NTNU
Innovation and Creativity

# Domain Specific Languages for Executable Specifications

**Kristian Alvestad**

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Tor Stålhane, IDI

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

Problem Description

Acceptance testing is a key practice in agile development. The ultimate goal is Executable Specifications:

Acceptance tests are written prior to development
The tests are functional specifications for the application feature (as opposed to the traditional verification)
The tests functions as documentation; preferably the only necessary documentation
The feature is complete when the test runs successfully
There are already tools and frameworks available:

FIT and FitNesse uses tables to declare requirements
BDD tools like RSpec and JBehave
The assignment, Domain Specific Languages (DLSs) for Executable Specifications should focus on:

Assessing FIT and FitNesse using tables to declare tests
Assessing BDD and BDD tools to declare tests
Research and propose more natural DSLs for declaring executable specifications
Look at dynamic languages like Ruby, which are flexible and suitable for creating DSLs
Look at declarative formats like YAML as an alternative to tables
Possibly create a tool or (extensions to existing tools) to support DSLs for Executable Specifications
Clarification on empirical work:

Build a testable case with various functional requirements and select two DSL frameworks for acceptance testing
Run an experiment on Executable Specifications against the test case
Compare the results of the experiment with the findings of literature studies to define areas of strength and weaknesses in existing solutions
Identify unsatisfied needs and points of concern to existing DSLs and address these through the main objective of this thesis

Assignment given: 19. January 2007
Supervisor: Tor Stålhane, IDI

## ABSTRACT

In agile software development, acceptance test-driven development is sometimes mentioned, and some have explored the possibilities. This study investigates if a non-technical individual can write executable specifications based on domain specific languages from three different frameworks. Fit, which is an acceptance testing framework based on HTML forms, CubicTest which is an acceptance testing framework that uses modeling through Eclipse, and RSpec, a BDD framework for specifying system behavior through examples.

This study involves an experiment where the perceived effectiveness and understandability of the three frameworks are evaluated. 10 students participated in a one and a half hour experiment for which they had prepared themselves for, by having one week to acquire overview of their assigned framework. The experiment was held in a computer laboratory at the Norwegian University of Science and Technology.

After results were gathered and analyzed, statistical hypothesis testing was unfortunately not able to reject the null-hypothesis of the study. No conclusions could therefore be drawn. The results of the study are discussed and possible improvements and further work is mentioned.

## PREFACE

This study was performed by Kristian Alvestad as the final thesis for the M.Sc. Computer Science degree at the Norwegian University of Science and Technology (NTNU) in Trondheim, for the Department of Computer and Information Science (IDI). The study was conducted over 20 weeks, from January to June 2007.

I would like to thank my supervisor Tor Stålhane, Professor at the NTNU Department of Computer and Information Science (IDI) and my co-supervisors Janniche Haugen, Consultant at BEKK Consulting AS and Uwe Kubosch, Senior Software Developer at Datek Wireless AS, for their help in getting me started on the experiment.

Also, my thanks go to ConfirmIT for giving me access to an online service for surveys and data collection, and Projectplace.com for hosting my project on their systems.

Last, but not least, I would like to thank the 10 anonymous subjects who volunteered to perform in the experiment.

Kristian Alvestad Trondheim, June 15th, 2007

## TABLE OF CONTENTS

## LIST OF FIGURES

## LIST OF TABLES

**Individuals and interactions** over processes and tools
**Working software** over comprehensive documentation
**Customer collaboration** over contract negotiation
**Responding to change** over following a plan

*Manifesto for Agile Software Development*

# Introduction

Agile software development is still a young term, but has gotten a lot of attention in recent years. While buzzwords like eXtreme Programming (XP) (Beck 2000) and Scrum (Schwaber 1995) gets praises from many, still many remain unconvinced. Often agile software development processes are viewed negatively due to the (or the perceived lack of) planning and documentation in processes like these. Some also question the quality in software that is developed in this manner.

When XP is mentioned, most people will instantly associate it with the practice of pair programming. XP is rather a collection of software development practices where pair programming is only one of 12 concrete techniques of software development. Pair programming is often ridiculed and viewed as waste of resources. Others think of it as a way to achieve better productivity and higher quality software.

