reason 1990)

3.3 Swiss Cheese Model

James Reason has done research on human error and found that it can be viewed in two ways: the person approach and the system approach [15]. The person approach focuses mainly on people executing the error, while the system approach sees the error more as a consequence than a cause.

Reason came up with the Swiss cheese model of system accidents. High technology systems have many defensive layers: some are engineered, others rely on people, and some depends on procedures and administrative control. Reason compared these layers as slices of Swiss cheese, with holes representing faults, see Figure 3.3. The presence of one hole in any of the slices of cheese does not need to cause a bad outcome. Usually it will only result in a bad outcome if there are overlapping holes in all the layers.

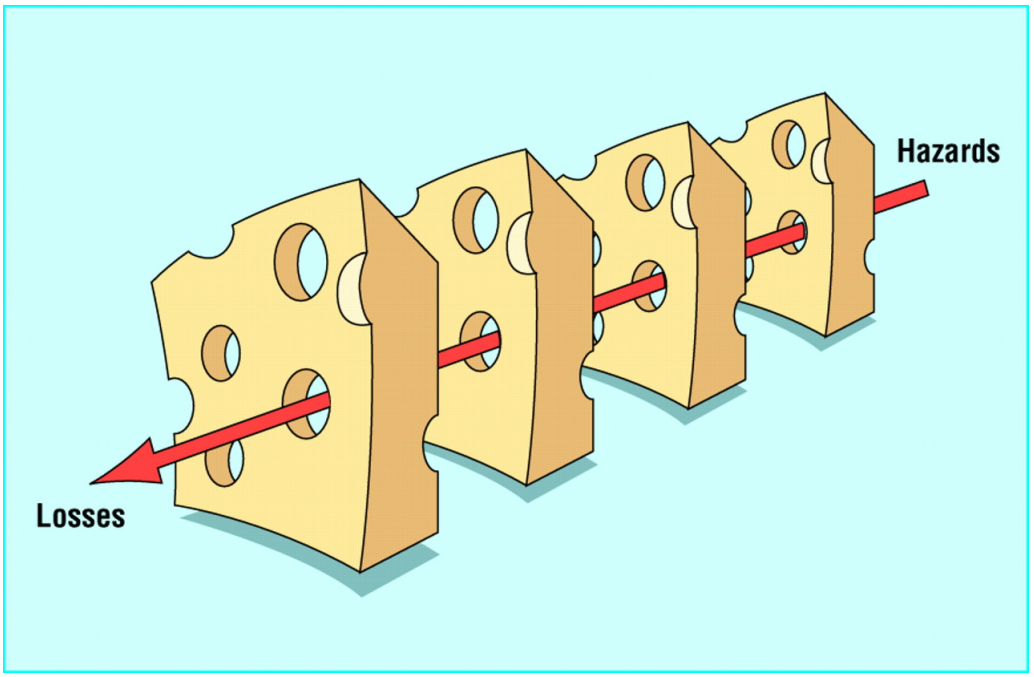

FIGURE 3.3: Swiss Cheese Model (Reason)

3.4 Disturbances on Human Performances

Humans are affected by everything going on around them. We are easily distracted, are generally unreliable and gets tired. Human behavior and performance has a vital part in the second generation of HRA. SPAR-H (see section 4.2), is mainly based on human behavior. Included in the model, there is a human behavior model (see Figure 3.4). The behavioral sciences literature reveals eight summary operational factors listed in the figure. These operational factors can be directly associated with a model of human performance [16].

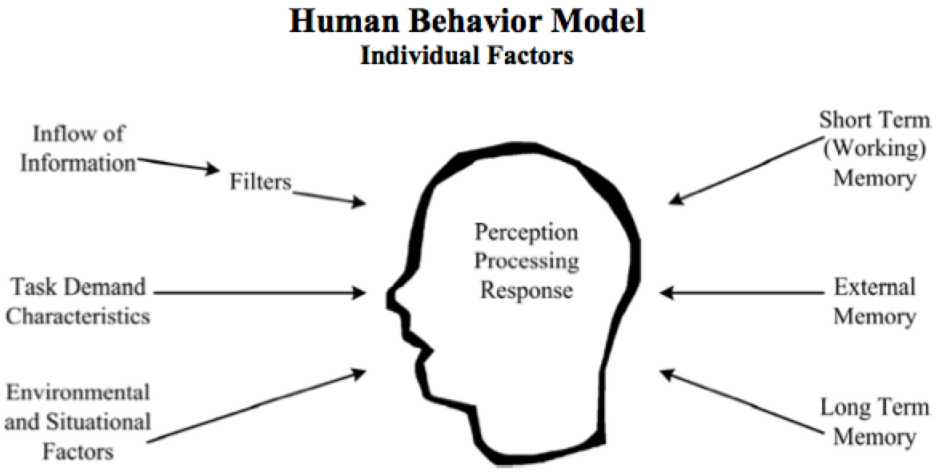

FIGURE 3.4: Human Performance Model

Robert J. Latino [17] wrote an important article on how sleep/wake cycles affect human's performance. There are several factors about our sleeping pattern that may have an effect, and one of these are how much sleep we get compared to how much we should receive. If we do not get enough, this will be registered in our brain and will disturb how we perform. Humans have a circadian rhythm which is the pattern of psychological and behavioral processes timed to about 24 hours. The circadian rhythm describes our sleep/wake cycles and influence human behavior in our workday cycles. It identifies fatigue points during our wake state. As human are more prone to human error when highly fatigue, we might be able to work around the critical points of fatigueness. When being aware of people's critical points we may be able to not assign critical tasks to people on their fatigue cycle. There has also been done some research indicating that human performance is lowest on the first day of work after workdays off. These disturbances are considered important in the second-generation models, like CREAM and SPAR-H.

Another research paper focuses on how noise disturb humans in their performance in their day to day work. Noise affects a wide area of human behaviors that have implications for health and well-being [18]. Over a long period of time, chronic noise exposure may even elevate to psychological stress. The research paper explains how it will affect the performance of a human being when exposed to high level of noise. Under noise, humans process information faster in working memory, but at the cost of capacity. An example is to memorize a list when exposed to noise. The subject will remember the last things they memorized, even better than when not exposed to noise, but greater errors occur farther back in the list [18].

This section has provided two examples of major disturbances on human performance. Human performance is affected by more factors than what mentioned in this section. However, it is important to keep in mind that humans are always affected by both

external and internal factors. Human behavior and performance will vary from person to person, and from day to day. The methods that help preventing human errors need to consider this in its design.

3.5 Software Errors

Humans are error prone, and will always make mistakes. Since humans make software there will occur human errors in software development. In software development these errors are referred to as bugs. A majority of bugs arise from misakes and errors made by people in either a programs source code or in its design.

HRA has been used in high-risk industries, and one might think that software development is a low risk industry. However, software errors are not just extremely expensive, but as the world is digitized, software becomes a bigger part in all other industries as well. There has also been software bugs with major consequences. A tech blog posted an article about 10 historical software bugs with extreme consequences [19]. Among them is a hole in the ozone layer that stayed undetected for a long lime due to software error. In 1994 a helicopter crashed leading to 29 lives lost, and this was all due to a system errors. Every year, software errors cause massive amounts of problems all over the world. We know that a lot of these could be avoided with more careful testing. Unfortunately, testing is the part of the software development process that are left out if there are some time restraints, or budget overturns. This might lead to bad software quality and a lot of expensive corrections after the release of a product.

Errors in software development can occur at any stage during the software development process. It is convenient to split errors made by software developers into two broad categories: development errors and debugging errors [20]. Development errors are made when developers are engaged in the developement of software, e.g. design and coding activities. Debugging errors occur when the developers try to fix a known error in the software, and the error is not corrected properly or the correction leads to new errors.

One error in software development might lead to several faults in the program. As an example, we can consider a scenario where the developer has misunderstood the syntax of the programming language that are being used, which is classified as a development error. Every time the programmer writes in this specific language, several faults are injected into the system.

G. Gordon Schulmeyer wrote about what he called net negative producing programmers, NNPPs [2]. He stated that in all teams there would always be at least one out of ten of the team members that were NNPPs. The NNPPs are said to spoil more than they produce, or in other words, that their spoilage exceeds their productions.

Schulmeyer states that in a team of ten, we can expect as many as three people to have a defect rate high enough to make them NNPPs. With a normal distribution of skills, the probability that there are no NNPPs in a team of ten is virtually zero [2]. In a high-defect project there might be half of the team that are NNPPs. It is important to mention that there are not just NNPPs that makes mistakes in a project. All humans will make some mistakes, as we are error prone. The study of NNPPs is relevant because they consider a lot of human factors in software development.

Chapter 4

Background Information

In this section background information about previous work we have done on this project is provided.

4.1 Specialization Project

This master thesis is a continuation of *TDT 4501 Specialization Project* conducted in the previous semester in the fall of 2013. General information about human reliability and human error is presented in the previous chapters. In this section the results from the specialization project are presented.

4.1.1 HR-methods

One of the challenges of the project was to find HRA models to consider. There are identified 71 different human reliability models [6], and there are probably more. The models that were evaluated in this experiment were seven of the best-known HRA methods. These methods were THERP, CREAM, HEART, SHERPA, SPAR-H, SRK and GEMS.

The HR-methods was evaluated based on a set of seven criteria's. These criteria's were:

CR1 How domain general is the model

CR2 How much training is necessary to use the method

CR3 How easy it is for a non HRA-expert to use the model

CR4 How much need for extra equipment, e.g. software, hardware

CR5 It should be possible to apply the method on different problems

CR6 How documantable is the method

CR7 How consistent is the method

Each criterion was assigned a weight, and then each of the HRA method was rated between one and five according to how well they met the criteria. Several of the methods got high scores in the evaluation. However, there were two models that stood out compared to the other models, SHERPA, with the highest score, and SPAR-H a few points below. In the specialization project the focus continued on SHERPA as it got the highest overall score.

SHERPA was originally developed for the process industry, see chapter 5 for more information about the HRA-method. As the process industry has different work approach than software development, it is likely there are parts that are unnecessary, and other parts in software development that will need support.

Few adjustments were made in the specialization project, but possible changes were identified as further work. The only change that was made concerned notation in one of the steps in SHERPA, and did not affect the method in a discernible way.

4.2 SPAR-H

SPAR-H scored three points below SHERPA in the evaluation of the HRA methods. SPAR-H is a second-generation method, unlike SHERPA belonging to the first generation of HRA. In further work in the specialization project we suggested to investigate SPAR-H further.

Standardized Plant Analysis Risk-Human Reliability Analysis [16] (SPAR-H), can be used both as a screening method and as a detailed analysis method [11]. The method has worksheets that allow analysts to provide complete descriptions of the tasks and capture task data in a standard format. HEPs are provided for four combinations of error type and system activity type, which are adjusted based on eight basic PFSs and dependency. The SPAR-H method is straightforward, easy to apply, and is based on a human information-processing model of human performance and results from human performance studies available in the behavioral science literature.

SPAR-H is an interesting HRA method and is possibly as applicable to software development as SHERPA. The method is well documented, and has been tested in a few domains and proved successful [16]. As SPAR-H is a second-generation method, while SHERPA is a first generation method it could be interesting to apply and tailor both methods to see the different results and approaches.

After further considerations we decided to keep all attention on SHERPA. If two methods were to be tested in the experiment, the number of participants in the experiment had

to be increased. SHERPA is a familiar method, as it was investigated in the previous semester. SPAR-H is a method with a lot of information, and time had to be set aside to learn the method thoroughly. Instead of using this time on learning SPAR-H, the time is rather spent getting to know SHERPA better and do more thorough investigation on adjustments needed.

Chapter 5

SHERPA

This chapter contains detailed information about SHERPA, the SHERPA procedure and gives an example for better understanding of the analysis.

5.1 SHERPA

The Systematic Human Error Eeduction and Prediction Approach, SHERPA, was developed by Embrey as a human-error prediction technique [18]. The technique is based on HTA as a description of normative, error-free behavior. The analysts use this description as a basis to consider what can go wrong during task performance. Basically SHERPA is a task and error taxonomy. The error taxonomy is continually under revision and development, and is thus considered as a work in progress. SHERPA uses hierarchical task analysis together with an error taxonomy to identify credible errors associated with a sequence of human activity [18]. The method works by indicating which error modes are credible for each task step in turn, based upon an analysis of work. Research comparing SHERPA with other human error identification methodologies suggests that it performs better than most methods in a wide set of scenarios [18].

Most of the human error prediction techniques, including SHERPA, have two key problems. The first one relates to the lack of representation of the external environment or objects [18]. Human error analysis technique has a tendency to treat the activity of the device and the material with which the human interacts in only a passing manner. Stanton claims that HRA often fails to take adequate account of the context in which performance occurs [18]. The second key problem is that th methods put a lot of responsibility on the judgment of the analyst. This will lead to different results from different analysts. Interanalyst reliability occurs when different analysts make different predictions regarding the same problem, while intraanalyst reliability is when the same analyst make different judgments on different occasions. This uncertainty may weaken the confidence in the predictions being made.

SHERPA has been used in seversl industrial sectors. It was initially designed to assist people in the process industry, like nuclear power, petrochemical processing, oil and gas extraction and power distribution. In 1994, SHERPA was applied to the procedure of filling a chlorine road tanker, and in 2000 it was applied to oil and gas industry. The domain has broadened in recent years, and is now including ticket machines, vending machines and in car radio cassette machines [18].

5.2 Procedure

SHERPA consists of eight steps. The explanations of each step is taken from [18].

Step 1: Hierarchical Task Analysis (HTA)

The process begins with the analysis of the work activities, using HTA. HTA is based on the notion that task performance can be expressed in terms of hierarchy of goals, operations, and plans [18]. Goals is what the person is seeking to achieve, operations are the necessary activities executed to achieve the goals, and plans are the sequence in which the operations are executed.

The analyst begins with an overall goal of the task, which is then broken down into sub goals. Further, plans are introduced to indicate in which sequence the sub activities are performed. The analysts decides when this certain level of analysis is sufficiently comprehensive, and will move on to scrutinize the next level.

An example of a HTA is in Figure 5.1 in section 5.3.

Step 2: Task Classification

Each of the operations found during HTA is classified based on the error taxonomy into one of the following behavior:

- Action: Action errors are classified into: process/operation, too long/too short, operation/process mistimed, operation in wrong direction, operation too little/much, misaligned, right operation on wrong object, wrong operation on right object, operation omitted, operation incomplete and wrong operation on wrong object.
- Retrieval: Retrieval errors are classified into: information not obtained, wrong information obtained and information retrieval incomplete.
- Checking: Errors in this category are classified into: check omitted, check incomplete, right check on wrong object, wrong check on right object, check mistimed and wrong check on wrong object.
- Selection: Errors in this category are classified into: selection omitted and wrong selection made.

- Information communication: These errors are classified into: information not communicated, wrong information communicated and information communication incomplete.

The explanations of the behaviors are from [21].

Step 3: Human Error Identification (HEI)

After each task is classified into a behavior in step 2, the analyst consider credible error modes associated with that activity. A credible error is an error that is judged by an expert, from the domain field the procedure is applied on, to be possible. For each credible error, a description of the error mode is given and noted with associated consequences.

Step 4: Consequence Analysis

The next step is a consequence analysis. The consequence of each behavior is considered, as the consequence has implication for the criticality of the error.

Step 5: Recovery Analysis

If there is a later task step at which the error could be recovered, it is entered here. If there is no recovery step, then this section can be skipped.

Step 6: Ordinal Probability Analysis

In this step the behavior is assigned an ordinal probability value, either low, medium or high. The classification of the probabilities is as follows:
- Low (L): the error has never been known to occur.
- Medium (M): the error has occurred in previous occasions.
- High (H): the error occurs frequently

The assigned classification relies upon historical data and/or a subject matter expert.

Step 7: Criticality Analysis

If the consequence is deemed to be critical, then a note is made of this. Criticality is assigned in a binary manner. If the error would lead to a serious incident, then it is labeled as critical, denoted by the symbol: "!". The serious incidents have to be defined clearly before the analysis start.

Step 8: Remedy Analysis

The final step in the process is to propose error reduction strategies. These are presented in the form of suggested changes to the work system which could have prevented the error from occurring, or possibly reduced the consequences. This is done in the form of a structured brainstorming exercise to propose ways of circumventing the error, or

to reduce the effects of the error. The strategies are typically categorized under four headings: Equipment, Training, Procedures and Organization.

As some of the remedies might be costly to implement, they need to be judged with regard to the consequences, criticality, and probability of the error. There are four criteria's to consider when analyzing the remedy:

1. Incident prevention efficiency: to which degree the recommendation would prevent the incident from occurring.
2. Cost effectiveness: the ratio of implementing the recommendations to the cost of the incident * the expected incident frequency.
3. User acceptance: to which degree workers and organization are likely to accept the implementation of the recommendation.
4. Practicability: technical and social feasibility of recommendation.

This evaluation then leads to a rating for each recommendation.

5.3 Example

SHERPA has previoulsy been applied to the task of programming a VCR. The following examples are from [18]. The first thing to be done is an HTA of the task. The example of HTA for programming a VCR is seen in Figure 5.1

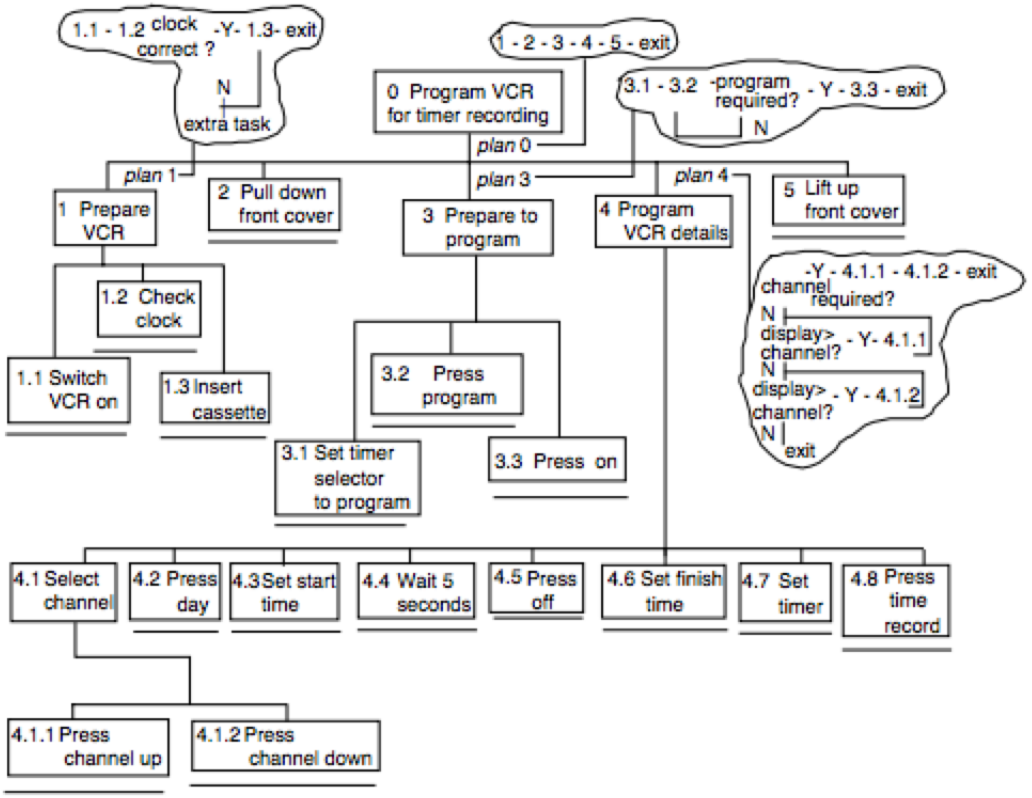

FIGURE 5.1: HTA example

After the HTA is conducted, the rest of the evaluation is performed. Each of the subtasks are evaluated in a SHERPA table. Figure 5.2 show the evaluation of the VCR example.

Task Step	Error Mode	Error Description	Consequence	Recovery	P [a]	C [b]	Remedial Strategy
1.1	A8	Fail to switch VCR on	Cannot proceed	Immediate	L	—	Press of any button to switch VCR on
1.2	C1 C2	Omit to check clock Incomplete check	VCR clock time may be incorrect	None	L	!	Automatic clock setting and adjust via radio transmitter
1.3	A3 A8	Insert cassette wrong way around Fail to insert cassette	Damage to VCR Cannot record	Immediate Task 3	L L	! —	Strengthen mechanism On-screen prompt
2	A8	Fail to pull down front cover	Cannot proceed	Immediate	L	—	Remove cover to programming
3.1	S1	Fail to move timer selector	Cannot proceed	Immediate	L	—	Separate timer selector from programming function
3.2	A8	Fail to press PROGRAM	Cannot proceed	Immediate	L	—	Remove this task step from sequence
3.3	A8	Fail to press ON button	Cannot proceed	Immediate	L	—	Label button START TIME
4.1.1	A8	Fail to press UP button	Wrong channel selected	None	M	!	Enter channel number directly from keypad
4.1.2	A8	Fail to press DOWN button	Wrong channel selected	None	M	!	Enter channel number directly from keypad
4.2	A8	Fail to press DAY button	Wrong day selected	None	M	!	Present day via a calendar
4.3	I1 I2	No time entered Wrong time entered	No program recorded Wrong program recorded	None None	L L	! !	Dial time in via analogue clock Dial time in via analogue clock
4.4	A1	Fail to wait	Start time not set	Task 4.5	L	—	Remove need to wait
4.5	A8	Fail to press OFF button	Cannot set finish time	—	—	—	Label button FINISH TIME
4.6	I1 I2	No time entered Wrong time entered	No program recorded Wrong program recorded	None None	L L	! !	Dial time in via analogue clock Dial time in via analogue clock
4.7	A8	Fail to set timer	No program recorded	None	L	!	Separate timer selector from programming function
4.8	A8	Fail to press TIME RECORD button	No program recorded	None	L	!	Remove this task step from sequence
5	A8	Fail to lift up front cover	Cover left down	Immediate	L	—	Remove cover to programming

FIGURE 5.2: SHERPA example

5.4 Pros and Cons

As all the other HRA models there are both advantages and disadvantages of using SHERPA.

Advantages:

- Structured and comprehensive procedure
- Taxonomy prompts analysts for potential errors
- Suitable for several domains

- No need for an HRA-expert
- Error reduction strategies offered as part of the analysis

Disadvantages:

- Extra work is involved if HTA is not already available
- Some predicted errors and remedies are unlikely or lack credibility, thus posing a false economy
- Different analysts may lead to different results

5.5 Validity

The biggest disadvantage of SHERPA is that it may become unreliable when used by different analysts, due to e.g different experience, education and opinions. Despite this disadvantage, SHERPA received the highest overall ranking of the human-error prediction techniques by expert users [22]. Some validity checking has been done by Baber and Stanton [23]. Predictive validity was tested by comparing the errors identified by expert analysts with those observed during 300 transactions with a ticket machine in the London Underground [23]. They found a validity statistic of 0.8 and a reliability statistic of 0.9 [23]. Another study, made by Stanton and Stevenage, found a validity statistic of 0.74 and reliability statistic of 0.65 in the application of SHERPA by 25 novice users for prediction of errors on a vending machines [21]. Validity statistics concerns to which extent a measure procedure is capabale of measuring what it is supposed to masure, and reliability statistics is estimated based on the consistency of the experiment [24]. Stanton and Young applied SHERPA on eight novice users for prediction of error on a radio-cassette machine, and reported a concurrent validity statistic of 0.2 and a reliability statistic of 0.4 [25]. These results corresponds to the disadvantages from section 5.4, and suggest that reliability and validity are highly dependent upon the expertise of the analyst and the complexity of the device being analyzed [26].

Part III

Research Methods and Research Design

Chapter 6

Research Methods

This chapter provides information about the different research methods used to collect data in this study. Firstly, an overall research methodology will be described, and then the detailed research design will be presented.

6.1 Qualitative and Quantitative Research

Qualitative research concerns studying objects in their natural environment and gathering information by observing. Quantitative research, on the other hand, concerns quantifying a relationship or to compare two or more groups [27], and are often conducted in a controlled environment.

6.1.1 Qualitative Research

Qualitative research has a flexible design, and mostly consist of qualitative data. Qualitative data includes all non-numeric data [28]. These data are e.g. words, images, sounds generated by case studies, action research and ethnography. There are no hard or fast rules on how to analyze qualitative research, which makes is hard to perform. As opposed to quantitative research which can draw upon well established mathematical statistics, qualitative research is dependent of the skill of the researcher to see patterns in the data. The advantage of qualitative research is that the analysis can be rich and descriptive. There is also a possibility of several alternative explanations, as opposed to where there are only one correct answer. A disadvantage of qualitative research is that the volume of qualitative data may feel overwhelming, as this kind of research provide large amounts of information. Another disadvantage is that the findings of the researcher rely heavily on the researcher's opinion and experience.

6.1.2 Quantitative Research

Quantitative research is a type of fixed design, which primarily consist of quantitative data. Quantitative data includes data, or evidence, based on numbers [28]. Quantitative data are generated by experiments and surveys, but can also be generated by other research strategies. The main idea of data analysis is to look for patterns in the data, and draw conclusions based these patterns. The data can be presented in different ways, as simple graphical representation like tables, graphs or charts, or on a next level of complexity with statistical techniques that allow more patterns to be found. The advantages of qualitative research are among other things that the analysis is based on measured quantities, which means that statistical tests can be used and checked by others, and give the same number. This advantage makes the research method scientifically respected. Some people find the use of quantitative data to be the only valid form of research. The disadvantages that needs to be considered are among others the danger of a lot of sophisticated statistical tests shadowing the original purpose of the research. It is important to keep in mind that the analysis can only be as good as the data initially generated.

In this master thesis a combination of qualitative and quantitative research methods will be used.

6.2 Focus Group

Focus groups are one of the many information-gathering methods available. It is a form of group interview that capitalizes on communication between research participants in order to generate data [29]. A group interview is a quick and convenient way to collect data from several people simultaneously, but focus groups use the group interaction as a part of the method. People are encourage to talk with each other, rather than participating in the question and answer routine used in a group interview. The method is particularly useful for exploring peopleÂŧs knowledge and experiences and can be used to examine not only what people think but how they think and why they think that way [29].

According to Kitzinger [29], the idea behind focus groups is that group processes can help people to explore and clarify their views in ways that would be hard to access in regular interviews. Focus groups are especially appropriate when the interviewer has open ended questions and seeks to encourage research participants to explore issues that are important to them, in their own vocabulary, generating their own questions and pursuing their own priorities. If the group dynamics work well, it might lead the research in new and unexpected directions.

Focus groups has many positive qualities, but as other forms of groups there are also some disadvantages. Dynamic groups may silence the individual voices of dissent. The presence of other research participants also compromises the confidentiality of the research session [29].

6.2.1 Context Selection

Focus groups studies are used in several situations and in several manners. It can consist of several groups, everything from a few to fifty, depending on the project and the resources available. Focus groups can also be combined with other data collection techniques.

Most focus group studies use a theoretical sampling mode, where participants are selected to reflect a range of the total study population or to test particular hypotheses [29]. Imaginative sampling is crucial. It is recommended to aim for homogeneity within each group in order take advantage of people's shared experiences. The groups can be naturally occurring, an example may be people that work together, or may be drawn together specifically for the research. Preexisting groups allows observation of fragments of interactions that approximate naturally occurring data, which is data that could have been collected by participant observation. Another advantage is that friends and colleagues can relate to each others comments to incidents in their shared daily lives. They may challenge each other on contradictions between what they profess to believe and how they actually behave.

