NTNU

Innovation and Creativity

BUCS: Patterns and Robustness

Experimentation with Safety Patterns in Safety-Critical Software Systems

Ingvar Ljosland

Master of Science in Computer Science

Submission date: June 2006

Supervisor: Tor Stålhane, IDI

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

Problem Description

In modern society, we are greatly dependent upon safely working software systems. The use of safety patterns can solve many of the safety requirements that such safety-critical systems have. Former research of a single case study suggested that safety patterns get stacked on top of the initial system and local design patterns. An experiment can try to generalize these findings, and make a qualitative statement about the difficulty level of solving a systems safety requirements compared to solving functionality related problems.

Assignment given: 20. January 2006
Supervisor: Tor Stålhane, IDI

Abstract

In modern society, we rely on safely working software systems. This is the final report in a masters degree project to reveal key issues in the science field of computer software architecture and design of safety-critical software systems.

A pre-study of a navigation system implied that functionality related problems and safety-critical problems do not stack one to one, but rather is a case of solving these aspects in different layers. This means that changes in software systems functionality do not necessary mean that change in safety-critical modules has to be done as well, and visa versa.

To further support the findings in the pre-study, an experiment was created to investigate these matters. A group of twenty-three computer science students from the Norwegian University of Science and Technology (NTNU) participated as subjects in the experiment. They were asked to make two functional additions and two safety-critical additions to a software robot emulator.

A dynamic web tool was created to present information to the subjects, and they could here answer surveys and upload their task solutions.

The results of the experiment shows that there were not found any evidence that the quality attributes got affected by the design approaches. This means that the findings of this study suggest that there is difficult to create safety-critical versions of software architectural design patterns, because all design patterns have a set of additions and concequences to a system, and all sides of the implications of the design pattern should be discussed by the system architects before used in a safety-critical system.

Keywords: *Software, design patterns, architecture, safety-critical, experiment.*

Preface

This is the written report of a masters degree project in computer science. The project is a contribution to the BUCS [1] research project at the Norwegian University of Science and Technology (NTNU), and conducted during the spring of 2006.

The project documentation consists of this paper, a web tool, and results in forms of spreadsheets and source code. The web tool source code, spreadsheets and programming task source code are included as a digital appendix to the report.

The web tool can be found at the research project homepage: http://www.ljosland.net/master/.

I would like to thank Tor Stålhane, my teaching supervisor, for valuable input during the research process.

I would also like to thank the third year students from computer science and communication technology at NTNU for doing a great job as subjects to the experiment, and also Marius Sommerseth, for being an assistant during the experiment day.

Trondheim, 16.06.2006
Ingvar Ljosland

[1] Business-Critical Software (BUCS) homepage: http://www.idi.ntnu.no/grupper/su/bucs/

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter will present the research, covering the motivation of the research, the problem definition, the scope, related research, and how the report is outlined.

1.1 Motivation

In modern society, we are greatly dependent upon safely working software systems. These systems can be anything from control systems in trains, ships, and airplanes to industrial chemical or nuclear plants.

For a developer of these types of systems, it is crucial to know how safety-critical additions may influence the system. These additions to a system can make it become more robust, more usable, more reliable, and more available and so forth, but there is always a possibility that an addition to a system can make one or more quality attributes suffer when others are improved.

A prestudy made by the researcher in [Ljo05] suggested that when adding new safety-critical requirements, the functionality aspects of a software system do not get much influenced by these adjustments. A case study of a navigation system illustrated how safety implementations got added on top of the functionality related classes.

To investigate this research further, there was suggested that a controlled experiment could supply the former research. The experiment should test how quality attributes such as differences in effort of making changes, or the correctness of the different implementations, got affected by a set of safety-critical additions.

1.2 Problem Definition

The goal of the study was to plan and execute a controlled experiment to investigate how safety-critical design pattern get affected by functionality related problems, and visa versa. A set of quality attributes was selected as measurement of the experiment.

The objective deriving from the goal is:

"Design and execute a controlled experiment where a set of safety-critical design pattern get tested against functionality related problems."

The research questions arising from the above objective is as follows:

R1. How are quality attributes affected when functionality problems are added to a set of safety-critical design?

R2. How are quality attributes affected when safety-critical design pattern get added to a system that has already a set of added functionality?

R3. How can we create a controlled experiment that can test research questions 1. and 2.?

1.3 Scope

Because the study is a part of a masters degree project in computer science, the time limit and the budget of the study is tight.

The experiment planning, execution, analysis and presentation had a time schedule from the 20.january 2006 to 16.june 2006.

Because of the tight schedule, the experiment execution could only be less than a days work for the participants, and only a couple of light-weight safety-critical design patterns can be tested in such a small-size system.

The low budget of the experiment made the subject selection by using Convenience sampling [WRH$^+$00, p.52], meaning that the nearest and most convenient persons are selected as subjects.

1.4 Related Research

There has been several research made with the use of experimentation in software engineering. A research made by [AS03] was testing how the effect of delegated versus centralized control style effected the maintainability of object-oriented software. The experiment is related in the test methods and subject population, only the research by [AS03] also includes professional software developers.

The prestudy made by [Ljo05] are focusing on the same problem definition, but uses a navigation case study to discover the effects of the safety-critical implementations.

This study used the results and findings of the [Ljo05] research and create a controlled experiment similar to the [AS03] experiment, only with a smaller budget.

1.5 Report Outline

The following chapter, Chapter 2, will describe the methodology of empirical studies and experimentation in software engineering related the research. Chapter 3 describes the theory of design patterns and how they can be used in software architecture, and why they are used in this research. Chapter 4 describes the design of the controlled experiment, and Chapter 5 describes the Experiment Web Tool. In Chapter 6, the results of the experiment are presented, followed by a discussion of the threats to the validity of the research in Chapter 7, and the discussion of the research as a whole in Chapter 8. Chapter 9 sums up the research, concludes and suggests further research that can be made.

Chapter 2

Research Method

This chapter presents the research method of the project, which is the methodology of experimentation in computer science. The first section will present general usage of experimentation in Software Engineering, while the following section will present the statistical models used in the data analysis.

2.1 Experimentation in Software Engineering

To justify the use of experimentation in software engineering research, we need to give a reasoning of the importance of empirical studies in this discipline. This section will give a brief introduction to the use of experimentation as an empirical study in the research area of software engineering.

2.1.1 The Background

As computer science evolved during the late 60s and the term "software engineering" was created to describe the engineering focus of developing software systems, empirical studies fitted into the software engineering context because of the need for systematic, disciplined and quantifiable approach of developing, operate and maintenance of software [WRH$^+$00, p.15].

For the software engineering to be a science, we need a well organized way to do research. With the use of the empirical studies, hypothesis that the software engineers have can be tested, and the more results of an empirical study that support the hypothesis, the stronger the hypothesis can be accepted as a scientific theory.

By doing scientific research in software engineering, the whole computer industry benefits it. As an example, if we have a hypothesis that object-oriented programming is more effective than functional programming, and do empirical studies that support this theory, the software industry can take this into consideration when choosing programming style in a software project.

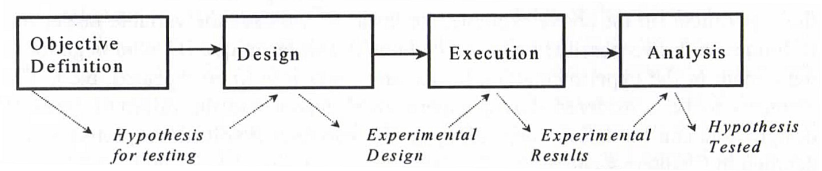


Figure 2.1: The phases of experimentation (taken from [JM01, p.49])

There are two types of approaches in empirical studies, the Qualitative research, and the Quantitative research [WRH$^+$00, p.7]. The qualitative approach is concerned with studying objects in their natural environment. As an example if the subject is a person, the data is collected by talking to the person and trying to understand his situation.

The quantitative approach is tying to quantify a relationship between groups to identify cause-effect relationships. The experiment is a type of empirical study that uses the quantitative approach when testing effects of a treatment, while a qualitative study will try to give an understanding why the results are as they are. That is why the two approaches are considered as complementary.

2.1.2 Experimentation Overview

The experiment is an empirical study with a high level of control. It is normally done in a laboratory environment, where subjects are assigned different treatments. When manipulating some variables and holding others at a fixed level, one can measure and analyse the effect of the manipulation with statistical models. The strength of an experiment is that it can investigate in which situations the claims are true and they can provide a context in which certain standards, methods and tools are recommended for use [WRH$^+$00, p.15].

[JM01, p.49] identifies four phases of experimentation:

1. Definition of the objectives of the experimentation
2. Design of the experiments
3. Execution of the experiments
4. Analysis of the results/data collected from the experiments.

The *definition of objectives* is a phase where the general hypothesis are transformed into formal hypothesis and formulated in terms of the phenomenon under examination. The plan of how to control the experiment environment, and how to measure the variables to make the hypothesis test possible, is determined in the *design phase*. In the *execution phase*, the experiment is run according to the experiment plan and the latter *analysis phase* use statistical models on the recorded data to either support or reject the defined hypothesis.

2.1.3 Hypothesis Statements

In the experiment definition phase, we try to formalize the research questions into hypothesis. The hypothesis are formulated in two parts, where the null hypothesis (H_0) states that there is no real underlying trends or patterns to the experiment setting. This means that there is only coincidental that some observations are different.

The alternative hypothesis (H_a, H_1, etc), is the hypothesis in favour of which the null hypothesis got rejected. The alternative hypothesis might as an example claim that one the effect of a treatment are less on one of the groups tested.

The conclusions from a hypothesis test can be validated with a significance level, where the errors are classified into two types, the Type-I-error and the Type-II-error [WRH+00, p.50].

The Type-I-error occurs when statistical tests indicates a pattern when there really is no pattern. The probability of committing a type-I-error is expresses as:

$$\text{P(Type-I-error)} = \text{P(reject } H_0 \mid H_0 \text{ true)}$$

The Type-II-error occurs when a statistical test indicates no patterns or relationship when there really is one. The probability of committing a Type-II-error is expressed as:

$$\text{P(Type-II-error)} = \text{P(not reject } H_0 \mid H_0 \text{ false).}$$

2.1.4 Threats to Experiments Validity

Before making any conclusions, one needs to carefully look at the threats to the validity of the experiment. [WRH+00, p.63-65] make following classifiction of threats to validity of the experiment results:

1. *Conclusion validity.* The validity concerned with the relationship between the treatment and the outcome. We want to make sure that there is a statistical relationship, i.e. with a given significance.

2. *Internal validity.* If a relationship is observed between the treatment and the outcome, we must make sure that it is causal relationship, and that it is not a result of a factor of which we have no control or have not measured. In other words, that the treatment causes the outcome (the effect).

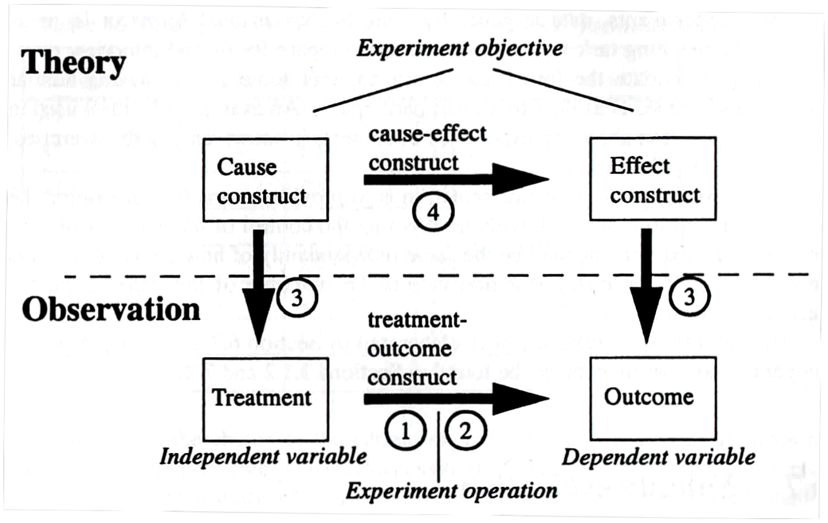


Figure 2.2: Experiment design and validity (taken from [WRH$^+$00, p.64])

3. *Construct validity.* This validity is concerned with the relation between theory and observation. If the relationship between cause and effect is real, we must ensure two things: 1) that the treatment reflects the construct of the cause well (left part of Figure 2.2) and 2) that the outcome reflects the construct of the effect well (right part of Figure 2.2).

4. *External validity.* The external validity is concerned with generalization. If there is a causal relationship between the construct of the cause, and the effect, can the result of the study be generalized outside the scope of our study? Is there a relation between the treatment and the outcome?

Figure 2.2 show how [WRH$^+$00] presents the important concepts of the validity classification.

2.2 Statistical Model

The experiment in this project is performed with the use of a completely randomized design. The design setup uses the same objects for both the treatments, and assigns subjects randomly to the two treatments. The design is called balanced if the same number of subjects is set to the treatments.

Table 2.1 show an example of assigning treatments to a randomized design.

The statistical model for an experiment using a parametric, one factor, two treatments, and completely randomized design is the t-test.

Subject	Treatment 1	Treatment 2
1	X	
2		X
3		X
4	X	
5		X
6	X	

Table 2.1: Example of assigning treatments to a randomized design (adopted from [WRH$^+$00, p.55]).

2.2.1 T-test

The t-test is a parametric test that compares two sample means. The t-test steps are described in [WRH$^+$00, p.99] as follows:

1. Input - two independent samples: x_1, x_2, .., x_n and y_1, y_2, .., y_m.
2. H_0 - $\mu_x = \mu_y$, i.e. the expected mean values are the same.
3. Calculate $t_0 = \frac{\bar{x}-\bar{y}}{S_p\sqrt{\frac{1}{n}+\frac{1}{m}}}$, where $S_p = \sqrt{\frac{(n-1)S_x^2+(m-1)S_y^2}{n+m-2}}$, and S_x^2 and S_y^2 are the individual sample variances.
4. (a) Two-sided (H_1: $\mu_x \neq \mu_y$): reject H_0 if $|t_0| > t_{\alpha/2,n+m-2}$.
   (b) One-sided (H_1: $\mu_x > \mu_y$): reject H_0 if $|t_0| > t_{\alpha,n+m-2}$.

This test can automatically be done by most spreadsheet programs. Microsofts Excel is used as statistical tool in this project.

Chapter 3

The use of Design Patterns in Software Architecture

Design patterns are commonly found in the field of Software Architecture. This chapter will give an introduction to what a design pattern is, and why it is important to the experiment. The first section gives a general overview of the role of design patterns in software engineering. The following sections presents software patterns in safety-critical systems, and the description of two design patterns used in the experiment.

3.1 Overview

A design pattern are described by [Dou03] as "a generalized solution to a commonly occurring problem". So, instead of inventing new solutions all the time when a problem occurs, we can use a generalized solution that is defined to meet one or more of quality attributes like: performance, predictability, scheduleability, throughput, reliability, safety, reusability, distributability, portability, maintainability, scalability, complexity, resource usage, energy consumption, recurring cost and development effort.

Software design patterns have arisen from a concept of the traditional architecture industry. In 1977, Christopher Alexander wrote the book "A Pattern Language: Towns, Buildings, Construction" [Ale77], where he describes practical, safe and attractive design implementations of all levels of detail in a building plan. He describes the design in a general language with pictures and example calculations.

[BC87] adopted the idea of a genuine architectural design language for describing software related problems in the same way as for the traditional architecture discipline. They presented the results of theire work at the OOPSLA-87 [1] work-

[1] Object-Oriented Programming, Systems, Languages & Applications (OOPSLA): An annual conference by The Association for Computing Machinery (ACM).

shop on the Specification and Design for Object-Oriented Programming.

The use of design patterns has increased during the latest years, and the famous “Gang of Four (GoF)” in computer science, Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, published the book “Design Patterns: Elements of Reusable Object-Oriented Software” in 1994 [GHJV94]. The book set a standard to the use of design patterns in the computer industry, and is still popular reading among software architects.

[GHJV94, p.3-4] presents four essential elements of a design pattern:

1. *The pattern name.* The name of a pattern is describing the pattern solution, and helps increasing the science vocabulary, helping software architects understanding the pattern in an unambiguous way.

2. *The problem.* This describes the problem, its context, and when to apply the pattern. It might describe classes, structures or a list of conditions that must be met before it makes sense to apply the pattern.

3. *The solution.* This is the elements that make up the design, their relationship, responsibilities and collaborations. The solution is a template to use, not a specific design implementation, because it can be applied in many different situations. The pattern provides an abstract description of the design problem, and a general way of arranging elements, such as classes and objects, to solve the problem.

4. *The consequences.* By applying the pattern there is a set of trade-offs and risks. The consequences of applying the pattern should be carefully discussed and evaluated by the system designers, and the pattern should only be implemented if the benefits of it are greater than the cost. A design pattern might, as an example, benefit system flexibility, while on the other hand lower the systems performance.

Software design patterns are under continuous evolution, and in annual conferences the current design patterns are revised and new patterns are added to the pattern catalogue if found appropriate.

3.2 Safety-critical Design Patterns

As for any other software, safety-critical systems are also dependent on correctly working design patterns. The design patterns for safety-critical software systems are often focused on quality attributes such as robustness, reliability, and safety.

Two safety-critical design patterns are used in the experiment, the Protected Single Channel and the Watchdog design patterns. These patterns are taken from [Dou03], and are two commonly used patterns in embedded systems. Bruce

Powel Douglass, the writer of [Dou03] and [Dou99], is in the Advisory Board of the Embedded Systems Conference [2], and a co chair of the Real-Time Analysis and Design Working Group (RTAD) [3] within the Object Management Group (OMG) [4], and consults a number of companies and organisations including NASA.

According to [Dou03, p.79-80], the safety and reliability architecture is concerned with correct functioning in the presence of faults and errors. The reliability attribute are a measure of availability of a system (to the user), and can be estimated by for instance the formula $A = \frac{uptime}{totaltime}$ or $\frac{MTTF}{MTTF+MTR}$, where A is the availability, MTTF is the mean time to failure, and MTR is the mean time to restore. Redundancy is one design approach that increases availability because if one component fails, another takes its place.

A safe system is a system that decreases a systems danger to people or equipment. A risk is the product of the severity of the incident and its probability. Take as an example flying in an aeroplane: The impact of a crash can be fatal, but the probability of a crash is so small, it makes the risk tolerable. Another example is the risk of getting electric shock when changing a battery in a camera or MP3 player; it is very likely to happen, but the consequences are so small that the risk is tolerable.

3.2.1 Protected Single Channel Pattern

The Protected Single Channel pattern is a lightweight design pattern for systems that needs some safety and reliability, but cannot afford total redundancy. It uses a single channel to handle sensing and actuation. Key points in the channel enhance safety and reliability trough checks where transient faults can be found and then threat.

Figure 3.1 show how the pattern structure in an open loop version, as used in one of the experiment tasks. The channel consist of input and output processing, internal data transformation and data integrity checks. The actuator is the hardware that performs the actions of the channel.

The pattern provides some level of safety and reliability against either systematic or random faults, such as corrupted data or data beyond the value bounds. In systems that should be able to operate in the presence of permanent fault, more heavy weight patterns such as Homogenous Redundancy [Dou03, p.415-421] or Heterogeneous Redundancy [Dou03, p.426-431] patterns should be used.

3.2.2 Watchdog Pattern

A dog that watches over hens in a chicken run, only concerns whether something is obviously wrong or not. The dog does not actually go into the henhouse and

[2] The Embedded Systems Conference, Home Page - http://www.esconline.com

[3] RTAD, Home Page - http://realtime.omg.org

[4] Object Management Group, Home Page - http://www.omg.org

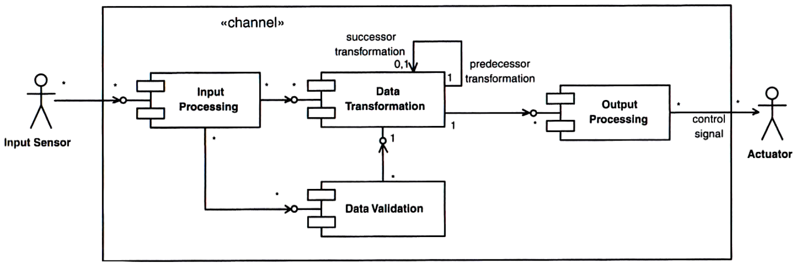


Figure 3.1: Protected Single Channel Pattern (taken from [Dou03, p.411])

check if the hens are laying eggs or not. This is how the software pattern got its name; because it watches over the processes in the system, but it does not actually checks that the internal computation processing is correct. This makes the Watchdog pattern a lightweight pattern that can add additional safety to a safety-critical system, and are often combined with other, usually more heavierweight patterns.

The Watchdog pattern structure is shown in Figure 3.2. The actuator channel operates pretty much independently of the watchdog, sending a liveness message to the watchdog every so often. This is called stroking the watchdog. The watchdog uses the timelines of the stroking to determine whether a fault has occurred, i.e. if a new stoke has not come within a defined time limit, the watchdog alerts the system, where further actions can be made.

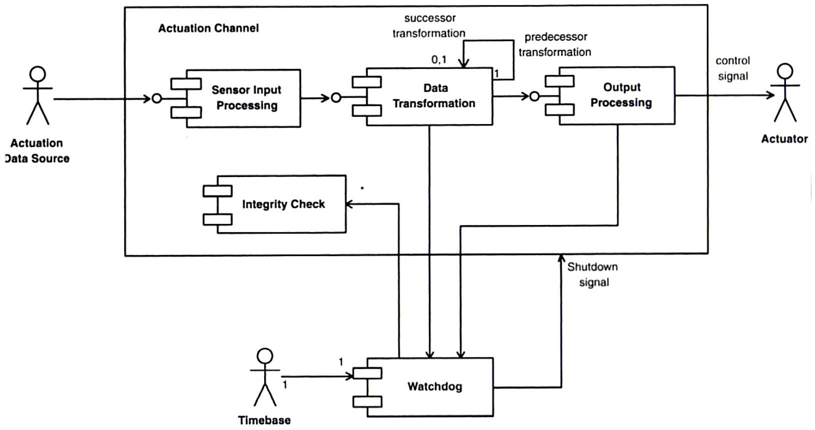


Figure 3.2: The Wathcdog Design Pattern (taken from [Dou03, p.445])

If the watchdog is to provide protection from timebase faults, a separate electric circuit must supply an independent measure of the flow time. The watchdog might also be used to improve deadlock detection, where strokes can be keyed or contains data to identify strokes from different computational steps, making it possible to identify in which step the fault occurred. This is called a Keyed Watchdog or a Sequential Watchdog [Dou03, p.448].