While agile software development is often associated with XP, Scrum has not received the same hype. This methodology is focused away from how one should produce code, e.g. techniques like Test-Driven Development (TDD) and pair-programming. Instead its focus is on how to structure an agile software development project. Scrum and XP are only two of the many known and less known agile software development methodologies. These two however, are often used in combination, complimenting each other so that the shortcomings of one are covered by the other. Scrum can successfully function as a management wrapper for XP practices.

One of these XP practices, testing, involves two types of testing techniques. TDD, which is already mentioned, and Acceptance Testing, which will be the focus of this study. Unit testing like the TDD technique is for the developers to ensure functioning production code, while Acceptance Testing is for the customer, to test that the developed system meets their criteria of acceptance.

Unlike TDD, which point is to make unit tests *before* production code, Acceptance Testing has no such criteria. The term Acceptance Testing is a widely used term in software development and is a customer's tool to accept or not accept a product at the end of a project. XP adds two criteria to this:

- Acceptance Testing should be linked to the story cards in the project
- The tests should be produced in and finished in the same iteration as the functionality they are written for.

This, by nature of the time that tests are written and also the nature of the word "test", implies *verification* of functionalities. Thus we look upon the motivation of this thesis; not acceptance testing for *verification*, but for *specification*.

## *Motivation*

In an ideal world, a person with limited technical but high domain knowledge shall be able to write functional specification documentation which is executable. In the real world, this is not yet possible without the assistance of expertise such as a developer team. We aim to research this area and determine how near this ideal world scenario we are with a selection of existing frameworks. We will then try to suggest improvements to the real world, to bring us closer to the ideal world.

## *Outline*

In the Prestudy chapter, we will describe the current state of the art. The research agenda chapter will describe the details of the goals and contexts of this study, while the experiment chapter describes the experiment plan and operation. The following results chapter describes the outcome with descriptive statistics and hypothesis tests while the analysis chapter discuss the statistics performed on the results. The discussion chapter follows with the chapter conclusion and possible improvements.

# Prestudy

## *What is Agile Software Development?*

Agile expresses the responsiveness and willingness to change: “the continual readiness of an entity to rapidly or inherently, proactively or reactively, embrace change, through high quality, simplistic, economical components and relationships with its environment” (Conboy and Fitzgerald 2004). Developers working by agile processes do not fear new and changing requirements. They rather see it as perfectly normal behavior of a project and embrace it (Fowler et al. 2001). Change is good, because it indicates that communications with the customer is good, and this should be harnessed to enhance the customers’ competitive advantage.

Agile Software Development is iterative by nature and favors iterations from a couple of months in length to as short as a week. Also, the developers working in an agile environment encourages very high customer participation in projects. Agile Software Development takes inspiration from Nonaka and Takeuchi (1995) and their theory of knowledge creation. An important principle from the Agile Manifesto is that the best communication happens face to face. Through this process of socialization, tacit knowledge can be transferred between individuals (Nonaka and Takeuchi 1995). The highest priority, however, is to satisfy the customer through early and continuous delivery of valuable software (Fowler et al. 2001; Conboy and Fitzgerald 2004)

Customer collaboration in agile Software Development is essential, and the principles of Agile Software Development states that business people and developers must work together daily. This is however easier said than done. A software developer may face difficulty in persuading a customer to sacrifice that much time for a project. Motivating a customer to participate actively is also a challenge. Customer collaboration may determine if a project will succeed or if it is doomed to fail, and the best measure of progress and success is the amount of working software that is produced (Fowler et al. 2001).

## *Test-Driven Development*

Test-Driven Development (TDD) is an approach to code development popularized by XP. The key is that the programmers write low-level functionality tests before production code. This testing approach is usually supported by a unit testing tool, such as JUnit for java and NUnit for .NET. Not only is TDD intended for unit testing, but also for specification. The tests are written before the developed software and becomes an important part of the specifications and design documentation (Canfora et al. 2006). Writing tests is a design activity, it specifies each requirement in the form of an executable example that can be shown to work (Freeman et al. 2004).