6.2.2 Planning of Focus Group

It is important to consider the appropriateness of a group for different study populations and to consider how to overcome potential difficulties. There is a safety in that there are several people in the group for those who are wary of an interviewer or is anxious about talking. The environment of the sessions should be relaxed. A comfortable setting, refreshments and sitting in a circle will help to establish the right atmosphere. The ideal group size is four to eight participants. The group is coordinated by a moderator or facilitator, who is often assisted by a co-researcher. The sessions should last one to two hours. If the session requires more time it can extend to an afternoon or a series of meetings. Before the focus group starts it is important that the researcher explains to the participants that the aim of focus groups is to encourage participants to talk to each other rather than to address themselves to the researcher. Kitzinger recommends the researcher to take a back seat at first, allowing for a type of structured eavesdropping. Later in the session, the researcher can adopt a more interventionist style, leading the group to further discussions. Disagreement within the group are likely to occur, and

should be used to encourage participants to elucidate their point of view and to clarify why they think as they do.

An important consideration in the data-collection process is the precise means by which data are recorded. Tape-recording is recommended, since it will leave the moderatorÂťs attention free to focus on the rest of the group. If a tape-recording is not possible, it is vital to take solid notes. Kreuger [30] recommends the facilitator to take written notes even when tape-recording is employed. This will protect against machine failure, and at the same time provide a means whereby observation of the non-verbal interaction takes place. Video has become a popular mean of recording, and could also be used during a focus group. Video recording will also catch the non-verbal interaction, but it may also have undesirable reactive effect.

The analysis of the focus group session is likely to follow the same process as for other sources of qualitative data [29]. When analyzing the data collected there are some issues to consider. One of these is that some of the participants in the group may be more articulate or assertive than other's, leading to some data being artificially suppressed. Members of the group with less self-confidence or are less articulated may be inhibited from expressing alternative viewpoints. The problem arises with the question of silence: do silence indicate agreement of represent an unwillingness to dissent? Skillful questioning by the moderator may assist in distinguishing these two possibilities [31]. If more than one focus group are conducted, then the combined result from each focus group will increase the reliability of the data.

6.3 Experiment

Experiments are used when we want control over the situation and want to manipulate behavior directly, precisely and systematically [27]. There are several advantages of experiments, one of them is the control of subjects, objects and instrumentation which help us to draw general conclusions. Another advantage is the ability to perform statistical analysis using hypothesis testing methods and opportunities for replication. When conducting a formal experience, we want to study the outcome when we vary some of the input variables to a process. According to Wohlin there are two kinds of variables in an experiment: independent variables and dependent variables [27]. Dependent variables are those we call response variables, and are the variables we want to study to see the effect of changes after the experiment is conducted. All variables in a process that are manipulated and controlled are independent variables.

Wohlin states that experiments are appropriate to investigate several aspects. These aspects includes:

- Confirm theories, to test existing theories

- Confirm conventional wisdom, to test peoples conceptions
- Explore relationships, to test that a certain relationship holds
- Evaluate the accuracy of models,
- Validate measures, to ensure that a measure actually measures what it is supposed to do.

The starting point of an experiment is insight, and the idea that an experiment is a possible way to evaluate what we are interested in. The experiment process can be divided into five main activities [27]. Scoping is the first activity, during this step the experiment is scoped in terms of problem, objective and goals. The next step is planning, where the design of the experiment is determined, the instrumentation is considered, and threats to the experiment is evaluated. Next is the experiment operation. In this activity, measurements are collected, before they are evaluated and analyzed in the analysis and interpretation activity, which is the next step. The last step is presentation and package where the results are presented.

6.3.1 Planning the Experiment

Planning refers to how the experiment is conducted. The experiments must be well planned, and plans need to be followed in order to control the experiment. In the planning phase the context of the experiment is determined in detail, which includes personnel and environment. The hypotheses are stated, including null hypotheses and alternative hypotheses. The planning of the experiment will be reflected in the result, poor planning may lead to bad results. The planning phase may be divided in seven steps [27]. First comes the context selection. In this step we select the environment in which the exercise takes place. Next is the hypothesis formulation and then the variable selection of independent and dependent variables take place. The selection of subjects is decided as a next step before the experiment design type is chosen. Instrumentation prepares for the practical implementation of the experiment. The final step is the validity evaluation which aims at checking the validity of the experiment.

6.3.1.1 Context Selection

When performing the context selection, it is always best to execute the experiment in large, real software projects with professionals. However, this is not always possible when research are at an early stage. Conducting experiments involves risks, which may make a project delayed or in some way make the project less successful.

An experiment can be characterized according to four dimensions [27]:

- Off-line vs. on-line
- Student vs. professional
- Toy vs. real problems
- Specific vs. general

Off-line experiments are conducted in controlled environments, with full control over the participants. In some cases the experiments may be unrealistic, as off-line experiments are conducted with pen and paper. On-line experiments, on the other hand, uses real computer tools on computer problems. This allows us to register data directly and will lead to a simpler analysis. A disadvantage with on-line experiments are that there are less control over the participants.

When using students in an experiment you are likely to get a large number of participants, which will give good statistical significance. However, students are not professionals yet, and may behave different than what professional would have done, which again makes the results difficult to generalize. When using professionals the results are realistic, and the result easy are to generalize. Unfortunately it is difficult to get a significant number of professionals to participate, which will give low statistical significance.

When selecting problems for the experiment there is the choice of whether to choose toy problems or real problems. Toy problems can be done in a short time, and the results are easy to analyze. The problems often get too simple and there is a risk of them being unrealistic. Real problems are realistic and will give a relevant result. It will generate a lot of data, but at the same time the results may be difficult to analyze. The real problems need long time to finish, compared to toy problems.

Specific experiments are easy to define, but the resukts are difficult to generalize. The data collected in these kinds of experiment are easy to define and analyze. In general experiments, the experiment is hard to define, but then also easy to generalize. In general experiments it may be hard to define relevant data.

The choice of which of the dimensions are selected depends on several factors, such as available resources (e.g. money, personnel, time), the need for generalization and the consequences of making wrong decisions.

6.4 Questionnaire

A questionnaire is a research tool that uses questions to gather information from multiple respondents [32]. It is a type of survey meant to allow a statistical analysis of the responses. Oats [28] defines questionnaires as a pre-defined set of questions, assembled

in a pre-determined order. Questionnaires are often associated with the survey research strategy, but are also used in other research strategies.

The questions in a questionnaire can be open-ended where the respondents are able to formulate his or her answer, or close-ended where a number of options are given for the respondent to choose. There are advantages and disadvantages for both kinds of questions. The open-ended questions give more information, but take longer to process. The close-ended questions are much easier to respond to. According to Ringdal, questionnaires has a high degree of standardization [33], especially in close ended questionnaires. The purpose of a high degree of standardization is to eliminate accidental measurement errors and give reliable data. Questionnaires consist almost entirely of close-ended questions. When constructing questionnaires it is important to use clear and precise words, correct grammar and correct punctuation [32]. Questionnaires are one of the data collection techniques used in surveys. The questionnaires can be provided in both paper form and in an electronic form.

Chapter 7

Validity of Research Methods

When analysing qualitative and quantitative data from information gathered in this thesis it is important to asses the validity of the data. Adequate validity refers to that the results should be valid for the population of interest [27]. This means that the results has validity for the population we would like to generalize the result of. Wohlin presents four types of threats to validity, identified by Cook and Campbell [34]. The characteristics are conclusion, internal, external and construct validity. The best research designs are those that can assure high levels of internal and external validity [35]. Figure 7.1 shows the different types of validities' relationship to eachother.

7.1 Conclusion Validity

Conclusion validity depends on the quality of the data. It is sometimes referred to as statistical validity, as it is desirable to ensure a statistical relationship. The threats that are associated with conclusion validity are issues that affect the ability to indentify statistical relationships in an experiment. These issues include choice of statistical tests, choice of sample size and so on.

7.2 Internal Validity

An experiment has good internal validity if the measurements obtained are indeed due to manipulation of the independent variable, and not to other factors [34]. Threats to internal validity concerns issues that may indicate a causal relationship, even though there are none [27]. Threats to internal validity are among others differences between the experiment and control group, history; events that are not noticed, interferes between pre-test and post-tests observations, badly designed instrumentation and so on. All

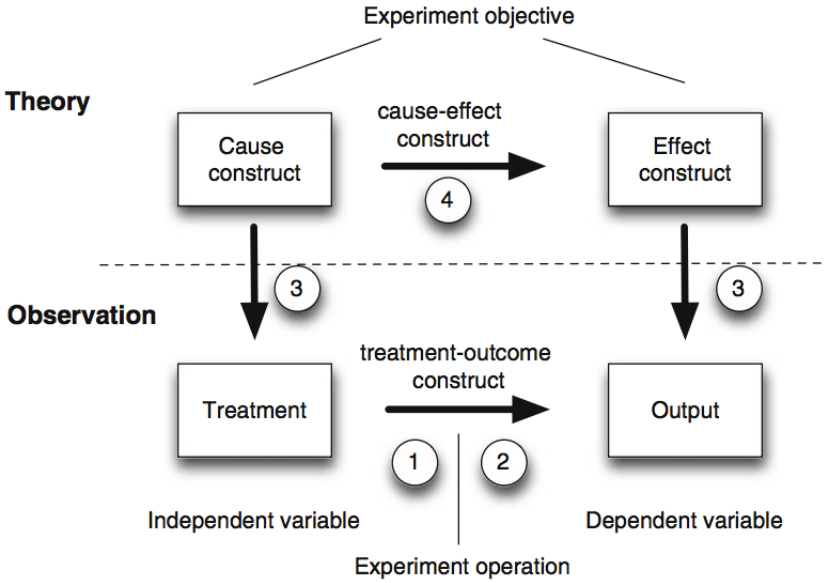

FIGURE 7.1: Validity

these factors, and more can make the experiment show results that are not due to what was tested in the experiment, but due to other disturbing factors.

7.3 Construct Validity

Construct validity concerns the measurements of the experiment. The measurement needs to be a good measurement for the situation in the experiment. Construct validity concerns generalizing the result of the experiment to the theory or concept behind the experiment. Threats to construct validity may be interaction between treatments in the experiment, or between treatment and testing, fishing for expectations or makeing your own expectation become too visible in the experiment.

7.4 External Validity

An experiment has good external validity if the results are not unique to a particular set of circumstances, but are also generalizable in other occasions. Experiments seek high external validity, and the best way to demonstrate generalizability is to repeat the experiments many times in many different situations. External validity is affected by the experiment design chosen, but also objects and subjects in the experiment. Threats to validity are among others too few participants, non-representative participants and non-representative test cases. The threats to external validity are reduced by making the experimental environment as realistic as possible.

Chapter 8

Research Design

This chapter will describe in detail how the research is performed.

8.1 Focus Group

The focus group will in this study be performed in conjunction with the experiment later in the project. It will cover the first part of SHERPA, which is the hierarchical task analysis, HTA. As the experiment will consist of participants from third year of a five-year degree, these participants may not have much experience through summer interns and other IT related work. HTA is only the first step of the SHERPA analysis, and needs to be conducted before the rest of the analysis. The focus group will consist of fellow students from the fifth grade with more experience than the participants in the experiment.

The group will be a naturally occurring group as the participants have studied together for a long time and know each other well. They are all interested in programming and has gained some experience through the years at the university in courses and projects as well as summer internships.

The purpose of this focus group is to do an HTA on five basic programming tasks. These tasks were presented in Schulmeyers article about NNPPs, which addresses the need for a programming behavior model. The basic tasks are:

1. Composition
2. Comprehension
3. Debugging
4. Modification

5. Learning

In the focus group the participants will discuss how they perform these tasks. In the project specialization paper, written during the previous semester, we localized some issues that needs to be considered. To tailor SHERPA to software development there are parts that need to be adjusted or removed. In the original format of SHERPA there are five behaviors that every operation are classified into. These are action, retrieval, checking, selection and information communication. The error mode does not match errors made in software development, this issue and other possible error modes are to be discussed in the focus group.

The focus group will also discuss what errors and how they occur when they work.

8.2 Experiment

The experiment performed in this master thesis will be an off-line experiment as the research is at an early stage with lot of uncertainties. With an off-line experiment we are able to control the environment and the variables better. The test subjects will be students, as students are cheaper and easier to "get" than professionals. There will be little risk when using students, and at the same time we will not have problems with getting a sizable group of students and schedule the experiment. The problems the participants will solve, are problems discussed in the focus group with more experienced students. The problems are realistic programming situations, but as the participants may have their own programming procedure, and the fact that none of the problems are complete, will to a certain degree make it a toy experiment.

8.2.1 Selection of Subjects

The test subjects for the experiment will be selected based on a convenience sampling. The test subjects will be a group from 4th semester computer science students, and a group from 6th semester or above from informatics at NTNU. The students will be compensated with 200 NOK each towards their class excursion.

The experience of the students may not be substantial, but it is important to keep in mind that they might have gained experienced in other places than the university and gained more experience than what is expected. As the students are attending different semesters, it will be interesting to see if there are differences in the results depending on which semester they are attending. We believe that when we choose simple programming tasks, the participants will be able to do the analysis without too much problems.

8.2.2 Location and Equipment

The experiment will take place in an auditorium at the university. The auditorium has 189 seats, which will make it possible to evenly distribute the participants throughout the room to prevent them from influencing each other when performing the experiment. The auditorium has a projector, which allows for a Powerpoint presentation to show certain parts of the experiment that is useful to help the participants along. The experiment demands little equipment, and will be conducted using pen and paper. The experiment material will be placed before the participants arrive, which will guarantee a distributed seating of the participants.

8.2.3 Experiment Design

The purpose of this experiment is to test the HRA method, SHERPA. In this experiment the participants are analyzing a set of subtasks predefined from the focus group conducted previously, see chapter 10. The participants attended an hour-long experiment. During the experiment, data was collected through questionnaires and the SHERPA table.

The experiment will be accompanied by two questionnaires, a pre-questionnaire and a post questionnaire.

8.2.3.1 Pre-Experiment Questionnaire

The purpose of the pre-questionnaire is to get information regarding the experience of the participants, and to gain general information about the participant. The result of the experiment is to some degree dependent of the experience of the person participating in the experiment. These data will be useful when analyzing the results.

8.2.3.2 SHERPA Table

The experiment starts after the pre-questionnaire is completed. The students starts by reading a step-by-step guide on how to fill the SHERPA table, and how to perform the analysis. The tasks that are to be analyzed are predefined and entered in the table. There are a total of 11 tasks to be analyzed during the experiment. The students decide for them selves whether the situation in the task is error prone or not.

8.2.3.3 Post-Experiment Questionnaire

The post-questionnaire regards the participants' perception of the method being tested. The questionnaire concerns the issues of how easy it was to conduct, and also how useful the participants found it to be.

Part IV

Research Procedure and Results: Focus Group

Chapter 9

Hierarchical Task Analysis

This chapter is an introduction to the main task to be done in the focus group. It contains detailed information about Hierarchical Task Analysis, and how it is conducted.

Hierarchical Task Analysis, HTA, is a core ergonomics approach with a pedigree of over 30 years continuous use [36]. The first paper written about HTA, Task Analysis (Department of Employment Training Information Paper No. 6), was published 1971 and was authored by Annett [37]. In this paper it was made clear that the methodology is based upon a theory of human performance. The theory is based on goal-directed behavior comprising a sub-goal hierarchy linked by plans. Originally, HTA had only three governing principles. The first is that at the highest level we choose to consider a task as consisting of an operation, and the operation is defined in terms of goals. Secondly, the operations can be broken down into sub-operations, each defined by a sub-goal. Third is the hierarchical relationship between operations and sub-operations [37]. Ergonomists are still developing new ways of using HTA which has assured the continued use of the approach for the foreseeable future [36]. According to Kirwan and Ainsworth, HTA is considered the best known task analysis technique [38].

The number of guidelines for conducting HTA are surprisingly few [36]. The methodology is based on a few broad principles, rather than a rigidly prescribed technique. According to Stanton there are 9 basic heuristics for conducting an HTA:

1. Define the purpose of the analysis
2. Define the boundaries of the system description
3. Try to access a variety of sources of information about the system to be analyzed
4. Describe the system goals and sub-goals
5. Try to keep the number of immediate sub-goals under any super-ordinate goal to a small number

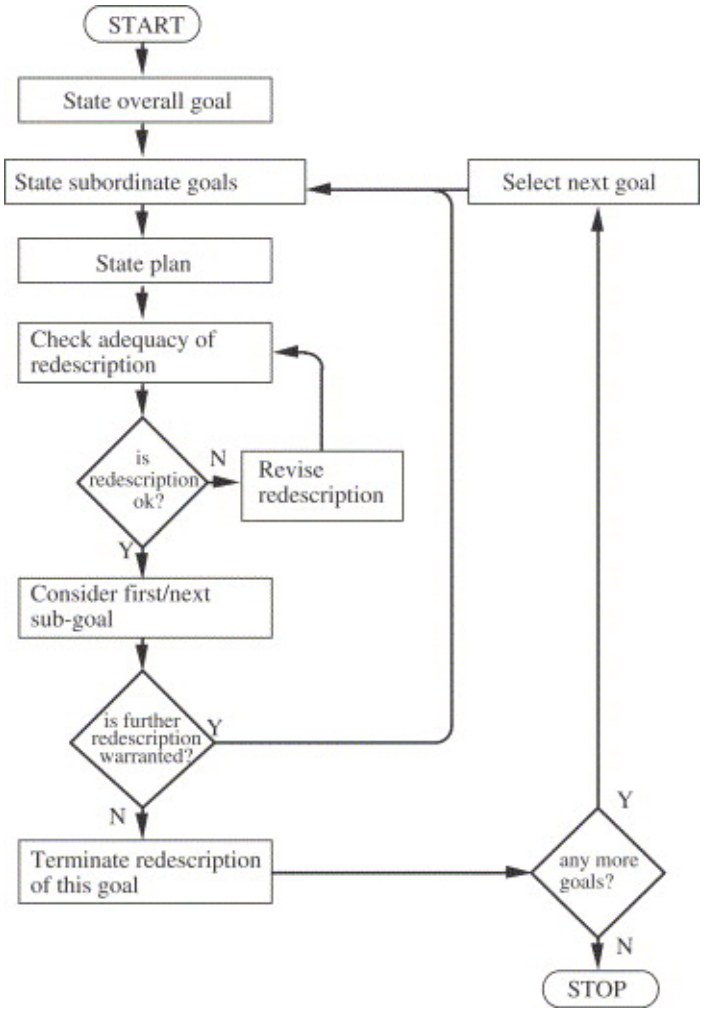

FIGURE 9.1: Procedure of breaking down the sub-goal hierarchy

6. Link goals to sub-goals and describe the conditions under which sub-goals are triggered
7. Stop redescribing the sub-goals when you jugde the analysis is fit for purpose
8. Try to verify the analysis with subject-matter experts
9. Be prepared to revise the analysis

Figure 9.1 presents a procedure of the steps above. The procedure only describes the steps 4-8, but offers a useful heuristic for breaking the tasks down into a sub-goal hierarchy. The notation of HTA may be presented in three ways, hierarchical diagrams, hierarchical lists and in a tabular format. Each of the notations has their own advantages and it is up to the analyst to choose between the three. The hierarchical diagrams make it easy to trace the genealogy of sub-goals for small scale analyses, but with larger scale analysis it can become cumbersome and unwieldy. [36] For these types of analysis a hierarchical list approach might be more useful. The hierarchical diagram and the hierarchical list presents the same information, but in different forms. The advantage of

the diagram is that it represents the group of sub-goals in a spatial manner, which gives a quick and straightforward overview of the HTA. The list presents the information in a more condensed format, which is useful in a large analysis. The tabular format provides more details, it is not a complete analysis but it provides notes on how different incidents were handled.

Chapter 10

Results From Focus Group

This chapter presents the procedure and the resulst and findings from the focus group session. The focus group was conducted with seven participants in addition to the facilitator, Friday 7th of March. It was located at the university, and lasted for 2 hours.

10.1 Procedure

This section describes how the focus group was conducted. A timetable for the session with estimated use om time is shown in table 10.1.

Activity	**Estimated time use**
Introduction	5 Minutes
Example of HTA and questions	10 Minutes
Debugging	15 Minutes
Composition: writing a program	15 Minutes
Comprehension: understaning a given problem	15 Minutes
Modification	15 Minutes
Learning	15 Minutes
Discussion of Software errors	25 Minutes
Total time	**120**

Table 10.1: Timetable for Focus Group Session

First, the purpose of the focus group was stated and explained. The participants would ask for and get the information they wanted. We started with HTA and an explanation of what it is. We discussed the matter further and studied some examples of HTA on one of the tasks that was going to be conducted.

FIGURE 10.1: Focus Group Session

We started with the task of debugging, and continued with the other tasks. The group started by discussing some of the main tasks in how to solve the problem. As most of them had never conducted an HTA before they found it a bit difficult to structure the tasks in a hierarchical structure. The main problem was that they felt that it was hard to draw an iterative process and keep it hierarchical at the same time. As developers they are more familiar with drawing state charts than drawing hierarchical tasks. The drawings ended up with being a merge of a state chart and hierarchical task analysis. The idea of this task was to get several views and experience of how the tasks are executed by the general developer. We collected a lot of interesting ideas, which will be helpful for further work before the experiment.

10.2 Findings

In this section, the findings of the focus group is presented.

10.2.1 HTA

During the discussions of the HTA it became clear that all the participants felt that there will be different ways of solving problems according to what kind of technology they used. At the task of debugging they all commented that the type of framework had a great impact on how they were to work, and that there are no universal way of debugging. Developers probably have their own process and habits when it comes to debugging. But there is a "main path" most of them follow. This statement points out what has been mentioned earlier, that when analyzing a situation that may seem similar to all, like debugging, it needs to be analyzed in the certain settings in order for all details to be correct. Different companies need to conduct their own analyze to deal with the problems in their own development team.

HTA is the first step of SHERPA, and the task of HTA is used to recognize all subtasks in different situations so that the analyst is able to perform the analysis on all the subtasks covering the entire situation. There are several ways of drawing HTAs, and it is possible to draw these as a merge as was done in the focus group. The important thing is to keep the subgoal and goals according to the rules of HTA. However, if there are other ways to find subgoals than drawing the (for developers) unfamiliar HTA, this may also be performed. If there are other ways that may give as much information as HTA does about subgoals, this method could also be performed as step 1 in SHERPA. This means that it is possible to use the form of analysis that is most suitable for the person performing it. However, the analysis should meet all the needs of SHERPA and should lead to the same tasks as HTA does.

10.2.2 Errors in Software Development

After the HTA was conducted, a discussion on error they usually commit, or has experienced from previous projects was discussed. Poor motivation was one of the identified problem areas. When things first start to go bad it is hard to keep motivation at a productive level. Motivation fails mostly because something else in the project or program already has failed. The setup of the programming environment is a major contribution to errors. The participants had all experienced problems occurring from this phase of software development. The errors committed during this phase may be serious and create severe problems throughout the project. An example some of the participants had experienced was that developers had imported wrong version of libraries to the project. This particular error may cause security issues and deprecated functionality.

There are differences in the severity of the errors that are committed during development. The severity will affect the time needed for correction. Small bugs will be corrected with little effort, while the big errors takes more time. Even though the amount of small errors is bigger than the amount of more severe errors, the severe errors are more time

consuming. As an example, the participants in the focus group had experienced that if there were something missing from planning, which made it inadequate, it would lead to the architecture being insufficient. These types of error are one of the most severe errors and demand a lot of time and refactoring.

The group agreed that the process they use in the project they are working on is important. Different processes serves different types of project, and choosing a wrong process might lead to low efficiency during development.

We also discussed pair-programming and pair-debugging. The participants felt that pair-programming might slow down the development somewhat, but may be useful when problems occur that are hard to solve alone. However, they were excited about pair-debugging. Pair-debugging helped them a lot and made both large and small debugging problems easier to perform. It is a bit time consuming, but it will probably take shorter time than when one developer is stuck with a problem alone.

We discussed how they solved problems that occurred in their daily programming work. They had different ways to solve their problems; quite often they asked each other or other developers for help to sort out their problem. The group also introduced a term called "rubber ducking". When you have a problem you are not able to solve, you put a rubber duck in front of you on your desk and explains the problem thoroughly to the duck. Hopefully, when you explain the problem you will understand it better and might be able to solve it yourself. For the participants it was important to always try to solve the problem themselves before they consult other developers. It is okay to ask other for help, but it is not appreciated if the person asking has not tried to solve it themselves first.

10.2.3 Error Modes

We discussed possible error modes in software development, and what needs to be covered in a behavioral model for developers. Some of the error modes in SHERPA might need some changes at the same time as new ones are presented.

One of the modes that are covered by the error mode action in SHERPA is timing. Timing is important in software development, as there are always time limits, and the fact that a lot of projects often exceed their time limits. There are several timing problems, some of them concerns the entire project, but there are also timing issues where a part of the problem takes more time than what was accounted for. Both of these should be covered by the error modes either together or as separate.

Software development is knowledge work. A lot of work done in plants can be characterized as different forms of actions, while most of the work done in software development involves a lot of thinking. We either use the knowledge we already have, or acquire new

knowledge to solve tasks. One problem that may arise, and often does, is that the developer does not have sufficient knowledge of the specific domain he is currently working in. Insufficient knowledge might lead to poor decisions in design and implementation.

Another issue that arises, especially in teamwork, is that developers work in different ways and in different pace. If there are some part of a system that is dependent on other parts, there may be delays and one of the developers may have to wait for others, which causes valuable time to be wasted.

Selection of improper technology was identified as another error mode. This occurs in several occasions like when choosing an improper framework or if a functional programming language is used when an object oriented language would be more appropriate. In the original version of SHERPA there is a category of error mode called selection, with sub-error modes: selection omitted and wrong selection made.

As in other knowledge sectors, the continuation of information is also important in software development. Developers obtain knowledge by asking developers with more experience in a certain domain or acquire knowledge by searching for information. As knowledge is an important part of programming, there should be an error mode that also includes the handling of knowledge.

Part V

Research Procedure and Results: Experiment

Chapter 11

Adjustments made in SHERPA

To make SHERPA applicable to software development there are some changes that need to be made. In this chapter the adjustments are presented, with the reasons why these changes were made. The changes are made on the basis of the results from the specialization project and the focus group conducted before the experiment.

11.1 Error Mode

There are five basic error modes in SHERPA; Action, Information Retrieval, Checking, Selection and Information Communication. Most of the error modes are also suitable for software development. However, the error mode "Action" is not that relevant since software development is considered knowledge work and not operational work. Hence there are four error modes left for task classification.

Three new categories of error modes are suggested, as well as one error mode added to one of the existing categories, and will be tested during the experiment.

11.1.1 Time

Timing was identified during the focus group as a needed category of error mode. Time is important in software development, relative to scheduling of projects, how much time is needed to perform a task and remain within time estimates. Another aspect of time that was identified during the focus group was that developers work at different pace. In some cases, like in teamwork, situations may arise where developers need to wait for each other before they are able to move on. Four error modes in the category Time are added, and these are:

T1 Underestimated schedule:
this concerns the estimation of the entire project, e.g project is not able to meet its deadline

T2 Underestimated workload:
this concerns time assigned to a task within the project, e.g not able to finish mockup before presentation

T3 Overestimated workload:
this concerns estimation of a task, e.g it took less time to make the functionality than accounted for

T4 Unbalanced workload:
this concerns those times when there is a delay in the project leading to one developer need to wait before he/she can start to work on their part

11.1.2 Knowledge

Knowledge is an important part of software development. When developers solve a problem, they use the knowledge they have, or acquire new knowledge. One problem that often arise is that developers do not possess enough knowledge about the specific domain they are currently working in. At the coding level, lack of knowledge in the programming language could result in an unduly complex program [39]. Other situations that may occur is that the developers overrate their own knowledge. Two error modes in the knowledge category are added:

K1 Insufficient Knowledge:
concerns the occasions where the developers do not possess the necessary knowledge to solve the problem at hand.