The design patterns presented in this chapter are used as basis for the experiment tasks, presented in the following chapter.

Chapter 4

Experiment Design

This chapter explains how the experiment was performed. The first section gives a formal experiment definition, and is followed by research goals in Section 4.1, and hypothesis formulations in Section 4.2. The last section presents the experiment tasks, and some practical information about the experiment conditions.

4.1 Experiment Definition

The formal experiment goal definition is set up from a goal definition template by [WRH^{+}00, p.42]. The goals are then further explained in the latter subsections.

4.1.1 Goal Definition

Analyse *source code, test results and survey answers*

for the purpose of *evaluate design approaches*

with respect to *correctness, effort and difficulty level of making changes*

from the point of view of *the researcher*

in the context of *M.Sc. students solving functionality and safety-critical tasks in a Khepera Robot Simulator Software System, and answering post and pre-task surveys. The study is conducted as a blocked subject-object study.*

4.1.2 Objects of Study

The objects studied are safety-critical and functionality tasks implemented in a robot emulator. The objects are source code, test results and survey answers. The source code is written by the experiment subjects, and uploaded to the experiment file server. The subjects will also answer a survey about their experience levels and skills in programming and design pattern knowledge. After each task, the subjects are given a post-task survey, where they answer how much time they spent on solving the task, how the effort of solving the task was, and how difficult they thought the task was.

4.1.3 Purpose

The purpose is to evaluate the approaches of adding safety-critical requirements to a software system versus adding new functionality to a system. The experiment will try to see if the safety patterns easily can be added as a layer on top of the existing software, or if safety patterns are most beneficial to be added as early in the design phase as possible.

4.1.4 Quality Focus

The quality focus is to find differences in correctness, effort and difficulty. This focus represents both quantity and quality measures, and will make it possible to make evaluations that treat the purpose of the experiment.

4.1.5 Perspective

The perspective is from the researchers point of view, meaning that the experiment will try to answer computer science research questions, where the goal is a better understanding of software architecture and the use of design patterns in safety-critical environments.

4.1.6 Context

The experiment is set up using M.Sc. students as subjects. It will be conducted in a computer lab at the NTNU campus. Surveys and tasks will be presented in a web-based environment, where the experiment output and test results can be logged and saved to a common database.

4.1.7 Research Goals

The research goals are presented in a Goal, Question, Metrics (GQM) approach [WRH$^+$00, p.23]:

- G.1. Discover if there are any differences in robustness, safety or reliability in software systems where safety-critical modules are added on top of functionality requirements, or if the opposite is more beneficial.

  - Q.1.1. Are there any difference in correctness of source code?
  - Q.1.2. Are there more failures when using one of the approaches?

- G.2. Discover whether it is more difficult to modify functionality modules to meet safety-critical requirements, versus modifying safety-critical modules to meet functionality demands in a software system.

  - Q.2.1. Are there any differences in the effort used by the subjects between the two approaches?
  - Q.2.2. Which of the approaches do the subjects find it easier to understand and modify?

Goal	G.1		G.2	
Question / Metrics	Q.1.1	Q.1.2	Q.2.1	Q.2.2
Numbers of code errors in a code inspection check-list	X			
Number of failures during execution of test sets		X		
Time measure of modification			X	
Survey answers				X

Table 4.1: Goal, Question and Metrics (GQM)

4.2 Hypothesis

This section will define the hypothesis of the experiment. As the subjects are divided into two groups, where the first group add safety-critical modules to a system, then solves functionality problems, and the second group solve the functionality problems first, then adds safety-critical modules, one would expect that the first group would have a more robust system from the beginning, thus score better on correctness when solving the functionality problems. This reasoning is represented by the H_1 hypothesis.

The H_2 hypothesis suggests that there should be less effort of changing functionality to a software system, when safety-critical modules are implemented, because of a more robust software system. In addition, the H_3 hypothesis suggests that the effort of adding safety-critical modules to a software system that has already been added functionality solutions should be greater then in a situation where no functionality has been added.

The difficulty of changing safety-critical aspects to a software system should rise as more functionality is added to the system. The H_4 hypothesis will represent this situation.

The null-hypothesis is thereby defined as follows:

$H0_1$ The Effect of Design Approach on Correctness: The number of correct solutions from a code inspection checklist of the two design approaches are equal.

$H0_2$ The Effect of Design Approach on Change Effort of Functionality: The effort of making changes to functionality in the software system are equal for both approaches.

$H0_3$ The Effect of Design Approach on Change Effort of Safety-Critical Design: The effort of making changes to safety-critical design in the software system are equal for both approaches.

$H0_4$ The Effect of Design Approach on Degree of Difficulty: The subjects find the difficulty level of the tasks given equal for both approaches.

4.3 Experiment Tasks

The experiment consist of five programming tasks; a pre-task, two tasks where functionality problems are to be solved, and two tasks where safety-critical modules are implemented to the software system. The software system is The WSU Khepera Simulator Suite (WSU KSuite) [1], created by the Wright State University. The WSU KSuite is a collection of Java programs that let users create and simulate controllers for the Khepera [2] robot.

The full listing of the task tekst are shown in Appendix B.

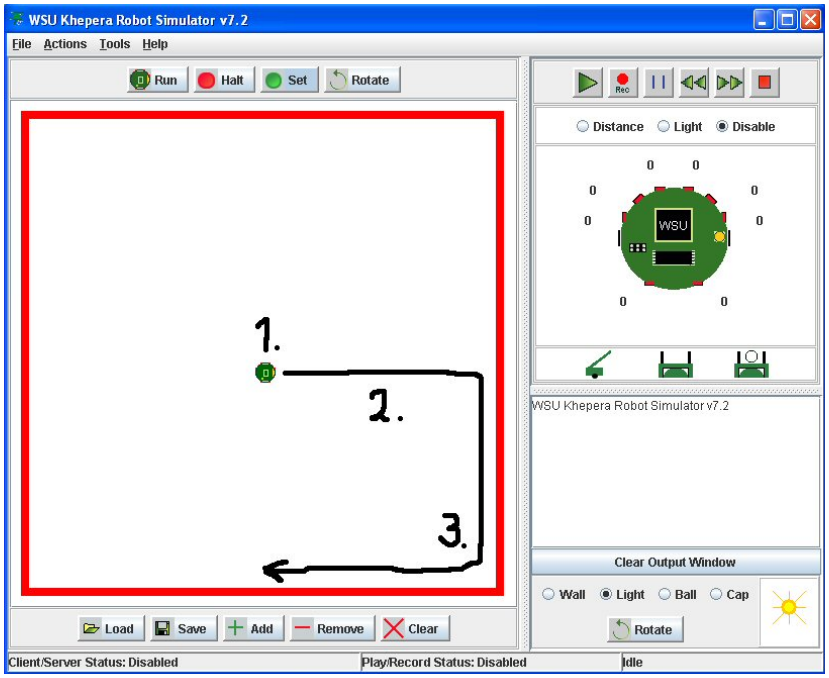
4.3.1 Pre-test / Trainig Task

In the pre-test task, the subjects were asked to implement a robot that follows a wall, as shown in Figure 4.1. The purpose of this task was to provide a common baseline for comparing the subjects programming skills, and to let the subjects get an introduction exercise for learning how the robot emulator works. The

[1] Khepera Simulator Home Page - http://carl.cs.wright.edu/reg//ksim/ksim/ksim.html

[2] Khepera Team Home Page - http://www.k-team.com/kteam/

subjects had to learn how to navigate with the use of distance sensors in order to solve the task.



○ Distance ○ Light ● Disable

WSU Khepera Robot Simulator v7.2

○ Wall ● Light ○ Ball ○ Cap

Client/Server Status: Disabled

Play/Record Status: Disabled

Idle

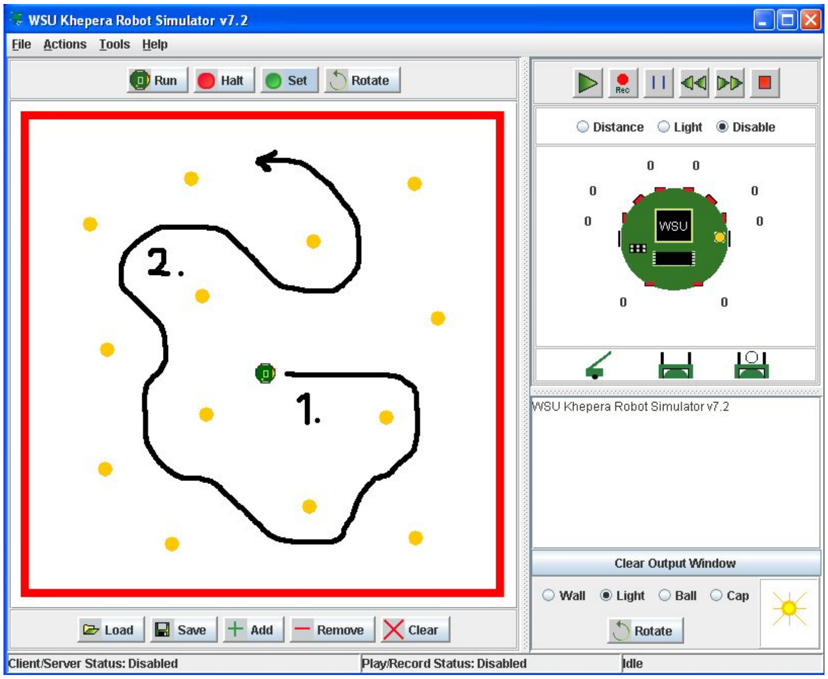
Figure 4.1: Pre-test: Wall Follower

4.3.2 Functional Solving Task 1 (FST1)

This is the first of the tasks where system functionality is implemented to the system. The subjects were asked to create a robot controller that walk around randomly and avoid any of the lights that was placed around the emulation map, as shown in Figure 4.2. The subjects had to learn to navigate with the use of light sensors in order to solve the task.

4.3.3 Functional Solving Task 2 (FST2)

The second task of solving functionality problems was to implement a robot controller that find its way through three light gates, pick up a ball, and returns to its home base. The task is illustrated on Figure 4.3. The subjects had to use the sensors for light detection from the previous task and the ball-picking function from the introduction exercise in order to complete this task.



○ Distance ○ Light ● Disable

WSU Khepera Robot Simulator v7.2

○ Wall ● Light ○ Ball ○ Cap

Client/Server Status: Disabled | Play/Record Status: Disabled | Idle

Figure 4.2: Functionality Solving Task 1 (FST1)

4.3.4 Safety-critical Task 1 (SCT1)

The SCT1 are the first of the two tasks for adding safety-critical modules to the software system. The subjects were given information and UML design of a Protected Single Channel design pattern to better validate the input from the robot sensors. The task was to implement this design pattern to the robot controller.

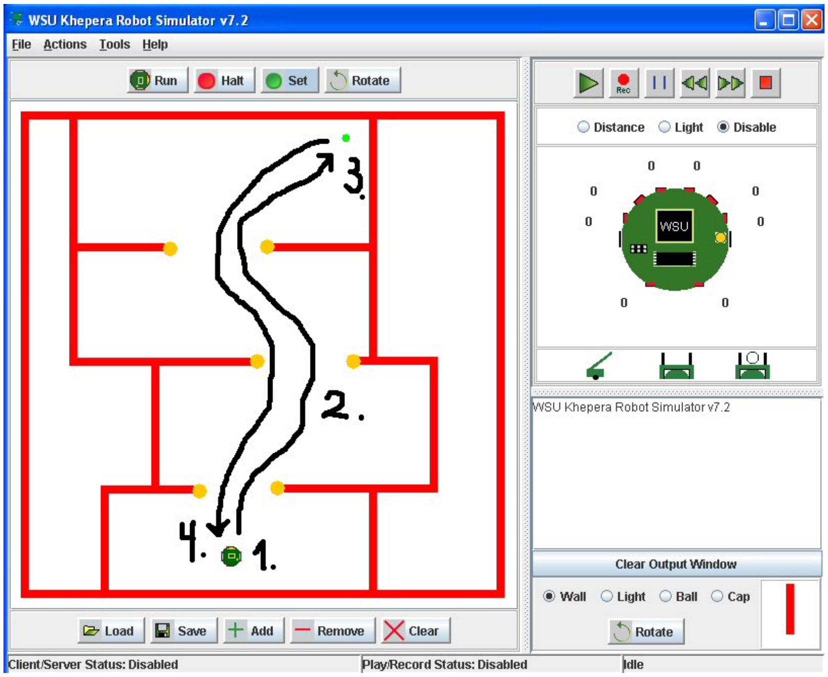
Figure 4.4 shows the UML design of the Protected Single Channel pattern. The Protected Single Channel Pattern is described further in Section 3.2.1.

4.3.5 Safety-critical Task 2 (SCT2)

In the last of the safety-critical task, the subjects were asked to implement a Watchdog pattern. The pattern are described in Section 3.2.2, and is added to the robot controller to detect that the processes are running, and the robot isn't stuck in a wall or any other obstacles. As Figure 4.5 shows, the subjects had to implement the Watchdog pattern with the Data Transformation process from the Channel class in the last safety-critical task.

When the Watchdog pattern has been fully implemented, the robot should detect when it's stuck, and run a routine that tries to free the robot from the obstacle.

In both the safety-critical tasks, the subjects were asked to test the imple-



○ Distance ○ Light ● Disable

WSU Khepera Robot Simulator v7.2

● Wall ○ Light ○ Ball ○ Cap

Client/Server Status: Disabled Play/Record Status: Disabled Idle

Figure 4.3: Functionality Solving Task 2 (FST2)

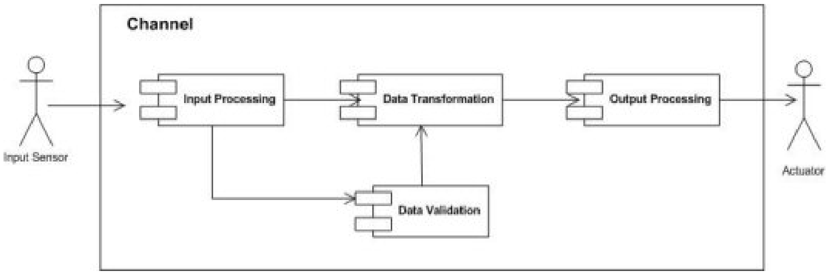


Figure 4.4: Safety-critical Task 1 (SCT1)

mentations in the robot controller that they used to solve the previous task.

4.4 Experiment Conditions

The experiment was conducted at the Norwegian University of Science and Technology (NTNU) in Trondheim, Norway. Because of the limits to economy and time in the experiment, the final number of subjects that participated in the experiment was twenty-three. The subjects were third year computer-science students that were saving money to a educational trip, and got paid 500 NOK each for participating in the experiment.

To make the experiment as realistic as possible, it was conducted at the students normal computer labs, where they used their usual Java development

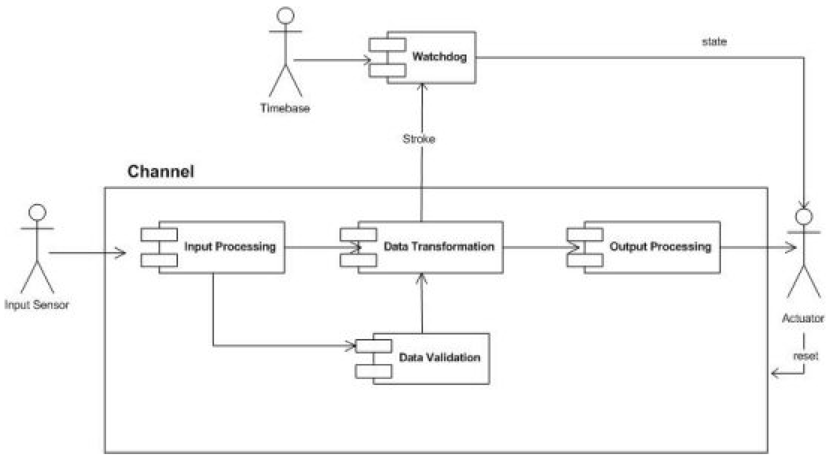


Figure 4.5: Safety-critical Task 2 (SCT2)

environment.

The subjects were divided randomly into two groups were the first group, the Type 1 group, had to implement the functional solving tasks first, then latter the safety-critical, and the second group, the Type 2 group, got the tasks the other way around.

To make the experiment run as easy as possible for the subjects, a web tool was created that could present information and tasks to the subjects. They could also use the web tool to upload their task solutions and survey answers.

Screenshots of the Experiment Web Tool are found in Appendix G, and the tool are presented in more details in the next chapter.

Chapter 5

Experiment Web Tool

A web tool was created to get an easy end effective experiment execution. This chapter will describe how the web-tool works and how it was built.

5.1 Web Tool Overview

The experiment was executed at a computer lab, with the subjects doing programming tasks and answering surveys. To make this process effective, a web-based tool seemed to be the best way of presenting information and helping the subjects upload their programs and survey answers to a common experiment database or fileserver.

Figure 5.1 shows how the communication between the subjects computers, the common database and file server, and the experiment researcher takes place. As shown in the figure, the subjects used computers at a computer lab with the use of a web browser as an interface to the experiment setup. Survey answers, login information and task answers were stored in the server MySQL[1] database. The source code from the programming tasks was uploaded to the file server, and could be accessed trough a File Transfer Protocol (FTP) connection by the researcher. The database could easily be exported to a spreadsheet that could be analysed with statistical tools.

5.2 Database Design

The experiment database was created to store necessary information about the subjects, their solutions to the programming tasks, and answers to surveys.

Figure 5.2 show the Entity Relationship (ER) diagram of the database. As one can see from the figure, the information about the subjects are username, password and an identification number. The passwords are stored encrypted

[1] An open source database system - official homepage: http://www.mysql.com

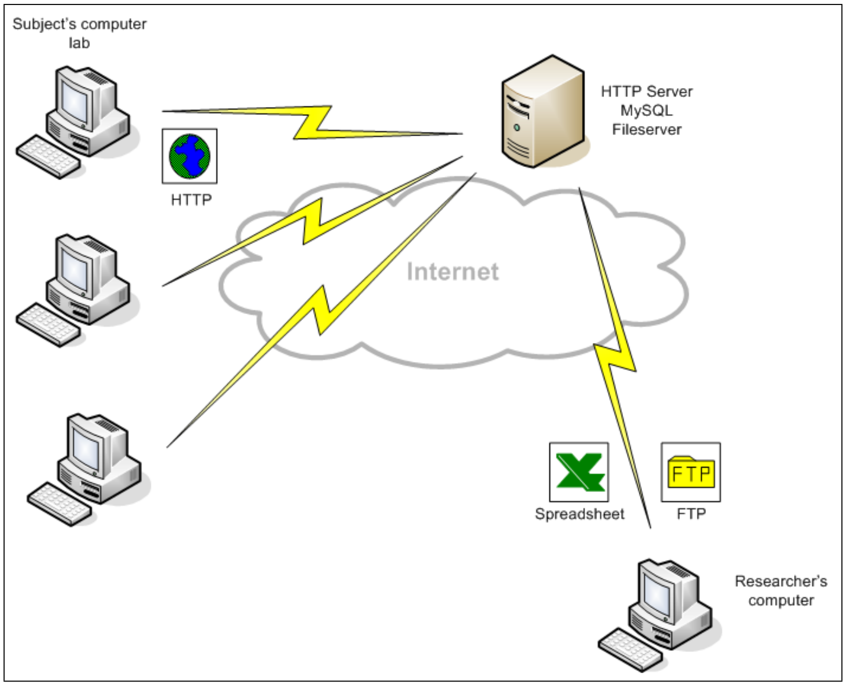


Figure 5.1: Web Tool Communication

with the use of MD5 algorithm. The answers from the pre-survey are stored in the Survey entity, where the subjects identification number connects the survey answer to the subject. The experiment entity have three attributes, a type field to store whether the experiment is done with a Type 1 or Type 2 group, an identification field, and a field that connects an experiment to a subject. The Task entity represent the answers of the post-task test, where the information stored are filename to the source code, start and stop time of task, and the other questionnaire answers. The experiment identification field connects the task to a specified experiment.

The database was built and implemented at the web server by the database administration tool called phpMyAdmin [2]. A complete copy of the database can be found in the digital appendix.

5.3 Web Interface

This section describes the web interface to the experiment tool. The web tool was written in the PHP[3] scripting language, which easily integrates database technology and html to make dynamical web pages.