Tests can be higher level and cross-cutting, but do not in general address testing at integration and system levels. TDD relies on no specific or formal criterion to select the test cases. Tests are written gradually during implementation, one at the time and by the same person developing the module (Erdogmus et al. 2005).

TDD is not intended to be a quality assurance technique or a testing technique, but rather a code development practice and unit testing has an important role in establishing what algorithms to adopt and other design decisions (Canfora et al. 2006).

## Effectiveness of Test-First Approach

Proponents of Test-Driven Development (TDD) assert that commercial software defect rates can be reduced from 18% to 50 % when tests are written at the beginning rather than the end of the development cycle (Maximilien and Williams 2003; Jones 2004).

TDD can be considered from several points of view: Feedback, Task-orientation, Quality assurance and Low-level design. Many consider tests as overhead, so what if the benefits outweigh the costs? Erdogmus et al. (2005) investigates the strengths and weaknesses of the Test-First approach to programming in the context incremental development.

The authors (Erdogmus et al. 2005) designed an experiment in the winter of 2002-2003 and conducted it in an academic setting in the spring of 2003. Their experiment focuses on product quality and programmer productivity. In addition to this, the authors summarize previous empirical studies about TDD. They comment previous studies and compare with their own to highlight potential mistakes and how they avoided them. For the experiment, they supplied only high-level requirements in terms of user stories. For quality assessment, they used a robust black-box acceptance testing approach. The objective of their design was not to test the effect of the presence of testing, but compare alternative techniques involving opposite testing dynamics.

Test-First works as follows:

1. Pick a story.
2. Write a test that expresses a small task within the story and have the test fail.
3. Write production code that implements the task to pass the test.
4. Run all tests.
5. Rework production code and test ode until all tests pass
6. Repeat 2 to 5 until the story is fully implemented.

Test-Last works as follows:

1. Pick a story.
2. Write production code that implements the story.
3. Write tests to validate the whole story.
4. Run all tests.
5. Rework production code and test code until all tests pass.

The goal of the experiment was to compare Test-First programming with Test-Last programming for the purpose of evaluating external quality and programmer productivity in the context of incremental development and undergraduate object-oriented programming course.

Their experiment revealed that Test-First programmers write more tests per unit of programming effort. In turn, a higher number of programmer tests lead to proportionally higher levels of productivity. Therefore, Test-First appears to lead to higher productivity. Test-First programmers did not achieve better quality on average, although they achieved more consistent quality results.

### The effect of TDD on students

TDD can have many benefits when used in conjunction with learning object-oriented programming. In a paper by Jones (2004), several experiments run on students are summarized (Jones 2004), including a study by Kaufmann and Janzen (2003) where a Test-First and Test-Last group are compared (Kaufmann and Janzen 2003; Jones 2004). It can be argued that the test and control group would not provide reliable results for several reasons; the groups were self-selected, the test-first group had more programming experience and the projects were not identical (Jones 2004). There are still interesting and statistically significant results. The Test-First group produced 50% more code while keeping complexity on the same level as the Test-Last group, indicating increased productivity. The Test-Last group also had a tendency to create overburdened classes. The Test-First group also reported higher confidence in the functionality of their project (Kaufmann and Janzen 2003; Jones 2004).

Some interesting learning benefits occur in the Steinberg study, where by using TDD and JUnit, the main() method was hidden inside the testing framework and allowed students to focus on writing tests, creating and accessing objects, creating and calling methods, and using variables. Students came to view an application as a "collection of services" (Steinberg 2001; Jones 2004). By writing tests first, students learned to write "code based on the specifications of the unit tests”. Steinberg concluded that TDD was the most important of the 12 disciplines of XP.

## *Behavior-Driven Development*

Behavior-Driven Development or BDD is not a revolutionary new way of development, but rather refinement and evolvement of development based on sound practices like Test-Driven Development and Domain-Driven Design (DDD).

### TDD Evolved

BDD is an evolution of TDD and is prompted onwards by the problematic perceptions of tests and the mental rewiring needed to apply TDD. BDD has a clear focus on “getting the words right”, and the use of the word “test” in TDD is misleading. Also unlike TDD, the purpose of BDD is to define behavior of an application rather than design and test the implementation.