K2 Overrated knowledge/Arrogance:
concerns the situations where the developer assume that his/her way is the best way to do it, while not investigating further.

11.1.3 Technical Error

In situations like setup of the development environment a lot of errors occur. The errors that occur in these situations cannot yet be covered by SHERPA. Two error modes are thus added to this category, namely:

E1 Wrong configuration:
concerns the configuration of software wrongfully, like adding an outdated library to the project

E2 Version control:
concerns the problems that arises when different versions of the project is used by developers, like when developers are debugging different versions of the system

11.1.4 Selection

The Selection category already consist of two error modes. During software development a lot of choices are made, and some of these are directly related to a selection of technology. To the category Selection, we add:

S3 Wrong technology selected:
concserns the situation of a selection of a technology, like when choosing an unsuitable language for the application

11.2 SHERPA Process

SHERPA consist of eight steps. In this experiment the participants will not conduct a complete SHERPA analysis. Certain simplifications needs to be made to make the experiment feasible. In this section information about changes in each step is provided.

Step 1: HTA
The first step of SHERPA is, as stated in chapter 5, hierarchical task analysis. In this experiment a simplified analysis will be tested. The subtasks that are normally identified during this step of SHERPA are predefined and already added to the SHERPA table prior to the experiment. These subtasks was identified during the focus group session, see the results in section 11.3.

Step 2: Task classification
The second step of SHERPA is task classification. In this step the subtasks defined during HTA is classified into one of error categories, also called behaviors. During the pilot testing of the experiment, it was suggested to remove this part from the experiment. The feedback given was that when the pilot testers had classified the subtasks into one of the categories, they got confused and ended up writing the error modes description from the error mode table into the error description in the SHERPA table, instead of writing the error they identified in the specific task. The testers found it strange that they were to classify the subtask to an error category before they had filled in the error in the SHERPA table. Given this feedback, and the fact that this step does not contribute anything to the SHERPA table, this step was removed during the experiment.

Step 3: Human Error Identification
This step was completed as normal without any adjustments. The only change is that it worked as step 1 during this experiment.

Step 4: Consequence Analysis
No changes are made to the consequence analysis.

Step 5: Recovery Analysis
In the original description of SHERPA there is a rule saying that it is not possible to select a previous step as a recovery step to fix a mistake that has been made. This step is considered as a particularly useful aspect of the SHERPA approach because of the determination of whether errors can be recovered immediately, at a later stage in the task, or not at all [3]. However, in software development the opearations are executed in an iterative process, where several operations are repeated until they reach a final state assumed to be correct. The subtasks analyzed during this experiment is not a complete process, but only selected parts of the process. Because of this we are not able to perform this step as it is intentionally described in SHERPA. In this experiment the participants are asked to write a recovery to their problem, without any rules.

Step 6: Ordinal Probability Analysis
No changes are made to the probability analysis.

Step 7: Criticality Analysis
In the original form of SHERPA, the criticality of each task is noted with the symbol:! indicating that the error identified is critical. In this experiment the criticality analysis will be noted as in the probability analysis with low, medium or high. The classification of consequences is as follows:

Low (L): little to no consequence

Medium (M) medium consequence

High (H) the consequence is severe

Step 8 Remedied Strategy
No changes are made to the remedied strategy analysis.

11.3 Experiment

In the original layout of SHERPA, the SHERPA table looked like in table 11.3.

Task step	Error Mode	Error Description	Consequence	Recovery	P	C	Remedied Strategy
2. Identify Problems	N/A	N/A	N/A	N/A	N/A	N/A	N/A

In this experiment, after doing prototype testing, error mode and error description changes positioning in the table has, see table 11.3. With this change, the columns in the SHERPA table follow the steps, provided in the guideline in the experiment, to the letter. In Appendix A a full version of the experiment is provided.

Task step	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2. Identify Problems	N/A	N/A	N/A	N/A	N/A	N/A	N/A

With these adjustments SHERPA was ready for the experiment. From the focus groups drawings of the HTAs, and their opinions on what part of software development were error prone, the following subtasks were analyzed during the experiment:

1. Choose programming language suited for your application
2. Set up development environment
3. Choose architectural pattern(e.g. MVC, observer)
4. Identify problems/uncertainties in requirements
5. Define goals from the requirements
6. Develop mockup/prototype of solution (to show to the customer)
7. Review codes behaviour: place breakpoints
8. Review codes behaviour: evaluate behaviour
9. Modification: identify new necessary functionality

10. Modification: draw connection between old code and new functionality
11. Create new functionality: code the changes

The new collection of error modes to be used in the experiment is provided in Table 11.1 In Appendix A a full version of the experiment is provided.

Error Mode	Error Description
Time	
T1	Underestimated schedule
T2	Underestimated workload
T3	Overestimated workload
T4	Unbalanced workload
Knowledge	
K1	Insufficient knowledge
K2	Overrated knowledge/arrogance
Technical Error	
E1	Wrong configuration
E2	Version control
Information Retrieval	
R1	Information not obtained
R2	Wrong information obtained
R3	Information retrieval incomplete
Checking	
C1	Check omitted
C2	Check incomplete
C3	Right check on wrong object
C4	Wrong check on right object
C5	Check mistimed
C6	Wrong check on wrong object
Information Communication	
I1	Information not communicated
I2	Wrong information communicated
I3	Information communication incomplete
Selection	
S1	Selection omitted
S2	Wrong selection made
S3	Wrong technology selected

Table 11.1: Error Mode

Chapter 12

Procedure

The experiment was conducted on Thursday 27th of March at 14.00. In the days before the experiment a total of four pilot tests was conducted. The purpose of the pilot tests was to make sure the tasks to be performed in the experiment was understandable, and that the amount of work was sufficient but not excessive. The pilot testers were fellow students, with far more general experience than the test participants. However, the experience of the pilot testers was in different fields and it was useful to get their opinion

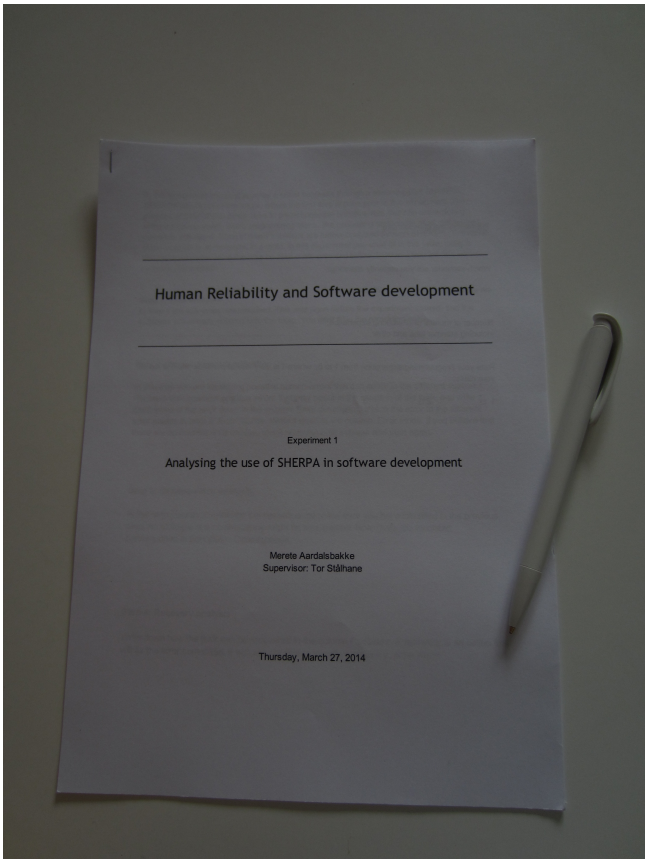

FIGURE 12.1: Experiment

on how they perceived the experiment. More important, none of the pilot testers had any experience with HRA, and knew little or nothing about SHERPA. Their feedback was valuable in helping to improve the standard of the experiment. A short but detailed step-by-step guide was provided in the experiment. During the pilot tests, several approaches to the experiment were used to test the step-by-step guide to SHERPA. One approach was that the testers themselves read through the description, and only asked questions after reading. The other approach was to give a short presentation before the test participants started. The first approach made sure that the guide was solid, and the second approach was a method that was approximately equal to the real experiment, which provided a good time estimate. A timetable estimated and planned for the experiment is provided in the table below.

Activity	**Estimated time use**
Introduction	8 Minutes
Pre-Questionnaire	2 Minutes
Experiment	45 Minutes
Post-Questionnaire	5 Minutes
Total time	**60**

TABLE 12.1: Timetable for Experiment

FIGURE 12.2: Participants conducting the experiment

A total of 41 students participated in the experiment. The experiment material were distributed evenly in the auditorium before the participants arrived. There was only one facilitator present, which had all the acting roles needed in an experiment. The session started with the facilitator presenting SHERPA. Each of the steps in SHERPA was explained and demonstrated by an example, also provided in the experiment. The

facilitator answered questions from the participants. There were a few question asked in this part of the experiment. The questions were mostly about how to fill in the form provided in the experiment paper.

After all questions were answered, the participants started filling in the pre-questionnaire about general information of their experience. After this they went straight on to the SHERPA-table and started to analyze the tasks. There were only two questions asked through the experiment. Both these questions regarded the task: Review Code: place breakpoints. Both the participants were uncertain of what breakpoints are.

The experiment went on without any major issues arising, and the participants were reminded to fill in the post-questionnaire towards the end of the time. The first participant was finished approximately twenty minutes before the time was up. A few followed, but most of the participants delivered the experiments paper after approximately one hour. The facilitator asked all participants to move on to fill in the post-questionnaire when five minutes of the estimated time remained.

Chapter 13

Results and Findings

In this chapter the results and findings from the experiment are presented. All of the participants filled in the post-experiment questionnaire, but two participants forgot to fill in the pre-experiment questionnaire. In section 13.2, the focus of the results will be on three of the columns from the SHERPA table, which is Error description, Consequence and Remedied Strategy. The Recovery analysis will not be presented in these results because of the simplifications done in chapter 11. There are uncertainties associated with this sub-analysis, and in this experiment the three other sub-analysis are more interesting.

As there were a lot of data collected through the experiment, the results will show examples from the raw data that are most representative for the results. All data from the experiment are provided in Appendix B.

13.1 Pre-Experiment Questionnaire

The primary focus of the pre-experiment questionnaire was to find general information about the participants and info on previous experience.

The participants were asked which semester they were currently attending at the university, see Figure 13.1. The majority (83 %) answered 4th semester. Two participants forgot to fill in the pre-experiment questionnaire. However, we knew that there were students from two different groups, and in one of the group there were seven participants that we in advance knew attended 6th semester or above. All of these seven filled in their form, thus the two unknown students currently attends 4th semester.

Further, we asked for IT-related experience. 56% of the participants answered that they had no experience from the IT-industry, 19 % had some experience, from one to five months, and a total of 25% answered that they had more than five months experience, see Figure 13.2. It was interesting to see that the few who had more than five months

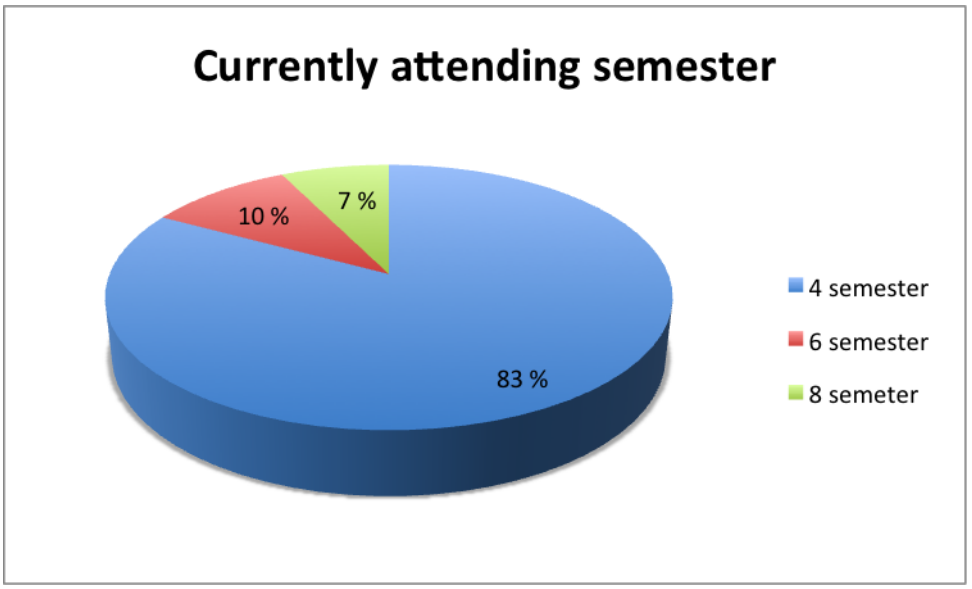

FIGURE 13.1: Currently attended semester

experience really had a lot of experience, one noted down 72 months. There was one participant who did not note his/her experience, in addition to the two who did not fill in anything in this questionnaire.

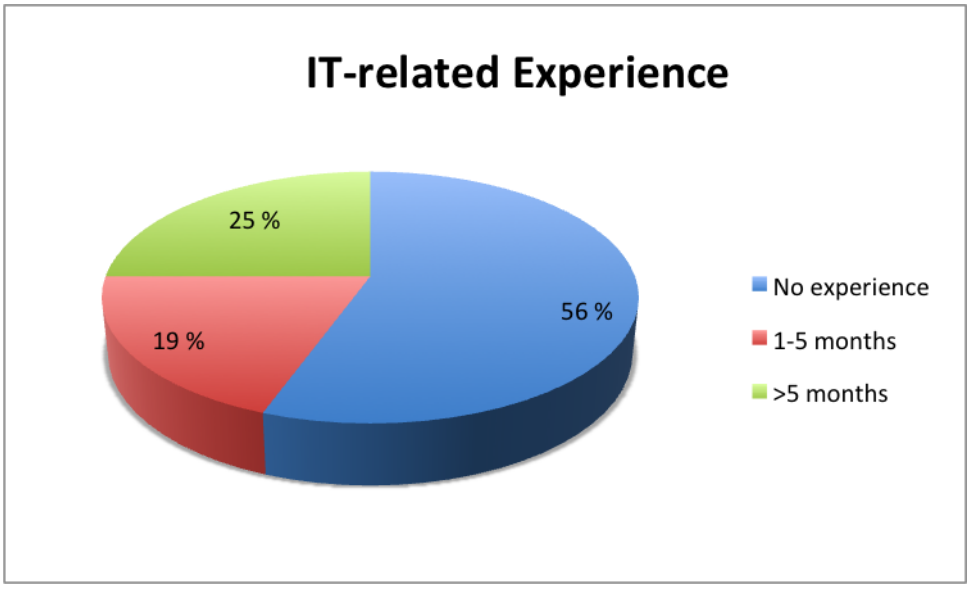

FIGURE 13.2: Nr of months with IT-related experience

The last question in the pre-experiment questionnaire was for the participants to rate their programming experience. They were asked to rate it on a scale between one and five, where one is very little experience, and five is top of your class. The results are presented in Figure 13.3. Most of the participants rated themselves in the middle at the third, or fourth level. There was one participant who rated his- or her-self as top of the class, and no one rate themselves as very little experience. Three students rated

themselves as two in programming experience. SHERPA is highly dependent on the expertise of the analysts, and due to these participants low expertise, these responses are disregarded further in this experiment.

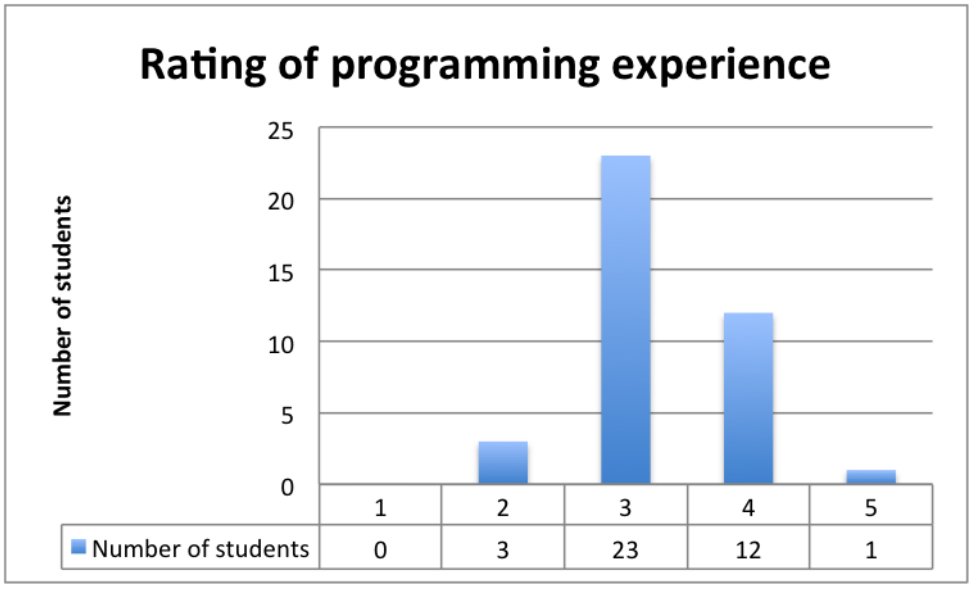

FIGURE 13.3: Rating of programming experience

13.2 Experiment

The experiment consisted of 11 subtasks, as stated in chapter 11. Figure 13.4 shows an overview of how many times each error mode was selected through all tasks in this experiment. The error mode selected in a task is strongly dependent on the person performing the analysis, which means that there are not one correct answer. However, some error modes are more suited for some tasks and errors than others. Figure 13.5 shows types of errors exposed in software development. Figure 13.4 shows that the error mode K1 is definitely the most used error mode. The error description of K1 is *Insufficient knowledge*. From this graph it seems as a lot of problems that occur during software development is related to insufficient knowledge. K1, *Insufficient knowledge*, was one of the error modes that was added to the list of error modes before the experiment.

The next most used error mode is E1, with the error description *wrong configuration*. The high number of E1 corresponds to what the focus group identified as an error prone part of development. Other popular error modes was T2 *underestimated workload*, S3 *wrong technology selected*, S2 *wrong selection made* and R2 *wrong information obtained* was all used more 25 times.

In Figure 13.5 the error modes within each category were added. In this graph we see how much the total category was used throughout the experiment. Knowledge stands out, and was used twice as much as the next most used error mode Category. In the

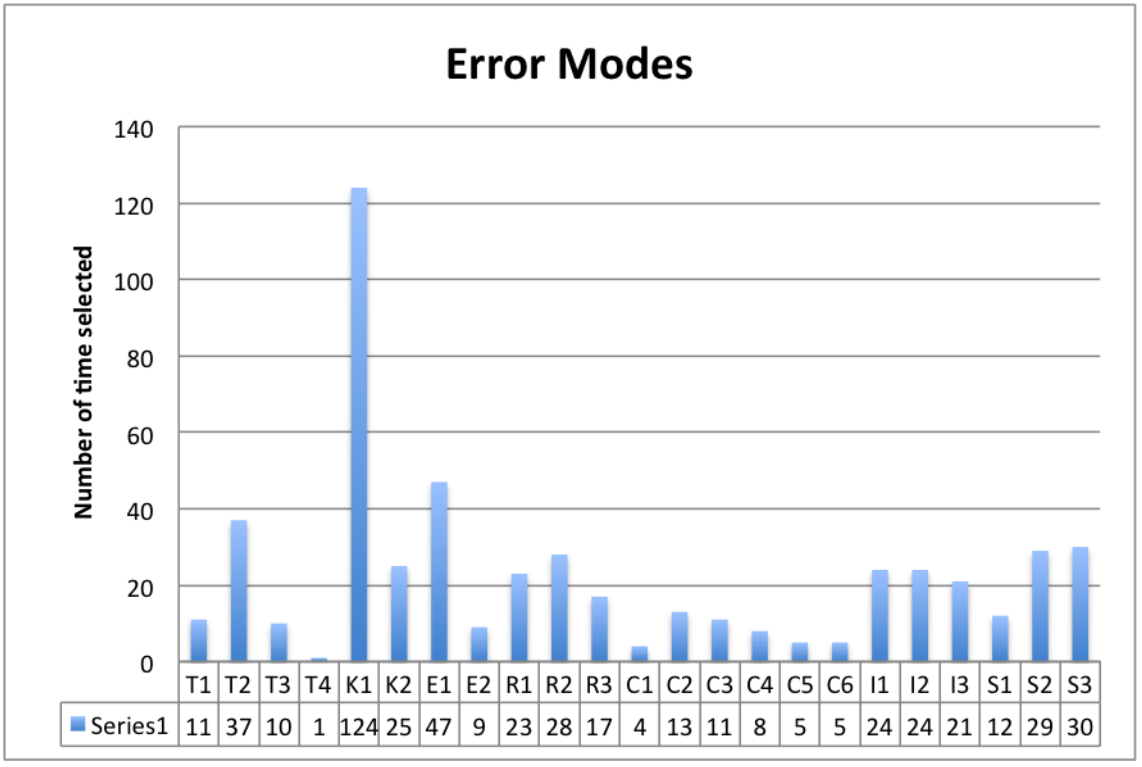

FIGURE 13.4: Error Modes

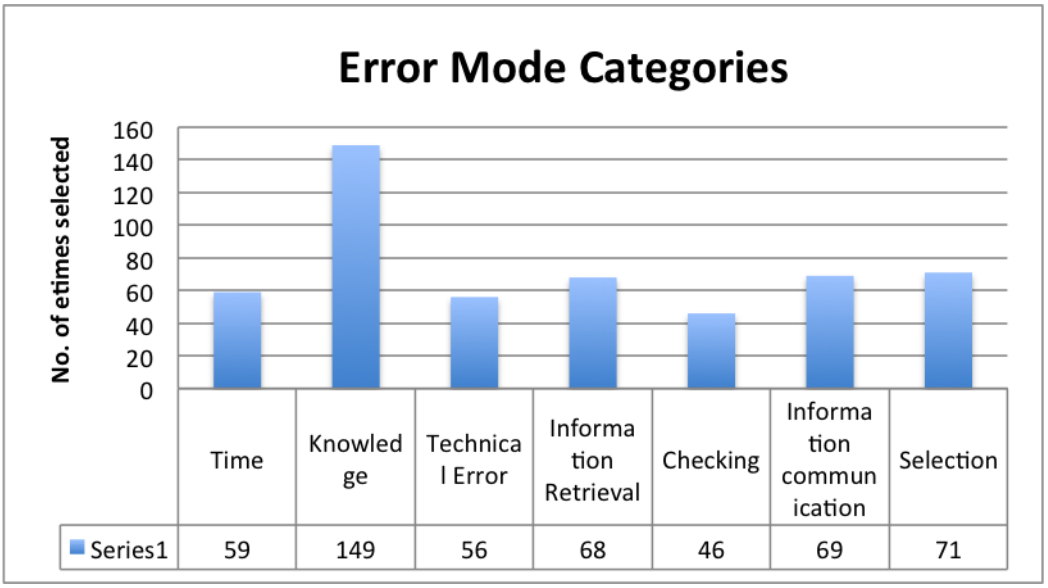

FIGURE 13.5: Categories of Error Modes

other categories the usage of the different error modes varies slightly. However, it is interesting to see that the checking category, with the largest number of error modes, was the least used category.

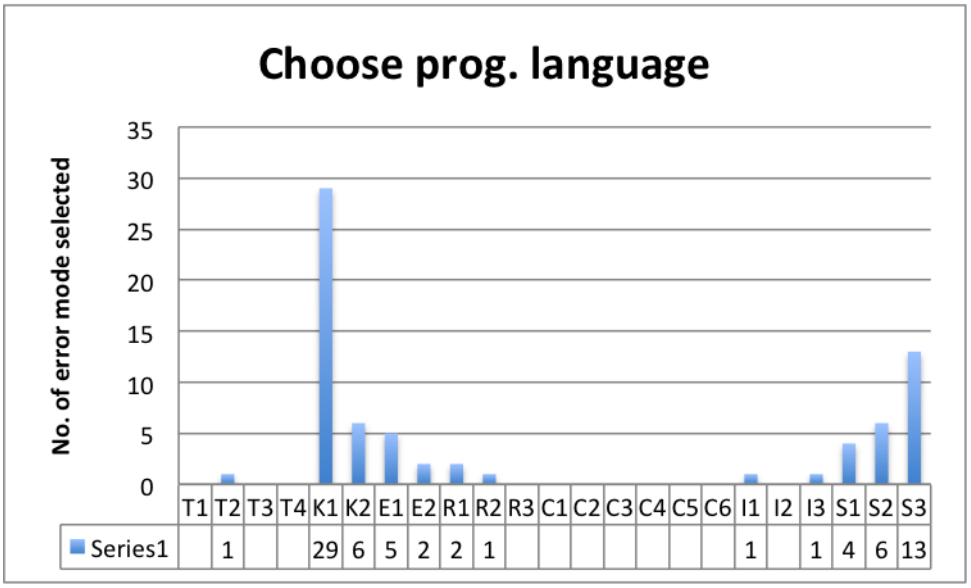

FIGURE 13.6: Error Mode: Choose programming language

13.2.1 Choose Programming Language

In the first subtask the participant were analyzing the subtask: choose programming language. A total of 67 descriptions of errors were filled in during this task. The majority of the error descriptions can be divided in two categories; due to lack of knowledge or a wrong choice was made. A lot of the answers identified that a wrong choice was made due to lack of knowledge. An example of an error description is "Wrong selection of programming language based on insufficient understanding of tasks". This error description has been matched to error mode S3, *wrong technology selected*. This error description could, however, also be matched to error mode S2, *wrong selection*, made and error mode K1, *Insufficient knowledge*.

It is interesting to see which type of error mode that is selected in errors concerning bad choices. Figure 13.6 gives an overview of the error modes selected in this task. K1 is definitely the most popular error mode. S3 is next, but it is used half as many times as K1. Most of the consequences of the errors concerns time lost, and that problems occur due to a bad choice. Time wasted because the development needs to start over, and time used to train the developers in an unfamiliar language are examples when time is identified as the consequence. When the consequence regarded a bad choice, the responses were e.g. hard to develop the needed functionality due to the restriction of the language.

In remedy strategy the participants were to find a strategy to avoid the errors they found to happen again. A lot of participants identified more training and experience as a strategy. This corresponds to the observation that a lot of the errors involved lack of knowledge. The strategies also suggest that more time and investigation should be

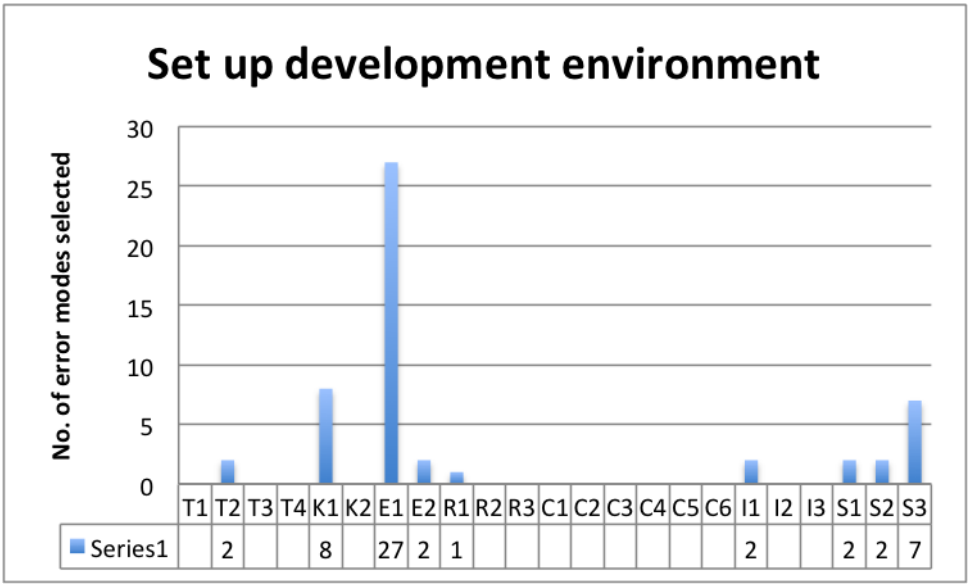

FIGURE 13.7: Error Mode: Set up development environment

provided in this part of software development, a few suggestions about the change of process was also provided.