[2] phpMyAdmin Home Page - http://www.phpmyadmin.net

[3] PHP: Hypertext Preprocessor - http://www.php.net

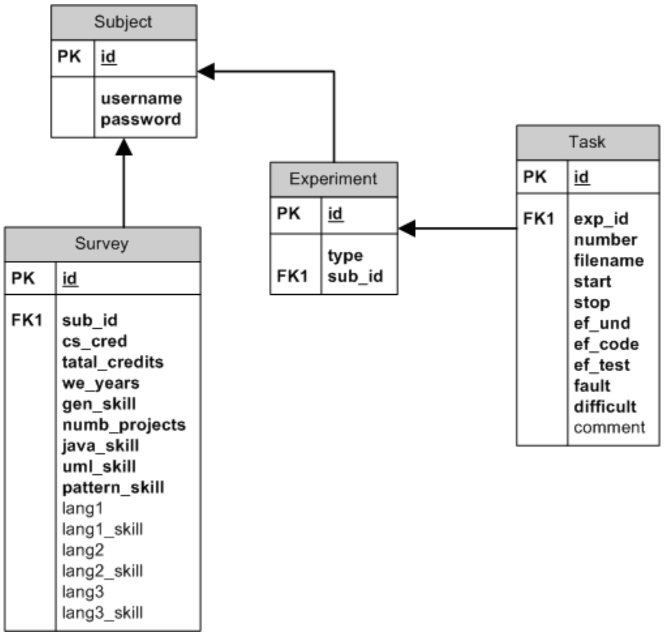
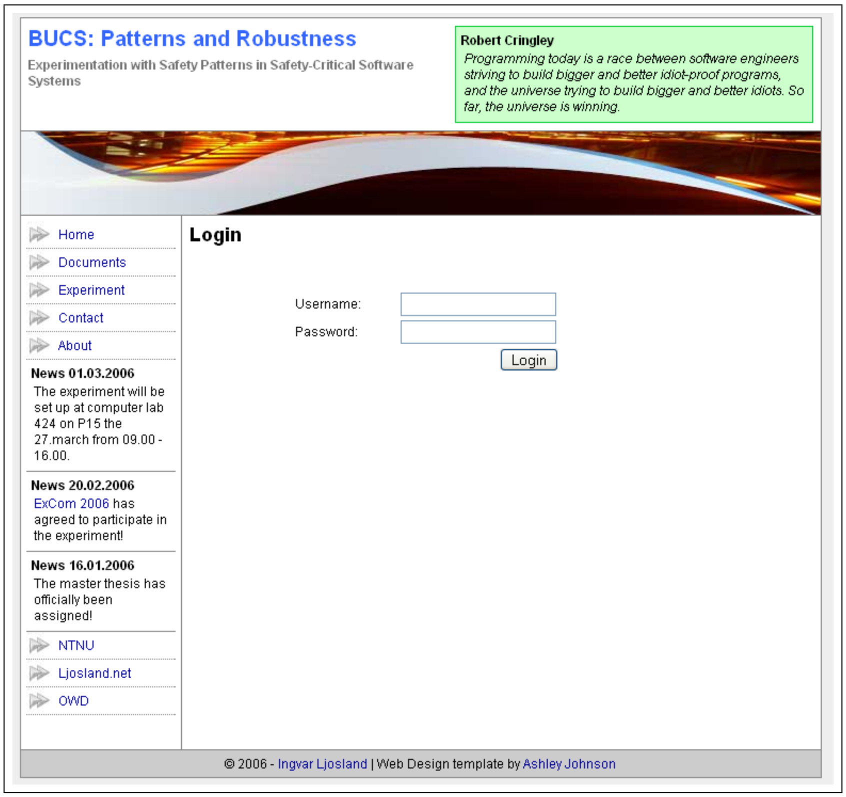


Figure 5.2: Experiment database ER diagram

5.3.1 Login

To identify each subject and to know the relation between the subject, tasks and survey answers, each subject got a username and password. They could use this username and password to log on to the experiment tool. A PHP session that stores their identification number during the experiment process get created automatically on login.

Figure 5.3 show the login screen. For test purposes, the login username “test1” with the password “test1” can be used to log in as a Type 1 experiment, while the username “test2” with the password “test2” can be used to log in as a Type 2 experiment.



BUCS: Patterns and Robustness

Experimentation with Safety Patterns in Safety-Critical Software Systems

Robert Cringley
Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far, the universe is winning.

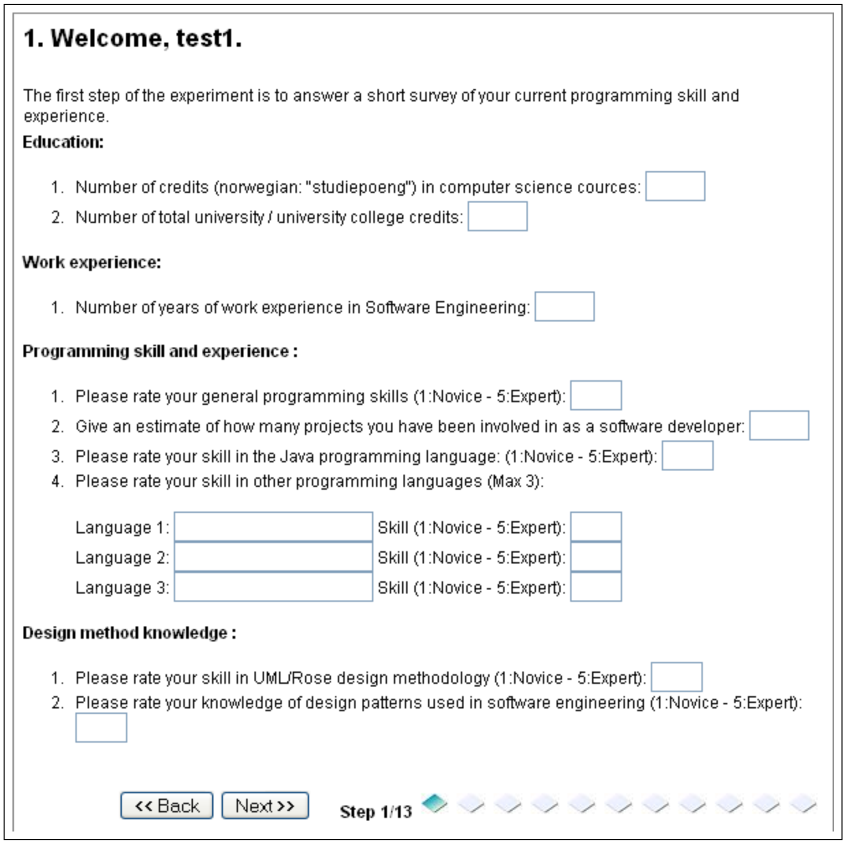
News 01.03.2006
The experiment will be set up at computer lab 424 on P15 the 27.march from 09.00 - 16.00.

News 20.02.2006
ExCom 2006 has agreed to participate in the experiment!

News 16.01.2006
The master thesis has officially been assigned!

© 2006 - Ingvar Ljosland | Web Design template by Ashley Johnson

Figure 5.3: Login web page



1. Welcome, test1.

The first step of the experiment is to answer a short survey of your current programming skill and experience.

Education:

1. Number of credits (norwegian: "studiepoeng") in computer science cources:
2. Number of total university / university college credits:

Work experience:

1. Number of years of work experience in Software Engineering:

Programming skill and experience :

1. Please rate your general programming skills (1:Novice - 5:Expert):
2. Give an estimate of how many projects you have been involved in as a software developer:
3. Please rate your skill in the Java programming language: (1:Novice - 5:Expert):
4. Please rate your skill in other programming languages (Max 3):

Language 1: Skill (1:Novice - 5:Expert):
Language 2: Skill (1:Novice - 5:Expert):
Language 3: Skill (1:Novice - 5:Expert):

Design method knowledge :

1. Please rate your skill in UML/Rose design methodology (1:Novice - 5:Expert):
2. Please rate your knowledge of design patterns used in software engineering (1:Novice - 5:Expert):

Figure 5.4: Survey web page

5.3.2 Surveys

When the subjects are identified, a survey can be made and the results are stored in the experiment database. The advantage of a web tool compared to pen and paper is that first of all, the results stored in the database can be exported to a spread sheet and analysed directly without any manual operations. Secondly, the web tool can check the integrity of the data, preventing any subjects from mistyping, as an example typing "6" to an answer where the range is "1" to "5".

Figure 5.4 shows how the subject experience survey is presented in the web tool.

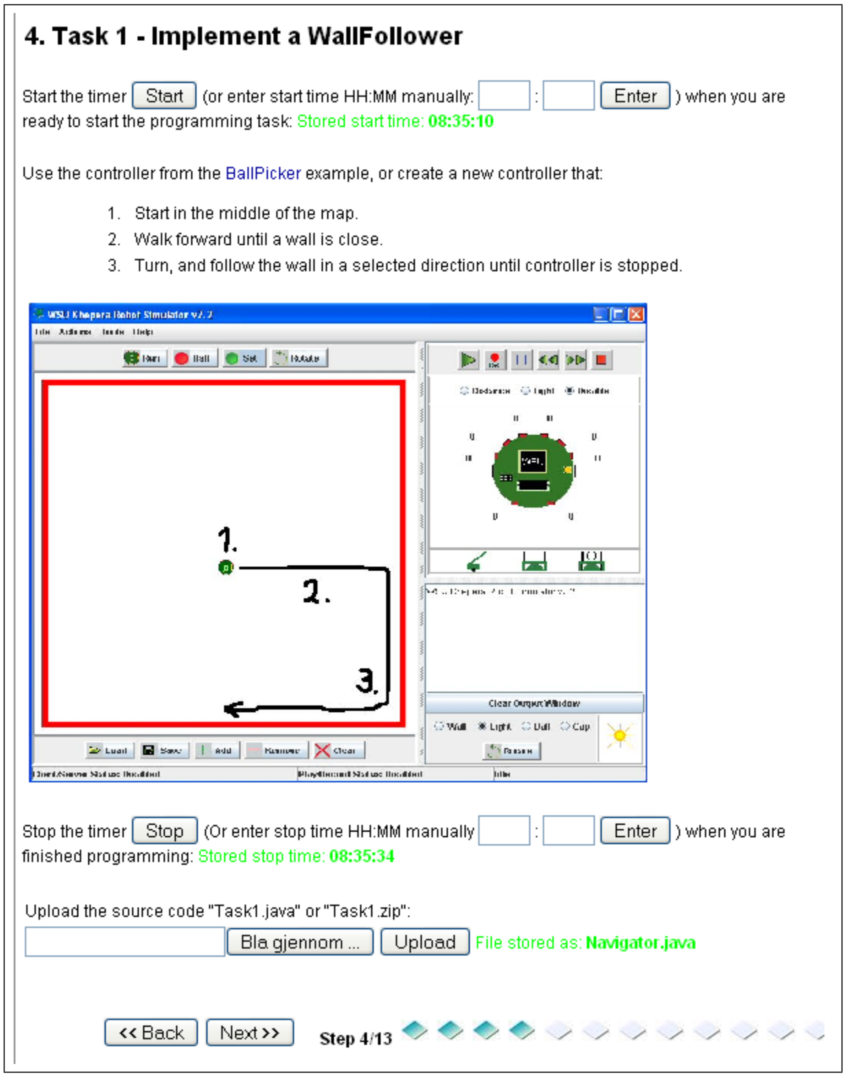
5.3.3 Tasks

Each subject belongs to either a Type 1 experiment or a Type 2 experiment. The web tool must query the database for the subjects experiment type before the correct task is presented. If the subject is in the Type 1 group, the first two functionality solving problems are presented before the safety-critical tasks, and visa versa for the Type 2 group.

Figure 5.5 shows how the pre-task is presented in the web tool. As the figure shows, there is actually more to the presentation of the task than just information. There is also a start and stop timer, and a task upload functionality implemented in the page. The start timer stores the time when the subject are ready to start solving the task in the experiment database. The stop timer saves the time when the subject has completed the task. This makes it possible to know how much effort, measured in working minutes, each subject has used to solve the tasks. When the subject has correctly has stopped the task timer, the stored value is shown in green.

The upload function makes it possible for the subjects to select a file that can be uploaded to the experiment file server. When they have completed the task, they can upload their task solution. The green typing shows the stored filename of the task, as shown on the figure.

After the subject has completed the task, they are presented a post-task survey. This survey is shown in Figure 5.6. The survey answers are stored in the experiment database in the task entity.



4. Task 1 - Implement a WallFollower

Start the timer Start (or enter start time HH:MM manually: __ : __ Enter) when you are ready to start the programming task: Stored start time: **08:35:10**

Use the controller from the BallPicker example, or create a new controller that:

1. Start in the middle of the map.
2. Walk forward until a wall is close.
3. Turn, and follow the wall in a selected direction until controller is stopped.

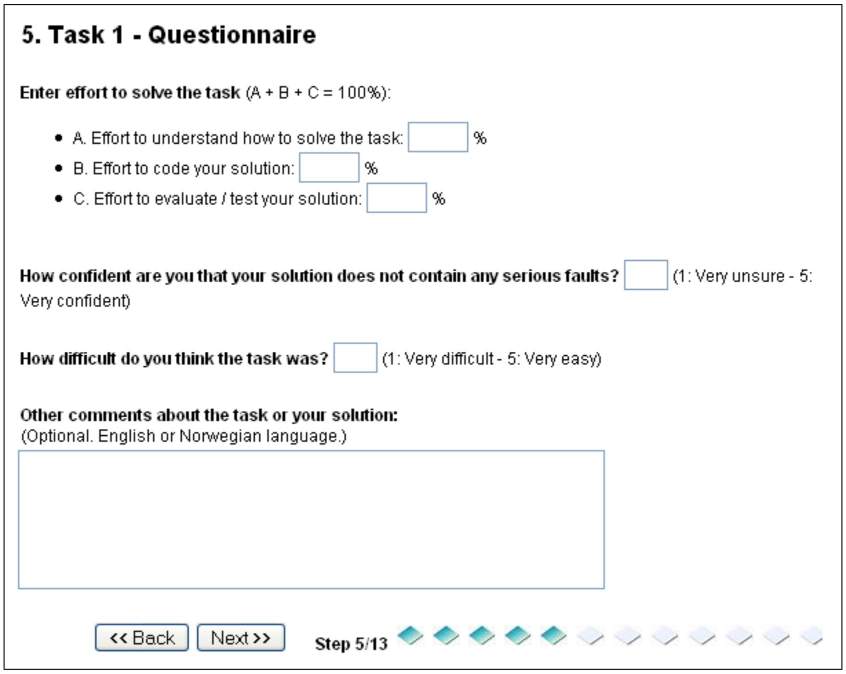
Stop the timer Stop (Or enter stop time HH:MM manually __ : __ Enter) when you are finished programming: Stored stop time: **08:35:34**

Upload the source code "Task1.java" or "Task1.zip":

Bla gjennom ... Upload File stored as: **Navigator.java**

<< Back Next >> **Step 4/13**

Figure 5.5: Task web page



5. Task 1 - Questionnaire

Enter effort to solve the task (A + B + C = 100%):

- A. Effort to understand how to solve the task: □ %
- B. Effort to code your solution: □ %
- C. Effort to evaluate / test your solution: □ %

How confident are you that your solution does not contain any serious faults? □ (1: Very unsure - 5: Very confident)

How difficult do you think the task was? □ (1: Very difficult - 5: Very easy)

Other comments about the task or your solution:
(Optional. English or Norwegian language.)

Figure 5.6: Post-task survey web page

Chapter 6

Results

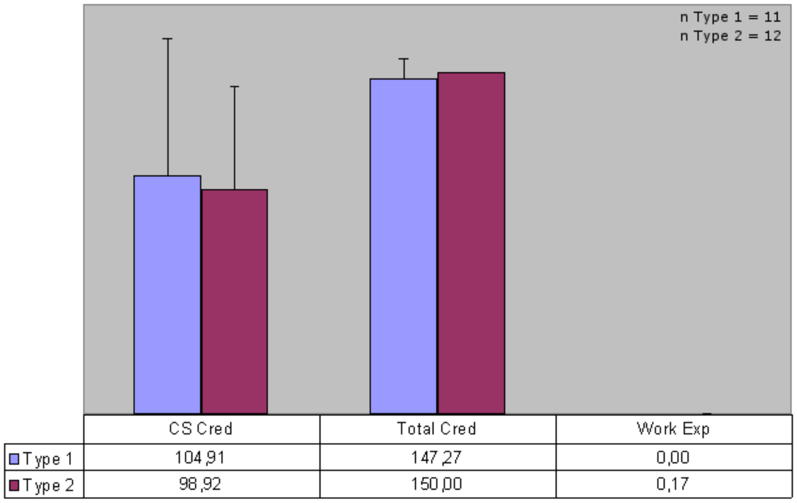
This chapter examines the results from the experiment. The first section gives some descriptive data of the subjects who participated in the experiment. This data are taken from survey answers. The following section formally tests the hypothesis outlined in Section 4.2, using the statistical model from Section 2.2. The final section sums up the results from the descriptive data and hypothesis tests, and interprets the results in the experiment context.

6.1 Descriptive data

The descriptive data give us some basic information about the subjects; their background in software engineering such as programming and design skills, university credits and working experience.

The subjects answered a survey at the first step of the experiment, as outlined in Section 4.1. Figure 6.1 show the results from this survey. The figure divides the subjects in two groups, the Type 1 group were the subjects that got the functionally tasks presented first, and the safety-critical tasks subsequently, and the Type 2 group that got the tasks in the reversed order. By showing some descriptive data about the two groups, we can see if there are some differences in skill level and experience between them.

We can see from the figure that there is not much difference in skill and experience between the two groups. This is as expected since both groups consist of third year computer-science students. The graph is explained as follows: The CS Cred column is the number of student points that the subject have in computer science courses, and the Total Cred column is the total number of students points that the subject has completed (30 points per term in the Norwegian university system). Work Exp is the number of years that the subjects have worked in a computer science or engineering related job. The table result shows that the students have little work experience (two subjects had one year each of work experience), as one can expect by the student population. Figure 6.2 shows some



	CS Cred	Total Cred	Work Exp
Type 1	104,91	147,27	0,00
Type 2	98,92	150,00	0,17

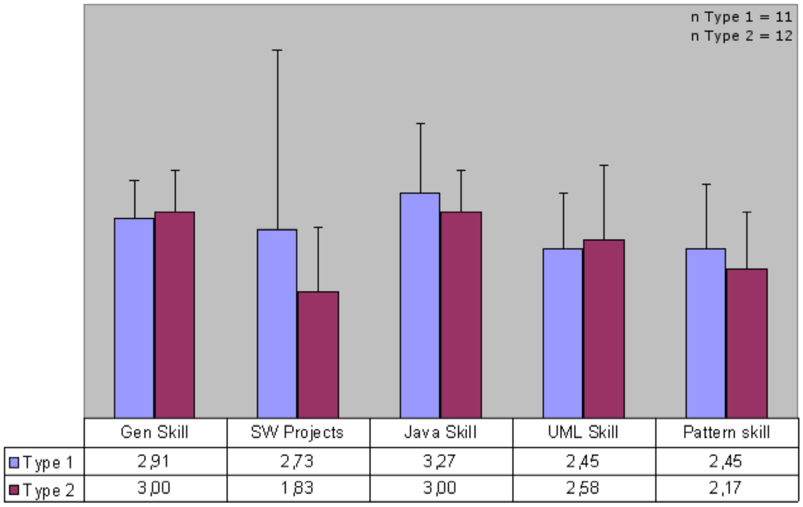
Figure 6.1: Descriptive data of working experience and university credits

descriptive data about the subjects programming and software architecture skills. The columns Gen Skill and Java Skill corresponds to what the subjects considered to be their own skill in the Java programming language, and more general programming skill. The students could also list up a few other programming languages, but these results where considered insignificant and the General skill column should be representative for these programming languages. As the subjects where supposed to rate their skill from one (Novice) to five (Expert), the table shows that both the groups have what would be considered as an average good programming experience (Mean value 3.27 in the Java programming language and 2.91 in general programming skill for the Type 1 group, and mean values 3.0 in Java and general programming skills for the Type 2 group).

As the tasks also involved adding safety-critical design pattern implementations, the subjects where asked to rate their skill in UML modelling language skill and their general knowledge of using architectural design patterns in software engineering. The result figure show that the subjects skill in these fields are less than the average skill in programming (Mean values 2.45 in UML skill and 2.45 in design pattern knowledge for the Type 1 group, and mean values 2.58 in UML skill and 2.16 in design pattern knowledge for the Type 2 group).

One noticeable result is that the number of software projects that the subjects have attended varies. One subject with experience from ten software projects draws the mean value of the Type 1 group up a bit.

There is no significant differences between the groups level of experience and programming skills. This are supported by t-tests, and can be found in Appendix D.



	Gen Skill	SW Projects	Java Skill	UML Skill	Pattern skill
■ Type 1	2,91	2,73	3,27	2,45	2,45
■ Type 2	3,00	1,83	3,00	2,58	2,17

Figure 6.2: Descriptive data about programming and software architecture skills

6.2 Task Results

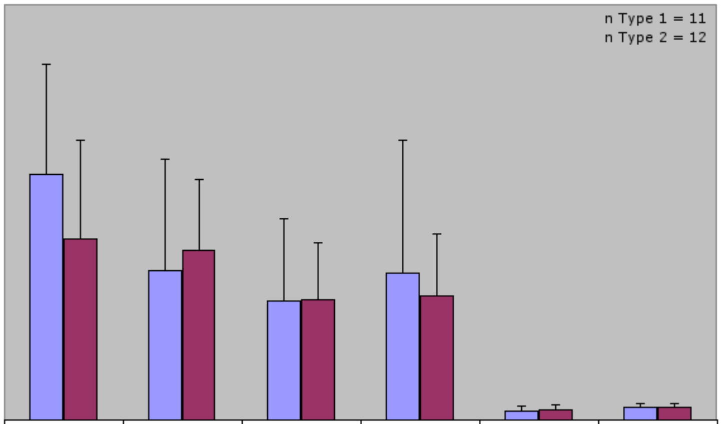
The experiment consisted of five programming tasks as described in Section 5.3.3. This section will give some statistical information about the results of the task in form of change effort, correctness and the subjects opinions on the post-task surveys.

6.2.1 Pre-task / training task

The pre-task had two purposes; to determine the subjects skill level in programming, and to be a training exercise for the robot emulator and the software development environment.

Figure 6.3 show the comparison between the Type 1 group and the Type 2 group on the time spent solving the task (column Time, measured in minutes), the effort to understand the task (column Ef und, measured in percentages), the effort to code the task solution (column Ef code, measured in percentages), the effort to test the solution (column Ef test, measured in percentages), the subjects rating of how confident they are that their solution do not containing any serious faults (column Fault, measured from 1 - Very unsure to 5 - Very confident), and the last column showing how difficult the subjects found it to solve the task (measured from 1 - Very easy to 5 - Very difficult).

As one can see by the figure, the Type 2 group has a slightly less mean time to solve the task. They also have the fastest solution time, 15 minutes, in contrast to a subject in the Type 1 group that used the maximum time of 124 minutes. This shows that there is some differences in the skill level between the students. Both of the groups rated the difficulty of solving the task at a medium level (mean



	Time	Ef und	Ef code	Ef test	Fault	Difficulty
■ Type 1	59,27	35,91	28,64	35,45	2,27	3,00
■ Type 2	43,58	40,83	29,17	30,00	2,58	3,08

Figure 6.3: Pre-task descriptive data

values 3.0 and 3.1).

Figure 6.4 shows the correctness of the solutions. The first columns represent the correctness in form of whether the correct functionality was implemented or not. When the solutions were evaluated, a one-value would represent a correct working solution, and a zero-value represents an incorrect or incomplete solution. Two subjects did not upload their solution to the experiment file server, and thereby got a zero score on the functionality test and did not get counted in the Overall correctness evaluation. This is further discussed in Section 7.2.2.

The Overall correctness column on the table shows how the subjects score on a Java code-inspection check list found in Appendix A. The t-test results, found in Appendix D, support that there is no significant differences in the time spent of solving the task, and the correctness of the source code.

Comments given by the subjects imply that they found it most difficult to understand how the robot emulator worked, with the use of sensors, controllers and debugging. Example of comments are (translated from Norwegian): “It took a long time to understand how to solve the task, and I had a hard time setting up the environment with the compiling/moving of files”, “It was difficult to find good (sensor) measurements”, “It was unclear to me where to make the changes, but all in all, it went pretty ok”, and “..it was difficult to understand how the sensor values vary to find the right solution.”

6.2.2 Functionality Solving Task 1

In this task, as described in Section 4.3.2, the subjects had to implement a robot controller that avoids lights. The Type 1 group where given this task straight after the pre-task, while the Type 2 group got this task after they had been given

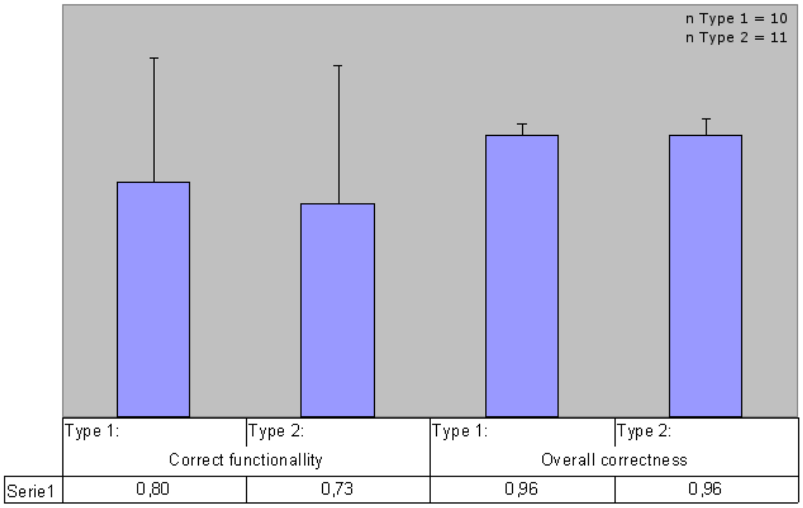


Figure 6.4: Pre-task correctness

two tasks of implementing safety-critical software patterns to the robot controller.

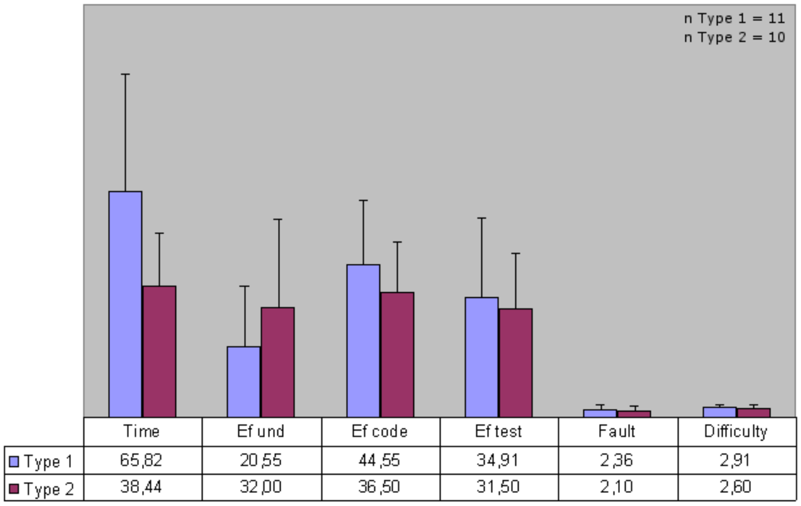
Figure 6.5 shows some statistical information of how much time the subjects spent on solving the task; how they used their effort on in solving it, how confident they are that their solution does not contain any serious faults, and how difficult they thought it was to solve the task. Column explanations are the same as for Figure 6.3 in the previous subchapter.

The figure shows that the Type 2 group uses slightly less time on solving the task (Mean value 65.8 minutes for Type 1, versus 38.4 minutes for Type 2). The Type 2 group also used slightly more effort to understand the task than the Type 1 group, hence less effort of coding and testing.

Figure 6.6 shows how many subjects that had a correct implementation of the functionality given in the task, found from the code inspection check-list. In the Type 1 group, one person did not upload the source code to the experiment file server, and for the Type 2 group this number raised to five. The reason for this is most likely that the Type 2 group did this task as task number four in the experiment, and some subjects did not get time enough to completely finish this task. However, nine of the twelve subjects from the Type 2 group answered the post-task survey, meaning that three of these subjects had this task completely blank.

The results from the correctness show that there is some difference between the two groups. The Type 1 group has slightly less correct implementations of functionality (70% for the Type 1 group versus 86% for the Type 2 group), but the standard deviation on 15% is too high to draw quick conclusions.

The t-test listed in Appendix D support the statement that there is no significant difference between the two groups in the correctness of the source code, but the test show a significant difference in the time used to solve the task. This



	Time	Ef und	Ef code	Ef test	Fault	Difficulty
🟦 Type 1	65,82	20,55	44,55	34,91	2,36	2,91
🟥 Type 2	38,44	32,00	36,50	31,50	2,10	2,60

Figure 6.5: FST1 descriptive data

is further discussed in Section 6.3.2.

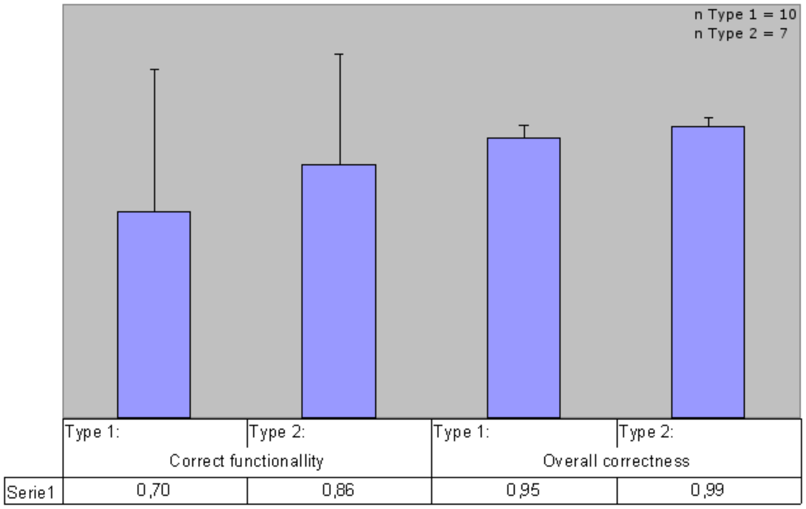
Comments given by the subjects imply that this task was difficult for them mostly because the emulation of the light sensors is variable and unreliable. There were also some difficulties in the logic programming of a robot walking randomly around the room. Examples of comments are (translated from Norwegian): "I think this was a difficult task. I could not solve it completely. I am still unsure of how the sensors work", "It was difficult to adjust to the light", "The light (sensors) does not work very well. The standard value is set to 500, but there is not much difference between far and close to light..", and "Random walking is more difficult than it might appear."

6.2.3 Functionality Solving Task 2

This was the second of the two tasks for solving functionality related problems. The task is described in Section 4.3.3, and involves creating a robot controller that finds its way trough three light gates, pick up a ball, and return to its original base.

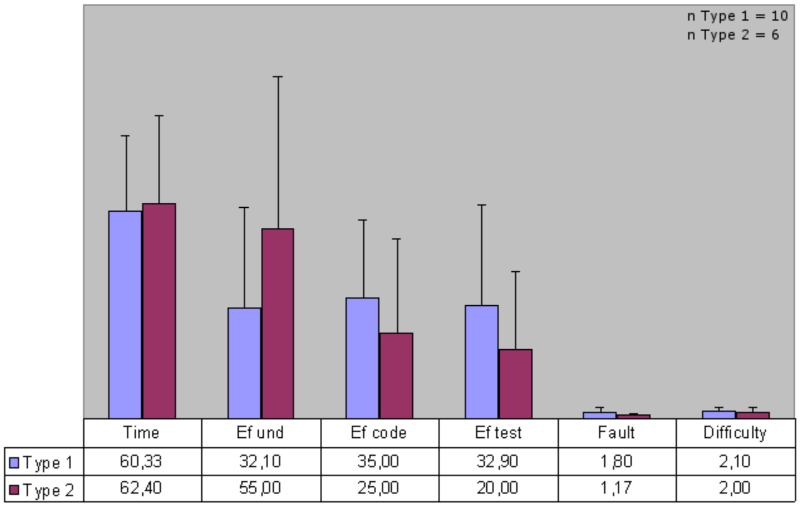
Figure 6.7 shows how the subjects performed on this task. The first thing to notice is that the average time spent on solving this task is quite similar between the two groups (Mean value 60.3 minutes for the Type 1 group, and 62.4 minutes for the Type 2 group). The shortest time spent on solving the task was done by a subject from the Type 2 group that used 19 minutes.

Another thing to notice is that the subject thought this task was more difficult than the previous functionality solving task (Mean values 2.1 and 2.0 on this task versus 2.9 and 2.6 on the FST1 task). They are also less confident that their solution is correct (Mean values 1.8 and 1.2 on this task versus 2.3 and 2.1 on



	Type 1:	Type 2:	Type 1:	Type 2:
	Correct functionallity		Overall correctness	
Serie1	0,70	0,86	0,95	0,99

Figure 6.6: FST1 correctness



	Time	Ef und	Ef code	Ef test	Fault	Difficulty
🟦 Type 1	60,33	32,10	35,00	32,90	1,80	2,10
🟥 Type 2	62,40	55,00	25,00	20,00	1,17	2,00

Figure 6.7: FST2 descriptive data

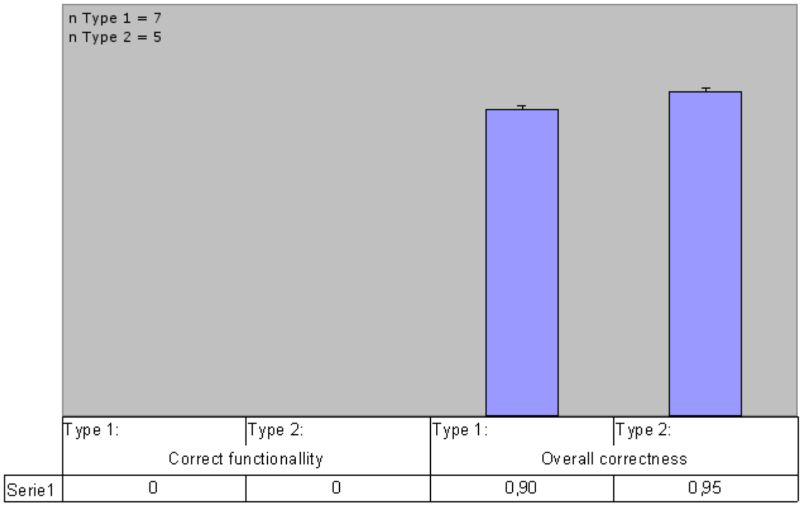


Figure 6.8: FST2 correctness

the FST1 task).

Since this task was the last task to solve for the Type 2 group, only five of the twelve subjects in this group did completely solve this task. The rest of the subjects did not have time to finish the task due to the experiment time limitation of six hours. As Figure 6.8 indicates, none of the subject from neither the Type 1 group nor from the Type 2 group did actually fully implement the functionality problems that was listed in the task.

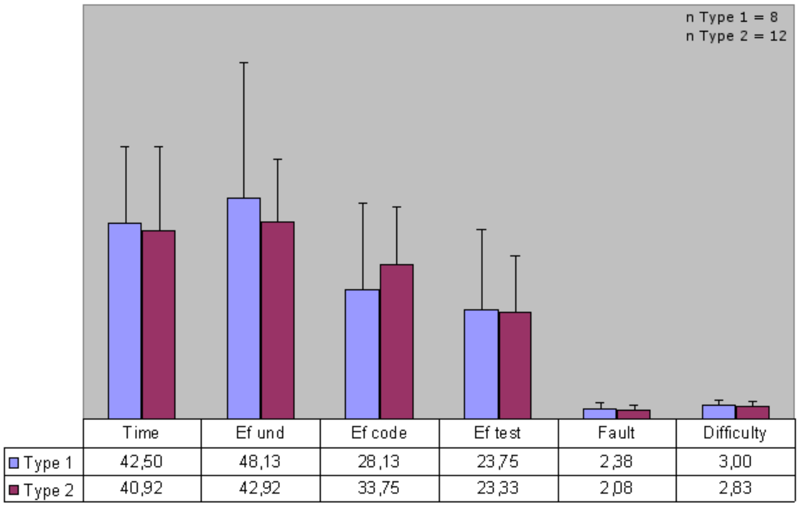
Figure 6.8 also shows that for those subjects that actually did deliver a solution to the task, the Overall correctness is still high, meaning that the subjects did not do much classical programming errors found in the code inspection checklist, the problem was rather how to logically solve the task with the use of robot sensors.

Comments given by the subjects support this argument, with examples like (translated from Norwegian): “I had problems with the lights in the simulation. It did not show much difference in light values even though the robot was right next to a light source..”, “.. I tried to navigate relative by the light sources, but I could not get it to work. (..) My next try was to navigate with the use of the walls and turn when it hit the light source, but I could not get it to follow the wall properly either, and I gave up.”, and “.. I did not know how to approach the task.”

6.2.4 Safety-critical Task 1

As the first of the safety-critical tasks, this assignment was to implement a Single Channel control pattern as described in Section 4.3.4.

Figure 6.9 shows some descriptive statistics about the task. The Type 1



	Time	Ef und	Ef code	Ef test	Fault	Difficulty
■ Type 1	42,50	48,13	28,13	23,75	2,38	3,00
■ Type 2	40,92	42,92	33,75	23,33	2,08	2,83

Figure 6.9: SCT1 descriptive data

group solved this task as task number four, thus some of the students from this group did not upload their solution to the experiment file server because they did not have time enough to finish the task. Two students that participated in the post-task survey have not filled in how much time they had spent on solving the task.

The time used of solving the task is quite equal for the two groups (Mean value 42.5 minutes for the Type 1 group, and 40.9 minutes for the Type 2 group). The same situation applies to the effort for understanding, coding and testing the task. One thing to notice about this task is that, compared to the task from the previous subsection, the subjects do now think that this task is a bit less difficult than the last one, raising the mean rating of difficulty from 2.0 and 2.1 in the FST1 task to 3.0 and 2.83 in this task (remembering the range of 1-Very difficult to 5-Very easy). The t-tests found in Appendix D support the statement that there is no significant differences between the two groups.

Figure 6.10 show how the subject performed in regards of correctness. Of the seven subjects that completed the task in the Type 1 group, only four of these had a correct implemented solution, as shown by the figure. The Overall correctness is still quite high, meaning that there were not much classical coding errors. The comments from the subjects are implying that they used a great deal of time to understand the task, but when they figured out how to solve it, it appeared as quite simple task to solve. Some problems during the testing of the code made the subject use a bit more time here, because the robot emulator does not use a safe implementation of Java threads, and the simulator had a tendency to crash after some minutes of testing.

Examples of comments written by the subjects are (translated from Norwegian): “I used some time to understand the task. A lot of time also went to

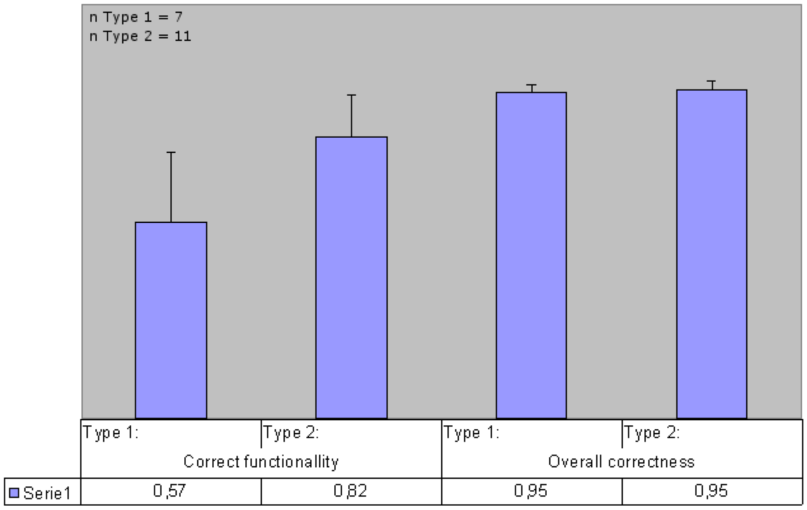


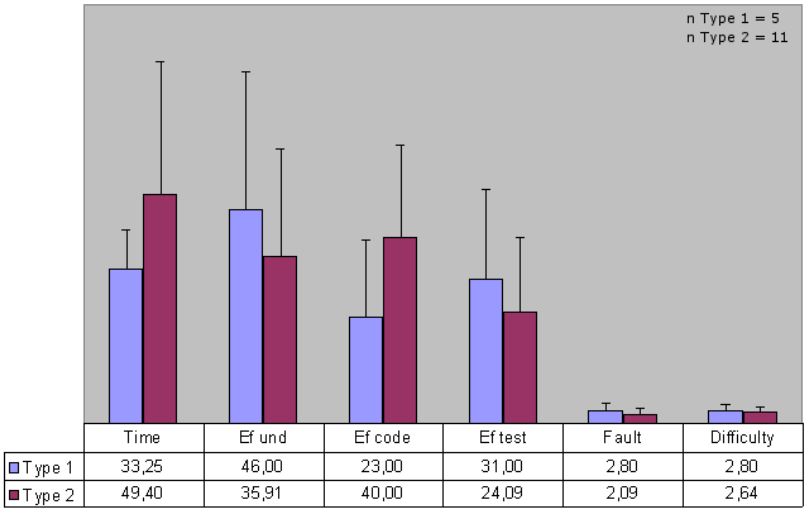
Figure 6.10: SCT1 correctness

testing the code because I had forgotten to update the distance vector.", "It wasn't so difficult when I took a look of how the light values were implemented", "I used a lot of time to understand the task..", "It was difficult to understand the task. I did not know how to solve it.", "I got a null pointer during the execution, and I don't know why.", and "My solution does not contain any thread safety, that's why the simulator crashes after a while".

6.2.5 Safety-critical Task 2

The second safety-critical task was to implement a Watchdog pattern as described in Section 4.3.5. Figure 6.11 shows some descriptive statistical data for the task. The first thing to notice is that for the Type 1 group, whitch did this task as the last task in the experiment, only five of eleven subjects answered the sub-task survey. Four subjects of this group registered the time used on solving the task while only two of these uploaded their solution to the experiment file server. On the other hand, the Type 2 group did this task as task number three, and only one subject did not answer the post-task survey. Two subjects did not register the time spent on solving the task, and these two did not upload their solution as well. The result implies that there were not much time left of the experiment when the Type 1 group came to this step in the experiment.

The figure also shows that the results of this task are varied: One subject used 15 minutes of solving this task, while the maximum time spent was 98 minutes. One subject stated that they used 100% of the effort to understand the task, while another clamed that he used 10% of the effort to understand the task, 50% of solving and 40% of testing. The difficulty and assurance of correct solution is also varied, given the fact that the minimum score on these questions was rated



	Time	Ef und	Ef code	Ef test	Fault	Difficulty
Type 1	33,25	46,00	23,00	31,00	2,80	2,80
Type 2	49,40	35,91	40,00	24,09	2,09	2,64

Figure 6.11: SCT2 descriptive data

one, and the maximum rate given by a subject was five (knowing that the range of this question was 1 - very unsure / very difficult to 5 - very confident / very easy).

As a result of the thin data, this task is pretty much useless as a comparison between the two groups. As Figure 6.12 shows, only one of the two subjects from the Type 1 group that uploaded their solution did actually have a correct implementation of the task.

Comments given by the subject support the fact that there was little time left to complete this task for the Type 1 group. The Type 2 group found this task not so difficult, and pretty fun task to do. Examples of comments are (translated from Norwegian): “It was pretty ok, I only messed up some, making it take a bit longer to solve”, “I used a bit time to tune the coding from the previous tasks”, “I’m not 100% sure that the method for getting the robot free from obstacles really work, but it knows at least when its stuck”, “I am unsure of how to solve this task. I was stuck at the end (of the experiment), and delivered what I had”, “This was too difficult, I haven’t enough programming experience to solve this”, “Not enough time”, and “This task was fun”.

6.3 Hypothesis Tests

In this section the hypothesis from Section 4.2 will be tested and evaluated. As the results suggest that the subjects did not have enough time to solve all the five task during the experiment time limit of six hours, the last tasks of the Type 1 group (SCT2) and the last task of the Type 2 group (FST2) are disregarded in the hypothesis tests, because of the thin data.