Most of the responses in this task make sense, and a lot of them concern the same type of error and consequences. However, there are a few responses that seem a bit off relative to the task. An example from the results is "The language works different on pc and mac".

13.2.2 Set up Development Environment

The responses in this task are more varying, and concerns a variety of problems that may arise. There were 52 responses, from 36 participants. E1, *wrong configuration*, was the error mode that selected to a majority of the error descriptions, see Figure 15.1. An error description that recur is *wrong configuration*, and a lot of the other responses relates to configuration problems. Some of the responses were more specific and detailed than just referring to the configuration. "Loss of data/conflicts" and "version control not working properly" are some examples. The errors also concern different aspects of errors that may occur during development setup. "No knowledge about software", "Unsupported OS" and "Problems with identifying packages and so on" are other examples from the results.

The consequences are, as the error description, quite varying. However, in a development set up, like when choosing programming language, time wasted is a major concern. The other consequences are related the to problems in error description. One example from the results is: error description "configured differently in the production environment than test environment", with the consequence "Code that works in development

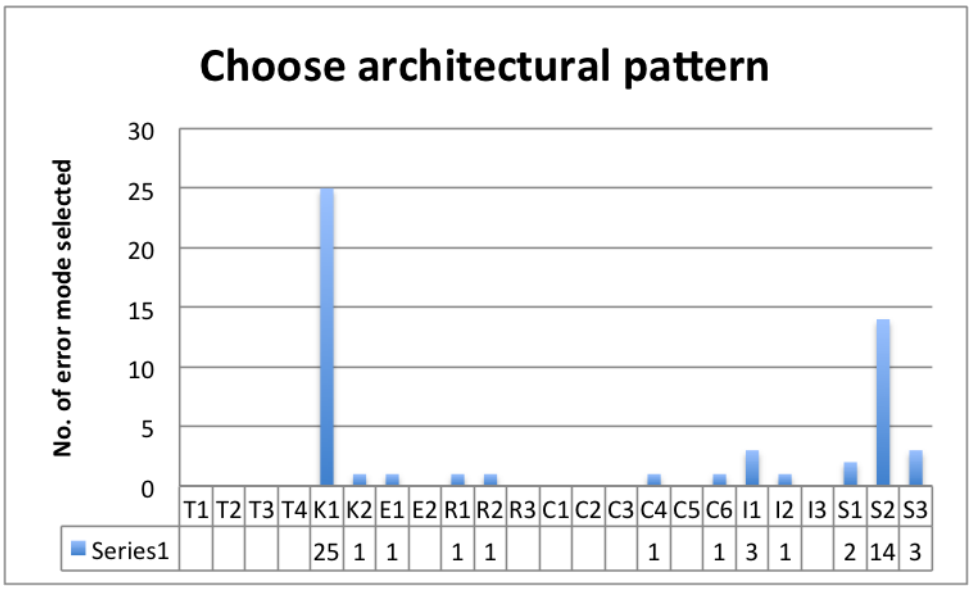

FIGURE 13.8: Error mode: Choose architectural pattern

environment does not work in production environment". A lot of the consequences are, like the example, related to the error description.

The remedied strategy suggestions relates mostly to experience, training and knowledge. There are also suggestions that relates to process and planning, like: guides on how to set up environments, and documents containing information on previous problems. Other suggests that the developers should use the time they need during the setup, and pay attention while they do the configuration.

In this task there were also some results that differed from the rest, where it appears as the participant had misinterpreted the task. One example from the results is "bad atmosphere among developers".

The fact that there are several issues related to this subtask corresponds to the conclusion of the focus group. The participants in the focus group had all experienced problems due to incorrect setup of development environment.

13.2.3 Choose Architectural Pattern

In this task there were 50 answers from 36 participants. Figure 13.8 show the selected error modes in this task. K1, *insufficient knowledge*, is the most used error mode followed by S2, *wrong selection made*. The selection of error modes in this task is similar to the error modes selected in the "Choose programming language" task. A lot of the error descriptions identifies that a wrong selection or an unsuited pattern were selected. Another error that several participants identified was too little knowledge about the pattern, or that a wrong choice was made due to lack of knowledge.

The consequences identified concerns, among others time, bad application and poorly written code. Time is a concern in several error descriptions, it can be related to wrong choices, like “time spent on finding a more suitable pattern”, or to try to fix a bad decision, like “Use time to write the pattern correctly”.

The remedied strategies suggested in this task include experience, training, planning and knowledge. The strategies suggest that the developers should get time to gain knowledge about the pattern that is used, preferably prior to the start of development. “Seek knowledge about the pattern before starting” and “Better knowledge of different patterns and a more thorough process of choosing pattern” are examples of remedies. Another strategy was that the developers in the company should know a variety of patterns, leading the company to possess broader experience within pattern knowledge.

Some of the responses seems a bit inappropriate in conjunction to the task. Examples of error descriptions are “Wrong check on wrong object” and “Different patterns used by different programmers”.

13.2.4 Identify Problems/Uncertainties in Requirements

In this task there were three main sets of error modes that were used most, Knowledge, Information Retrieval and Communication Retrieval, see Figure 13.9 for more details. There were 53 responses in this task by 39 participants. The errors identified in this task concerns misunderstanding of requirements, misinterpretation of requirements, problems or requirements not identified and incomplete requirements. Typical comments in error description are “Incorrect interpretation of requirements”, “requirements are unable/unreasonable to comply to” and “Requirement lists is incomplete, client wishes to add additional requirements”.

The consequences identified that there is a risk that the end-product will not be complete, or at all what the customer expected. If problems and uncertainties are not identified early in the project it may lead to more serious problems later in the project. Another consequence was time spent on recovering, and that the project may exceed its estimate.

The remedy strategies suggested include, among other suggestions better communication with customer, a thorough understanding of the requirements and requirement analysis. However, most of the remedies strategies concerns better communication between developers and customers. “Maintain good communication with client. Ask questions if unsure about requirements” and “Thorough dialogue with the customer about expectations and requirements” are some examples. Other suggestions concerns the quality of the requirements, and that a thorough process of checking the requirements is necessary.

Most of the responses in this task are reasonable. However, there was one response that distinguished themselves from the others. The error description of this response was “Do wrong test” which is not a representing error description for this task.

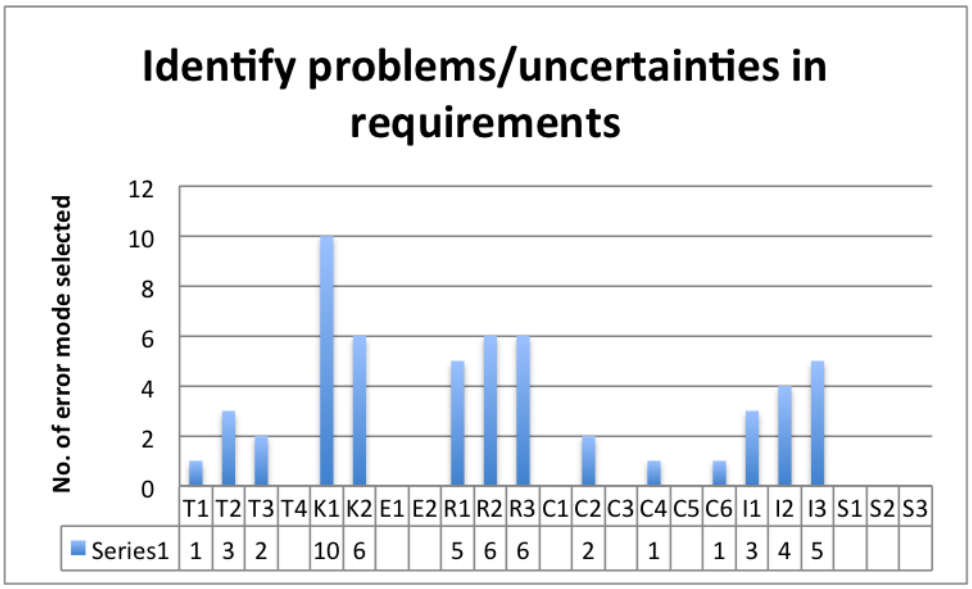

FIGURE 13.9: Error Mode: Identify problems/uncertainties in requirements

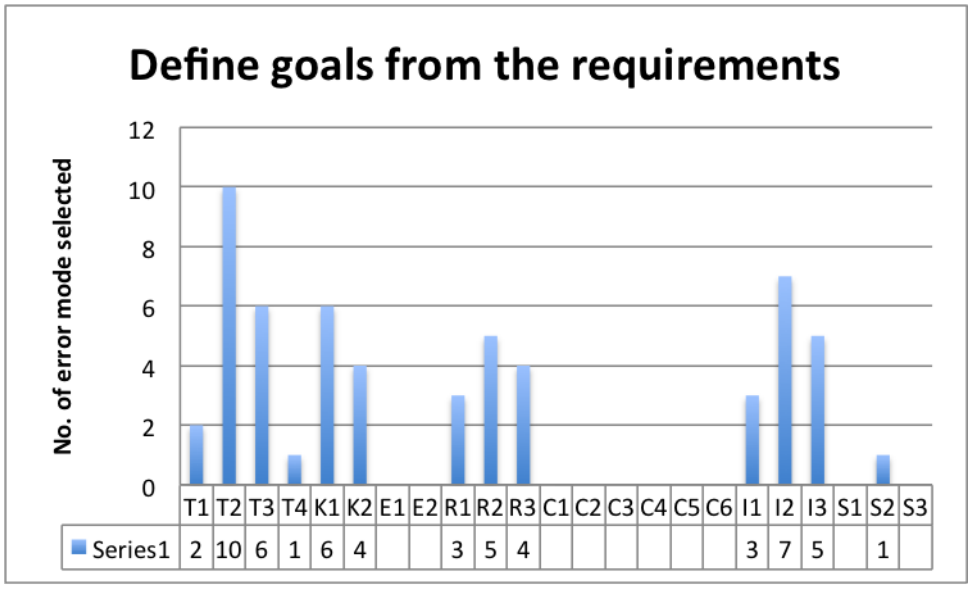

FIGURE 13.10: Error Mode: Define goals from requirements

13.2.5 Define Goals from the Requirements

From Figure 13.10 we can see that Time, Knowledge, Information Retrieval and Information Communication were the error modes most used. There were 53 responses from 37 participants. The error descriptions in this task include goals that are not identified, time estimating, either too ambitious goals or goals that lack ambition. There are also concerns regarding misunderstanding of requirements leading the goals to be ambiguous and not specific enough. "Define wrong goals based on insufficient understanding of product requirements", "too short deadlines for each goal" and "goals are larger/more complex than originally assumed" are examples from the results.

The consequences regarding the errors that were found are in accordance with the error descriptions. A lot the the responses are related to that the end-product will not be complete, and that it does not meet customers need. “Missing functionality discovered at a later stage of development”, “unsatisfied customer/bad results” and “Goals not reached in time” are examples. Most of the responses are similar or addresses about the same problems as these examples.

The remedied strategies concerns better communication between customer and developers, time management, and processes of identifying goals. One of the remedied strategies suggests that the customer should be involved in every major step of the development, another example is “Communicate well with client. Seek confirmation that you are on the right path before implementing”. More experience with time management and “get more experience with how long different tasks take” are example on strategies concerning time. Another good example from the result is “More time spent on researching the project, talking with the customer and gaining an overall good overview of the size of the project”.

13.2.6 Develop Mockup/Prototype of Solution

Time and Information Communication were the most popular categories of error modes in this task, see Figure 13.11. There were 53 responses by 40 participants. The responses in the error description concerns that the prototype might be too good, leading the customer to believe that the end-product is almost done, or that the prototype does not meet the customers expectation and lack functionality. Another concern is related to how realistic the prototype is relative to the end-product, example: “Mockup does not provide a realistic image of what the app can do”. The next major concern is about timing. Not being able to complete the prototype within time is a recurring concern. Another error description worth noticing is “Prototype has functionality that is hard to implement”.

A consequence identified is that the project will need more time to finish the product, “will not complete in time or have to work extra” is an example from the result. Unsatisfied customer is another consequence identified by the participants, which recurred in the responses. Another consequence concerns the need for additional changes on the prototype to make the customer satisfied.

Remedied strategies includes time management, better communication with customer and planning. Better communication with the customer is identified in several responses. Better communication is believed to give the customer more realistic expectations, one example from the result is “More communication with customer”. More frequently contact could lead to smaller adjustments in time estimation along the development, and mistakes/misunderstandings would be fixed at an early stage. There were also strategies to resolve the time issues, one example is “better planning, but more importantly plan

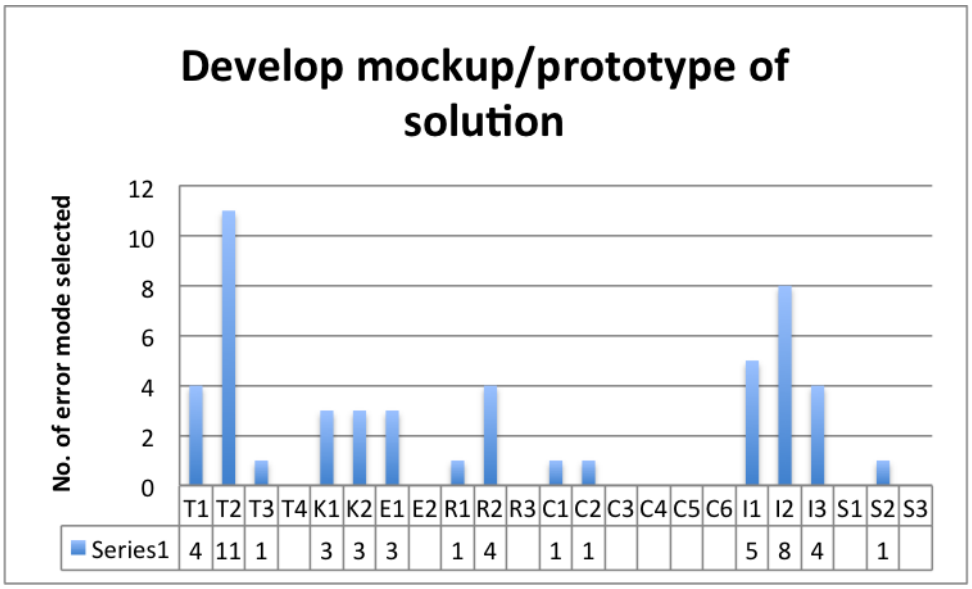

FIGURE 13.11: Error Mode: Develop mockup/prototypeof solution

for underestimated schedule and underestimated workload happening, no matter how well the project is".

13.2.7 Review Codes Behaviour

In Figure 13.12 we can see that K1, *Insufficient knowledge* was the most used error mode in this task. The second most used error mode category is Checking, where C3 *right check on wrong object* was most frequently used. There were 38 responses by 30 participants. The most repeated error description was "Place breakpoints at wrong places". Some of these error descriptions are more detailed, like "place breakpoints at places where the code runs as it should". Other concerns are about the inability to find the expected behavior or problem, too many or too few breakpoints or that the developer does not understand the code to be reviewed.

The consequences of the errors concerns timing, code not properly tested and that the problems that was identified need to be fixed. Examples from the results are "Time spent reviewing code", "Functionality is not properly tested" and " Code does not run as expected, harder to test as wanted".

Remedied strategies include more training and experiment, documentation and more and better focus on code review. The need for training and experience is identified by several participants. These strategies include concrete statements like "More experience", but also strategies like "Be aware of what parts of the code are to be/need to be tested". Better knowledge on placing breakpoints and better knowledge about the code that is reviewed are strategies concerning lack of knowledge. One strategy was to use

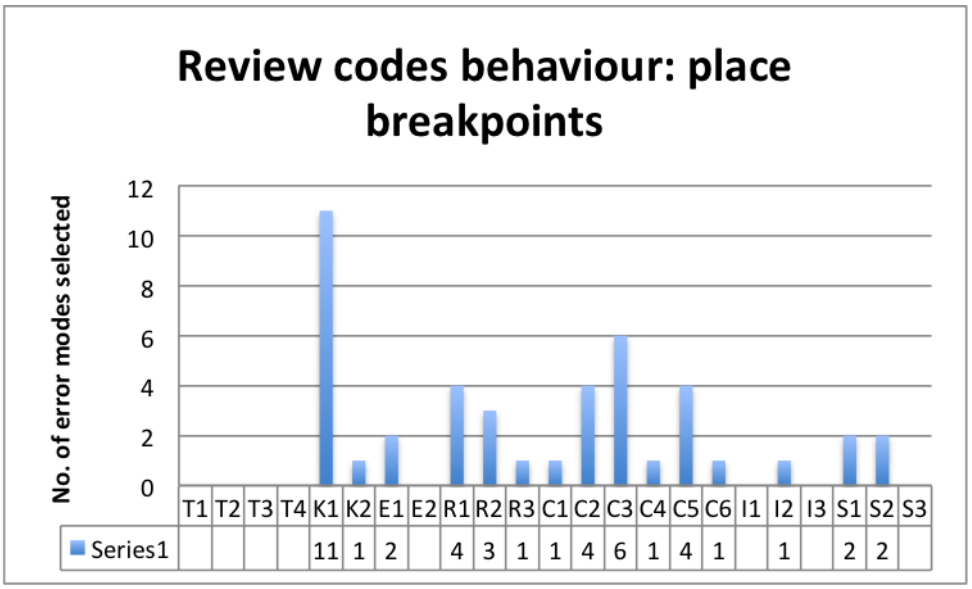

FIGURE 13.12: Error mode: Review codes behaviour: place breakpoints

indentation, while another strategy with error description concerning code not properly tested says "Change attitudes toward code reviews".

In this task there are several error descriptions that diverge from the task. Examples from the results are "Programmer does not understand how code works. Look at unimportant breakpoints" and "Can't use the tool".

13.2.8 Review Code: Evaluate Behaviour

Figure 13.13 shows the error modes selected in this task. K1, *insufficient knowledge* and the entire Checking category was used most. There were a total of 36 responses by 28 participants. The error descriptions identified in this task are among others that there were not enough extensive testing, the code does not behave as expected and tests performed on the wrong elements. Examples from the results are "Check that the code behave as thought, but not as required through the specifications", "Not enough heavy testing (number of users etc.)" and "Accept poor behavior".

The consequences identified in this task are related to time fixing problems, undiscovered errors leading to poor software, and the end-product's inability to meet the requirements. An example from the result concerning the quality of the code is "has untested and potentially wrong functionality in the code". Other consequences was the time used to fix the software when it did not behave as expected, like "Time spent on finding and fixing the error". Another example related to failure in the evaluation was "Has untested and potentially wrong functionality in the code".

Remedied strategies suggested in this task are more training in code reviews, extensive testing based on the project requirements, and the need for new routines when evaluating

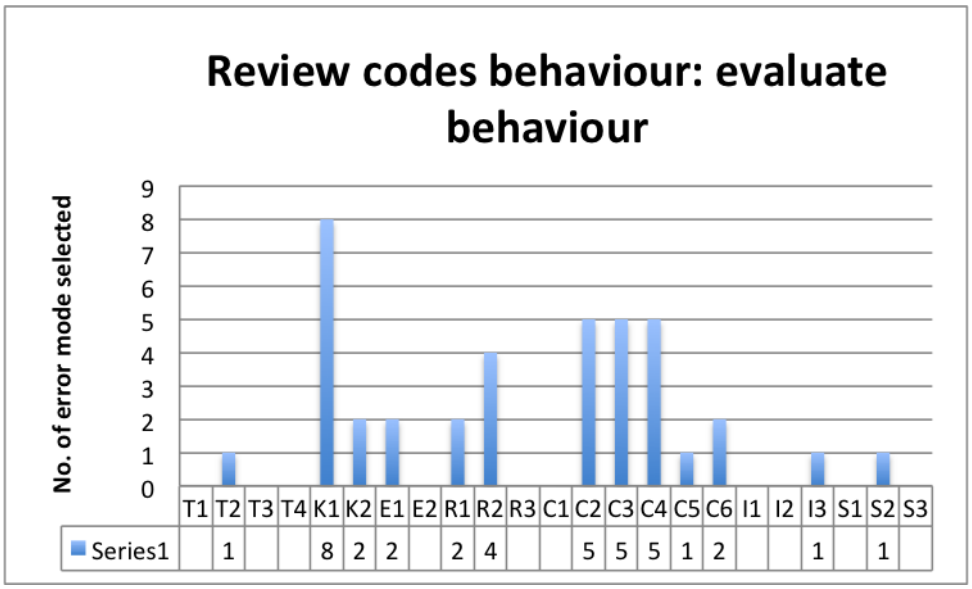

FIGURE 13.13: Error Mode: Review code: evaluate behaviour

code. Some suggestions are about the programmer's ability to see when their knowledge is not sufficient and ask for help, an example is "More knowledge about code review. Ask other developers for help when needed. See your own limits". Other suggestions were "Peer reviewing of code" and "Better routines of code reviewing".

13.2.9 Modification: Identify New Necessary Functionality

In this task K1, *insufficient knowledge*, was the error mode used most. The categories Information Communication and Selection was also selected as error modes in a substantial part of the responses, see Figure 13.14. There were 40 responses in this task made by 29 participants. The error description in this task includes existing functionalities added again, unnecessary functionality added and that necessary functionality is not identified. Other errors are about the developers' ability to code the new functionality, like "Developers do not know how to implement new functionality", or that the existing code does not support the new functionality to be added.

Time is identified as a consequence of some of the errors. "Time wasted on creating functionality" and "Time used on changing product" are some examples. Other concerns about time is that time does not suffice for the changes that needs to be done. Other consequences are about redundancy of functionality, and that the necessary functionality is not added, either due to it not being identified or the developers lack experience.

Better communication within the development team and with the customer are suggested remedied strategies. Another focus for several of the responses concerned requirements. "A well structured design-phase where requirements are well-defined and approved by

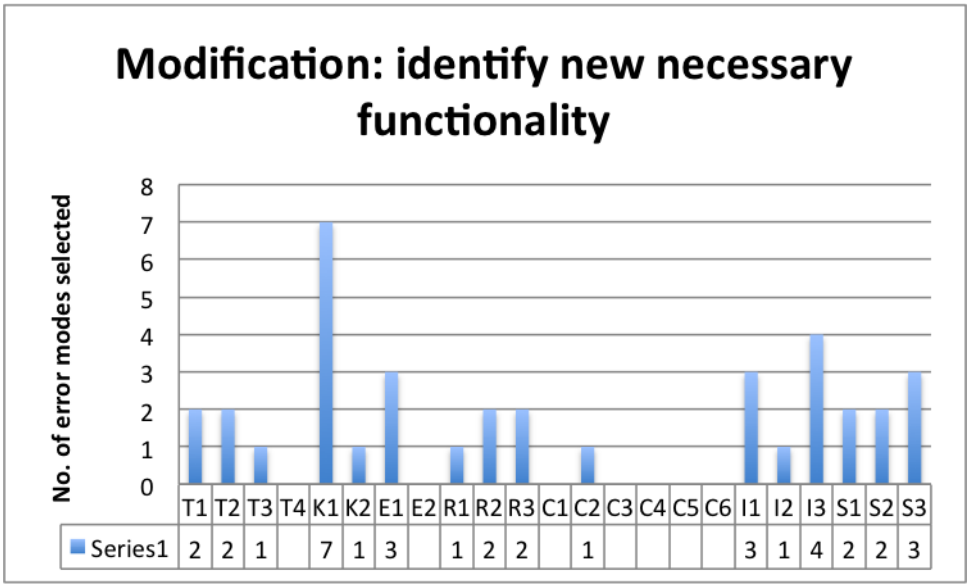

FIGURE 13.14: Error Mode: Identify new necessary functionality

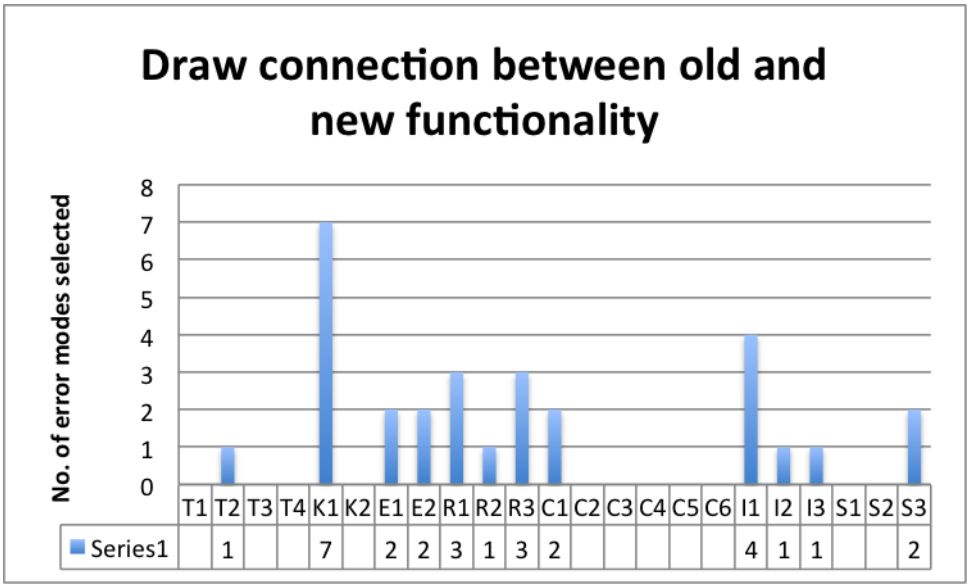

FIGURE 13.15: Error Mode: Draw connection between old and new functionality

customer" and "Check requirements before deciding upon new functionality" are examples from the responses. Time Management accordance to the new functionality is another suggestion, and to always maintain modifiability when writing code.

13.2.10 Modification: Draw Connection Between Old and New Functionality

In Figure 13.15 we see that K1, *insufficient knowledge*, was the most used error mode, followed by I1, *information not communicated*, and the information retrieval category. In this task there were 30 responses made by 28 participants.

Recurring error descriptions in this task concerns trouble with combining the old code with the new functionality. Other errors descriptions are about the quality of the old code making it hard to add new functionalities. Examples from the responses are “Not doing modification properly due to difficulties understanding the old code” and “poorly documentation of code”. Another error description identified is that the old code is misunderstood and the new functionality will thus not work as intended due to these misunderstandings. “Old code is ignored”, “New functionality requires changes in old code” and issues with time it takes to rewrite old code are other examples from the results.

A majority of the consequences identified concerns the time used to solve the problems at hand. “More time than expected is used” and “Not sure where to make changes, time spent to figure it out” are examples about time. Incompabilities between old code and new functionality is also a recurring consequence. One example from the responses is “New functionality does not work as it should, or ruins old functionality”.

The remedied strategies in this task concerns documantation, modifiability and testing of existing code. Examples from remedied strategies suggested are “have good overview of code. Draw class diagrams” and “Make sure the new functionality will not cause any problems”. Proper documentation when writing code and to keep the code modifiable when coding is stressed in several responses. Another strategy is “test old code before use. Review documentation”.

In this task there are responses that differ from what was expected as error descriptions. Examples of these are “Old and new code have little in common” and “New code written in wrong language”.

13.2.11 Create New Functionality: Code the Changes

In this task K1, *insufficient knowledge*, and T2, *underestimated workload*, was the error modes used most and in that order, see Figure 13.16 . There were 32 responses made by 30 partcipants.