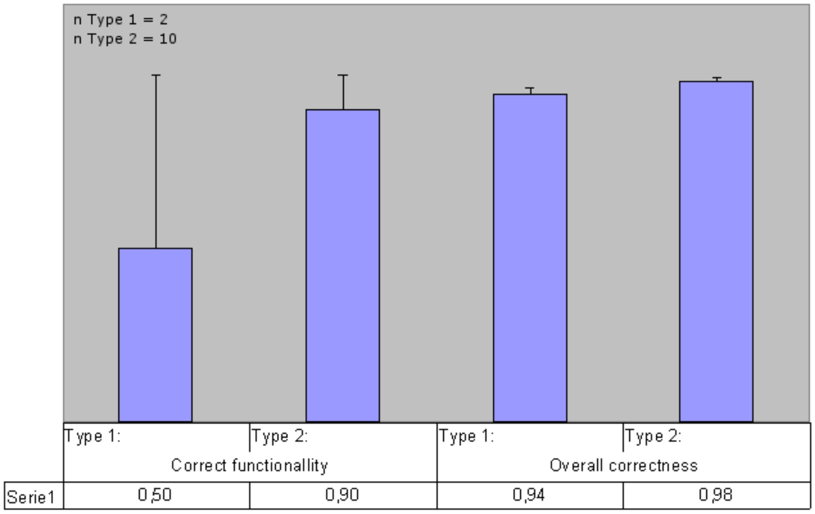


Figure 6.12: SCT2 correctness

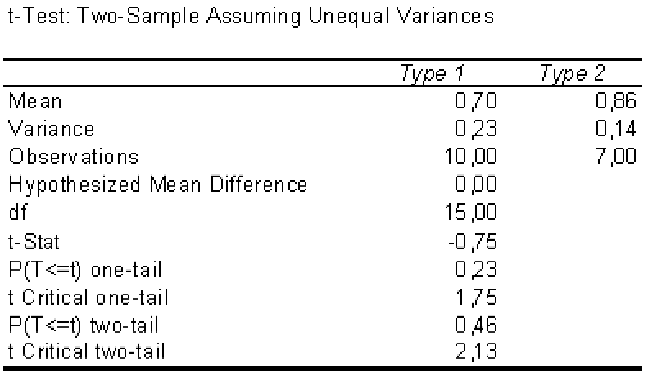
6.3.1 H0$_1$: The Effect of Design Approach on Correctness

As the tasks were evaluated, the solutions that was correctly compiled and tested got a one-rating on the correct functionality column, while the solutions that had not fully or incorrect implementations got a zero-rating at this test. The null hypothesis suggests that the mean difference between the subjects that solved the functionality related problems first and the group that solved the safety-critical problem first should be equal.

The t-test result from the first functionality solving task (FST1) is shown in Table 6.1. As one can see by the table, the t-Stat absolute value of 0.75 is less than the t Critical one-tail value of 1.75, meaning that the null hypothesis cannot be rejected in this case. In addition, the P(T≤t) one-tail value of 0.23 suggests that if we do reject the null hypothesis, it is a 23% possibility that we do a wrong decision, so that is obviously too risky, and would make a high posibilty of getting a Type-I-error.

When looking at the first of the safety-critical tasks (SCT1), we can see that the same situation applies in this case. The t-test for this task listed in Table 6.2. With a t-Stat value of 1.05 that is less than the t Critical one-tail value of 1.81, the null hypothesis cannot be rejected for this task either. The P(T≤t) one-tail of 0.16 shows that there is a 16% chance of being wrong if we do reject the null hypothesis.

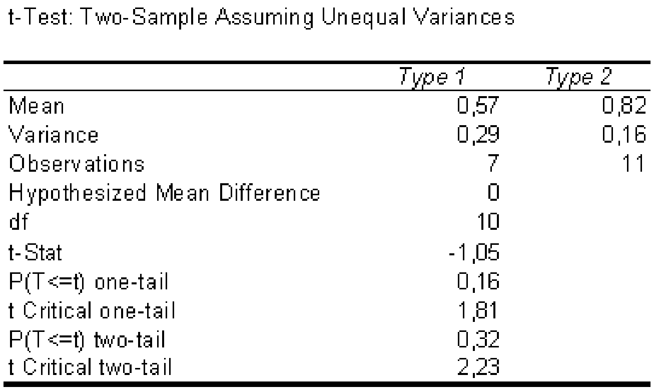
This means that we cannot reject the null hypothesis and say that the design approach effected the correctness.



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	0,70	0,86
Variance	0,23	0,14
Observations	10,00	7,00
Hypothesized Mean Difference	0,00	
df	15,00	
t-Stat	-0,75	
P(T<=t) one-tail	0,23	
t Critical one-tail	1,75	
P(T<=t) two-tail	0,46	
t Critical two-tail	2,13	

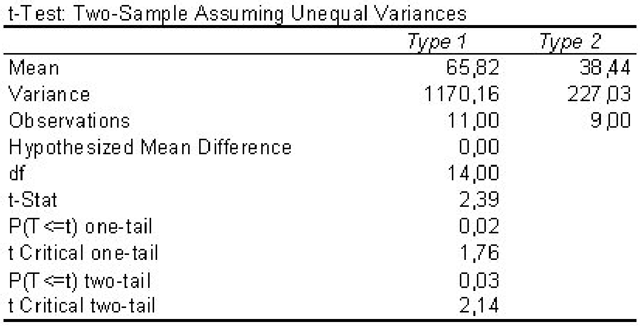
Table 6.1: t-test of FST1 on correctness



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	0,57	0,82
Variance	0,29	0,16
Observations	7	11
Hypothesized Mean Difference	0	
df	10	
t-Stat	-1,05	
P(T<=t) one-tail	0,16	
t Critical one-tail	1,81	
P(T<=t) two-tail	0,32	
t Critical two-tail	2,23	

Table 6.2: t-test of SCT1 on correctness



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	65,82	38,44
Variance	1170,16	227,03
Observations	11,00	9,00
Hypothesized Mean Difference	0,00	
df	14,00	
t-Stat	2,39	
P(T<=t) one-tail	0,02	
t Critical one-tail	1,76	
P(T<=t) two-tail	0,03	
t Critical two-tail	2,14	

Table 6.3: t-test on effort of solving FST1

6.3.2 $H0_2$: The Effect of Design Approach on Change Effort of Functionality

The change effort was measured as time needed to complete the task. When the subjects started on a task, they were asked to start a task timer. When they had completed the task, and uploaded their solution to the experiment file server, they could stop the timer. The subjects also answered a post-task survey where they weighted their efforts to solve the task in three categories: Effort to understand the task, effort to code the solution, and effort to test the solution.

The t-test results from the mean time effort, listed in Table 6.3, shows that there is a significant difference in the change effort between the two groups. The t Critical one-tail value of 1.76 is smaller than the t-Stat value of 2.39, meaning that we can reject the null hypothesis. The P(T≤t) one-tail show that we can be 98% confident of this decision.

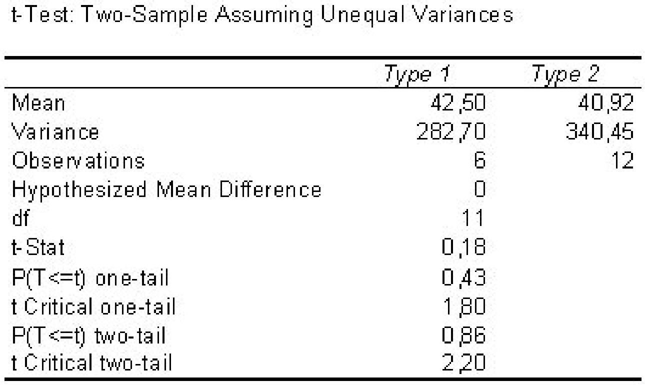
The Type 2 group uses significantly less time effort of making changes to the first functionality task, than the Type 2 group.

6.3.3 $H0_3$: The Effect of Design Approach on Change Effort of Safety-Critical Design

This hypothesis suggests that the effort of adding safety-critical modifications to a system should be higher the more complexity and functionality that has been added.

The results from the SCT1 are analyzed with a t-test, and are listed in Table 6.4. The table shows that the t Critical one-tail value of 1.80 is too high compared to the t-Stat value, meaning that we cannot reject the null hypothesis. If we do reject the null hypothesis, the P(T≤t) one-tail value is showing us that there are 43% chance that this would be a wrong decision and make a Type-I-error.

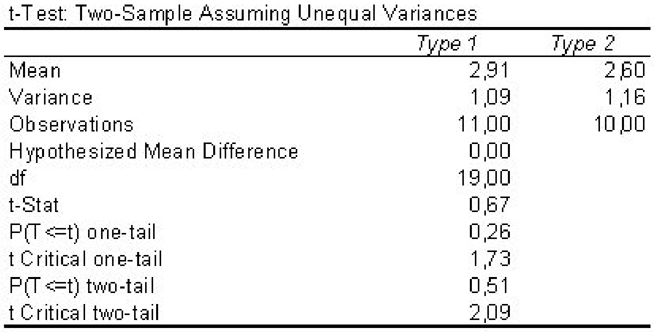
This means that there are not found any evidence that there is differences of change effort when solving a safety-critical task, whether additional functionality has been implemented or not.



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	42,50	40,92
Variance	282,70	340,45
Observations	6	12
Hypothesized Mean Difference	0	
df	11	
t-Stat	0,18	
P(T<=t) one-tail	0,43	
t Critical one-tail	1,80	
P(T<=t) two-tail	0,86	
t Critical two-tail	2,20	

Table 6.4: t-test on effort of solving SCT1



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	2,91	2,60
Variance	1,09	1,16
Observations	11,00	10,00
Hypothesized Mean Difference	0,00	
df	19,00	
t-Stat	0,67	
P(T<=t) one-tail	0,26	
t Critical one-tail	1,73	
P(T<=t) two-tail	0,51	
t Critical two-tail	2,09	

Table 6.5: t-test on difficulty of solving FST1

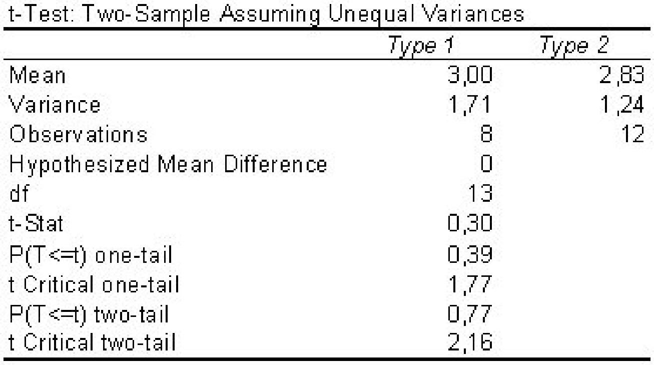
6.3.4 H0$_4$: The Effect of Design Approach on Degree of Difficulty

The degree of difficulty was measured from the post-task survey answers. The hypothesis suggests that there should be, as for the change effort and correctness, more difficult to add safety-critical modules when a software systems complexity has risen with the added functionality. There should also be more difficult to solve functionality problems when the complexity of safety-critical additions has been added.

Table 6.5 shows the t-test on the mean difference between the answers of difficulty on the first of the functionality solving tasks. The t Critical one-tail value of 1.73 is too high compared to the t-Stat value of 0.67, so we cannot reject the null hypothesis in this case either.

The same situation applies for the t-test of the first of the safety-critical change tasks, as presented in Table 6.6. The t Critical value of 1.77 is greater than the t-Stat value of 0.30, and rejecting the null hypothesis is not possible.

The results from these t-tests show that there are not enough differences be-



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	Type 2
Mean	3,00	2,83
Variance	1,71	1,24
Observations	8	12
Hypothesized Mean Difference	0	
df	13	
t-Stat	0,30	
P(T<=t) one-tail	0,39	
t Critical one-tail	1,77	
P(T<=t) two-tail	0,77	
t Critical two-tail	2,16	

Table 6.6: t-test on difficulty of solving SCT1

tween the two groups, and we cannot reject the null hypothesis. This means that the subjects did not feel that making functionality additions to a software system is more difficult after the two safety-critical additions were made.

Additionally, the subjects that implemented two functionality tasks first did not find it more difficult to add the safety-critical task than group that had not made the changes.

6.4 Summary of Results

This section sum up the results presented in the above sections.

The descriptive data about the subject population showed that they, in their own opinion, had a medium programming experience with a mean score of three from a rating of one to five. Only two of the twenty-three subjects had one year or more of working experience. This is as expected since the subjects were third-year students.

The experiment consisted of five tasks, where one task was a pre-test task and training task to the experiment environment, two tasks was related to functionality solving, and two tasks was to improve the system with safety-critical additions. The subjects had six hours of solving the five tasks, and this was a little too tight time limit for some of the students. The subjects were split in two groups, and the first group (Type 1) solved the functional related tasks first, then the safety-critical task, and the other group (Type 2) solved the tasks in the reverse order. In the Type 1 group, only two of the eleven subjects did actually deliver a solution to the last task. For the Type 2 group, only five of twelve subjects completed the last task.

In the functional solving tasks there were not much programming errors found by using the code inspection check-list. The errors found were logic related, like reading the correct robot sensors, and understanding of the robot controller tasks. There were also some bugs in the robot emulator, like unstable light values and missing thread safety, which created some extra difficulties for the subjects.

There were four hypothesis presented in Section 4.2. The first hypothesis suggested that the subjects that implemented the functionality solving tasks first should have less correctness in the safety-critical implementations, due to higher system complexity. The subjects that implemented the safety-critical additions first and then solved functionality problems should have better performance when solving functional problems due to better system robustness. The results showed that the data did not support this hypothesis, and we cannot say that there were any differences in the correctness of these tasks.

The second hypothesis suggested that the effort of changing functionality in a system should be lower with robust safety-critical implementations. The result data of the first functionality solving task (FST1) support this hypothesis by stating that there are 98% certainty that the Type 2 group used less time of solving this task than the Type 1 group.

The third hypothesis suggests that the effort of adding safety-critical implementations should rise when a systems complexity has risen with functionality additions. The results from the first of the safety-critical tasks (SCT1) show that there is not any significant differences in the mean effort between the two groups, meaning that the hypothesis is not supported by the results.

The final hypothesis claims that the difficulty of making safety-critical additions to a system rise with added functionality. Also, with safety-critical additions, the difficulty of changing functionality related problems should rise. The results from the tasks post-test answers show no significant differences between how difficult the subjects find the tasks, meaning that the data does not support this hypothesis either.

Chapter 7

Threats to Validity

When conducting experiments, it is important to identify any threats to the validity of the experiment before making any conclusions. This chapter will discuss the possible threats to the construct, internal, external and conclusion validity of the experiment.

7.1 Construct Validity

The construct validity is, as mentioned in Section 2.1.4, the validity concerned with the relationship between the treatment and the outcome.

7.1.1 Classification of the Tasks

The first threat to the construction validity is whether the treatment and the variables in the experiment design actually was representative for the concept studied or not. Are the functional solving tasks really only focused on solving functionality and nothing else? Are the safety-critical tasks only concerned with adding additional safety and not adding functionality as well?

To have some sort of realistic environment, a robot emulator was chosen to represent the task framework of the experiment. The first functionality solving task was to avoid light. To solve this task, the use of the robot sensors had to be used to navigate. The second functionality task was to make the robot navigate trough three gates, pick up a ball, and return home. Both of these tasks assume that the values from the robot sensors are correct, and the challenge in solving these tasks lay on the subjects logical understanding of the robot controller and its sensors.

In the safety-critical tasks, there were no indications of what functional problems the robot had to solve, only additions that made validation checks of the sensor readings or other safety-critical additions. The first safety-critical task was to implement a Single Channel Pattern, and the second task was to implement a Watchdog Pattern, as described in Section 3.2.1 and Section 3.2.2 respectively. The subjects were asked to test their solutions with robot controllers from previous tasks, making no implications that additional functionality should be added to the system. These patterns were taken from [Dou03], and are considered to be good representations of light-weight safety-critical additions.

7.1.2 Classification of Treatment Groups

To make sure that the effect of the treatment did not get affected by external sources, the group of subjects was divided into two equally large groups. The results from the survey answers in Section 6.1, and the pre-test results from Section 6.2.1, show that there is no significant difference between the skill level or experience of the two groups, and we will assume that this applies for the whole experiment.

7.1.3 Measures of Variables

Several variables were measured in the experiment. The measure of correctness was how well the task solutions scored on a functional test, and results from a check-list. If the task was correctly solved the subjects got a one, otherwise they got a zero value on the test result. The check-list is shown in Appendix A, and is concerned with traditional code inspection checks such as all loops can be ended, the appropriate functions are called, return values are correct, and so forth.

The effort of solving the tasks was measured as time. The subjects were asked to start a timer when they started solving a task, and stop the timer when the task was completed. Because the experiment lasted over several hours, the subjects were asked to take breaks when needed, but were told to take the breaks between two tasks if possible. A lunch was also served during the experiment. Some subject comment that they used some extra minutes on a task because of the lunch break, and if an exact number of minutes were provided, these were subtracted from the stored time value.

These variables can be considered to be a good representation of typical quality attributes in a software design, and should not represent any threat to the construct validity.

7.2 Internal Validity

The internal validity of the experiment is, as mentioned in Section 2.1.4, concerned with whether the conclusions are drawn on the basis of a casual relationship and not the result of a factor that we have no control over or have not measured.

7.2.1 Task Solving

The experiment was conducted in a computer lab. The researcher and an assistant were present at the experiment to make sure that everything went according to the plan. The subjects had personnel present to ask if they did not understand the given information, or had other practical questions, but no help was given to solve the actual programming tasks.

The size of the computer lab made it impossible to watch every single subject all the time, making it possible for subjects with bad motivation to have cheated and copied other subjects answers.

The motivation factor is important for the integrity of the task solutions. If the subjects are not motivated, they could either copy solutions from other, or not make an effort at all for performing normal at the experiment.

The subjects were a group of students that got 500 NOK in remuneration for each participant. They were saving money to an educational trip to Malaysia, and need the extra income. This would probably be a good motivation for the subjects, backing each other up to achieve collective goods.

Another motivation factor is the fact that some of the subjects fond the experiments tasks to be challenging and fun to solve. This was stated verbally by the subjects at the experiment day, and by comments in the task comment field in the post-task questionnaire, listed in the digital appendix. Motivation among subjects is, however, mixed and some students performed much better than others.

The Learning Effect could also be considered as a risk to the internal validity of the experiment, as there are five programming tasks, and the two groups of students get presented the tasks in different order. As of this, it might be found that when comparing two tasks, one of the groups has already solved two or more programming tasks, making it a possibility that they have learned the emulator and task style better, so they would probably solve this task faster than the other group.

7.2.2 Tools

The tools used during the experiment may affect the internal validity. To make the experiment environment as realistic as possible, the subjects were asked to use their own personal development tool for solving the programming tasks. The choice of tools and the experience level in using these tools were not tested in the experiment, making it a possible threat to the internal validity.

Another tool used during the experiment was the robot emulator. The participants got some instructions about the emulator and an example program at the start of the experiment. A treat to the internal validly at this stage is whether some of the subjects have been familiar with the emulator from other projects or attended university courses. The emulator is known to be used in a fourth year software architecture course, but none of the subjects have reached this stage in the studies yet. During the experiment day, none of the student said that they were familiar with the emulator, so there is a little risk that this might have been a great threat to the validity.

On the other hand, a greater threat to the internal validity is the fact that several bugs were found in the emulator during the experiment. First off all, the light sensors was very unstable, and on the FST2 task, none of the students got the light detection to work well. Secondly, there was some problems with the thread management of the emulator, making the robot simulation sometimes randomly crash during testing. These bugs made some extra effort to the students that likely have influenced the results, making it a possible threat to the internal validity of the experiment. As an example, how much effort in time did the subjects use extra because of the emulator bugs, are an unknown factor to the experiment results.

Another tool used during the experiment was the Experiment Web Tool, described in Chapter 5. The tool was meant to make experiment execution more user-friendly to the subjects. There is a possibility that there could be misleading information or ambiguous design that would make the students give another answer than the one in fact was intending at some survey or questionnaire. In some of the tasks, there were a couple of students that did not successfully upload their solutions to the experiment file server. This might be caused by misunderstandings of the web tool, or just sloppiness of the subjects. One might consider making the web tool more fail-safe by denying subjects to proceed to the next experiment step before the finished task is successfully uploaded. The unsuccessfull source code upload is a possible threat to the internal validity.

All inn all, there were listed several threats to the internal validity from the tools used in the experiment, but the issues found are constant to all of the experiment participants, meaning that the threat is not that significant after all. We can assume that the same issues of experiment tools apply for all the experiment participants.

7.3 External Validity

The external validity of an experiment is concerned with whether the results can be generalized outside this study, such as other safety-critical environments, tools and development situations.

7.3.1 Subject Sample

An important question regarding the generalization of the experiment is if the subjects were representative for software engineering development teams in real-life projects. As the subjects of the experiment were third year students, there was not that high experience level of the population as one might expect that developers of real-life projects have. The results from the experience survey show an average skill level of programming and a less than average skill in UML and design patterns. Safety-critical systems are considered as one of the most difficult areas of software engineering [BCK03], and a high experience level is crucial for building systems that have high quality and safety.

Another threat to the external validity is the fact that the experiment performed the programming tasks individually to avoid threats to the internal validity such as copying of other subjects source code. In a real-world development situation, there would probably be a team of developers that work together to solve the problems.

7.3.2 Experiment Tasks

A question to raise here is if the experiment tasks are representative for a safety-critical environment. The experiment uses a robot emulator that emulates a quadratic map setting with walls and objects. These objects are either lights or balls, and can be identified by the robot by using light sensors and distance sensors. The safety-critical tasks was to implement light-weight design patterns that improves the reliability of the sensor readings trough the Protected Single Channel pattern, and the availability of the robot by detecting that the robot is stuck, by the Watchdog pattern.

As the real-world systems could often be much bigger in size and complexity, one might argue that the two tasks of the experiment would not be representative for these kinds of systems.

But than again, the emulator has some of the basic concepts of a real-time system, like input sensors, wheels, moving robot arms, and controllers, making the system a miniature of a complete real-time system.

The treats to the external validity can be considered to be appreciable enough for us not to draw the conclutions to other contexts. For large software systems there is a great posibility that the effect of the threatments can have a much greater impact than found in this light-weight safety-critical system.