The error descriptions in this task concern the developers’ abilities to code the changes. Other concerns are about how new code affect the old code in an undesired manner. “New code overwrites old code” and “Changes in code leads to new unforeseen faults” are examples from the result where the new code affects the existing code in a bad way. Other errors identified concerns the quality of the old code, an example from the result is “Spaghetti code, many code changes needed for small functionality change”. How long time it takes to make the changes are another error description that recurred through this task.

There are mainly two consequences identified in this tasks, and these are time overuse and that the program does not work as intended. The consequences about time overuse

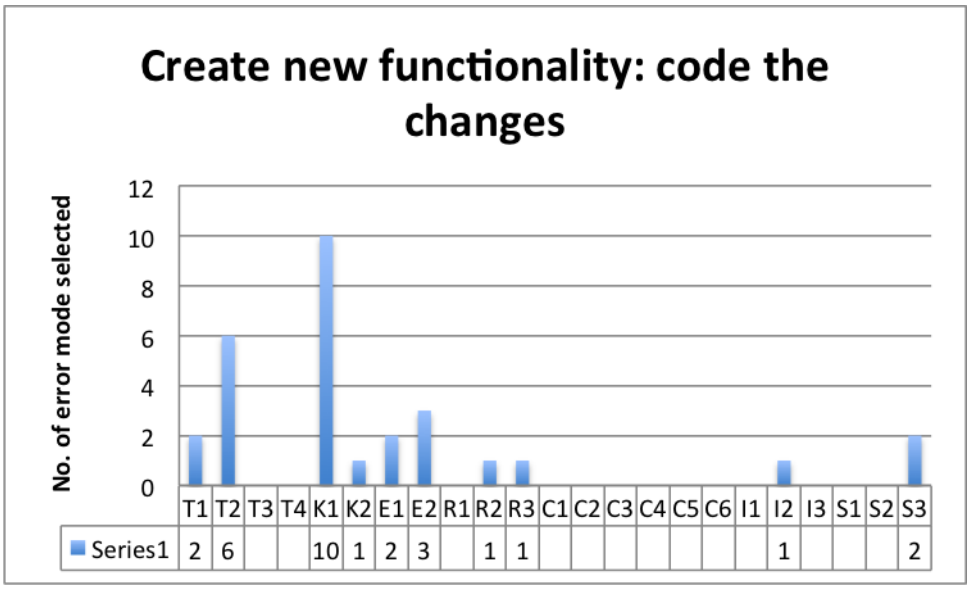

FIGURE 13.16: Error Mode: Create new functionality: code the changes

concerns to a large extent that the task takes longer time than accounted for, or the fact that more work requires more time. Examples from the responses are "More work,which takes more time", "workplan needs to be revised" and "delays". The quality of the software after adding new functionality was the other major concern. "Old functionality is destroyed, needs to be fixed" and "Non-functioning software" are other examples.

In remedied strategies better testing and more training are suggested. "Test more aspects of the code than the ones directly connected to the change" and "Check compatibility before writing code" are examples from the results. Better time management is also suggested as a strategy to resolve the timing issues in projects. Another example from the results are to "separate the functionality, use own/new variables" and "Comment/-document code while reviewing old code".

13.3 Post-Experiment Questionnaire

The primary focus of the post-experiment was to get the perception of SHERPA. All of the respondents answered the questions in this questionnaire. There were one case where the student had checked of Agree on all of the questions.

Figure 13.17 shows results from the post-experiment questionnaire. In this figure the results are merged into three categories in stead of five which it is in its originally form. In section B.4, the raw data is provided. In Figure 13.17 provided in this section the categories agree and strongly agree is merged into agree, likewise are disagree and strongly disagree merged into one column disagree.

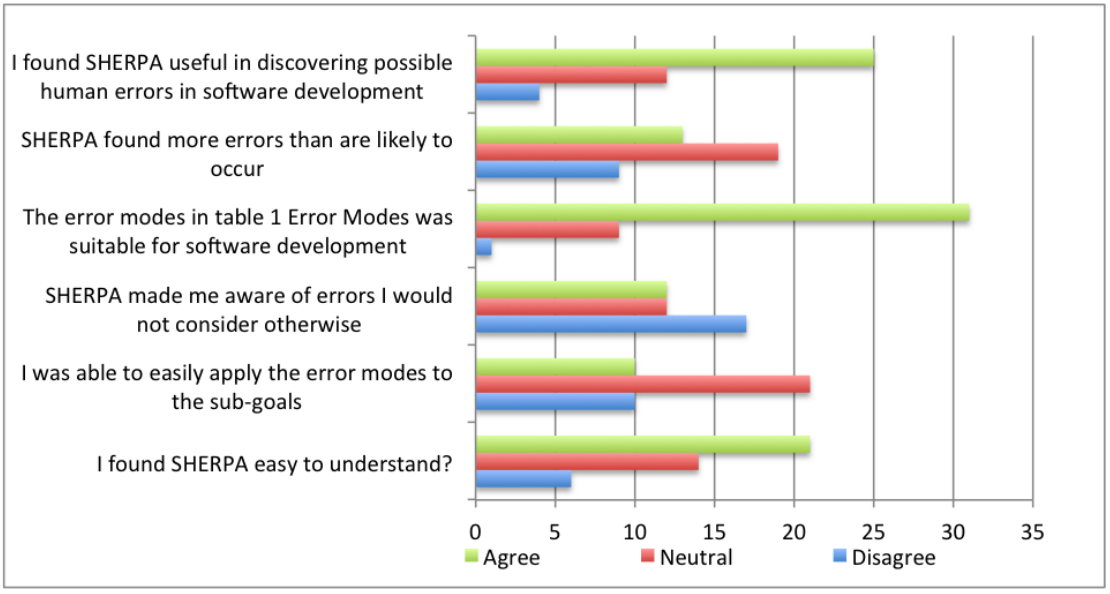

FIGURE 13.17: Results from Post-Experiment Questionnaire

There are two questions where there are significant positive responses, as seen in Figure 13.17. These questions were "I found SHERPA useful in discovering possible human errors in software development" where 10% answered disagree, 29% neutral and 61% agree, and "The error modes in table 1 Error Modes was suitable for software development" where 2% answered disagree 22% neutral and 76% agree. In one question there were a majority of negative answers relative to positive and neutral, this was the question "SHERPA made me aware of errors I would not consider otherwise", where 41% answered disagree, 29% neutral and 29% agree.

The category neutral has quite high response rate in the questionnaire. In two questions there were a higher per centage of neutral, than agree and disagree. These questions were "SHERPA found more errors than are likely to occur" and "I was able to easily apply the error modes to the sub-goals".

Part VI

Discussion and Conclusion

Chapter 14

Discussion

In this chapter a thorough discussion of the results and findings are presented. The participants in this experiment had a wide range of experience. In chapter 13 we see that over 50% of the participants had no IT working experience, other than courses and projects attended at the university. SHERPA is highly dependent on the analysts opinion and experience. Through the experiment there were different quality on the responses provided. In some cases where the participants had little experience, the responses seemed to have little to do with the task given. In the result chapter the participants that scored two in programming experience was disregarded, due to that SHERPA is as dependent on the analyst as it is. If we choose to use participants with low programming expertise, this would be in contradiction to SHERPAs validity statements.

Even though there were some strange answers, there were several valuable responses contributing to form an image of how SHERPA could be used to prevent errors in software development. Altogether, there were acceptable responses in all tasks.

14.1 Error Modes

In this experiment three new categories of error modes was added to the list of error modes used in SHERPA, in addition to one category that was removed. The three new categories were Time, Knowledge and Technical Error, and the one removed was Action. In chapter 13 we see Figure 13.4 presenting the use of error modes during this experiment. The error mode K1, *insufficient knowledge* is definitely the most popular error mode and was used 124 times. Next is E1 *wrong configuration*, followed by T2 *underestimated workload*. The three most used error modes are all new error modes added in this experiment. These were necessary and contributes to tailor SHERPA better to software development. In the result of the post-questionnaire 76% of the respondent answered that they found the error modes in the experiment suitable for software development.

14.1.1 Time

When adding the category Time to the error modes it was expected that the error modes in this category would be used a lot during the experiment. When looking on the results they were used less than expected. The category has four error modes T1, *underestimated schedule*, T2, *underestimated workload*, T3, *Overestimated workload*, and T4, *unbalanced workload*. When looking at the overall figure of error modes used, in Figure 13.5, we see that it was used less than some of the other error categories.

In Figure 13.4 in section 13.2 we see that T2, *underestimated workload*, was most used out of the four erros modes within the category. T4 was only used one time through the experiment. These numbers indicates that all of these error modes within the Time category are unnecessary. T1 and T2 are very similar to each other, and perhaps only one of these are necessary.

14.1.2 Knowledge

The category Knowledge was added to cover the need of knowledge work in software development. As expected, the use of this category was widely popular during this experiment. K1, *insufficient knowledge*, was used more than K2, *overrated knowledge/arrogance*. The Knowledge category is a suitable choice in several tasks, as it is a huge part of software development. This category can be linked to the Selection category, and it was intriguing to see whether the participants would choose Knowledge or Selection when an error occurred due to a wrong selection. This issue would arise especially in the two tasks were there was a choice to be performed. From the results in subsection 13.2.1 we see that both K1 and S3 was selected as error modes. S3, *wrong technology*, and S2, *wrong selection made* was used mostly when it was specified that there were a selection, and K1 was used when the selection was bad, due to lack of knowledge in task "Choose a programming language". However, in the task "Choose architectural pattern" there is a mix of when the different modes are used. Both error mode categories, either Knowledge or Selection, are considered to be appropriate choices of error modes in these tasks. Another question to consider is whether these two error modes within the category Knowledge are sufficient and cover all aspects of knowledge work in software development.

14.1.3 Technical Error

E1, *wrong configuration*, was used several times during the experiment, but when going through the results it seems as this error mode was used in places it should not have been. It seemed, that when respondents were not quite sure on what error mode was most suited, they just picked E1. However, it was expected that this error mode should

have been used in the task "Set up development environment", and as expected E1 was used 27 times in this task, out of the 47 total times it was used in the entire experiment. See section B.4 for more details.

E2 version control was used in a total of nine times during the entire experiment. It was expected that this error mode would have been used primarily in two task, "Set up development environment" and "Create new functionality: code the changes". Even though the error mode was not used a lot of times, the overall impression is that the times it was actually used they were a necessary contribution to the error modes, where no other error mode could have covered the needs.

14.1.4 Information Retrieval and Information Communication

The categories of Information Retrieval and Information Communication, is approximately equally used throughout the experiment. While Information Retrieval concerns issues like getting the applications requirement from the documents, Information Communication concerns issues like communication between customer and development team. In the results these two different categories are used interchangeable. As an example, in error descriptions describing situations where I1, *information not communicated*, would be the most suitable error mode, R1 *information not obtained* is selected. It seems, as the respondents are not aware of the difference between these two error mode categories. The reasons why there may be confusions are numerous. The students are not trained in SHERPA, and only received a short introduction to the method. There were not provided an explanation of the meanings of the error mode in the introduction or in the experiment paper. Perhaps if the error modes were described thoroughly this misconception could been prevented. Another reason for this confusion may be because of the simplification done in section 11.2. The second step of SHERPA, where the sub-tasks are classified into one of the categories, was removed in this experiment. If the sub-tasks had to be classified within a category prior to the selection of error mode, perhaps this confusion could be avoided.

14.1.5 Checking

The Checking category was the least used category. The error modes within checking were used primarily in the two tasks about code review. Checking will in software development be related to testing of the code, which is related to code review. There has not been done any changes to the checking category in this experiment, as it was presumed to be suitable to software as it is. Even though it was not used a lot in this experiment, we believe it contributes to SHERPA and make it complete in accordance to software development. However, all of the six different error modes is clearly not necessary.

14.1.6 Selection

Selection is provided in SHERPAs original form. However, one error mode was added in this experiment, which is S3, *wrong technology selected*. S2, *wrong selection made*, and S3, *wrong technology selected*, was equally used throughout the experiment. These two error modes are similar, and it is important to consider whether both these are necessary or not. These error modes are used interchangeably, but in the task of "Choose programming language" there is a predominance of S3, while in "Choose architectural pattern" S2 is predominated. It is important to consider if the parting of a selection or a technology selection provides more information than the risk of confusion with two almost equally error modes.

14.2 Discussion of the Results

In the responses from the experiment we see that in some cases the respondents have copied the error description from the table Error Mode, provided in the experiment paper, and inserted it into the SHERPA table. To give an example, in the task "Choose programming language" they wrote "insufficient knowledge" in the error description column and "K1" in the error mode column. These responses do not provide enough information, and we would prefer what kind of error occurs rather than the description of error mode. These responses will be disregarded in the analysis of this experiment.

There are several reasons why this confusion may happen. Both in the error mode table and in the SHERPA table, the term "Error description" is used. Perhaps if the description column in the error mode table was called error mode description, this problem could have been avoided.

Another reason for this problem might be the simplification we did in chapter 11. We removed the second step of the original form of SHERPA, because there were confusions where the same problem arose as discussed here. It is possible that this simplification was not as effective as we hoped it to be, or created more confusion.

In the results there are cases where there seem to be little connection between the error mode and the error description. All the error modes has a letter and a number that indicates what category it belongs to, and what description that accompany the error mode. It is conceivable that the participants have filled in wrong information because of a mistake. In these cases the participants has noted C1, instead of S1, which would have been a more credible error mode. In other cases the error mode selected seem as a solution only because there were no other modes they felt were appropriate. These observations seem credible in accordance with the responses the participants gave in the post-examination questionnaire. Most of the participants, 51% stated that they were neutral to the question about how easy it was to apply error modes to the sub-goals.

In this experiment there were a total of 23 different error modes. If there are too many choices confusion may arise, as the saying *Too many cooks spoil the broth.* Some of the error modes was used less than other, and are probably not that relevant to software development.

In chapter 11, in the section about recovery, there were made changes to the analysis. I decided to only ask for what was needed to recover from the error, and provided no more information about this step in the analysis. In the responses there seemed to be confusion about the difference between recovery and the remedied strategy. Respondents could write the same in both columns, or in other cases only in one of them. Chapter 11 provides a discussion about software development being an iterative process. In this experiment the participants was able to write a recovery suggestion that had no relation to what would intentionally be the next step in the development process, which is inconsistent with the original SHERPA analysis. Because of the confusion found in the responses, this new approach might not be the best solution to the problem. I have not placed great emphasis on recovery suggestions through this experiment, other than notice how this new approach could work.

It is important to state that most of the participants during this experiment was students with little to no experience with software development(except courses and project at the university), see Figure 13.2, and that none of the participants had previous experiment with HRA. The lack of experience for some of the participants may be the reason why a lot of the respondents answered neutral to the questions asked in the post-examination questionnaire. When people does not feel that they have enough knowledge on the subject they tend to lean toward neutral responses.

The number of responses in each task is declining through the experiment. The participants was asked to fill in the table if they were able to identify a possible error, or leave the task blank and move on to the next task if did not. One reason why there were fewer responses in the latter part of the experiment may be that they did not have enough time to analyze all the tasks. There is a distinction after the six first tasks, where there are significantly more responses on the first six than the five following. Another reason may be that the five last tasks can be considered to be harder than the previous ones. Most likely it is a mix of both these reasons. Even though the number of responses decreased, there were not any responses were the participant only answered the first six, and let the remaining five blank. Usually, they answered two to three of the five last tasks, but it varied from participant to participant which of these tasks they responded to.

When looking at the responses and the results, there are three reasons for error that occurred frequently throughout the experiment. Knowledge, Information Retrieval and Information Communication are according to the responses in this experiment the most

important contributors to errors in software development. When looking at the Figure 13.5 in chapter 13 we see the same, at least when considering the fact that information retrieval and information communication was used interchangeable throughout the experiment, as discussed previously in this chapter. Even though some of the responses in the experiment seemed to be a bit off, and that some of the error modes did not correlate with the error description, these results shows an overall image of what causes errors during software development.

The consequences identified concerns the errors that were found. The consequences are coherent with the errors. It is hard to specify certain consequences, but there were one recurring consequence throughout the experiment, which was time. Time was also one of the categories within the error mode, and before the experiment ot was expected that this category would be used at a higher rate than it was. But the problem of time, which is a common area of trouble in software development, was rather identified as a consequence to other errors than the error itself. The observation of trouble that arise will cost more time, is a good and likely observation.

Through the experience a lot of the remedied strategies suggested are about training, experiment and better communication within the development team and with the customer. These strategies correlates with the error modes that were most used through the experiment, which indicates that SHERPA is consistent. Other recurring strategies were about the need for new processes for the problem areas stated above, and the need for more experience with time management. The remedied strategy part of the experiment is considered to be the hardest part of the analysis. More training and experience are easy strategies to suggest. Even though there is need for more experience and training, it is possible that a lot of the participants wrote it because it was easy. Other wrote more specific strategies that might be more helpful in a real world situation.

All the training these participants received about SHERPA before the experiment a ten minutes lasting introduction. Training time within SHERPA is estimated to be approximately three hours [40]. The participants received far from this amount of training, and even though we made simplification, it is not that strange that there was some confusion through the experiment. The participants did not get to analyze their own work process as the HTA was conducted by the focus group with their programming precedure. It is probable that the analysis would have been easier to conduct if the same person performed the entire analysis.

In this experiment we decided to let all the participants conduct it on their own, to get as much data as possible. In hindsight we see that a group-based experiment might possibly have yielded better results, especially considering the experience of the participants. Teamwork could have strengthened the creativity, and SHERPA is usually performed with several participants. At the same time, it was important to get a high number of responses. In addition, it was valuable to get everyone‘s own opinion in the post-experiment questionnaire, without being influenced by others.

All things considered, the overall impression of the responses were good. Even though there were some peculiar responses, there were good contributions in all of the tasks.

14.3 Research Questions

In this section the results to the reseach questions, provided in section 1.2, will be discussed in detail,

1. *RQ1: Is it possible to successfully apply HRA to software development?*

   This research question was transferred from the specialization project conducted in the previous semester. It concerns if it is possible to apply HRA to software development. In this master thesis, SHERPA was the only HRA model investigated and the answer to this question.

   In the post-experiment questionnaire the participants was to evaluate SHERPA, one of the questions asked was whether SHERPA was useful when discovering possible human errors in software development. 61% answered that it was, 29% were neutral to the question, and 10% did not find SHERPA useful. These numbers indicates that it is possible to apply SHERPA to software development. However, there are much work needed before this question could be answered thoroughly. If more research was done on SHERPA, I believe this approach could be a useful tool in the IT industry. However, it is important to consider if the time spent on analyzing is worth the effort. In this study only one HRA method was tested, and even though I believe SHERPA is useful, there are several other HRA-models that could be more useful.

2. *RQ2: What adjustments are needed for SHERPA to be better tailored to software development?*

   During this experiment we have tested a few adjustment to tailor SHERPA to software development. The Action category was removed from the error modes, because software development was not covered in any of the error modes described within the category. As a result of the discussion and the new changes stated above, three new categories of error modes are added to the collection. The following changes are made to the error modes in SHERPA:

   Time category:
   T1 *Underestimated workload*
   T2 *Overestimated workload*

   Knowledge category:
   K1 *Insufficient knowledge*

K2 *Overrated knowledge/Arrogance*

Technical error:
E1 *Wrong configuration*
E2 *Version control*

Checking category:
C1 *Check incomplete*
C2 *Right check on wrong object*
C3 *Wrong check on right object*

Selection category:
S1 *Selection omitted*
S2 *Wrong selection made*

One other small adjustment was added to SHERPA, namely the criticality step in the SHERPA procedure. In the original form of SHERPA, the criticality of each task is noted with the symbol:! indicating that the error identified is critical. In this experiment the criticality analysis will be noted as in the probability analysis with low, medium or high. The classification of consequences is as follows:

Low (L): little to no consequence

Medium (M) medium consequence

High (H) the consequence is severe

3. *RQ3. Will a set of non trained students be able to conduct SHERPA on a set of problems?*

   This research questions concerns whether untrained participants are able to perform the analysis without any further training in SHERPA. The participants in this experiment received a short introduction to SHERPA lasting about ten minutes. Usually, the training for SHERPA lasts approximately three hours. With these prerequisites one would believe that the students could have difficulties through the experiment. In the post-experiment questionnaire the participants answered the question whether they found SHERPA easy to understand. 51% of the participants answered that they did, and 15% did not, while the remaining was neutral to the question. These numbers shows that most of the participants was able to perform an analysis without too much problems.

   Each task received different amounts of errors, and there were different numbers of participants who answered each task. The participants had a wide variety of experience, and it is likely that the ones with the most experience found the analysis easier to conduct than other participants with less experience. However, all

the questions received 30 or more responses. They reached satisfying conclusions in the tasks, which guides us to the answers to this research questions: Yes, overall the students gave reasonable responses in all tasks they were given.

4. *RQ4. Will the student reach similar solutions?*

   This research question concerns how useful the method is. If the participant identified total different problem areas, this is a bad quality that comes wisth SHERPA.

   There were various answers to the tasks in this experiment. However, there were some errors and consequences that were recurring throughout the experiment. Knowledge, information retrieval and information communication are identified as contributors to errors in software development. A lot of the participants also identified that inadequate time was a consequence to a lot of the errors. SHERPA is considered to be dependent on the person conducting the analysis. This means that there may be different results depending on the person doing the analysis. As this is an area of concern it is reassuring that the participants actually reached nearly the same conclusion. The remedied strategies, and the recoveries will obviously not be equal as these are even more dependent on the analyst's habits and experience. As long as the errors and consequences are approximately the same the results of the analysis is satisfactory

5. *RQ5. Will these solutions be useful?*

   This research question concern whether the results found in this experiment is useful or not. In this experiment, as stated in the previous research question, the experiment identified three contributors to error in software development, namely Knowledge, Information Communication and Information Retrieval.

   Software development is the processing of knowledge in a very focused way, and experience plays a major role in any knowledge related activity [39]. In 2009, incomplete requirements and changing requirements and specifications where two out of top six reasons for software project failures [41]. Effective communication among the stakeholders of a software development project is crucial to its success [42]. Frequent, recurring problems related to the lack of adequate communication among those involved in the development process has been documented as in important reason to for failure in software projects [43].

   All these statements above are previous research made within software development. SHERPA found the same reasons for error during this experiment conducted by students with limited experience relative to professionals in the IT business.

Chapter 15

SHERPA

After the discussion the new version of SHERPA tailored to software development is ready.

15.1 Error Modes

In the post-examination questionnaire we asked whether the participants found the error modes suitable for software developement, and 76 % answered that they did, and only 2% answered no. Even though the error modes was approved by the participants, the previous discussion indicates that there are still changes to be made to improve the method further.

Time
The category Time was used a sufficient amount altogether to keep it as a category in SHERPA. However, the error mode T2, *underestimated workload*, was used more than the other error mode in the category altogether. I would recommend to reduce the time category to the following two error modes:

T1 Underestimated workload
T2 Overestimated workload

Knowledge
Knowledge was the most popular error mode category. Both these error modes should be kept as a part of SHERPA tailored to software development.

Technical
The Technical Error modes are kept as they are considered to be useful, and was used several times during the experiment. E2, *version control*, was not used a lot of time, but it is not covered by the other error modes, and is considered useful. The category change name to Technical.

Checking

The checking category was used less than any other error mode category. However, the checking category is important in the testing part of software development. Currently there are six different error mode within this category, this is clearly unnecessary when used in software development. After some consideration these three error modes are suggested:

C1 check incomplete
C2 Right check on wrong object
C3 wrong check on right object

Selection

Two of the selection error modes, used in this experiment, are similar. Eventhough a choice made was done about a technology, it is still a choice. After considerations I would like to retract the added error mode, S3 *wrong technology selected*, leaving the two in their original form:

S1 selection omitted
S2 wrong selection made

In table 15.1 the final collection of error modes is presented. In the previous draft there were a total of 23 error modes, with these new changes there are now a total of 17 error modes. Hopefully, this reduction will make the process of applying the error modes to the sub-tasks easier.

Error Mode	Error Mode Description
Time	
T1	Underestimated workload
T2	Overestimated workload
Knowledge	
K1	Insufficient knowledge
K2	Overrated knowledge/arrogance
Technical	
E1	Wrong configuration
E2	Version control
Information Retrieval	
R1	Information not obtained
R2	Wrong information obtained
R3	Information retrieval incomplete
Checking	
C1	Check incomplete
C2	Right check on wrong object
C3	Wrong check on right object
Information Communication	
I1	Information not communicated
I2	Wrong information communicated
I3	Information communication incomplete
Selection	
S1	Selection omitted
S2	Wrong selection made

Table 15.1: Final Error Mode

15.2 The SHERPA Procedure

The procedure of SHERPA is as it was in its original form. It still has the same steps through the analysis. The simplifications done in the experiment are set back to its original form. The recovery analysis was the biggest change in of the steps in the experiment. This change caused confusion during the experiment, and are thereby

Step 1: Hierarchical Task Analysis (HTA)

The first step if SHERPA is HTA, to break down the goals into subgoals.

Step 2: Task Classification

Each of the operations found during HTA is classified based on the error taxonomy into one of the following behavior:

- Time
- Knowledge
- Technical
- Information Retrieval
- Checking
- Information communication
- Selection

Step 3: Human Error Identification (HEI)

After each task is classified into a behavior, the analyst consider credible error modes associated with that activity. The error modes are provided in Table 15.1.

Step 4: Consequence Analysis

The next step is a consequence analysis. The consequence of each behavior is considered, as the consequence has implication for the criticality of the error.

Step 5: Recovery Analysis

If there is a later task step at which the error could be recovered, it is entered here. If there is no recovery step, this section can be skipped.

Step 6: Ordinal Probability Analysis

In this step the behavior is assigned an ordinal probability value. The classification of the probabilities is as follows:

Low (L): the error has never been known to occur.

Medium (M): the error has occurred in previous occasions.

High (H): the error occurs frequently

The assigned classification relies upon historical data and/or a subject matter expert.

Step 7: Criticality Analysis The criticality of the error is assigned to the task. The classifications of consequence is as follows:

Low (L): little to no consequence

Medium (M) medium consequence

High (H) the consequence is severe

Step 8: Remedy Analysis

The final step in the process is to propose error reduction strategies. These are presented in the form of suggested changes to the work system which could have prevented the error from occurring, or possibly reduced the consequences. This is done in the form of a structured brainstorming exercise to propose ways of circumventing the error, or to reduce the effects of the error.

15.3 SHERPA in a Software Development Task

In this section an example of SHERPA analysing a subtask is provided. We are analysing the task: *set up development environment for a web project in Eclipse*. This is a small, but realistic task during software development. Firstly a HTA of the task is conducted, before the analysis starts. The HTA is, in this case, analyzed and divided to a low level of subtasks. In regular analyses, the subtasks would not be reduced to this level. The sub tasks are entered and anayzed with SHERPA. The top level tasks, are never evaluated and remains empty in the table. The tasks were no errors are identified remains empty as well.