7.4 Conclusion Validity

The conclusion validity is concerned between the treatment and the outcome. Are there really any statistical relationship?

As the experiment uses t-test to compare the means of two samples, the statistical significance value can be read directly from the test results, as shown in Section 6.3.

One of the problems of getting reliable statistical results in this experiment is possibly because due to the small sample sizes. As the experiment subjects were selected by convenience sampling, only 23 subjects were able to participate. The group of subjects was split in two, meaning only two groups of 11-12 subjects could be assigned different treatments, which makes small sample sizes for use in statistical analysis.

The t-tests results are sufficiently enough to make the conclusions needed to answer the research questions and hypothesis of the experiment.

Chapter 8

Discussion

This chapter will discuss the projects research questions, the experiment design and the results of the experiment.

8.1 Comments to the Research Questions

The introduction chapter introduced a set of research questions that this project would try to answer. The first research question (R1) was: "How are quality attributes affected when functionality problems are added to a safety-critical design?"

The term quality attributes represents a number of attributes that affect a system. Examples of quality attributes are, as mentioned in Section 3.1, performance, predictability, scheduleability, reliability, safety, reusability, maintainability, and so forth.

As the quality attributes would be tested using a controlled experiment, mentioned in the third research question (R3), only a selection of the attributes was chosen for the experiment, such as development effort and reliability. The development effort was chosen because it is crucial to know how much effort in time (and money) it takes for developers to develop a software system. The development effort was measured by finding out how long it took for a subject to complete an experiment task, and also how difficult they though the task was. The reliability was measured as correctness of the software implementation using a source inspection check-list and testing.

The second research question, R2, was concerned with how the quality attributes were affected when safety-critical design patterns was added to a system were functionality already was added. The same quality attributes as the first research question was tested.

8.2 Comments to the Experiment Design

To give different treatments to the subjects, they were split into two equally large groups. One group, the Type 1 group, got the functional tasks presented first, and the safety-critical second. The second group, the Type 2 group, got the experiment tasks presented in reverse order. Both groups were asked to answer a survey about their experience level and programming skills, and both groups did the same pre-task test. The survey answers and the pre-task test showed that there was no significant difference between the two groups, as presented in Section 6.2 and Section 6.2.1. This is a good basis for further tests, knowing the subject experience and skill levels are alike.

Because of the small scope of the experiment, only two safety-critical and two functionality solving tasks got tested. The design patterns were selected because of the light weight implementations, and were convenient for a small system. A real-world project would be much more complicated, making the change effort and correctness much more complex than they was in the experiment tasks.

To make the experiment as close to a real-world project as possible, the subjects were asked to use their usual development environment to solve the programming tasks. The task was to solve typical safety-critical problems by using a robot emulator that consists of a robot with input sensors, arms, wheels and control mechanisms.

The subjects were selected by convenience sampling, meaning that the nearest and most convenient persons are selected as subjects. The subjects were third year computer science and communication technology. They had, as presented in Section 6.2, generally good programming skills, but had not much experience in UML and design patterns. The subjects experience level are good enough to justify the internal validity of the experiment, but to generalize the findings to a real-life context, one could argue that the experience level of many developers is higher and the results might be different for that population.

8.3 Comments to the Experiment Results

The results of the experiment were presented using statistical models showing mean values, standard deviations and variances. The results were tested using a parametric test that compares two sample means, the t-test. This test is well known and commonly used in empirical studies, and can be used even with small sample sizes. The results are presented in Chapter 6.

There were some problems reported by the subjects, as mentioned in Section 6.2.4, with software bugs in the robot emulator. As is discussed in Section 7.2.2, these problems might have affected the internal validity of the experiment, but are the implications so strong that we cannot use the results? As an example with the first functional solving task, the results presented in Section 6.2.2 shows that 7 of 12 subjects from the Type 2 group, and 10 of 11 subjects from the Type 1 group solved the tasks, and the correctness was 70% for the Type 1 group and 86% for the Type 2 group. These results are showing that despite the emulator problems, most of the subjects delivered a correct solution to the task. Hence, there is no clear evidence that the emulator bugs made the internal validity suffer significantly. We can assume that the emulator difficulties were representative for the whole population, which also support the last statement.

Chapter 9

Conclusion and Further Work

This chapter will try to answer the research questions presented in Section 1.2, and make an overall conclusion. The last subsection will propose further research that can be made to extend the research done by this research project.

9.1 Answers to the Research Questions

How does the experiment contribute to answering the research questions? This section will try to justify the choices made, and will help us conclude the findings of the research.

9.1.1 R1: How are quality attributes affected when functionality problems are added to a safety-critical design?

There are several quality attributes, and only a selection of them was chosen to be tested in the research. These were development effort and reliability. The development effort was measured in minutes needed to complete an experiment task, and the reliability was measured by the number of errors found in source code from a code inspection check-list and functional tests.

The results from the experiment presented in Section 6.3, and the discussion of these results in Section 8.3, shows that the quality attributes does not get much affected when additional functionality are added to a system were we have already added safety-critical design. The result shows that only in one case did the group that had already implemented the safety-critical design patterns use significantly less effort in time to solve the task (mean time difference of 27 minutes). This could, however, be a result of the learning effect, discussed in Section 7.2.1.

This means that all in all, there was not much significant differences in how much time the subjects of the experiment used in implementing a functionality task, whether they had already implemented safety-critical design or not. There were not much difference in how difficult they found it to solve the tasks either, nor did the experiment results discover any differences in correctness of the source code implementations.

9.1.2 R2: How are quality attributes affected when safety-critical design patterns get added to a system that has already a set off added functionality?

The same quality attributes that were used in R1 were also tested when adding safety-critical design patterns.

The results presented in Section 6.3, and the discussion in Section 8.3, suggest that neither in this case was there any significant differences on the tested quality attributes between the two design approaches.

This means that we did not find any evidence in this study that suggest that there is more difficult, more time consuming or give better correctness when making safety-critical additions to a system whether extra functionality are added or not.

9.1.3 R3: How can we create a controlled experiment that can test research questions R1 and R2?

The experiment design in Chapter 4 presents how an experiment could be set up to answer the research questions.

Because of limitations to economy and time for the research project, only light-weight safety-critical design patterns could be tested, and the experiment participants was selected with the use of convenient sampling.

Several threats to the experiment validity are discussed in Chapter 7.

There were found several sources of threats, but none of them seems to affect the experiment results so much that the results should be rejected.

What we found most difficult is to generalize the findings of the experiment to other contexts. This is discussed in Section 8.2. How we can investigate these questions further is presented in Section 9.3.

9.2 Overall Conclusion

Based on the answers from the research questions, we can sum up the research and make some overall conclusions.

The test results from Section 6.3 shows that there is no significant difference in the selected quality attributes between the two groups solutions. Only in one case did one of the groups use less effort in minutes to solve a task. Despite that the subjects did not have high experience in safety-critical software systems, and that there were some problems with bugs in the robot emulator software, the results clearly imply that there is not much difference in the tested quality attributes between the two different treatment groups.

As of this, one cannot say that making functionality additions is more or less time consuming, difficult or has better or worse correctness if safety-critical issues have already been implemented. Hence, one cannot say that making safety-critical additions to a system is more or less time consuming, difficult or have better or worse correctness whether a systems functionality has been changed or not.

This regards real-time systems that are small and have a limited complexity. One cannot draw the conclusions further to larger systems with extensive complexity and system demands. Each software system lives its own life, and should be treated different.

Software design patterns are templates to be used when making software solutions to a system. There are design patterns for performance enchanting as well as for making systems with better maintainability or safety. This experiment shows that one cannot make, as an example, a general solution to a safety-critical implementation of a performance design pattern. The design pattern that enchants the quality attributes can be mixed, with better or worse results regarding of what consequences that the design pattern introduce, as mentioned in Section 3.1. This means that a design pattern that enchants, as an example, performance might have a consequence that the systems reliability suffers. The system developers must consider all positive and negative effects of a design pattern before it should be implemented to a system. This is why one cannot make a general solution to a safety-critical version of a design pattern that, as an example, enhances performance.

9.3 Further Work

To generalize the findings of this experiment, we must do some further research. First of all, the experiment could be tested with more experienced system developers. This would improve the external validity of the results. Secondly, the subject sample size could also be increased to help making the findings more reliable concerning the conclusion validity.

Since the experiment tested only two safety-critical design patterns in a limited size real-time system, it is difficult to claim that the findings of the experiment also counts for larger and more complex systems. An experiment that test larger and more complex systems is a good way to supply the research. Such a large scale experiment might be too time-consuming and expensive, hence less expensive empirical studies, such as case studies or surveys, could be used for testing systems with larger complexity.

If the experiment should be replicated, one might consider choosing another environment than the Khepera robot simulator until the emulator bugs mentioned in Section 7.2.2 are eliminated. One could replicate the experiment with other safety-critical design patterns and functionality solving tasks to supply additional data to the research.

Appendix A

Java Code Inspection for Safety-Critical Software

1. **Specification / Design**

   1. 1 Is the functionality described in the specification fully implemented by the code?

   1. 2 Is only specified functionality implemented with no additional functionality added?

   1. 3 Does the code conform to the class coding standard?

   1. 4 Is the code free of ”smells?” (Duplicate code, long methods, big classes, breaking encapsulation, etc.)

2. **Initialization and Declarations**

   2. 1 Are variables and class members of the correct type and appropriate mode?

   2. 2 Are variables declared in the proper scope?

   2. 3 Is a constructor called when a new object is desired?

   2. 4 Are all object references initialized before use?

3. **Method Calls**

   3. 1 Are parameters presented in the correct order?

   3. 2 Is the correct method being called, or should it be a different method with a similar name?

   3. 3 Are method return values used properly?

4. **Arrays**

   4. 1 Are there no off-by-one errors in array indexing?

4. 2 Have all array (or other collection) indexes been prevented from going out-of-bounds?

4. 3 Is a constructor called when a new array item is desired?

5. Object Comparison

5. 1 Are all objects (including Strings) compared with ”equals” and not ”==”?

6. Computation, Comparisons and Assignments

6. 1 Check order of computation/evaluation, operator precedence and parenthesising

6. 2 Are all denominators of a division prevented from being zero?

6. 3 Is integer arithmetic, especially division, used appropriately to avoid causing unexpected truncation/rounding?

6. 4 Are the comparison and boolean operators correct?

6. 5 If the test is an error-check, can the error condition actually be legitimate in some cases?

6. 6 Is the code free of any implicit type conversions?

7. Exceptions

7. 1 Are all relevant exceptions caught?

7. 2 Is the appropriate action taken for each catch block?

8. Flow of Control

8. 1 In a switch statement, are all cases by break or return?

8. 2 Do all switch statements have a default branch?

8. 3 Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?

Appendix B

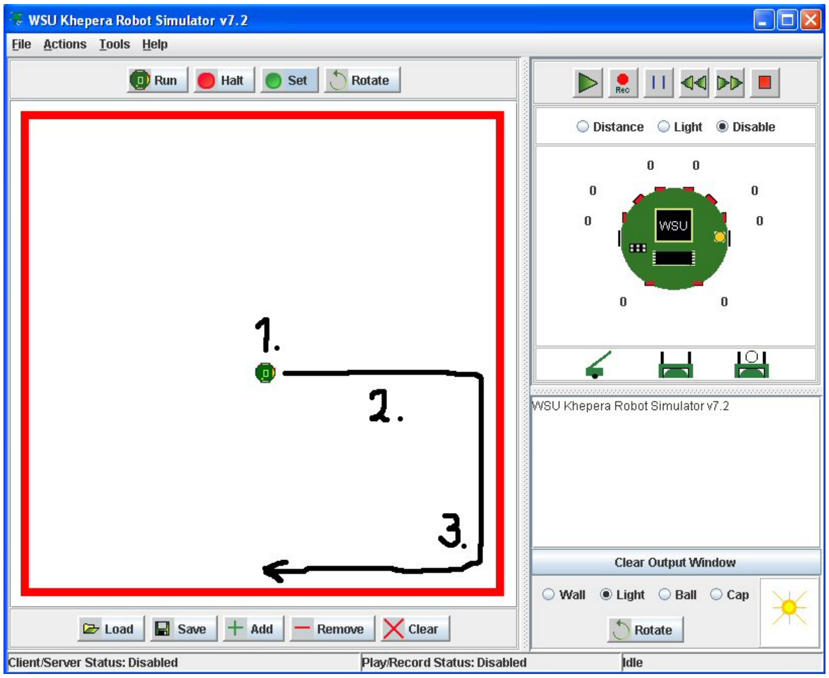
Task Descriptions

B.1 Pre-test / trainig task

B.1.1 Implement a WallFollower

Use the controller from the BallPicker example, or create a new controller that:

1. Start in the middle of the map.
2. Walk forward until a wall is hit.
3. Follow the wall in a selected direction until controller is halted.


WSU Khepera Robot Simulator v7.2
File Actions Tools Help
Run Halt Set Rotate
Rec
○ Distance ○ Light ● Disable
0 0
0 0
0 0
0 0
WSU
1.
2.
3.
Clear Output Window
○ Wall ● Light ○ Ball ○ Cap
Rotate
Load Save Add Remove Clear

WSU Khepera Robot Simulator v7.2

Client/Server Status: Disabled

Play/Record Status: Disabled

Idle

B.2 Functionality Solving Tasks

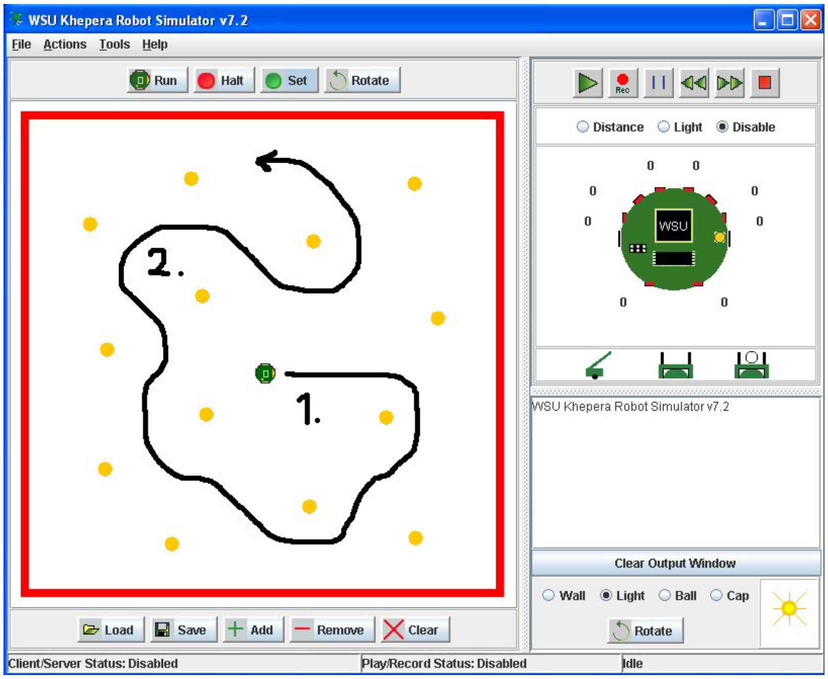
B.2.1 FST 1 - Implement a Robot that Avoid Light

Load map “lights.map.” Use the contoller from the last task, and create a robot controller that:

1. Walks randomly around the map.

2. Avoid the lights and wall.

3. Run until controller is halted.



○ Distance ○ Light ● Disable

WSU Khepera Robot Simulator v7.2

○ Wall ● Light ○ Ball ○ Cap

Client/Server Status: Disabled Play/Record Status: Disabled Idle

B.2.2 FST 2 - Gates

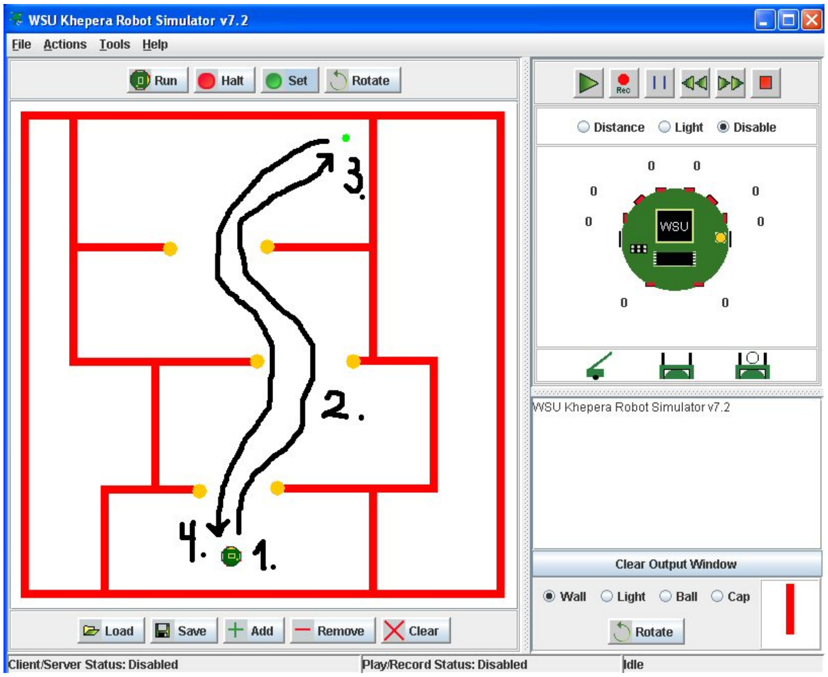
Load map “gates.map.” Use the controller from the last task, and create a robot controller that:

1. Start in home base.

2. Navigates trough the light gates to the ball room.

3. Picks up the ball.

4. Return to home base.



○ Distance ○ Light ● Disable

WSU Khepera Robot Simulator v7.2

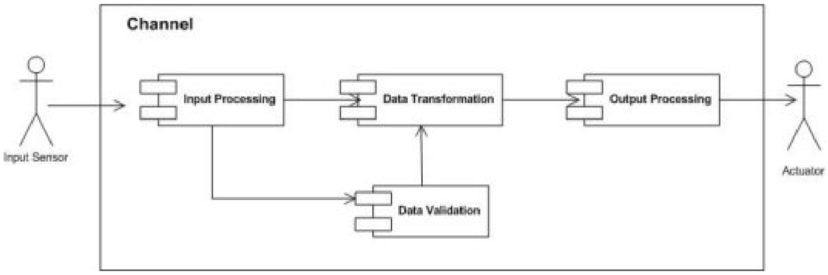
● Wall ○ Light ○ Ball ○ Cap

Client/Server Status: Disabled Play/Record Status: Disabled Idle

B.3 Safety-critical Tasks

B.3.1 SCT1 - Implement a Protected Single Channel Pattern

To better validate the input from the robot sensors, your task is to implement a Protected Single Channel Pattern.



You have to use the Channel class from the file “Channel.java.” The Channel class has these elements:

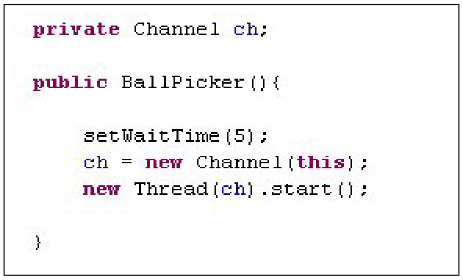
- Input Processing: Processes input from the RobotController class. The input are light sensors, and the methods to get these from the RobotController is “int getLightValue(int sensorID)” and “int getDistanceValue(int sensorID).”
- Data Transformation: The input data are stored in a input buffer with the last five input values.

- Data Validation: The input data are validated against a set of limits to ensure correctness. The value limit for getDistanceValue() is as an example >= 0 and < 1024.

- Output Processing: Instead of using the RobotController.getDistanceValue(int sensorID), it should be possible to use a channel.getDistanceValue(int sensorID), where the output is the mean of the five input values stored in the buffer (if they are validated to be ok).

- Actuator: The RobotController implementation that uses the sensor values.

The Robot Controller have to initialize a new Channel as a Thread so that it can run seperatly (Inserted into the BallPicker example):



```
private Channel ch;

public BallPicker(){

    setWaitTime(5);
    ch = new Channel(this);
    new Thread(ch).start();

}
```

Your task is to :

1. Implement the getDistanceValue(int sensorID) in the Channel class so that the mean value of five sensor inputs are returned.

2. Use the robot controller from the last task, and implement the Channel class for input handling. Confirm that the robot behavior remains the same.

B.3.2 SCT2 - Implement a Watchdog Pattern

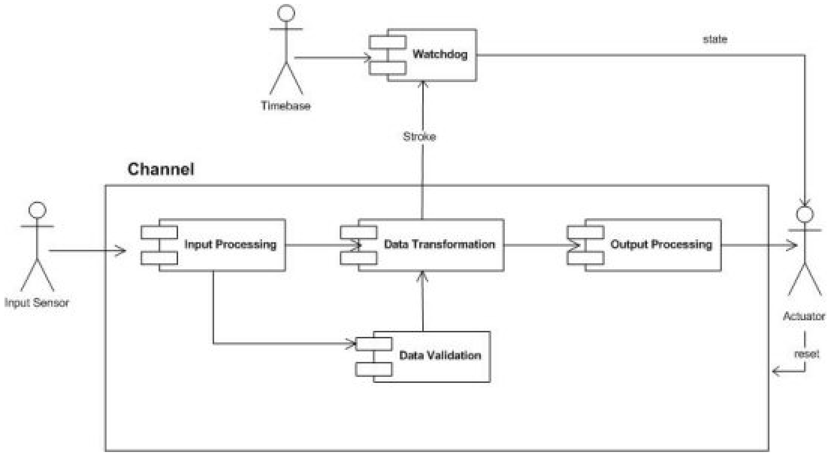
Abstract

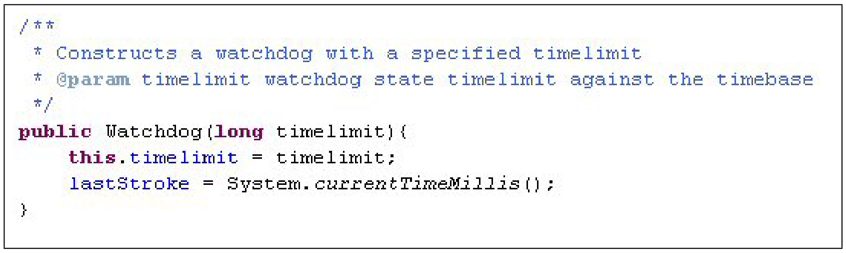
A watchdog, used in common computing parlance, is a component that watches out over processing of another component. Its job is to make sure that nothing obviously is wrong.

The Actuator Channel operates pretty much independently of the watchdog, sening a liveness message every so often to the watchdog. This is called stroking the watchdog.

The watchdog uses the timeliness of the stoking to determine whether a fault has occurred.

Implementation



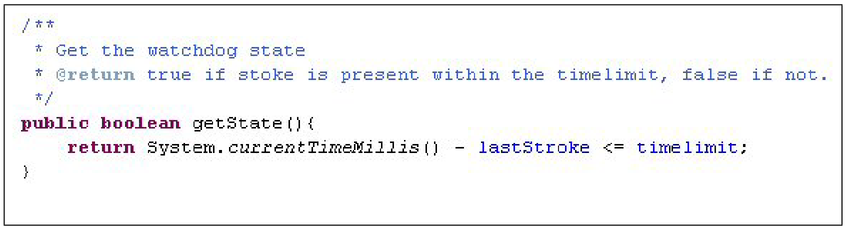


```
/**
 * Constructs a watchdog with a specified timelimit
 * @param timelimit watchdog state timelimit against the timebase
 */
public Watchdog(long timelimit){
    this.timelimit = timelimit;
    lastStroke = System.currentTimeMillis();
}
```

A watchdog pattern that uses System.currentTimeMills() as a timebase has been implemented in the Java class Watchdog.java.

The watchdog get initialized to a defined timelimit (in milliseconds):

To check the state of the watchdog, the getState() method returns false if a stroke is absent within the timelimit:



```
/**
 * Get the watchdog state
 * @return true if stoke is present within the timelimit, false if not.
 */
public boolean getState(){
    return System.currentTimeMillis() - lastStroke <= timelimit;
}
```

Your task is to:

1. Extend the Channel class to use the Watchdog class in addition to the RobotController.

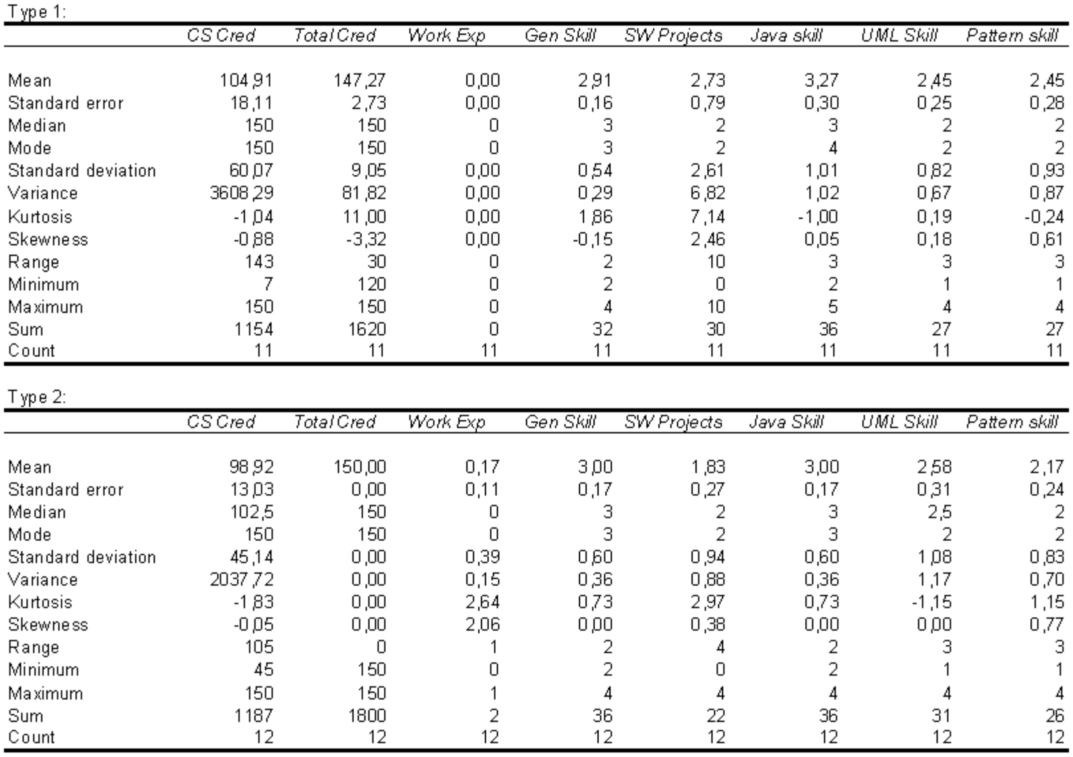
2. Add a method in the Channel class that also checks the RobotController getRightWheelPosition() and getLeftWheelPosition(), so that a stroke is sent to the watchdog every time the robot wheel positions are changing.

3. Add a test in the main controller class (from the solution of your last task) that checks the watchdog state, and if the state is false (robot is stuck), an

attempt to get the robot free from the obstacle should be done. Define a appropriate timelimit (example 5 seconds = 5000 ms).

Appendix C

Descriptive Statistics of the Subjects



Type 1:

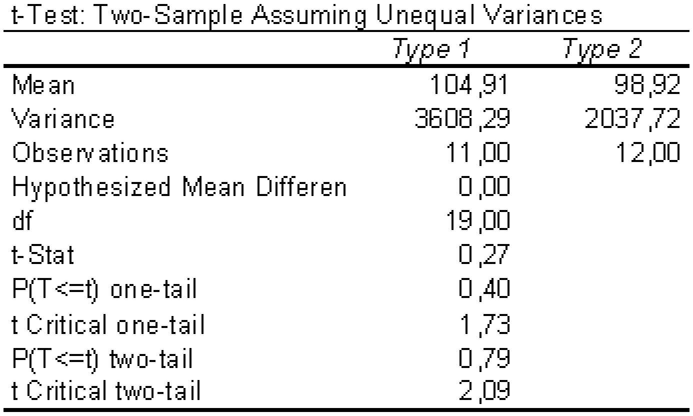
	CS Cred	*Total Cred*	*Work Exp*	*Gen Skill*	*SW Projects*	*Java skill*	*UML Skill*	*Pattern skill*
Mean	104,91	147,27	0,00	2,91	2,73	3,27	2,45	2,45
Standard error	18,11	2,73	0,00	0,16	0,79	0,30	0,25	0,28
Median	150	150	0	3	2	3	2	2
Mode	150	150	0	3	2	4	2	2
Standard deviation	60,07	9,05	0,00	0,54	2,61	1,01	0,82	0,93
Variance	3608,29	81,82	0,00	0,29	6,82	1,02	0,67	0,87
Kurtosis	-1,04	11,00	0,00	1,86	7,14	-1,00	0,19	-0,24
Skewness	-0,88	-3,32	0,00	-0,15	2,46	0,05	0,18	0,61
Range	143	30	0	2	10	3	3	3
Minimum	7	120	0	2	0	2	1	1
Maximum	150	150	0	4	10	5	4	4
Sum	1154	1620	0	32	30	36	27	27
Count	11	11	11	11	11	11	11	11

Type 2:

	CS Cred	*Total Cred*	*Work Exp*	*Gen Skill*	*SW Projects*	*Java Skill*	*UML Skill*	*Pattern skill*
Mean	98,92	150,00	0,17	3,00	1,83	3,00	2,58	2,17
Standard error	13,03	0,00	0,11	0,17	0,27	0,17	0,31	0,24
Median	102,5	150	0	3	2	3	2,5	2
Mode	150	150	0	3	2	3	2	2
Standard deviation	45,14	0,00	0,39	0,60	0,94	0,60	1,08	0,83
Variance	2037,72	0,00	0,15	0,36	0,88	0,36	1,17	0,70
Kurtosis	-1,83	0,00	2,64	0,73	2,97	0,73	-1,15	1,15
Skewness	-0,05	0,00	2,06	0,00	0,38	0,00	0,00	0,77
Range	105	0	1	2	4	2	3	3
Minimum	45	150	0	2	0	2	1	1
Maximum	150	150	1	4	4	4	4	4
Sum	1187	1800	2	36	22	36	31	26
Count	12	12	12	12	12	12	12	12

Appendix D

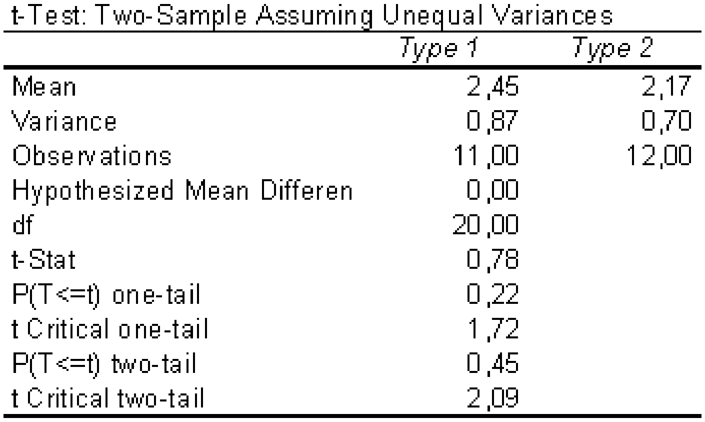
T-test Results



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	104,91	98,92
Variance	3608,29	2037,72
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	19,00	
t-Stat	0,27	
P(T<=t) one-tail	0,40	
t Critical one-tail	1,73	
P(T<=t) two-tail	0,79	
t Critical two-tail	2,09	

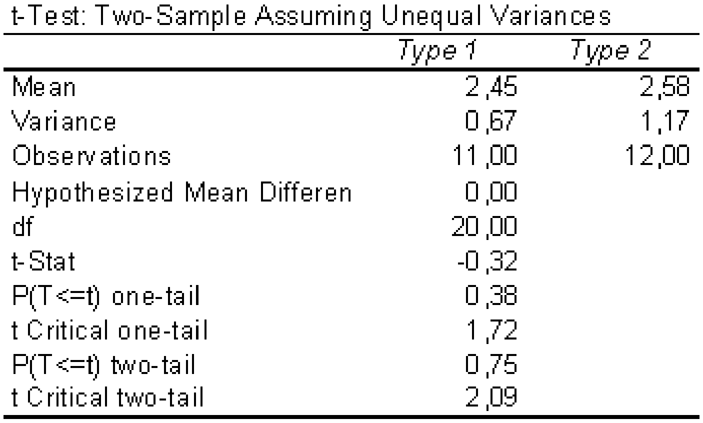
Figure D.1: T-test of credits in computer science



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	2,45	2,17
Variance	0,87	0,70
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	20,00	
t-Stat	0,78	
P(T<=t) one-tail	0,22	
t Critical one-tail	1,72	
P(T<=t) two-tail	0,45	
t Critical two-tail	2,09	

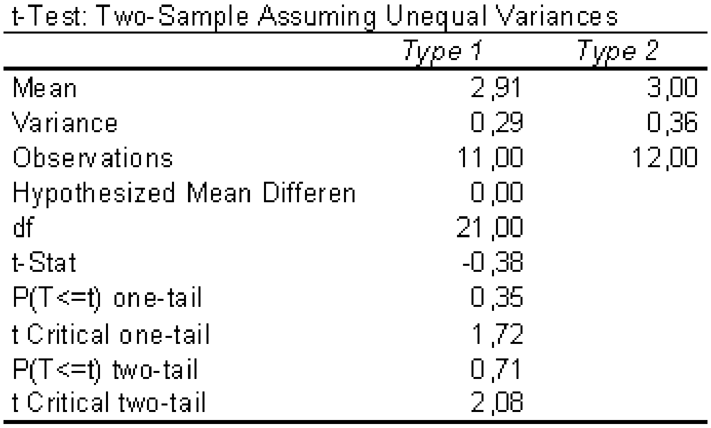
Figure D.2: T-test of design pattern knowledge



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	2,45	2,58
Variance	0,67	1,17
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	20,00	
t-Stat	-0,32	
P(T<=t) one-tail	0,38	
t Critical one-tail	1,72	
P(T<=t) two-tail	0,75	
t Critical two-tail	2,09	

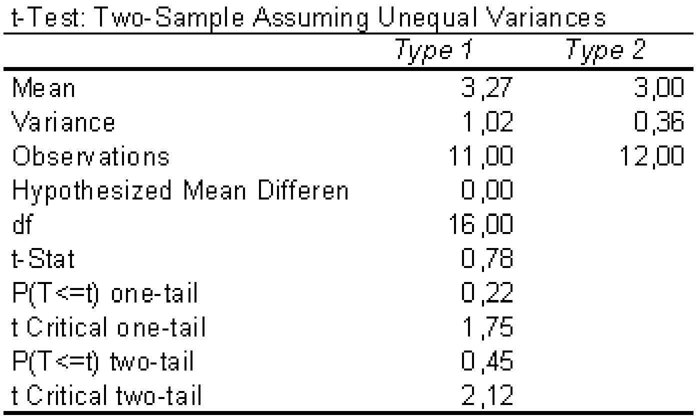
Figure D.3: T-test of UML skills



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	2,91	3,00
Variance	0,29	0,36
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	21,00	
t-Stat	-0,38	
P(T<=t) one-tail	0,35	
t Critical one-tail	1,72	
P(T<=t) two-tail	0,71	
t Critical two-tail	2,08	

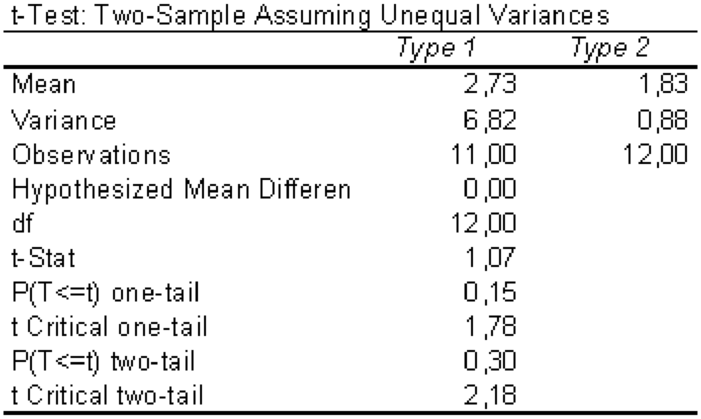
Figure D.4: T-test of general programming skills



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	3,27	3,00
Variance	1,02	0,36
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	16,00	
t-Stat	0,78	
P(T<=t) one-tail	0,22	
t Critical one-tail	1,75	
P(T<=t) two-tail	0,45	
t Critical two-tail	2,12	

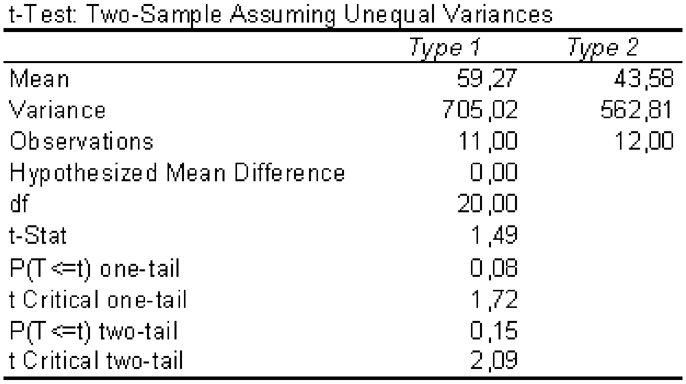
Figure D.5: T-test of Java programming skills



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	2,73	1,83
Variance	6,82	0,88
Observations	11,00	12,00
Hypothesized Mean Differen	0,00	
df	12,00	
t-Stat	1,07	
P(T<=t) one-tail	0,15	
t Critical one-tail	1,78	
P(T<=t) two-tail	0,30	
t Critical two-tail	2,18	

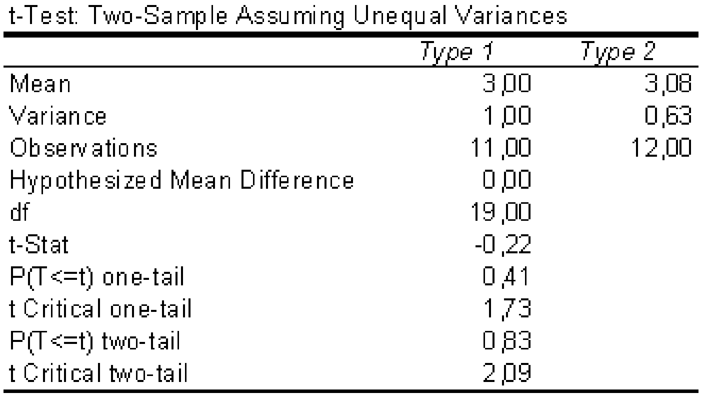
Figure D.6: T-test of involved computer science projects



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	59,27	43,58
Variance	705,02	562,81
Observations	11,00	12,00
Hypothesized Mean Difference	0,00	
df	20,00	
t-Stat	1,49	
P(T<=t) one-tail	0,08	
t Critical one-tail	1,72	
P(T<=t) two-tail	0,15	
t Critical two-tail	2,09	

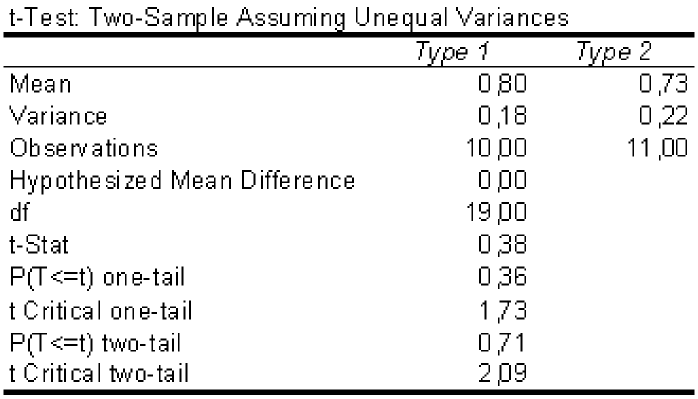
Figure D.7: T-test of effort of solving pre-test / training task



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	3,00	3,08
Variance	1,00	0,63
Observations	11,00	12,00
Hypothesized Mean Difference	0,00	
df	19,00	
t-Stat	-0,22	
P(T<=t) one-tail	0,41	
t Critical one-tail	1,73	
P(T<=t) two-tail	0,83	
t Critical two-tail	2,09	

Figure D.8: T-test of difficulty of solving pre-test / training task



t-Test: Two-Sample Assuming Unequal Variances

	Type 1	*Type 2*
Mean	0,80	0,73
Variance	0,18	0,22
Observations	10,00	11,00
Hypothesized Mean Difference	0,00	
df	19,00	
t-Stat	0,38	
P(T<=t) one-tail	0,36	
t Critical one-tail	1,73	
P(T<=t) two-tail	0,71	
t Critical two-tail	2,09	

Figure D.9: T-test of correctness pre-test / training task

Appendix E

Post-task Questionarre

1. Enter effort to solve the task (A + B + C = 100%):

- A. Effort to understand how to solve the task:
- B. Effort to code your solution:
- C. Effort to evaluate / test your solution:

2. How confident are you that your solution does not contain any serious faults? (1: Very unsure - 5: Very confident)

3. How difficult do you think the task was? (1: Very difficult - 5: Very easy)

4. Other comments about the task or your solution: (Optional. English or Norwegian language.)

Appendix F

Subject Experience Survey

F.1 Education

1. Number of credits (norwegian: “studiepoeng”) in computer science cources:

2. Number of total university / university college credits:

F.2 Work Experience

1. Number of years of work experience in Software Engineering:

F.3 Programming Skill end Experience

1. Please rate your general programming skills (1: Novice - 5:Expert):

2. Give an estimate of how many projects you have been involved in as a software developer:

3. Please rate your skill in the Java programming language (1:Novice - 5:Expert):

4. Please list up and rate your skill in other programming languages you have knowledge of (Max 3) (1:Novice - 5:Expert):

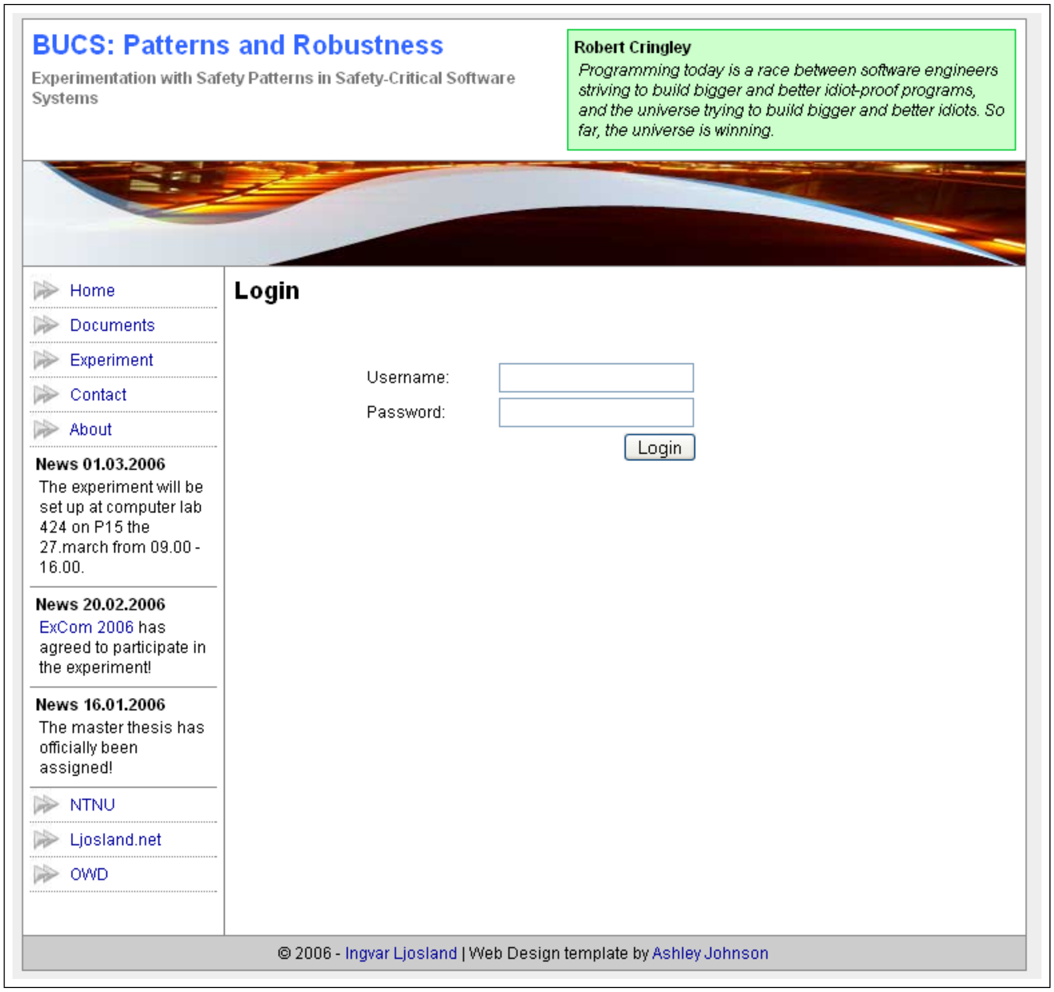
F.4 Design Method Knowledge

1. Please rate your skill in UML/Rose design methodology (1:Novice - 5:Expert):

2. Please rate your knowledge of design patterns used in software engineering (1:Novice - 5:Expert):

Appendix G

Experiment Web Tool Screenshots



BUCS: Patterns and Robustness

Experimentation with Safety Patterns in Safety-Critical Software Systems

Robert Cringley

Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far, the universe is winning.