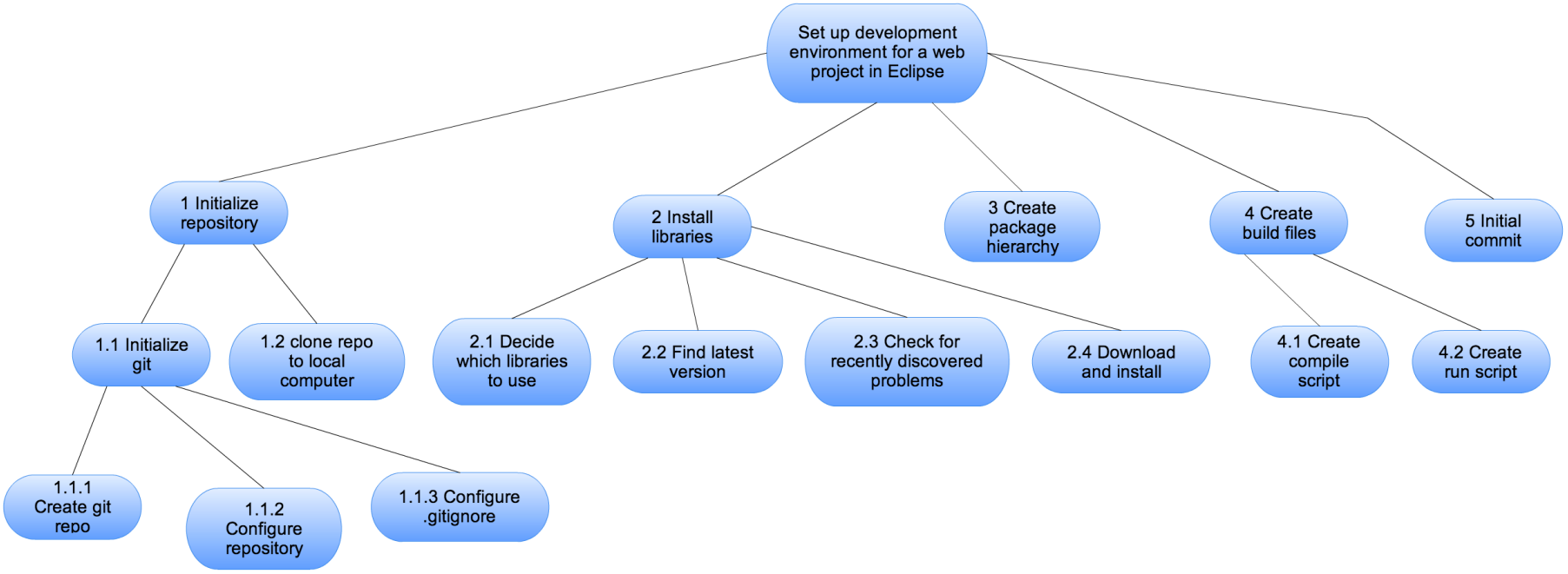

FIGURE 15.1: HTA: Set up development environment

Table 15.2: SHERPA

Task step	Error Mode	Error Description	Consequence	Recovery	P	C	Remedied Strategy
1 Initialize repository							
1.1 Initialize git							
1.1.1 Create git repository	C1	Repository is shared with uncleared people	Uncleared people gets access to source code	None	L	H	Always get another developer/ project manager to look over this step
	E1	Wrong permissions are applied	People not cleared are able to change the source code	None	L	H	
1.1.2 Configure repository	E1	Made repository publicly available	Uncleared people get access to repository	None	L	H	Always get another developer/ project manager to look over this step
1.1.3 Configure .gitignore	K1/ E1/ R1/ R2	Files that should remain hidden are not added to .gitignore	Sensitive data is exposed	None	H	H	Need of a new procedure, where sensitive files are marked as such prior to project
	K1	Wrong files are added to .gitignore	Other developers will not see the work done in these files	None	L	M	
1.2 Clone repo to local computer							

2 Install libraries							
2.1 Decide what libraries to use	S2	Inappropriate library chosen	At a later stage during development, you recognize that the library lack functionality	2.3	L	H	Do better research on what is necessary in your project
2.2 Find latest version	R1/ R2	Beta version is selected, but should have chosen a stable release	Unstable functionality	2.3	L	M	Stay focused while researching
2.3 Check for recently discovered problems	R1/ R3/ K2	Information about problems are ignored or not obtained by the developer	Problems later in the development	None	L	M	Procedure: always check the web for problems, eventhough you are specialist on the subject
2.4 Download and install	K1	Downloaded to wrong folder	Need to be imported to project	None	L	L	
3 Create package hierarchy							
4 Create build files							

Chapter 15 SHERPA

4.1 Create compile script	K1 K1 E1	Wrong class path Wrong main file Compiled with wrong configuration	Not able to compile source code or not able to run source code Wrong parts of code compiled Source code is compiled with debug-parameters	None	M L M	L L M	Coursing in writing compile script
4.2 Create run script	K1 E1	Wrong main file	Not able to run code / Wrong parts are ran	None	L	L	Coursing in writing run script
5 Initial commit							

In Figure 15.1 the HTA of the task is depicted, and the analysis is provided in Table 15.3. The analysis shows that there is only one subtask that is really critical in this task. This task is *1.1.3 Configure .gitignore*. The person conducting this analysis should thus focus on this sub-task, and provide a new procedure on how to avoid making these mistakes in the future.

Chapter 16

Validity

This experiment was performed with students with a wide variation of experience. A small part of the paricipants were students who has attended six or eight semesters at Informatics, and the remaining, larger part of the participants were students attending the fourth semester at Computer Science, see Figure 13.1. The majority of the participants in this experiment had no IT-related work-experience. SHERPA is dependent on domain expertise to get a thorough analysis of the different task. When a large part of the participant has little to no IT-work experience, it is important to consider the validity of the results. There were 41 participants in the study. Before the results can be generalized, SHERPPA should be analyzed by several participants and preferably in a more representative group.

The participants in this study were students with limited experience, both related to courses taken at the university and real IT-work. These facts decrease the external validity of the results, and have been accounted for when analysing the results. To enhance the external validity, the results found were compared with previous research in the field of software development. SHERPA did find similar problem areas within software development and prevoius research.

Internal validity concerns whether the outcome is a causal relationship. To ensure internal validity of the experiement the participants in the experiment that scored two or lower on programming expertise was disregarded from the experiment. In the focus group session, were the HTA was conducetd, the participants were students attending their tenth and last semester at the university. A more experienced group was used to make sure that the task were reliable tasks within software development. To increase internal validity, the questions in the questionnaires were formulated to not allow the participants to cross of every box at one “side of opinion”, to satisfy either their own opinions or what they thought was preferred by the facilitator. The participants in this experiment did not get to conduct a full analysis of SHERPA. SHERPA is thereby not tested in its entirety which decreases the construct validity to this experiment.

Threats to construct validity refer to the extent to which the experiment setting actually reflects the construct under study [27]. To ensure the quality of the results from the experiment, the error modes selected in the task, and the textual responses from the SHERPA table, were compared. The textual responses of error descriptions, should, to som extent correlate with the error modes selected in the task. As an example, if most of the error descriptions in a task concerned insufficient knowledge, the selected error modes should essentially be from the Knowledge category.

Threats to conclusion validity concerns issues that affect the ability to draw correct conclusion between the treatment and outcome of an experiment [27]. The result from the experiment was inserted into tables by the author of this master thesis. The participants filled in the forms by hand during the experiment, the results thereby had to be interpreted by the author before insertion. The results are either in a textual form, or in numbers. When evaluation and analyzing the textual responses it is important to be aware of the threat of fishing, were the analyst is fishing for a specific result.

Triangulation is an attempt to map out, or explain more fully, the richness and complexity of human behaviour by studying it from more than one standpoint [44]. In other words, triangulation relies on using different methods to reach the same results. In this study, data was gathered through focus group, experiment and previous research.

Chapter 17

Conclusion

The goal for this master thesis was to explore whether the human reliability method, SHERPA, could be applied to software development. We conducted a focus group followed by an experiment to evaluate the method. The main purpose of the focus group was to conduct the first part of SHERPA, the hierarchical task analysis. The focus group consisted of students attending their tenth semester at the University, with more experience than the participants in the experiment. The HTA made during the focus group was used in the experiment. The experiment was conducted with 41 participants, with students from 8th, 6th but mostly 4th semester at the university.

RQ1 aims to find out whether HRA methods can be successfully applied to software development. More than half the participants found SHERPA useful in discovering human errors. However, for the time being, it is too soon to answer the question. SHERPA is definitely off to a good start, but more research is needed to reach a conclusion.

Before the experiment a few adjustments was made to make SHERPA tailored to software development. The majority of these changes concerned the error modes. These adjustments were discussed and analyzed before a final version of SHERPA was presented in chapter 15. Section 15.1 provides a detailed description of the adjustments, which answers RQ2.

It is important that training necessary, before developers able able to conduct an analysis, is limited, if SHERPA are to be used within software development. The students in the experiment received a short introduction to SHERPA before they conducted the analysis. They were still able to perform the analysis without any major problems, which answers RQ3.

One of the disadvantages of SHERPA is that it is heavily dependent on the experience of the analyst performing the analysis. Through the experiment there were mainly three error mode categories, also referred to as behaviors, which recurred. These were Knowledge, Information Retrieval and Information Communication. The consequences

these errors caused were time used to recover. These results recurred through several tasks in the experiment. All in all, the responses within each task were related to each other and concerned the same issues. Hereby, the answer to RQ4 is: all things considered, the students reached approximately the same conclusion in the experiment.

Another important issue to consider is if the results from the experiment will be useful, which is addressed in RQ5. The results SHERPA provided in this experiment gave useful information about the tasks provided. The overall responses, stated above, can be related to research done in software development, where the same problem areas are identified.

All things considered, SHERPA seems to be a useful tool to predict human error within software development. The results from the experiment are credible relative to previous research within software development. However, there is still remaining work to do with the method.

Chapter 18

Further Work

During the focus group the participants found it hard to draw hierarchical task diagrams. Software developers are more familiar with drawing state charts and class diagrams, rather than HTA. Another issue that arose was that the focus group participants found it hard to draw the tasks hierarchical, at the same time as iterative. HTA is used to identify all subtasks in work procedures. If there exist any other methods that break down the tasks equally as HTA, which is more familiar for software developers, this could also be used as step 1 in SHERPA. However, it is important to state that HTA has a very good reputation, and has been proved useful several times through history. The potentially new method need to tested and compared with HTA, before any changes in step 1 of SHERPA are made.

In this experiment the recovery step was disregarded from the analysis. In the specialization project it was suggested that this step could be adjusted to make SHERPA take iterative processes into account. In this experiment the participants were free to suggest a recovery, regardless whether the recovery was an upcoming task or not. This created confusion about the difference of recovery and remedied strategies. In further work it could be interesting to test if this confusion will occur if the participants got a thorough introduction to SHERPA, or if there exist another way to support iterative processes in SHERPA. If a new method of identifying sub-goals will support iterative processes, then current version of the recovery analysis would work. However, at this point, both of these aspects should be considered in further work on SHERPA.

In this master thesis, SHERPA was tested on students from the university with limited experience. SHERPA should be tested in a real software company where they have discovered some issues within the development team. During this experiment the participant did not conduct SHERPA in its entirety. In further work of tailoring SHERPA to software development, a complete analysis needs to be conducted and tested. More excessive testing is also necessary.

Previous, in other domains, SHERPA has been conducted with an HRA expert, and a domain expert. A suggestion is to test SHERPA with both an HRA expert and one or more software developers. The HRA expert could guide the developers if any problem or questions occur. However, it is important to state that it is not necessary to have an HRA expert present when conducting the analysis.

Bibliography

[1] Software bugs cost more than double eurozone bailout. URL `http://www.businessweekly.co.uk/hi-tech/14898-software-bugs-cost-more-than-double-eurozone-bailout`.

[2] GG Schulmeyer. The net negative producing programmer. *American Programmer*, 6, 1992.

[3] Barry Kirwan. Human error identification in human reliability assessment. part 1: Overview of approaches. *Applied ergonomics*, 23(5):299–318, 1992.

[4] Alan D Swain and Henry E Guttmann. Handbook of human-reliability analysis with emphasis on nuclear power plant applications. final report. Technical report, Sandia National Labs., Albuquerque, NM (USA), 1983.

[5] Winfried Hacker. *Allgemeine Arbeitspsychologie: Psychische Regulation von Arbeitstätigkeiten*. H. Huber, 1998.

[6] Julie Bell and Justin Holroyd. Review of human reliability assessment methods. *Health & Safety Laboratory*, 2009.

[7] Kyung S Park and Kwang T Jung. Considering performance shaping factors in situation-specific human error probabilities. *International Journal of Industrial Ergonomics*, 18(4):325–331, 1996.

[8] Adham Mackieh and Canan Cilingir. Effects of performance shaping factors on human error. *International journal of industrial ergonomics*, 22(4):285–292, 1998.

[9] Barry Kirwan. The validation of three human reliability quantification techniquesï£ · ï£ · ï£ · therp, heart and jhedi: Part 1ï£ · ï£ · ï£ · technique descriptions and validation issues. *Applied ergonomics*, 27(6):359–373, 1996.

[10] Human reliability analysis. URL `http://www.nopsema.gov.au/resources/human-factors/human-reliability-analysis/`.

[11] FT Chander, YH Chang, A Mosleh, JL Marble, RL Boring, and DI Gertman. Human reliability analysis methods: Selection guidance for nasa. *NASA Office of Safety and Mission Assurance, Washington, DC*, 2006.

[12] David Embrey. Understanding human behaviour and error. *Human Reliability Associates*, 1:1–10, 2005.

[13] Mike Falla et al. Advances in safety critical systems: Results and achievements from the dti/epsrc r&d programme in safety critical systems. *Falla M.(ed)*, 1997.

[14] Human error slips and mistakes. URL `http://www.interaction-design.org/encyclopedia/human_error_slips_and_mistakes.html`.

[15] James Reason. Human error: models and management. *BMJ: British Medical Journal*, 320(7237):768, 2000.

[16] David I Gertman, Harold S Blackman, JL Marble, JC Byers, and CL Smith. *The SPAR-H human reliability analysis method*. US Nuclear Regulatory Commission, 2005.

[17] The effects of distractions on human performance. URL `http://www.hcpro.com/QPS-211165-234/The-effects-of-distractions-on-human-performance.html`.

[18] Neville STANTON, Allan HEDGE, Hal W HENDRICK, Eduardo SALAS, and Karel BROOKHUIS. *Handbook of human factors and ergonomics methods*. 2004.

[19] Historical software bugs with extreme consequences. URL `http://royal.pingdom.com/2009/03/19/10-historical-software-bugs-with-extreme-consequences/`.

[20] Martin A Stutzke and Carol S Smidts. A stochastic model of fault introduction and removal during software development. *Reliability, IEEE Transactions on*, 50 (2):184–193, 2001.

[21] Neville A Stanton and Sarah V Stevenage. Learning to predict human error: issues of acceptability, reliability and validity. *Ergonomics*, 41(11):1737–1756, 1998.

[22] Barry Kirwan. Human error identification in human reliability assessment. part 2: Detailed comparison of techniques. *Applied ergonomics*, 23(6):371–381, 1992.

[23] Christopher Baber and Neville A Stanton. Human error identification techniques applied to public technology: predictions compared with observed use. *App lied Ergonomics*, 27(2):119–131, 1996.

[24] Statistics.com. URL `http://www.statistics.com/`.

[25] Neville A Stanton and Mark S Young. What price ergonomics? *Nature*, 399(6733): 197–198, 1999.

[26] Neville A Stanton and Christopher Baber. Error by design: methods for predicting device usability. *Design Studies*, 23(4):363–384, 2002.

[27] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer, 2012.

[28] Briony J Oates. *Researching Information Systems and Computing*. SAGE, 2006.

[29] Jenny Kitzinger. Qualitative research. introducing focus groups. *BMJ: British medical journal*, 311(7000):299, 1995.

[30] Richard A Krueger. *Focus groups: A practical guide for applied research*. Sage, 2009.

[31] Jo-Ellen Asbury. Overview of focus group research. *Qualitative health research*, 5 (4):414–420, 1995.

[32] Differencebetween.net. URL `http://www.differencebetween.net/miscellaneous/difference-between-questionnaires-and-surveys/`.

[33] Kristen Ringdal. *Enhet og mangfold: samfunnsvitenskapelig forskning og kvantitativ metode*. Fagbokforlaget, 2001.

[34] Thomas D Cook, Donald Thomas Campbell, and Arles Day. *Quasi-experimentation: Design & analysis issues for field settings*. Houghton Mifflin Boston, 1979.

[35] Anol Bhattacherjee. Social science research: principles, methods, and practices. 2012.

[36] Neville A Stanton. Hierarchical task analysis: Developments, applications, and extensions. *Applied ergonomics*, 37(1):55–79, 2006.

[37] J Annett, KD Duncan, RB Stammers, and MJ Gray. Task analysis (department of employment training information paper no. 6). *London, UK: Her Majesty's Stationary Office (HMSO)*, 1971.

[38] B Kirwan and LK Ainsworth. A guide to task analysis, 1992. *Tailor & Fransis*.

[39] Pierre N Robillard. The role of knowledge in software development. *Communications of the ACM*, 42(1):87–92, 1999.

[40] Neville Stanton and Mark Young. Is utility in the mind of the beholder? a study of ergonomics methods. *Applied Ergonomics*, 29(1):41–54, 1998.

[41] M Bishop. Standish group chaos report: Worst project failure rate in a decade, 2009.

[42] Michael E Atwood, Bart Burns, Dieter Gairing, Andreas Girgensohn, Alison Lee, Thea Turner, Sabina Alteras-Webb, and Beatrix Zimmermann. Facilitating communication in software development. In *Proceedings of the 1st conference on Designing interactive systems: processes, practices, methods, & techniques*, pages 65–73. ACM, 1995.

[43] Bill Curtis, Herb Krasner, and Neil Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.

[44] Louis Cohen, Lawrence Manion, and Keith Morrison. *Research methods in education*. Routledge, 2013.

Appendix A

Experiment

Human Reliability and Software development

Experiment 1

Analysing the use of SHERPA in software development

Merete Aardalsbakke
Supervisor: Tor Stålhane

Thursday, March 27, 2014

Pre-experiment questionnaire

Which semester are you currently attending?

4th semester □ 6th semester □ 8th semester □

Number of months of IT working experience,
including summer jobs and other ……

Rate your programming experience from 1 to 5, where 1 is very little experience and 5 is top of your class:

1 □ 2 □ 3 □ 4 □ 5 □

In this experiment you shall analyse a set of subtasks through a method called SHERPA. SHERPA consist of seven steps, where the first step is done prior to this experiment. The purpose of SHERPA is in our case to predict possible human errors that can occur during software development. Basic programming tasks, like debugging and modification, are broken down into sub-tasks. Each of these subtasks are further analysed through SHERPA. The result of the analysis is documented in a table. In this experiment you shall fill in this table, table 2: SHERPA table, based on the step by step guide provided below. In table 2 you will find an example of a subtask

Step 1: Task analysis
In step 1 the sub-tasks are identified. This was done before the experiment started, and the subtasks are already entered in to the table. You shall fill in the remaining columns.

Step 2: Human-Error Identification

In this step you are identifying possible human-errors that can occur in the different subtasks. For each task consider credible errors that may occur in the situation of the task, and write a description of the error down in the column: Error description. Match the error to the different error modes in table 2: Error Modes. Write it down in the column: Error Mode. If you believe that there are no credible error modes, move on to the next subtask and start again.

Step 3: Consequence analysis

In this step you are to consider the consequence of the error you have identified in the previous step. An example of a consequence might be time wasted. Note down the identified consequence in the column Consequence.

Step 4: Recovery analysis

Write down how the task can be recovered in the column Recovery. A recovery is an action that will fix the error committed. If you are not able to identify a recovery, enter None.

Step 5: Probability analysis

How probable is the possible error? The probability of the error is to be noted as low, medium or high.

- Low: error will to almost never occur
- Medium: error has occurred in previous occasions
- High: error occurs frequently

The probability is noted in the column P.

Step 6: Criticality analysis

The criticality of the error are to be assessed. How critical is the error? The criticality is also noted as low, medium or high. The criticality is noted in the column C.

- Low: little to no consequence
- Medium: medium consequence
- High: the consequence is severe

Step 7: Remedy analysis

The final step is to propose error reduction strategies. These are suggested changes to the work process which could prevent or reduce the consequences.

Table 2: SHERPA table

Task Step	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2.1.2 Locate problem in code when test failed	Could not locate problem Not able to solve the problem R1	R1 K1	A lot of time spent on finding the error Problem not solved, test not green	None Need help from another developer	L M	M M	New process of writing unit testing, or general code. Needs to be more tedious. More training needed
Choose programming language suited for your application							

Task Step	Error description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
Set up development environment							
Choose architectural pattern (e.g MVC, observer..)							
Identify problems/ uncertainties in requirements							

Task Step	Error description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
Define goals from the requirements							
Develop mockup/ prototype of solution (to show to customer)							
Review codes behavior : place breakpoints							

Task Step	Error description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
Review codes behavior: place brakepoints							
Review codes behavior: evaluate behavior							
Modification: identify new necessary functionality							

Task step	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
Modification: draw connection between old code and new functionality							
Create new functionality: code the changes							

Post experiment questionnaire

	Disagree strongly	Disagree	Neutral	Agree	Agree strongly
1. I found SHERPA easy to understand	☐	☐	☐	☐	☐
2. I was able to easily apply the error modes to the sub-goals	☐	☐	☐	☐	☐
3. SHERPA made me aware of errors I would not consider otherwise?	☐	☐	☐	☐	☐
4. The error modes in table 1 Error Modes was suitable for software development	☐	☐	☐	☐	☐
5. SHERPA found more errors than are likely to occur	☐	☐	☐	☐	☐
6. I found SHERPA useful in discovering possible human errors in software development	☐	☐	☐	☐	☐

Thank you for participating in this experiment!

Table 2: Error Modes

Error Mode	**Error Description**
Time	
T1	Underestimated schedule
T2	Underestimated workload
T3	Overestimated workload
T4	Unbalanced workload
Knowledge	
K1	Insufficient knowledge
K2	Overrated knowledge/arrogance
Technical error	
E1	Wrong configuration
E2	Version control
Information Retrieval	
R1	Information not obtained
R2	Wrong information obtained
R3	Information retrieval incomplete
Checking	
C1	Check omitted
C2	Check incomplete
C3	Right check on wrong object
C4	Wrong check on right object
C5	Check mistimed
C6	Wrong check on wrong object
Information Communication	
I1	Information not communicated
I2	Wrong information communicated
I3	Information communication incomplete
Selection	
S1	Selection omitted
S2	Wrong selection made
S3	Wrong technology selected

Appendix B

Responses from experiment

B.1 Pre-experiment questionnaire

Wich semester are you currently attending?	4 semester	6 semester	8 semeter		
Number of students	34	4	3	39	
Number of months with IT working experience	No experience	1-5 months	¿5 months		
Number of students	20	7	9		
Rating of programming experience from 1 to 5	1	2	3	4	5
Number of students	0	4	22	12	1

TABLE B.1: Data from Pre-experiment Questionnaire

B.2 Post-experiment questionnaire

	Votes				
Questions	Disagree Strongly	Disagree	Neutral	Agree	Agree Strongly
I found SHERPA easy to understand?		6	14	14	7
I was able to easily apply the error modes to the subgoals	2	8	21	9	1
SHERPA made me aware of errors I would not consider otherwise	4	13	12	11	1
The error modes in table 1 Error Modes was suitable for software development		1	9	25	6
SHERPA found more errors than are likely to occur		9	19	13	
I found SHERPA useful in discovering possible human errors in software development		4	12	23	2

Table B.2: Data from Post-experiment questionnaire, part 1

Questions	Disagree	Neutral	Agree		Disagree	Neutral	Agree
I found SHERPA easy to understand?	6	14	21		15 %	34 %	51 %
I was able to easily apply the error modes to the subgoals	10	21	10		24 %	51 %	24 %
SHERPA made me aware of errors I would not consider otherwise	17	12	12		41 %	29 %	29 %
The error modes in table 1 Error Modes was suitable for software development	1	9	31		2 %	22 %	76 %
SHERPA found more errors than are likely to occur	9	19	13		22 %	46 %	32 %
I found SHERPA useful in discovering possible human errors in software development	4	12	25		10 %	29 %	61%

Table B.3: Data from Post-experiment questionnaire, part 2

B.3 Error Modes

Table B.4: Data from Error Modes

	T1	T2	T3	T4	K1	K2	E1	E2	R1	R2	R3	C1	C2	C3	C4	C5	C6	I1	I2	I3	S1	S2	S3	Total
Choose programming language suited for your application		1			29	6	5	2	2	1								1		1	4	6	13	71
Set up development environment		2			8		27	2	1									2			2	2	7	53
Choose architectural pattern (e.g MVC, observer..)					25	1	1		1	1					1		1	3	1		2	14	3	54
Identify problems/ uncertainties in requirements	1	3	2		10	6			5	6	6		2		1		1	3	4	5				55
Define goals from the requirements	2	10	6	1	6	4			3	5	4							3	7	5		1		57
Develop mockup/ prototype of solution (to show to customer)	4	11	1		3	3	3		1	4		1	1					5	8	4		1		50

Review codes behavior : place breakpoints					11	1	2		4	3	1	1	4	6	1	4	1		1		2	2		44
Review codes behavior: evaluate behavior		1			8	2	2		2	4			5	5	5	1	2			1		1		39
Modification: identify new necessary functionality	2	2	1		7	1	3		1	2	2		1					3	1	4	2	2	3	37
Modification: draw connection between old code and new functionality		1			7		2	2	3	1	3	2						4	1	1			2	29
Create new functionality: code the changes	2	6			10	1	2	3		1	1								1				2	29
Total	11	37	10	1	124	25	47	9	23	28	17	4	13	11	8	5	5	24	24	21	12	29	30	518

B.4 Choose programming language

Table B.5: Choose programming language

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Not all developers know the language	K1	time spent on learning	Train developers	M	L	Change training requirements

2	The language limits the development	S3	Code not able to be completed	Change language	L	H	Know reqs better before choosing language
3	choose language with lack of functionality	S3	use time to change language/work around		L	M	Maintain knowledge on what different language can do, and what you need it to do
3	language works different on pc and mac	E2	use time to rewrite because it is written on mac/pc		M	M	Have the same version of the language
4	Lacking knowledge in suitable language	K1	New problems may occur due to unsuitable language	Spend time on fixing unnecessary problems	M	M	Gain knowledge in various programming languages
4	Wrong selection of programming language based on insufficient understanding of task	S3	New problems may occur due to unsuitable language	Spend time on fixing unnecessary problems	H	M	Spend more time on understanding the task prior to choosing programming language
5	Programming language insufficient for application	K2	Time spent trying to make work-arounds to complete application	None	L	H	Read into the language to confirm it is suitable for the application
5	Language doesn't work with other parts of the application	I3	time spent looking for a solution	possible only if solution is found	M	M	check documents for possible conflicts/shared libraries
6	wrong configuration	E1	lot of time used to solve the problem	change programming language	L	M	Find a language that produces less mistakes
7	no previous experience with the selected programming language	K1	A lot of time spent on trying to figure out and learn new language	try and fail until you make it. Ask someone for help. Google it	M	H	More training needed
7	misunderstanding of application functionality(programming language lacks critical functionality)	R2	A lot of time will be spent on developing a non-runnable app	must start over, and possibly reuse code in new project	L	H	Further discussion with whomever provided the requirement specification
8	Too little knowledge about other programming languages that may be more suitable	K1	The chosen language may be or are about to be outdated, the support may be poor	reimplement in a new language	M	M	More people, with different experience should contribute when planning
9	Wrong language selected	S3	Long time used to code, when it could be done more easily on another language	Seek help/tips from an other developer	L	M	More knowledge about what is to be done before language is selected
10	Developers not trained in language	K1	Time spent learning the language	Need help from trained language	L	M	More training needed
10	Language is not fast or extensive enough for application	K1	Slow or unnecessary complicated code	switch programming language	L	H	More knowledge about prog. languages, Process of choosing language need to more extensive
11	choosing based on personal preferences and not relevancy. making wrong choice	S3	Changing language half way, huge time waste	None	L	H	Spend the time needed to pick the right language for the task
12	chosen language was not suited for the application	S3	Depending on time since the choice, it may lead to severe consequences, such as needing to rewrite the entire program	Rewrite code, or make it work	L	H	Use more time on selecting language, and understand the applications requirement
13	Language not supporting application requirements	S3	a lot of time used to find-/create a "fix" to make it run	A workaround could be available	L	H	Better knowledge on different programming languages. Better knowledge on what the application requires

13	programmers does not posses the knowledge of the prog. language	K1	a lot of time used to learn programmers the language	skilled programmers that can quickly teach the unskilled	M	M	Use a language that is already supported by the programmers. Teach language to programmers before project start
14	Choose wrong language	S3	too technical solution	None	L	H	analyze the applications technical requirement carefully, familiarize oneself with the languages possibilities and limitations
14	Lack of knowledge of the selected language	K1	Poor code, poor application, hard to maintain	need help from other developers, may need to rewrite loads of code	M	H	Careful planning, make sure that everyone got enough and correct training
15	Not able to find an appropriate language that satisfies the needed requirements of the application	R1	the application requirements to functions needs to be changed	None	L	H	a more realistic approach to what is possible when developing applications
15	No experience with the language that satisfied all needed reqs of the application	K1	not able to fulfill these requirements	get help from a more experienced developer	L	M	More experience and knowledge
16	The selected progr. language is new for the developers	K1	Lot of time used to learn it	get help from a more experienced developer, or read about the language online	M	M	More training/ instructions needed
16	New upgrade of programming language	E2	Spend time on learning new functionality of the prog. language	Read more, ask for help from somebody who knows it better	L	L	Pay attention to new versions, and their new functionality
17	Not sufficient knowledge about language/paradigm	K1	time spent on learning	Coursing of developers	M	M	Choose language/paradigm based on the teams knowledge
17	Developers believe they have more knowledge than they have, leads development in wrong direction	K2	hard to develop new functionality	consider to discard code, and start over again with new and improved wisdom	L	H	Use time, let more people consider in the process of selecting technology
18	language chosen is unsuited	S3	time wasted on wrong language	change language	L	H	More training
19	Can't choose a language	K1	Use too much time on selecting programming language	Need help from different developer	M	M	More training
20	chosen language was not suited for the application	S3	Use unnecessary long time and/or can't finish in time	None	L	M	More research
20	Do not make a choice	S1	Code does not fit, which causes a mess	None	L	H	Improved process that ensures a good plan
21	Lack of knowledge about the prog. languages functionality	K1	lot of time spent on learning the language. the probability of that an unsuitable language is selected is high	help from someone with more experience	H	M	More experience
21	Unsuitable language is selected	S3	cumbersome solutions, slow system	change language, look for workarounds with the selected language	L	H	More experience
22	Lacking knowledge in suitable language	K1	Unnecessary time used to development	Consult other developers	M	L	Gain more experience
23	Choosing a language no one knows	K1/T2	Time spent learning the language	Read Docs	L	L	More training
23	Choose unsuitable language for the task	K1	Not being able to fulfill the task	Change language	M	M	Read the spec

25	Get stuck on technicality	K2	Some time spent reading up on it	Need help from others	L	L	More honestly about real vs. imagined knowledge
25	Do not know the language well	K1	Lots of time spent reading up on it	Help from others. Find good examples	L	M	More careful selection of language to use and hours to train on using it
25	wrong language selected	S2	difficulty implementing	accept incomplete or bad solution			Attempt to rethink choice of language
26	Not ideal language chosen because of lack of knowledge	K1	More time used to implement	None	M	M	Deeper knowledge about several different languages
27	No suitable languages found	K1	development never starts	keep looking	L	H	Better research needed
27	Made a bad choice	S3	Application not successfully developed	Choose another language and start over	L	H	Better research needed
27	Disagreement in what language to use		Argument delayed developments	Someone make the choice	M	M	someone to make the choice
28	Little/no experience with preferable language	K1	Time spent learning the language	Need help from coworker	L	M	Gain experience in different programming languages
30	chosen language was not suited for the application	S2	Worse implementation (takes longer time)	switch programming language	L	M	More training, knowledge in the different languages
31	The prog. language does not support necessary functionality	E1	Extra time spent, app will not get desired properties	Use another language or sort it out in another way	M	M	Need more investigation and planning
32	Bad choice of language	S2	Less effective than other languages	Discard code, and rewrite it in a more suitable language	L	M	More experience with choosing language that fits your task
32	Do not pick Python	S2	Less effective than with Python	Rewrite code to Python	L	H	Always use Python
33	Wrong language selected	S2	A lot of time spent trying to make it work	Choose another language and start over	L	M	Get more knowledge about the language before development starts
34	No suitable languages found	S1	application can not be developed	none	L	H	change requirements
34	Unsuitable language selected	S2	time used on facilitation and things that don't work	change language	L	H	do research
35	Not familiar with suitable language	K1	Do not possess enough skills	Learn more languages	L	M	Make sure developers know enough languages on a general basis
35	selected language was unsuited	K2 I1	Does not work/ hard	change language	L	H	Do more thorough work when selection language
36	not able to program in the selected language at a sufficient level	K1	slow progress	need help from another developer	L	M	more training required
37	Wrong/ bad choice	S2	Wasted time	start development over	L	M	Do a pre-study
37	"Fanboyism"	K2	Wasted time	start development over	L	L	Do a pre-study
38	Wrong choice	R1	Time lost	Need to change language	L	M	Learn more about language
39	Lacking syntactic knowledge	K1	Time wasted on learning syntax	Study chosen language	M	M	More training
39	Ineffective language for a specific task	S3	More time used than necessary	Restart project with different language	L	H	Gain more experience in different programming languages
40	Language doesn't support right functionality	E1	Change language, rewrite code	Need help to make a right decision	L	H	More experience needed
40	language is outdated	E1	Change language, rewrite code	Need help to make a right decision	L	H	More experience

41	Lack of experience with most suited language	K1	Not able to make the program	Get help from another/ internet	M	H	Need to learn the language

B.5 Set up development environment

Table B.6: Set up development environment

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Environment not working properly, configuration	E1	Takes a lot of time to correct it	None	M	H	Use these experiences next time
2	Loss of data/conflict	E2	Time for recovery	None	L	M	train employees in use
5	wrong version/ configuration	E1 S2 S3	IDE does not run/ work	recheck settings and/ or guides on internet	L	L	Pay more attention when downloading or installing new software
5	IDE does not work with chosen language	S3	Language non-compatible with IDE	Chose different IDE or a language that works with chosen IDE	L	M	Pay more attention when downloading and reading about
7	Development environment different on different operating systems	S3	Trouble when merging code	swap dev. environment	M	H	select more familiar development environment/ have same OS
7	Lack of experience with development environment	K1	Trouble when merging code	train more	M	M	select more familiar development environment
8	configured differently in the production environment than test environment	E1	Code that works in dev, does not work in production	Code needs to be rewritten and environment needs to be set up correctly	M	H	Make sure all environments are the same before starting development.
9	Bad support for the selected language	S3	a lot of time used on finding errors and so on	Change dev. environment to one more appropriate for the selected language	L	M	More knowledge about language before selecting environment
10	Correct setup takes more time than assumed	T2	More work	None	M	L	use proper roll-out of IDE so that each developer does not have to set up IDE individually.
10	Problems with IDE setup	E1	version control, system etc. not working properly	Reinstall IDE etc.	L	L	use proper roll-out of IDE so that each developer does not have to set up IDE individually.
11	Not understanding the setup process	K1	waste a lot of time	Ask for help from another development	M	L	Make sure everyone who don't have experience with the technology have the support they need
12	configured wrongly	E1	Problems may occur	Reconfigure	M	L	Comes with experience and/or training
13	Programmers using different IDEs	E1	extra time on smaller configurations for each IDE	Adding minor fixes to different IDE	M	L	State preferred dev. environment before starting project
13	Programmers not familiar with the de. environment	K1	time used on teaching the developers	help from those skilled in the IDE	M	L	create guides for programmers to follow
14	wrong choice of IDE/compiler	S2	bruker mye tid	change IDE/compiler	M	L	more careful planning
14	development environment is not configured correct in all platforms	E1	loads of faults and problems when developers work together	reconfigure with same version for all developers	M	M	Agree upon versions, identify possible issues with the different platforms
15	All necessary programs are not supports on every OS	E1	Time used to solve problems	find alternative program/ change OS	L	M	Use environments that support most platforms

16	Bad atmosphere between the developers		The level of motivation and commitments decreases	Superiors needs to find a solution to the problems	L	L	
17	Not able to configure environment	K1	time spent, need help from other	get help from someone who knows the IDE	M	L	Let the developers who are familiar with IDEs, make a documentation containing problems that has occurred previously.
17	wrong version of software creates problems	E1	odd errors, things do not work properly	reinstall correct version of IDE	H	L	pay attention to the version number. Do not make major updates, if not everybody does it
18	D.E unsuited	S3	time wasted on setting up D.E	Set ut new D.E	L	M	Use generalized D.E
18	version control not working properly	E2	time wasted on making version control work	none	M	M	more training in version control
20	Code does not run properly	E1	New code can't be written	Correct the configuration	M	L	More training
20	No one knows how to set up the environment	K1	Code can't be run or written	Give training	L	L	More training
21	problems with identifying packages and so on	R1	takes time to configure	take the time needed, ask someone more experienced for help, Google it	M	L	find a good user manual, make bulleted lists to next time, a step by step guide
22	Does not get preferred development environment.	K1	Less effective writing and testing of code	Try to find alternative solutions	M	L	Get a clear overview of what is needed, and find something suitable.
22	Don't use the IDE for what it is worth	E1	Less effective writing and testing of code	Change settings	L	L	Get familiar with IDE
23	Unsupported OS	E1	Not being able to setup	Change OS	M	M	
25	Cant build	E1	Time spent	help from others, or start over	M	L	Find better instructions
25	No access to source code management	E1	cant work while solving	Grant access	L	L	Better control over team members
25	cant submit changes	E2	Team cant work	Copy changes manually	L	M	Better training
27	Make config errors in environment	E1	Dev environment will not work as intended	Fix config or set up dev environment again	M	M	Needs some training in setting up development environments
30	Do not have the right equipment/software	S3	Not able to solve the task in a good way	Get hold of necessary equipment, or do the most of what you have	L	M	Always have newest software versions available, in addition to equipment one might need
30	Errors on equipment or software	E1	Can not be used	Fix the error	L	H	Always check that the software/equipment works as it should beforehand
31	Wrong configuration	E1	Extra time	None	M	H	Plan
31	Not able to fix the problems	K1	Problem is not resolved	Need help	M	M	More practice and training
32	Bad choice of development software	S2	Less effective development	Install and use better software	M	M	More experience
32	Bad choice of version control system	S2	More problems than with good version control				
33	Bad configuration of D.E	E1	Things does not work as it should	Reconfigure D.E	M	M	More training in the environment selected for this project
34	incompabilities	none	time spent on configuration, while things doesn't work	change to different software	M	M	do research about familiar compatibilities
34	no knowledge about software	K1	wrong configuration	have someone redo it	M	L	proper training, courses
35	wrong program	E1 S3	doesn't work	try another program	L	L	better knowledge about the programs

36	lack of communication within the development environment	I1	slows progress down	management takes action	L	H	scrum meetings, etc.
37	takes longer time	T2	time wasted	none	M	M	pre-study
37	components doesn't work	E1	detailed development	find compatible solution	L	L	seek guidance from experienced developers
38	Wrong configuration	E1	Not able to run code	Change configuration	L	L	Gain more knowledge
39	Compiler wrongly configured	E1	Time used on finding solution/reconfigurement	Restart project, switch software	M	M	Gain more knowledge about what is suitable/-good software
40	Missing knowledge about configurations	K1 E1	Program doesn't run	seek help for configuration	L	M	Use more time in start-up phase
41	The D.E doesn't work on you OS	E1	Not able to start working	Get new OS	L	M	Make sure you have all the correct resources in advance

B.6 Choose architectural pattern

Table B.7: Choose architectural pattern

I ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Inappropriate pattern chosen	S2	Not working	Extra work: details or rewrite	L	H	Choose based on more experienc
3	Different patterns used by different programmers	I1	Hard to get an overview and change each others code	Start over and use the same patterns	M	L	Decide before coding, and let everyone know how it works
4	Wrong selection made due to not understanding the task	S2	Writing the code will get more difficult	Make new selection based on new information	M	M	Spend time on analysing the architechture needed prior to selection
5	Model is not updated upon change of view or opposite	I1	nothing happens, changes are not updated	implement methods that check for changes and update	L	M	Always include an update method that fores upon changes
5	wrong architechtual pattern chosen for assignment	K1	Does not work as intended/ at all	Ask another programmer for help/ search for help	M	M	Keep to something you know, read more about the certain architechtural pattern
6	wrong check on objects	C4 C6	wrong arcitechture	build a new architecture	M	H	
6	wrong information about relations in the code	R2 I1	time used to find new and correc relations	find the correct relations between objects			
7	lack of experience with pattern	K1	Mistakes may be done, time spent on it	train for pattern	M	M	Choose a more familiar pattern
8	Users lacks knowledge about the architecture	K1	code is not written according to architecture, leading to a destroyed structure. May lead to messy code	Code needs to be rewritten and more tedious	M	M	Give good training in the architecture that is to be used
9	Misuse of selected pattern	K1	hard for other developers to see and understand what has been coded	None	M	M	Seek knowledge about the pattern before starting

10	Developers are not familiar with pattern	K1	Slow development, incosistent use of pattern or code more prone to bugs	Need help for another developer or course in pattern	L	M	More training needed with different patterns
10	Pattern not suited for application	K1	code not working or unneseca	Rewrite code with better suited pattern	L	H	Better knowledge of different patterns and a more thorough process of choosing pattern
11	Picking an architectural pattern thats incomplete with the development framework	S2	have to redo the architectural design. time wasted	change development framework	M	L	know the ins and outs of the pattern and frameworks you are using
13	Choosing an uncommon pattern	S2	A lot of time used on explaining how it works	help from programmers skilled in the certain pattern	M	M	Give explanation for the choice of pattern with a guide on how to use it
14	wrong architechtual pattern chosen	S2	time wasted	change architecture	L	H	developers in the firm should know different patterns, make time for preporatory work
16	incorrect setup of pattern	K1	Use time to write the pattern correctly	Needs to learn how the pattern works	L	M	More experience needed
16	Wrong selection of pattern	K1	time spent on finding a more suitable pattern	Discuss with other developers on what pattern to use	L	L	More experience needed
17	Hard to implement pattern in the technology	S3	Poor solutions to get the solution required	change patter, technology, or alt leas learn from your mistakes	M	H	Do thourough research before staring
17	Pattern does not satifiy demands	S2					
18	A.P chosen not ideal for project	K1	time wasted on wrong A.P.	need to restructure code	L	M	more training
18	Developers are not familiar with A.P.	K1	time wasted on not following A.P.	need to restructure code	L	M	more training
20	Unsuitable choice of architecture	S2	Increased use of resources and demotivating work	Early in development: Choose different architecture. Otherwise: None	M	M	Better planning
20	Does not pick any specific architecture	S1	Messy code	Choose an architecture and refactor code	L	M	Better planning
21	Uncertainties about what is the most suitable selection	K1	an unsuitbale A.P. is selecected	None	L	H	gain more experience with different architecture
21	Lack of experience in patterns	K1	architecture is unsuiatable, copy pasting of code/code doesnt work as wanted	Try to fix it, make it right	M	M	learn relevant architecure more thouroughly
22	Choose an inpractical pattern	K1	Hard to develop as first planned	Work around problems and find solutions.	L	M	More experience
23	Unsuited pattern	K1	Poorly designed software	Change architecture	M	M	Learn more about design patterns
25	Wrong choice	S2	Slow progress	Work longer	L	H	change arch. pattern
25	Developers not familiar with pattern	K1	time spent reading	help from other, like courses	M	L	more training up front
28	Architecture does not support expected tasks	S3	(A lot of) time used finding preferred architecture	Help from developers that have worked with similar problems	L	H	Gain more insight in how different architectures differ from one another
27	Choose pattern not suited for chosen language	S2	App will be harder to develop	Chagne apttern	L	M	More research

30	Choose a pattern that are not suitable for the application	K1	Worse implementation, and longer time spent implementing	Switch pattern or make the best out of what you have	M	M	Better knowledge. More training in different patterns
31	Faults in the architecture	E1	Extra time spent	Redo the architecture	L	M	Plan and do it carefully
31	Use the pattern incorrect	K1	Get load of error messages	Read the error messages and fix it	M	H	More experience
32	Bad choice of achitecture	S2	Less effective development	Rewrite code to follow better architecture	M	M	More experience with which architecture that fits which task.
32	Lack of knowledge in different architectures	K1	Bad architecture	None	M	M	Gain more experience and knowledge within different types of architectures
33	Wrong pattern selected	S2	Messy code, lot ot unecessary writing	Change pattern, to a more suitable	L	M	Gain more knowledge about the different patterns
33	PAttern does not work as it should	K1	uneffective program		L	M	Training
35	the architecture doesnt match the functionality	S2 I2	need a lot of work to make it work	change architecture	L	H	better planning
36	chooses a subpar architectural pattern	S3	slow down progress, creates a weaker result	choose again	L	M	Have sufficient information regarding the different architectural patterns
36	all developers are not experienced with this pattern	K1	slows down progress	Training by experienced developers	L	M	Additional training
37	unsuited choice	S2	wasted time	re-visit decision	L	M	proper planning
38	wrong choice	R1	Time lost	Need to be changed later on	L	L	Get more knowledge
39	Architecture does not cover requirements as specified	K1	Don't cover the requirement specification for this project	Make a solution that covers it as good as possible	M	M	Gain more insight into the task
40	chosen architecture doesnt meet all requirements	K1	change architecture, rewrite code	Help with making a new choice	L	H	Time spent on planning, check requirements thourogh
41	Personel lack experience, not able to make the choice	K1	might choose unsuited pattern	Get external help	M	M	Learn

B.7 Identify problems/uncertainties in requirements

Table B.8: Identify problems/uncertainties in requirements

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Not identifying all requirements	R1	Important work left out	Implement requirements	M	M	Make sure employees know their requirements
2	customer not certain of requirements	R3					
3	misunderstand the requirement	R2	do unnecessary coding, do not meet the requirement	clarify the misunderstanding and fix the code. ask the customer	M	H	Read thorough through reqs, and discuss them in groups
3	overcomplicate requirements	K2	unnecessary coding	go through the reqs again	M	M	Do not overthink, Gain knowledge on language to find the easiest way

4	Unable to identify problems in requirements	R3	The code will not work as intended	Extensive testing and error-detection	H	H	Focus on requirements and scenarios to higher probability to identify problems
5	requirements not understood	I3 K2	something completely different is produced	ask problem giver if there are uncertainties	L	H	make sure you know what they want before starting to make it
5	requirements are unable/ unreasonable to comply to	K1	Product does not function/ requirements do not give working product	remove/ reconfigure some requirements	L	H	Know what can and will work
7	uncertainties not discovered	R3	Problems arise later in the project	Discuss thoroughly all aspects of the requirements	L	M	Discuss beforehand
7	uncertainties misunderstood	R1	mismatch in development within team	Discuss thoroughly all aspects of the requirements	M	M	Discuss beforehand
7	uncertainties viewed as problem by some of the team	R2	some of the team spends a lot of time trying to solve the problem	Discuss thoroughly all aspects of the requirements	M	M	Discuss beforehand
8	The makers of the reqs are non-technical and makes unrealistic/ hard reqs	K1	Reqs must be removed or development will exceed the time estimated	reqs are removed	H	H	Always have technical personnel on the teams that makes the reqs
9	No problems found	K1	Product has severe deficiencies	None	L	H	Constant control check ups of requirements. Good communication with customer. Keep what the customer want in focus
10	Problem/uncertainty not recognized	R3	Problems are noticed in a later stage of development	Fix problem	L	H	More thorough check of requirements
11	Not seeing	R1			L	M	
12	could not solve problem	K1	can not continue without fully understanding the requirements or solving the problem	Get help with a problem from another developer, or talk with the customer about the requirements	M	L	Better line of communication with the customer. Unsolvable problem: needs more training
13	not able to identify all requirements	K1	project slows down when req cannot be met	adding req to program	H	M	
15	Flaws in requirements are not identified	R1	Time used	None	L	L	Be more careful
16	Incorrect interpretation of requirement	T2	Time spent on rereading the requirements	Ask customer/or fellow workers to clarify the requirements	H	M	Better communication between customer and developers
16	contradictory requirements	I1	Time spent on interpreting the requirements	Ask customer/or fellow workers to clarify the requirements	M	M	Better communication between customer and developers
17	Unrealistic requirements	T1/T3	Not able to finish the solution	go through reqs with customer, or get more time	M	H	time spent on requirements analysis
17	Not able to identity requirement(which should be identified)	K1 R1/R2	Not able to finish the solution, a lot of stress	go through reqs with customer, or get more time	M	H	Choose development models which identifies these problems, gain more competence
18	problem not identified until implementation	none	time spent communicating with customer regarding requirements	modify requirements	H	L	none
18	mistake in requirements	I2	time spent communicating with customer regarding requirements	modify requirements	L	L	Better communication with customer
19	Can't find problems or uncertainties	C2	Becomes a problem later	Need to fix the problem later	L	M	Either more training, or better requirement specification from customer

20	Can't find problems or uncertainties	K1/I3	Delivered product that does not work as expected	Implement missing functionality	H	M	Use better time on analyzing requirements
20	Requirements are misinterpreted	I3	Delivered product that does not work as expected	Implement missing functionality	M	M	Better communication with customer. Allocate more resources to achieve this, and include customer during the project
21	customer has undefined requirements	R3	Hard to no what solutions to implement	ask customer for more details, show examples, make a paper prototype	H	L	
21	Customer misunderstood	R2	Code implemented in unwanted way, customer does not approve	ask customer for more details, show examples, make a paper prototype	H	M	
22	Interpret requirements wrong	K2	Implement wrong requirements	Find the meaning of the requirement and correct it	L	M	Ask customer/boss if there are any uncertainties.
23	Misinterpret requirements	I1	Must reimplement features		M	M	Read the specification carefully
25	fail to find big risk	C2	potentially breakdown	None	L	H	Better risk analysis
25	risk wrongly assessed	C4	time spent recovering	Longer working day	L	M	Better risk analysis
26	Wrong assumptions made on unclear requirements	K2	End product does not comply with requirements	Parts of the code has to be rewritten	L	M	More people can read through requirements, in order for every one to understand. Unclear requirements can be discussed with customer/rest of development team
28	Misunderstand the requirements	K2	Extra work providing functionality that fulfills requirements	More work	L	M	Thorough dialogue with the customer about expectations and requirements
27	Miss problem/uncertainty	R2	Problem will be embedded in app	Fix problem	L	M	Have more people check the requirements
30	Misinterpret requirements	I2	Result does not become as wanted	Ask customer to explain	M	H	Requirements should be more clear and precise beforehand Better communication.
30	Don't get enough information about the task	I3	Result does not become as wanted		L	H	Always get the information received confirmed with customer, and improve communication
31	Not able to locate the problem	K1	Not able to solve the problems	Need help from another developer	M	M	Practice
31	Know what's wrong, but it takes a long time to fix the error	T2	More time used than estimated	None	H	H	Careful planning
32	Uncertainties regarding implementation details	R1	Some functionality won't get its development started	Contact the customer and ask for clarification	H	L	Make sure that the requirements are clear from the beginning
33	Neglect problem		Not able to develop the program as it should have been done		M	H	Read reqs more carefully, and several times
34	problem underestimated	K2	underestimated timing	get more man power	M	M	spend time analyzing requirements
35	problems/uncertainties identified	I2	not able to/ hard do satisfy requirements	clarify requirements with customer	M	M	ensure the quality of the requirements
36	Created solutions the program didn't need	T3	wastes time		M	M	Communication with the company that ordered the program
37	wrong assumptions	I2	not the desired product	none	L	M	Good dialogue with customer
38	Identify errors too late	K1	run into problems in implementation	Change requirements	M	M	Check requirements thoroughly
39	One of the requirements are not good enough described	R3	Requirement can be implemented wrongly	Find new solution for the requirement			Better communication with customer regarding the requirement specification

40	ambiguities are not identified	R2	Customer doesn't get what they want	Overtime on the developers	L	H	Better communication with customer
41	Not able to contact customer/ or other people writing requirements	I1	there's are risk of the program not supporting any requirements	None	L	H	Make sure to establish good connections with customer

B.8 Define goals from the requirements

Table B.9: Define goals from the requirements

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Too ambiguous goals	T3	Project not finished	More time	H	M	less ambitions and/or more time
2	not defining all goals	K1	Project not finished	More time	H	M	make sure employees knows the reqs
3	Define too few goals	T2	bad overview of remaining work	define the goals in minor goals	M	M	Keep a minimum amount of goals, and an overview of time planned to use in each goal
3	Add unimportant goals	T4/T1	The time used should rather be used on important goals	work after priority	M	L	Prioritize the goals
4	Define wrong goals based on insufficient understanding of product requirements	R3	Not able to develop product specified	start defining goals al over later in the process	L	H	Spend time on understanding the requirements
5	Unable to reach goals	T1	goals not reached	get more help/make smaller goals	M	M	Experience make time management easier
6	overestimated workload	T3	extra work to complete the reqs	only used estimated time when performing the task	L	M	
7	Misunderstanding of requirements	R3	Discussion, time spent	Ask customer	M	L	Have discussion of req. before development starts
7	Disagreement of high priority req	K2	time spent on discussion	Ask customer	M	L	have discussion of req. before development starts
9	Poorly defined reqs	T2	Bad goals in accordance with time estimation	Needs more developers to finish on time	M	M	More experience with requirements, and more and better communication with customer
10	Not all required goals defined	R3	Missing functionality discovered in a later stage of development	Implement missing functionality	L	H	Better process for defining goals from requirements
12	The goals does not match the customers requirement	R2	Wrongly defined goals. May lead to unwanted program	redefine. If a lot of time was spent before noticing/being notified, then redoing work as a possibility	L	M-H	Meet the customer before every major step in the project
13	choosing wrong goal	S2	product not the way customer wants	talk to customer, create new version	M	H	Keep good connection to customer through the entire development process
14	goals are defined in wrong order	K1					
14	unattainable goals						

15	Poorly defined reqs	I3	Goals not defined	clarification in requirements	L	M	identify flaws in requirements before defining goals
16	Goals not specific enough	R1 I3	The end-product may not meet customer demands	Contact customer for more specific goals	M	M	Better communication between customer and developer
16	Unrealistic goals (from customer)	I2	Not able to deliver according to customers need	Need to inform customer about unrealistic goals, and agree upon something else	L	M	More experience (for customer) needed, and better communication between developers and customers
17	Misunderstanding of requirements, leading to wrong goals	K2	Unsatisfied customer/ bad result	start over	M	H	Use time. Use a reasonable development model
18	requirements misunderstood	I3	goals not fulfilling requirements	communication to clarify reqs/goals	L	M	make requirements very clear
18	goals doesn't solve problems in practice	K1	project not fulfilling requirements	set new goals	M	M	make requirements very clear
19	Too ambitious goals	T2	Can not finish in time	None	M	H	Need more training
20	Requirements misinterpreted	I3	Product does not work as expected	Implement missing functions	H	M	Better communication between customer and developer
20	No goals found	I1	Remaining development can not start	Talk to customer, identify goals	L	L	None
21	Too ambitious goals/ lacks details		details omitted during development, bad overview of progress	divide into smaller goals	M	L	
21	too short deadlines for each goal	T2	project more expensive than estimated, lack of effort on each task	None	H	M	more experience in time management
21	too much time is estimated on each goal	T3	project seems too expensive, loosing tender	None	H	M	more experience in time management
22	Set up fast schedule	T2	Can't deliver at the times estimated, or work overtime	Use more time than planned	H	L	Get more experience with how long time different tasks take
23	Define wrong goals	T2	Time wasted	Redefine goals	M	M	
25	Requirements incomplete	I3	Restructure code	None	H	L	More conversations with product owner
25	Requirement overlooked	R2	Time spent implementing	None	L	M	More thorough examination of requirements
28	Wrong time estimate	T2	Work plan need to be revised	Extra work, if this is an option	M	H	More time spent researching the project, talking with the customer and gaining an overall good overview of the size of the projects
30	Too ambitious goals	T2	Goals are not reached (in time)	Use more time than planned	H	H	Be realistic when defining goals. Include margins
30	Goals with lack of ambition	T3	Could have reached a better result	Define more ambitious goals during the project	L	M	Be realistic when defining goals. Open up for more work during the project
30	Goals too ambiguous	K1	Hard to measure progress	Adjust goals during the development.	L	L	Define goals more accurately.
31	Underestimate workload	T2	Not able to accomplish the goals	None	H	H	More planning
31	Over rated knowledge	K2	Need more time, more errors	Get help from someone with more knowledge	M	M	More learning
32	Define wrong goals from requirements	I2	The application gets the wrong functionality	Define goals correctly.	L	H	Gain more experience with defining goals
33	Reqs are misunderstood		system is incorrect	change the code	L	H	talk to customer, communication