Login

Username:

Password:

News 01.03.2006

The experiment will be set up at computer lab 424 on P15 the 27.march from 09.00 - 16.00.

News 20.02.2006

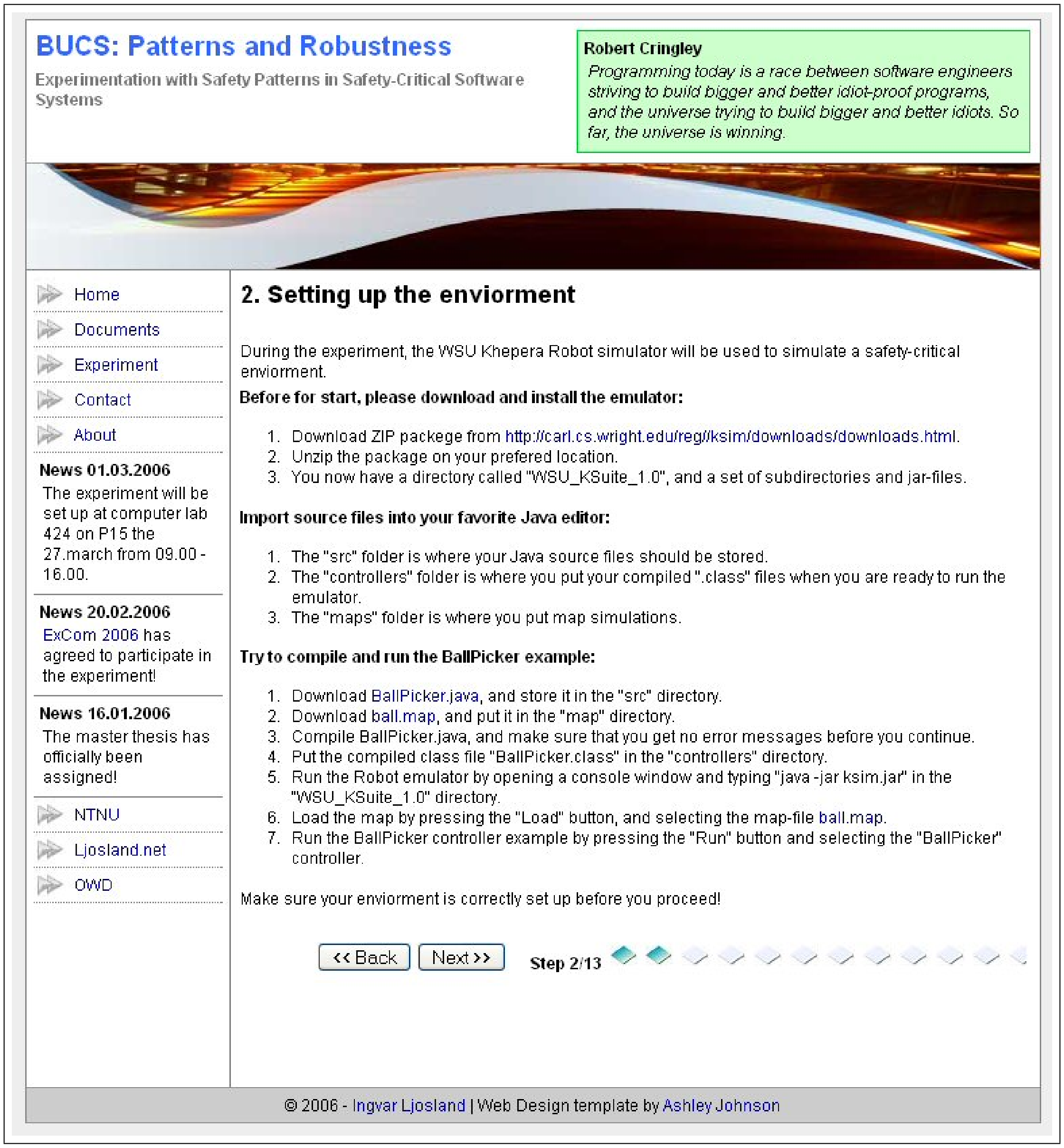
ExCom 2006 has agreed to participate in the experiment!

News 16.01.2006

The master thesis has officially been assigned!

© 2006 - Ingvar Ljosland | Web Design template by Ashley Johnson

Figure G.1: Screenshot of login web page



BUCS: Patterns and Robustness
Experimentation with Safety Patterns in Safety-Critical Software Systems

Robert Cringley
Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far, the universe is winning.

Home
Documents
Experiment
Contact
About

News 01.03.2006
The experiment will be set up at computer lab 424 on P15 the 27.march from 09.00 - 16.00.

News 20.02.2006
ExCom 2006 has agreed to participate in the experiment!

News 16.01.2006
The master thesis has officially been assigned!

2. Setting up the enviorment

During the experiment, the WSU Khepera Robot simulator will be used to simulate a safety-critical enviorment.

Before for start, please download and install the emulator:

1. Download ZIP packege from http://carl.cs.wright.edu/reg//ksim/downloads/downloads.html.
2. Unzip the package on your prefered location.
3. You now have a directory called "WSU_KSuite_1.0", and a set of subdirectories and jar-files.

Import source files into your favorite Java editor:

1. The "src" folder is where your Java source files should be stored.
2. The "controllers" folder is where you put your compiled ".class" files when you are ready to run the emulator.
3. The "maps" folder is where you put map simulations.

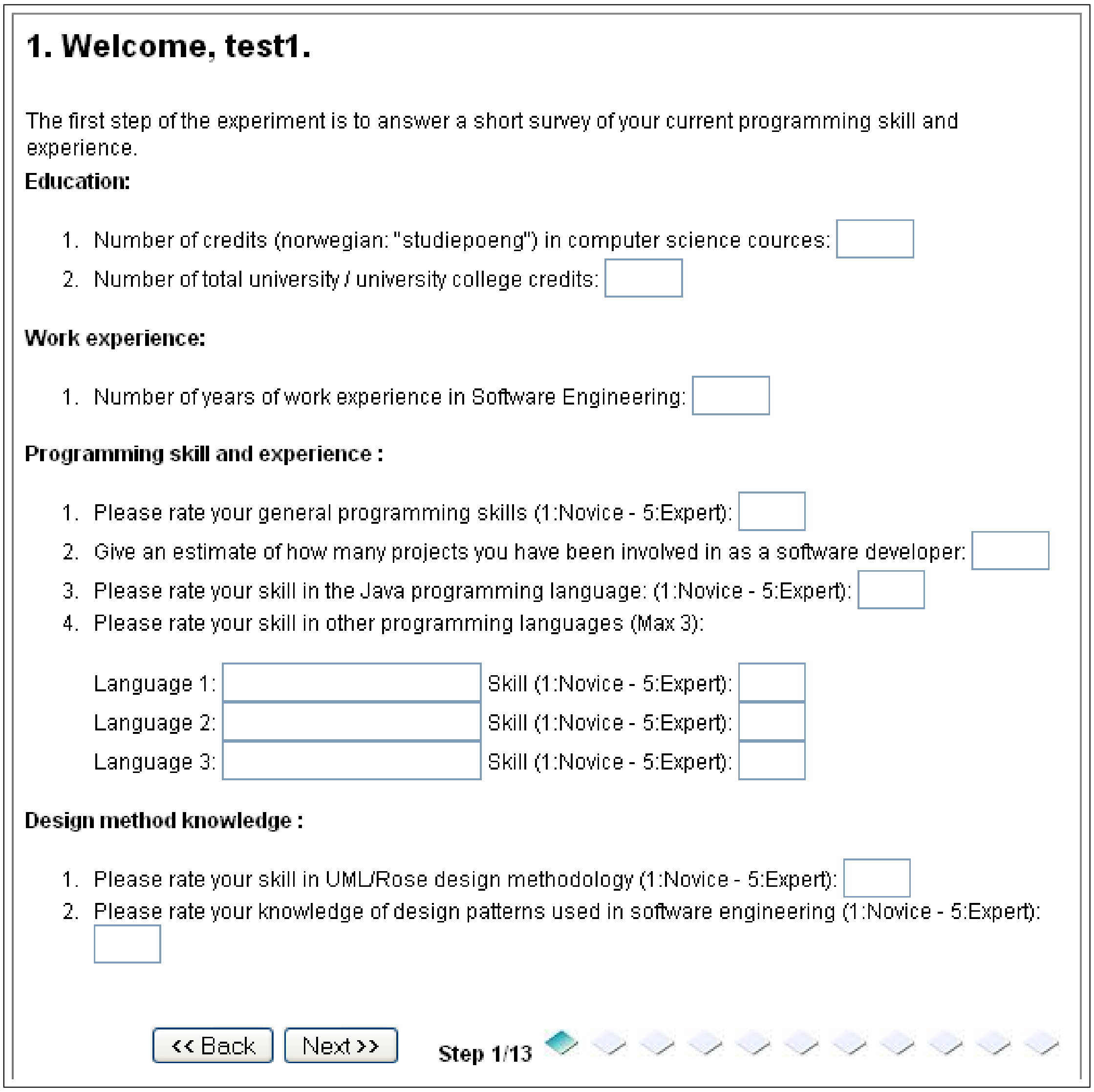
Try to compile and run the BallPicker example:

1. Download BallPicker.java, and store it in the "src" directory.
2. Download ball.map, and put it in the "map" directory.
3. Compile BallPicker.java, and make sure that you get no error messages before you continue.
4. Put the compiled class file "BallPicker.class" in the "controllers" directory.
5. Run the Robot emulator by opening a console window and typing "java -jar ksim.jar" in the "WSU_KSuite_1.0" directory.
6. Load the map by pressing the "Load" button, and selecting the map-file ball.map.
7. Run the BallPicker controller example by pressing the "Run" button and selecting the "BallPicker" controller.

Make sure your enviorment is correctly set up before you proceed!

© 2006 - Ingvar Ljosland | Web Design template by Ashley Johnson

Figure G.2: Screenshot of information presentation



1. Welcome, test1.

The first step of the experiment is to answer a short survey of your current programming skill and experience.

Education:

1. Number of credits (norwegian: "studiepoeng") in computer science cources:
2. Number of total university / university college credits:

Work experience:

1. Number of years of work experience in Software Engineering:

Programming skill and experience :

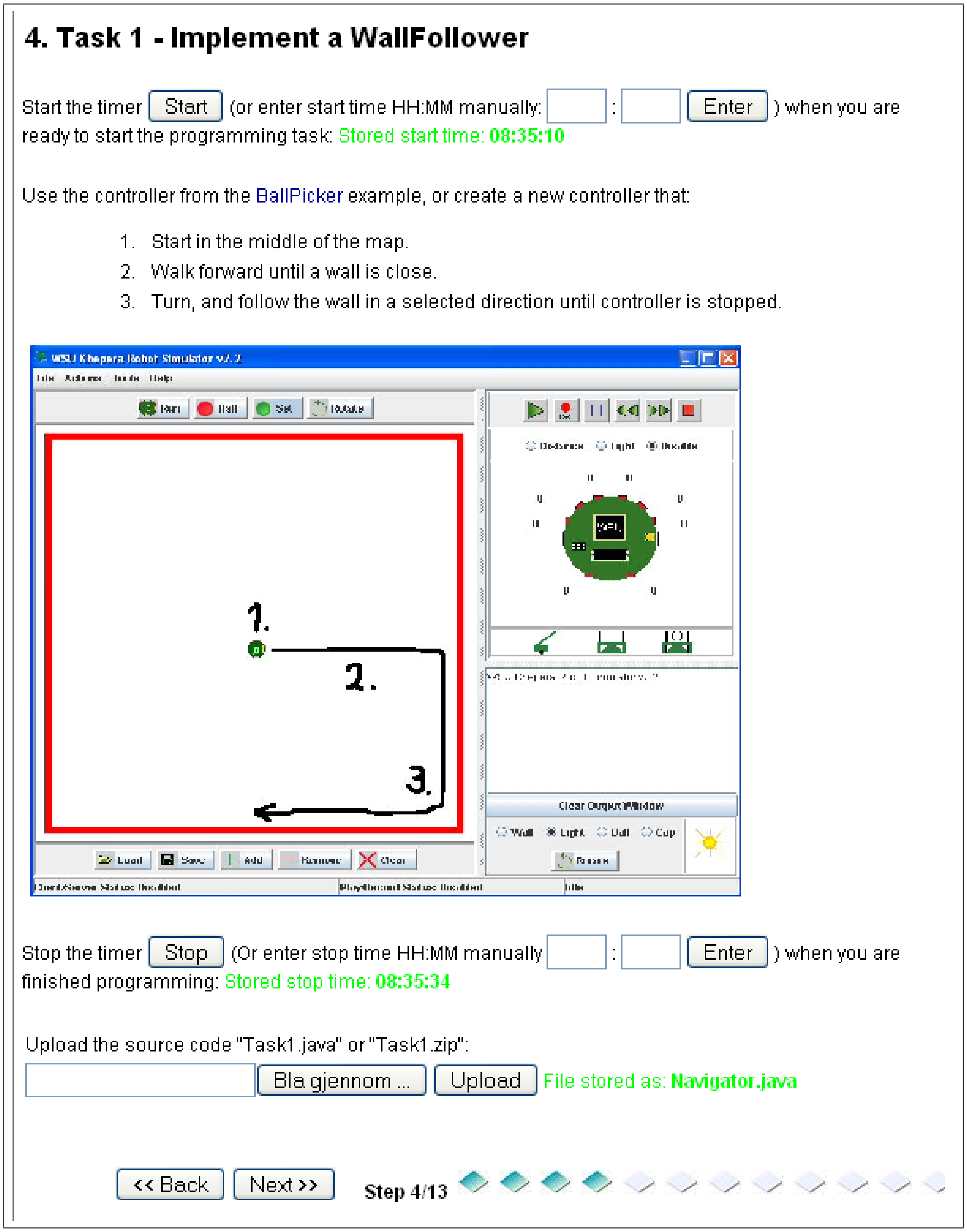
1. Please rate your general programming skills (1:Novice - 5:Expert):
2. Give an estimate of how many projects you have been involved in as a software developer:
3. Please rate your skill in the Java programming language: (1:Novice - 5:Expert):
4. Please rate your skill in other programming languages (Max 3):

Language 1: Skill (1:Novice - 5:Expert):
Language 2: Skill (1:Novice - 5:Expert):
Language 3: Skill (1:Novice - 5:Expert):

Design method knowledge :

1. Please rate your skill in UML/Rose design methodology (1:Novice - 5:Expert):
2. Please rate your knowledge of design patterns used in software engineering (1:Novice - 5:Expert):

Figure G.3: Screenshot of the subject survey



4. Task 1 - Implement a WallFollower

Start the timer [Start] (or enter start time HH:MM manually: [ ] : [ ] [Enter]) when you are ready to start the programming task: Stored start time: **08:35:10**

Use the controller from the BallPicker example, or create a new controller that:

1. Start in the middle of the map.
2. Walk forward until a wall is close.
3. Turn, and follow the wall in a selected direction until controller is stopped.

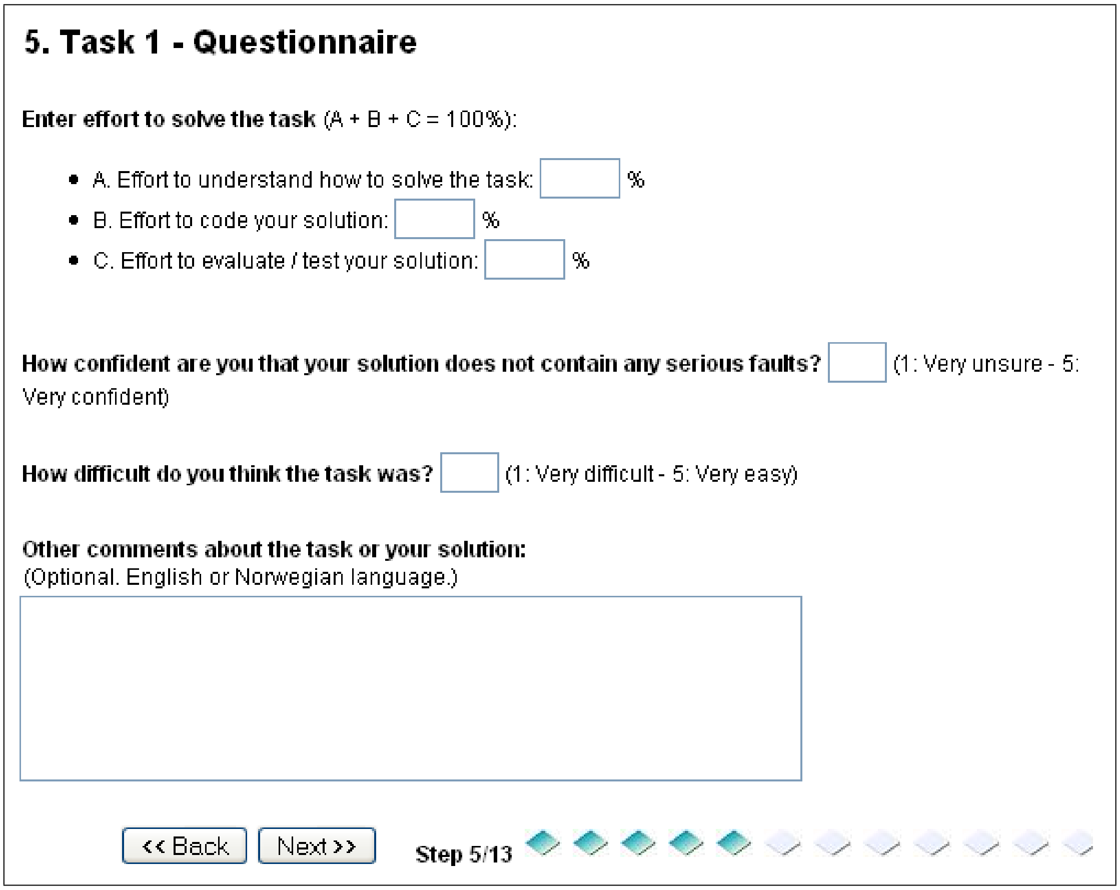
Stop the timer [Stop] (Or enter stop time HH:MM manually [ ] : [ ] [Enter]) when you are finished programming: Stored stop time: **08:35:34**

Upload the source code "Task1.java" or "Task1.zip":

[ ] [Bla gjennom ...] [Upload] File stored as: **Navigator.java**

[<< Back] [Next >>] **Step 4/13**

Figure G.4: Screenshot of a experiment programming task



5. Task 1 - Questionnaire

Enter effort to solve the task (A + B + C = 100%):

- A. Effort to understand how to solve the task: ____ %
- B. Effort to code your solution: ____ %
- C. Effort to evaluate / test your solution: ____ %

How confident are you that your solution does not contain any serious faults? ____ (1: Very unsure - 5: Very confident)

How difficult do you think the task was? ____ (1: Very difficult - 5: Very easy)

Other comments about the task or your solution:
(Optional. English or Norwegian language.)

Figure G.5: Screenshot of post-task survey

Bibliography

[Ale77] C. Alexander. *A Pattern Language: Towns, Buildings, Construction.* Oxford University Press, New York, 1977.

[AS03] E. Arisholm and D.I.K. Sjberg. A controlled experiment with professionals to evaluate the effect of a delegated versus centralized control style on the maintainability of object-oriented software. Technical report 2003-6, Simula Research Labaratory, 2003.

[BC87] K. Beck and W. Cunningham. Using pattern languages for object-oriented programs. Technical report no. cr-87-43, OOPSLA-87 workshop, 1987.

[BCK03] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice.* Addison - Wesley Publishing Company, 2. edition, 2003.

[Dou99] B. P. Douglass. *Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks and Patterns.* Addison - Wesley Publishing Company, 1999.

[Dou03] B. P. Douglass. *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems.* Addison - Wesley Publishing Company, 2003.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison - Wesley Publishing Company, 1994.

[JM01] N. Juristo and A.M. Moreno. *Basics of Software Engineering Experimentation.* Kluwer Academic Publishers, 2001.

[Ljo05] I. Ljosland. Bucs: Patterns and robustness - a navigation system case study. In-depth study project in software engineering, NTNU, 2005.

[WRH$^+$00] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln. *Experimentation in Software Engineering - An Introduction.* Kluwer Academic Publishers, 2000.