33	Make too few goals	T2	deficient program	add more goals when you become aware of the situation	L	M	Read reqs thoroughly
34	Goals not specific enough		they don't help, unable to work towards them	remove goals	M	L	Have a template for goals
35	Goals lack ambition	K1	Requirements are not met	Redefine goals	M	M	More thorough work with quality assurance of goals
36	Misunderstands the requirements	I2	program becomes incomplete	communicate with company	L	M	good communication with the customer
37	unclear goals	I3		redefine goals	L	L	
37	missing goals	I1					
38	wrong goals	R2/I2	Time lost	Change goals	L	M	Read requirements thoroughly
39	Don't understand goals	K1	Wrong implementation	Clarify goals with customer	M	H	Clarify the meaning of the requirements with customer
40	Not able to define milestones from requirements	K1	the development is not goal oriented	get help from an experienced developer	L	M	More training in getting clear goals
41	Choose unrealistic goal	R2	Not able to finish all goals	redefine goals during project	M	H	more experience

B.9 Develop mockup/ prototype of solution

TABLE B.10: Develop mockup/ prototype of solution

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Insufficient functionality	K1	Bad prototype- bad results	More adjustments ASAP	L	M	train employees/ hire better employees
2	Too advanced model	T2	Unimplementable / ununderstandable	take it down a few notches. redesign	M	M	Be more self-oriented
3	Unsatisfied customer	I3	need to make new prototype	gain information about what the customer dislikes, and fix it	M	M	Keep good communication with customer
4	Not able to develop a prototype within the time available	T2	Will need to finish the prototype later than intended	None	M	M	Try to overestimate the workload of individual work packages slightly
5	Customer unable to understand prototype	I1	customer is lost	KISS- keep it simple stupid	M	M	experience, make sure it is at customer level, preschool/rocket-scientist
5	Prototype has faults	C1	Prototype doesn't work	Test prototype before showing customer	L	M	Remove features with major bugs to keep prototype functioning in the future
6	Information not communicated	I1	the prototype is wrong		M	H	Keep good communication with customer to avoid misunderstanding
6	Information not communicated	R1	use time to get info		L	M	Keep good communication with customer to avoid misunderstanding
7	Mockup does not provide a realistic image of what the app can do	I2	Time spent	Mockup must be redone	M	M	Discussion before mockup is developed. clarification of which reqs are high priority and available technology

7	Lack experience to perform functionality	K1	Time spent	use time to train	M	L	Google it
8	The GUI looks finished, leading the customer to think the project is almost complete, while loads of work on backend remains	I1	customer thinks development is almost complete		H	H	Hard to explain the difference between front- and back-end. If possible try to get in contact with a person with experience from the customer
9	Messy and bad mockup that not meets customers expectations	I3	More resources due to a rebuild of mockup to meet customers expectation	Well-defined goals	H	M	More communication with customer. More frequently contact could lead to smaller adjustments at time, and mistakes/misunderstandings would be fixed at an early stage
10	Mockup far from satisfactory to customer	I2	customer not happy	Extensive dialogue with customer to fix or redevelop mockup to his needs	L	H	Better communication with customer before and during development
11	mockup look too finished	I2	customer less likely to criticize something that looks completed	None	L	M	Make sure the mockups UI looks unfinished without sacrifying usability
11	mockup doesn't reflect requirements	I2	you get feedback on how the prototype differs, not how to make it better		L	H	stick to the requirements
12	Customer not satisfied		Must redo the mockup and reconsider solution	None	M	M	The customer may have unreasonable expectations. not much to be done
13	time consuming	T2	time spent making prototype only worth it if customer approves	none	M	M	good connection to customer and skilled programmers
14	Prototype is not feasible	K2	Wasted time and work	Make new/ or change the old one	L	H	
14	prototype does not cover requirements		unsatisfied customer	add missing functionality	M	M	ensure the quality better
14	prototype is not intuitive enough						
15	Not able to develop a prototype within the time available	T2	prototype delayed	need to work overtime, or use more developers	L	M	Better estimation of workload
16	mockup does not meet customers expectations	R2	Time spent on creating a new prototype	Need to discuss with customer what they expect, what is a good prototype?	M	M	Better communication between customer and developers. Walk through the rq with the customer
16	Not able to develop a prototype within the time available	T2	not able to deliver what the customer asked for/ get fired	Need to explain the situation, and ask for more time	M	H	Better communication between customer and developers. Better planning
17	The customer is misunderstood, or customer is bad at explaining what they want	I1 I2	Unsatisfied customer, need to use more time	new evaluation process, repeat	M	M	
18	Customer not happy with result	S2	time wasted	rewrite project behavior	L	H	Better communication with customer
18	prototype not ready in time	T1/T2	project delayed, more expensive	plan more, plan for other delays	H	H	better planning, but more importantly: plan for T1/T2 happening, no matter how well planned the project is
19	Lacks highly prioritized requirements	I1	Develop new prototype	Talk to customer	M	L	Better specification from customer
19	Error in solution	I2	Develop new prototype	Talk to customer	M	L	Better specification from customer

20	Prototyping takes too much time	T2	Project budget is overrun, and too late delivery	Drop functionality	L	L	Make the prototypes less comprehensive
20	Customer believes the product is almost finished	I2	Customer not happy and not able to understand. Lack of trust between customer and developers.	None	M	M	Make sure the prototypes don't look finished when showed to a customer
21	Prototype has functionality that is hard to implement	K2	The end product will differ from prototype	None	M	M	Make sure there is a plan on how to develop the functionality before presenting the prototype to the customer. Know you have the resources you need
22	Use too much time	T2	Give customer a prototype late, use too much time on a prototype.	Make the prototype simpler, by putting less functionality in to it.	M	L	Find a balance as for how much time one may put into the prototype compared to further development.
23	Make a prototype that does not work	C2	Angry customer	Fix it.	M	H	Spend more time testing
25	Prototype wont run	E1	Time spent	None	L	L	Better testing and planning
25	Not finished overtime	T2	Need extra time	None	L	M	Better planning and estimation
26	Prototype is nicer than necessary	T1	Too much time spent on a task that gives little in return	None	L	M	Make it clear as for what each iteration of the prototype should test
27	Task too time consuming	T2	Will not complete in time or have to work extra	Have more people do the task on work extra hours	M	H	New process of early app development
28	Customer does not approve of mockup/prototype	E1	Mockup/prototype need to be revised	Ask customer what should be changed	M	L	None. Customers needs are hard to estimate
31	Underestimate the time it takes to create the mockup	T1	Not able to finish on time	Work faster/ deliver an unfinished product	H	M	Not spend unnecessary time on things, more planning
31	Unsatisfied customer	E1	Everything/ some things need to be redone	Try to convince the customer that this is the best solution	H	M	Listen to the customer
32	Develop bad prototype	K1	Customer not happy.	Develop better prototype	L	M	Gain more experience with creating better prototypes
33	Make a poor prototype	R2	Do not get good feedback from customer	Make new mockup	M	M	Let more people join the prototype development process
34	mockup based on wrong requirements	R2	it becomes useless	redo it	L	H	proper communication between customer and team
35	prototype does not cover requirements	R2	a lot of wasted work	start over from an early state of the project	M	H	Quality assurance of prototype along the project, so that we are sure that the project is moving in the right direction
36	its not what the customer wanted	I2	prototype needs to be reamade	change the prototype	M	M	Good communication with customer
37	faster than expected	T3	down-time	reschedule	M	L	
38	too little time	T1	Not completed	get more time	M	M	better time estimate
39	Requirement interpreted wrongly	I3	Product contains wrong functionality	Start all over	M	H	Gain better understanding of requirements
40	prototype not ready in time	T2	Customer will not see the prototype	More work for all involved parts	L	M	Estimate more time
41	Prototype does not reflect a realistic image of the end-product	I2	Customer not able to give useful feedback	None	L	H	Make sure the prototype reflect the goals

B.10 Review codes behaviour: place breakpoints

TABLE B.11: Review codes behaviour: place breakpoints

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Inappropriate placement of breakpoints	K1	Test not working	Place again	M	L	Think more before placing
3	breakpoints placed badly	C2	loose some of the code while testing, or get faults on right code		L	L	Check frequently and test several times
6	place breakpoints at places were the code runs as it should	S2	use to much time on debugging, not finding the real errors		M	M	
7	Breakpoints wrongly placed	C3	compilation error	undo placement, try again	M	L	Indentation
8	Code badly documented, hard to get what is going on	I2	takes a lot of time	Documentation of code, and good names on variables and functionality.	H	H	Make it a business strategy to document, and keep a standard for class-, function- and variable-names
9	Not able to find expected behavior	R1	A lot of time spent looking for wanted behavior	Ask another developer for help	M	L	Better knowledge on placement of breakpoints, and better knowledge on the code that is reviewed
11	place breakpoints the wrong location	K1	don't see what you need to see	ask for help	L	M	Make sure anyone with little experience with bug testing get the support they need
12	Did not find any problems	K2	will lead to bad functionality	ask for help	M	M	understand the application better/ more training
13	programmer does not understand how code works. look at unimportant breakpoints	K1	code not properly checked	have someone with the knowledge of the program redo the review	M	M	make sure the one reviewing has the required knowledge
15	Not able to place breakpoints	E1	time used to locate the error	ask a more experienced developer/ internet	L	L	Better introduction to the program
16	Not able to locate errors, place breakpoints wrongly	R1	time spent locating the error	Need help from another developer	L	L	More experience needed
17	placed incorrect	E1 K1	bad design/analyze process	repeat, with more	H	L	Learn from mistakes, get crew with experience
18	unable to understand behavior	K1	code review not finished	Need help from another developer	L	M	More training
20	Can't use the tool	K1	code review not finished	More training or other QA-method	L	L	More training in use of tool
20	Jumps past code and assume that it works	C2	Parts of the code will not be reviewed	Check remaining code	L	L	Change attitude towards code reviews
22	Put breakpoints at the wrong place	S1	Don't check what you want to	Switch breakpoint.	H	L	More training
22	Forget to remove breakpoints	S1	Code does not run as expected, harder to check as wanted	Remove unnecessary breakpoints.	H	L	More training
23	Place breakpoints at wrong places	K1	Non optimal debugging	Replace them	L	L	Be careful.
25	checking wrong part	C3/C6	Takes longer than necessary	help from others	L	L	More practice

25	Fixed error, but created another bug	C2	may holt development later	Pair programming	M	L	Code reviews
27	Unwanted/unexpected behavior found		Need to fix problem	Fix problem	H	L	Write better test/code
28	Breakpoint set too early	C3	Functionality is not properly tested	A new set of eyes should look on the code.	L	M	Good documentation and careful review of the code
30	Breakpoints on wrong position	C5	Parts of code will not be tested.		L	H	More experience, rather put too many than too few
30	Too few breakpoints	C5	Parts of code will not be tested.	Insert more breakpoints	M	H	More experience, rather put too many than too few
30	Too many breakpoints	C5	Debugging takes more time	Remove unnecessary breakpoints	L	L	More experience
31	Information is not communicated	R1	Needs to be fixed	Fix it	M	L	
31	Wrong info	R2	Results are not as expected	Find the error and solve it	H	M	Understand the task and the code properly
31	Info is not received	R3	results are not as expected	Find the error and solve it	H	M	Understand the task and the code properly
31	Not able to solve the problem	K1	Extra time spent	None	L	H	More training
32	Do not review own code properly.	C1	Miss out on errors that could have been avoided	Review code and write tests for all functionality	M	M	
33	Place breakpoint wrong	S2	Poor progress in development	change placement of breakpoints	L	M	Understand the tasks scope better
35	Wrong breakpoints	C5	Too much/ to little code is tested	move breakpoints	L	L	Be aware of what parts of the code are to be/ need to be tested
37	Developer does not know how/why	K1	not done properly. inefficient code	help from another developer	M	L	More experience
40	breakpoints placed badly	C3	Wrong code is checked	Get help to do better placements	M	M	More training in setting breakpoints
41	Does not understand old code	R1	Not able to review	rewrite code, use more time interpreting old code	M	M	Make sure to write modifiable code

B.11 Review codes behaviour: evaluate behavior

Table B.12: Review codes behaviour: evaluate behavior

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Not able to identify problems	K1	Time used to find problem	Ask more experienced programmers	L	M	Better training
3	Misunderstand the codes functions	C2-C6	write new code which behaves the same. delete code that works	discuss with the author of the real code	M	L	test several times, in different ways
4	Not testing enough scenarios	C2	Error might not be discovered	None	M	H	Do extensive testing based on product requirements
6	Wrong information obtained	R2		Change the code	L	L	None

7	cannot provide tests that checks that the behavior is correct	C2	Test fails	write new test	M	L	None
7	wrong check on right object	C4	test fails	find right check	M	L	None
7	right check on wrong object	C3/C4	Test fails	find right object	M	L	none
9	Wrong evaluation of behavior	C4	Strange behavior no one understands, because the impression is that the code is flawless	Evaluate again	L	M	More knowledge about code review. Ask other developers for help when needed. See your own limits
11	Insufficient understanding of information received	K1	Not able to fix issues	ask for help	M	M	Make sure anyone with little experience with bug testing get the support they need
13	Reviewer does not make correct documentation for code	R2	code not working according to correct documentation	review the entire code with correct documentation	L	M	Always keep correct and updated documentation in an accessible place
16	Code does not behave as expected	K1	Time spent to find out what is going wrong	Help from another developer	M	L	More experience needed
17	Incorrect debug process	R1	no new knowledge	Change strategy	M	L	New experience
18	behavior not as expected	I3	project not fulfilling requirements	modify code behavior	L	H	Better communication
20	Check that the code behave as thought, but not as required through the specification	K2	Code not working as it should. Components don't fit	Rewrite code. Read through requirements and plan	M	M	Make requirements understandable and easy to read. Change attitude.
22	Run wrong test	C4/C3	Does not get to check the wanted functionality	Modify / add test	H	L	Gain more experience
22	Forget to test parts of the functionality		Has untested and potentially wrong functionality in the code.	Add tests, try to cover as much as possible	H	M	Gain more experience, and test product before delivery.
21	Not enough heavy testing(number of users etc.)	R2	program doesn't work when strain is high	Launch a beta version and test it with many users	M	H	
21	The test works with test data, but data that is not tested can make the program crash	C2	Program doesn't work as intended	Launch a beta version and test it with many users	M	M	Problems may most likely be fixed quickly with a patch
23	Accept poor behavior	C2	Bad software	Reevaluate	M	M	Improve evaluation scheme
25	Code misunderstood	K2	Takes longer to incorporate	help from others	L	L	Better orientation and training
28	Program does not function properly	E1	Extra time spent trying to fix	Ask another developer	H	L	Be thorough and consistent when writing schemas and code.
30	Wrong behavior in code	C2	Have to reimplement parts of the code	Fix error	H	M	Gain more programming experience
30	Evaluation wrongly conducted	C3/C4/C6	Errors could be missed, found errors could be "wrong" (An error identified is not really an error)		L	H	Better routines on code evaluation
31	Code does not behave as expected	E1	Time spend on finding and fixing the error	None	M	H	Get the help you need in time, before the project suffers
35	behavior not as expected	R2	Faults in the code	Fix the errors	H	K	Code better
36	Bugs		Program runs poorly	Find what causes the bug	H	L	

36	Code doesn't work properly	K1	Problem not solved	Find logical flaws	L	M	Properly trained staff
37	Bugs missed	K1	Bugs	Patching	H	M	Peer reviewing of code
39	Implementation gives wrong functionality	K1	Modify code	Reserve more time to implemented code.	M	M	Better code understanding
39	Unknown/unwanted behavior of code	K1	Time spent on searching for bug	Find a developer who can help you	H	M	Use longer time on writing understandable code
40	Wrong behavior is expected from the code	C3	code is marked wrong	Investigate system tests better	L	M	Read documentation
41	Misinterpret behavior	R2	rewrite code that really does work	None	L	M	Make sure the test for behavior are good

B.12 Modification: identify new necessary functionality

Table B.13: Modification: identify new necessary functionality

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Wrong solutions to the problems	S2	Wasted time	Modify solutions	M	M	Choose better
2	Not finding the solutions	K1	wasted time	Ask for help			Get more experienced people to do this
3	Add unnecessary functions	I3/K2	unnecessary time spent		M	M	Discuss with others(customer or developers)
4	Unable to identify functionality the customer will need	I3	Product might not meet customers expectations	Obtain information on needed functionality from customer	M	M	keep a close dialogue with the customer throughout he development process
5	same task is repeated every time		pointless to have check/ task	task is omitted or automated for simplification	M	L	Check for redundant tasks
6	Overestimated	T3	project not complete at estimated time	None	L	M	Implement the most necessary reqs first
6	wrong selection of functionality	S2	add unnecessary functionality		M	M	Check reqs before deciding upon new functionality
7	functionality already implemented in another place in the app	I1	functionality might be added twice and unnecessary time spent	none, except deleting redundant code	M	L	Better communication within development team
7	customer does not agree	I3	functionality is dropped	none	M	L	None
7	functionality exceeds time scope	T1	functionality must be dropped	prop other tasks to do this functionality	M	L	Plan better use of time. Take new functionality into consideration
9	Functional deficiencies are not discovered	R3	Leads to faults/deficiencies in the end-product. Not satisfied customer	Communicate with customer	L	H	A well structured design-phase were requirements are well-defined and approved by customer
11	identifying a new functionality which is already supported	I1 K1	redundancy	talk to the other developers when making code decisions	L	L	Keep the code simple and well commented so its easy to understand what is happening
12	the new functionality takes too much time	T2	may not meet deadline	extension of deadline	M	H	Research the necessary functionality

13	Adding unnecessary functionality	I3	time wasted on creating functionality				
16	Pattern selected does not fit with new functionality	S3	Time spent on rewriting the pattern	Help from another developer	L	M	More experience needed
16	Wrong interpretation of functionality	I3					
17	Misinterpreted customer	I2	wrong requirements	repeat	M	M	
17	unfamiliar with the technology stack	K1	wrong reqs, strange requirements	let somebody else take over	L	H	Use time and work together
18	new functionality causes bugs	none	project buggy	fix bugs	H	L	none
18	No knowledge for new functionality	K1	functionality not added	need help from another developer	L	H	more training
19	Unnecessary functionality added	S2	Time wasted	None	M	L	Need more training
20	Can't find necessary functionality	R1	Necessary functionality not implemented.	Try again and implement missing functionality.	L	M	
23	Not being able to identify	S1	No new functionality	None	L	L	None
23	Identify too much	S1	Lots of work	Remove some requirements	M	L	Be more critical
25	Need not discovered	I3	Missing functionality	None	L	M	More rigorous checking of needs
31	The entire program needs changes	T1	Time spent doing all the changes	None	M	H	None
31	Do not know how to do what needs to be done	K1	Not able to perform the changes	Get help	L	M	More training
33	Add unsuited functionality	S3	Time spent on something that is not going to work	drop the functionality	M	M	Gain more knowledge about the technology used
33	Choose too many new functionalities	T2	Too little time to finish	skip certain functionalities	M	L	Better time management
34	not able to implement due to lack of experience	K1	cant be implemented, unhappy customer	bring in more expertise	M	H	do a good prestudy, so that it is easier to know what expertise is needed in the project
35	existing code does not support new functionality	E1	not able to implement new functionality	change existing code	M	M	be aware of the functionality from the start of project
36	Developers doesn't know how to implement new functionality	K1	cant solve problem	Help by another developer	L	M	More training
37	difficult to implement	E1	Extra work	substantial refactoring	M	H	have all the functionality planned in the beginning
38	wrong feature identified	R2	time lost	Identify right features	L	H	identify missing feature better
39	New requirement from customer	R3	Time used on changing product	Modify product	M	M	Keep "natural requirements" in the back of mind during development.
40	doesn't see the need for new functionality	K1	System lacks functionality, doesn't meet the requirements	run more tests	L	H	investigate requirements further
41	Identifies existing functionality as deficient	R2	Write same functionality twice	None	L	M	Keep an overview of what is implemented

B.13 Modification: draw connection between old and new functionality

TABLE B.14: Modification: draw connection between old and new functionality

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Wrong assumptions about new functionality	S3	wasted time/errors	change code	L	M	Think better
4	Not doing modification properly due to difficulties understanding the old code	R3	Modification will not work properly	Will need to analyze the old code when unexpected problems occur	M	M	Spend time analyzing the old code, talk to other developers if any
5	old code is ignored	C1	functions don't work, code not needed	if code is needed the problem must be solved, else code can be scrapped	M	L	Pay attention when coding
7	critical objects missing to do the connection	R3	connection cannot be drawn	Create critical objects	H	L	have good overview of code. Draw class diagrams
8	Old code is badly structured		Lot of work needed to do small changes	code needs to be rewritten, or do it possible to make a loose connection between old and new code	M	H	Make good standards for the developers to follow
9	Hard to find connection in the code/poorly connection	I1	Hard to see the changes that are made	None	L	M	Maintain a good version control of the code
11	Insufficient knowledge of code	K1	Redundancy and poor optimization	get someone else to take over	M	M	make sure every developer is aware of everything happening
11	messy code	R1	even more messy code				
12	100 new bugs		Application not working	Debug	H	M	Make sure the new functionality will not cause any problems
16	Poorly documentation of code	R1	Not sure were to make the change, time spent to figure it out	Need to contact developers of the old code	M	M	Keep a focus on documentation of code
17	Reach wrong consequences	K1 K2					
18	old incompatible with new	K1 K2	code not working together	modify old code	H	L	more training
19	Can't connect old code and new functionality	R3					
22	No good way to connect		Ugly code				
26	Old code misunderstood. New functionality uses old methods wrongfully	R2	New functionality does not work as it should, or ruins old functionality.	Old code should be reviewed and corrected if necessary.	L	M	Test old code before use. Review documentation
27	No connection found	R1	Developing new functionality will be difficult	None	M	M	Write more easily changeable code
28	Old and new code uses different platforms (are incompatible)	E1	Time spent trying to make the code compatible	Ask a developer with similar expertise	L	M	Check compabilities before writing code
30	Can't connect old code and new functionality		Can't add new functionality				
31	Old code and new code are not compatible	E2	Need changes, more time	Change old code	H	M	

32	Hard to read code you have never seen before or written a long time ago	K1	Waste of time	Read and understand the code again	H	M	Get better at documenting and commenting.
31	Communication doesn't work	I1,I2,I3	Need to spent time finding the error	None	M	H	Try to write simple code, that works with different functionality
33	New code in written in wrong language	S3 K1	Need to rewrite everything	change language	L	H	Do research before you begin
35	Not compatible	E1	not able to support new functionality	change the code	M	M	Be aware of what is to be done to support the new functionality
36	New functionality requires changes in old code	E2	New functionality does not work	update old code	M	L	Teamwork
37	not familiar with old code	K1	unable to perform task	read documentation	H	M	proper documentation
39	Underestimate workload needed to translate old code	T2	More time than expected is used	Reserve more time for the task	H	M	Gain more knowledge of old code and new functionality
40	Not able to see the connection	K1	incoherent code	get help from more experienced developers	M	H	Investigate the connection further
41	Conclude wrongly	K1	New functionality is not implemented	get help from developers who knows the old code, and new functionality better	L	M	Get to know old code and new functionality before you start

B.14 Create new functionality: code the changes

Table B.15: Data: Create new functionality

ID	Error Description	Error Mode	Consequence	Recovery	P	C	Remedied Strategy
2	Not able to code all changes	K1	Wasted time	Ask for help	L	H	Think better
3	New code overwrites old code		old functionality is destroyed, needs to be fixed		M	L	separate the functionality, use own/new variables
4	Not changing all necessary parts of code	R3	New functionality will not work as intended	solving problems due to the insufficient code changes	M	M	Test more aspects of the code than the ones directly connected to the change
5	application crashes	K1	application crashes	test as changes are implemented to make sure it works	M	H	Remember to test as small/ new code is added
6	wrong technology	S3	Technology does not exist	find new technology to code the changes	L	M	
7	Merge conflict between new code and project	E2	Merge conflict. Project does one compile	Rewrite previous working versions	H	L	pull and push often
9	Changes in code leads to new unforeseen faults	K2	More work/ which takes more time	Ask for help	H	M	Experience
10	New functionality not complete with old code		cannot implement functionality without changing old code	Rewrite old code	M	M	Write code with low coupling

11	Underestimated, time required	T2	wasted time	scrap new feature	H	M	Don't be naive when estimating time
12	changes not fit for the chosen language	E2	can not implement without spending time on the problem	None	L	M	Rethink the functionality
16	The old code is bad/ unsuitable pattern is used	K1	Time spent on walking through the old code	Help from other developers	M	M	Better communication between developers. Rules on when to use different patterns
18	old code affected, bugs appear	K1	code not working together	modify old code	H	L	more training
21	Code that ran flawless previously stops working	E2	Program does not work as intended				
23	Break old code	C1	Non-functioning software	Fix it	M	H	Better tests
23	Fail to implement	K1		Get help from others	M	M	
26	Old code hard to understand (Not intuitive)	R3	Modification takes long time. Lots of testing on old code wastes more time.	None	M	M	Comment/document code while reviewing old code.
27	Spaghetti code, needed many code changes for small functionality change	T2	Much time lost fixing bad code	Need help from more developers	M	M	Better coding techniques and better documentation
28	Implementation takes longer time than expected	T2	Work plan needs to be revised	Ask a developer with similar expertise	M	M	Check compabilities before writing code
30	Not able to implement	K1	Changes not implemented	Ask a developer with similar expertise	M	M	More training
31	Uncertainties about how to code the changes						
33	Not able to code all changes	K1 T2	Discard functionality, not able to reach time estimate		L	H	Become a better programmer
34	spend too much time	T1	delays	drop non-vital functionality	H	M	be more generous when making plans and scheduling
35	Old code does not support old functionality	E1	Hard to code new functionality	Code	M	L	Make modifiable code
36	Developer is not capable of coding the changes	K1	Problem not solved	needs help from another developer	L	L	
37	takes long time	T2	production delayed	more developers assigned to task	M	M	Better planning of time-management
38	flaws in code	K1	Feature doesn't work	Fix it	M	L	Learn to code better
38	Too little time	T1	Feature omitted	Get more time	L	H	learn better time-estimation
39	Underestimate implementation	T2	More time used than planned	Reserve more time, or get more developers on your team	M	M	Gain experience with similar implementations.
40	Wrong changes are implemented	I2	Wrong functionality in system	recode the changes	L	L	Gain a better understanding of the functionality for coding
41	New code is not implementable in the system	S3	not able to implement	None	L	H	Make sure to write all code modifiable