When asked to write tests, common statements from developers include:

- “It’s really simple code, it doesn't need to be tested”
- “Testing is a waste of time”
- “I’ve done this (loop/data retrieval/functionality, etc) millions of times”
- “That’s what we have testers for”

The word “test” in it self implies verification of code or functionality. A common misconception is that TDD is a test activity, and it is not difficult to see why. The real benefit of TDD is design. BDD aims to remove many of these confusing concepts of TDD by influence and appliance of popular psychology, Domain-Driven Design and Neuro-Linguistic Programming.

## Language and Neuro-Linguistic Programming

With his 1976 English language translation of Dionysus (Kerenyi 1976), Carl Kerenyi wrote:

> "The interdependence of thought and speech makes it clear that languages are not so much a means of expressing truth that has already been established, but are a means of discovering truth that was previously unknown. Their diversity is a diversity not of sounds and signs but of ways of looking at the world."

Neuro-Linguistic Programming (NLP) is a school of psychology in which language and the way we use language to express ourselves is key to understanding the behavioral patterns of humans. The Sapir-Whorf hypothesis (Whorf 1956) also sheds some light on the link between language, mentality and behavior:

> "There is a systematic relationship between the grammatical categories of the language a person speaks and how that person both understands the world and behaves in it."

NLP practitioners can identify human behavior and mental patterns by recognizing how a person uses language. But this is not a one-way street. What is truly remarkable about NLP is the ways practitioners can influence human behavioral patterns by varying their use of language. By consciously using language, words, rhythms and gestures, the practitioner can influence thought patterns in a way that makes the practitioners suggestions and view on a subject dominate. NLP is in fact closely related to hypnosis.

From the important role of language, some TDD practitioners have recognized problematic behavioral patterns associated with such words as "test". BDD is influenced by NLP and tries to induce desired behavioral patterns that better suit BDD activities. By changing the use of language and syntax from TDD to BDD, proponents hope to shift focus from writing tests to specifying behavior.

## Domain-Driven Design

Another source of inspiration to BDD is Domain-Driven Design (Evans 2004), or DDD, a method for tackling complex domains through modeling. The model developed with the customer isn't just a reflection of the domain and the domain experts knowledge, but also "it is a rigorously organized and selective abstraction of that knowledge" (Evans 2004). Through the modeling process, a common language is cultivated based on the model. DDD places heavy emphasis on establishing a common vocabulary between the developers and the customer and is described as "Ubiquitous language", establishing a common consistent language structured around the problem domain that can be shared by the business users of the system and the developers.

Preservation of knowledge and knowledge transfer is a very important aspect of DDD. The domain model also functions as a tool for domain knowledge transfer. As more and more knowledge is discovered, the model evolves. In the old waterfall method, the business experts talked to analysts who digested and abstracted the knowledge and then passed it on to programmers. With such typical design approaches, the code and documents don't express the knowledge that was so hard to earn. So when oral transition is interrupted for any reason, the knowledge is lost.

The domain model can be the core of a common language for a software project. Models, text documents and UML diagrams can all be valuable in different situations. But for any of this documentation to work, knowledge has to be transferred with it. We have to have a common and shared language to express this. The use of language is therefore considered al-important in DDD (Evans 2004). To achieve this, DDD describes "Ubiquitous Language" as previously mentioned.

## RSpec

RSpec is a Ruby library that provides a domain specific language intended for BDD. It provides a way of describing the behavior of a system by using examples of how it should work. In RSpec, the dynamic programming language Ruby is used to describe the behavioral examples.

```
describe Account, " when first created" do

  before do
    @account = Account.new
  end

  it "should have a balance of $0" do
    @account.balance.should eql(Money.new(0, :dollars))
  end

  after do
    @account = nil
  end

end
```

**Figure 1: RSpec example of BDD behavior**

Note in this example of creating a bank account, how syntax like "@account.balance.should eql" is used to emphasize that this is an example of a desired behavior, rather than a test.

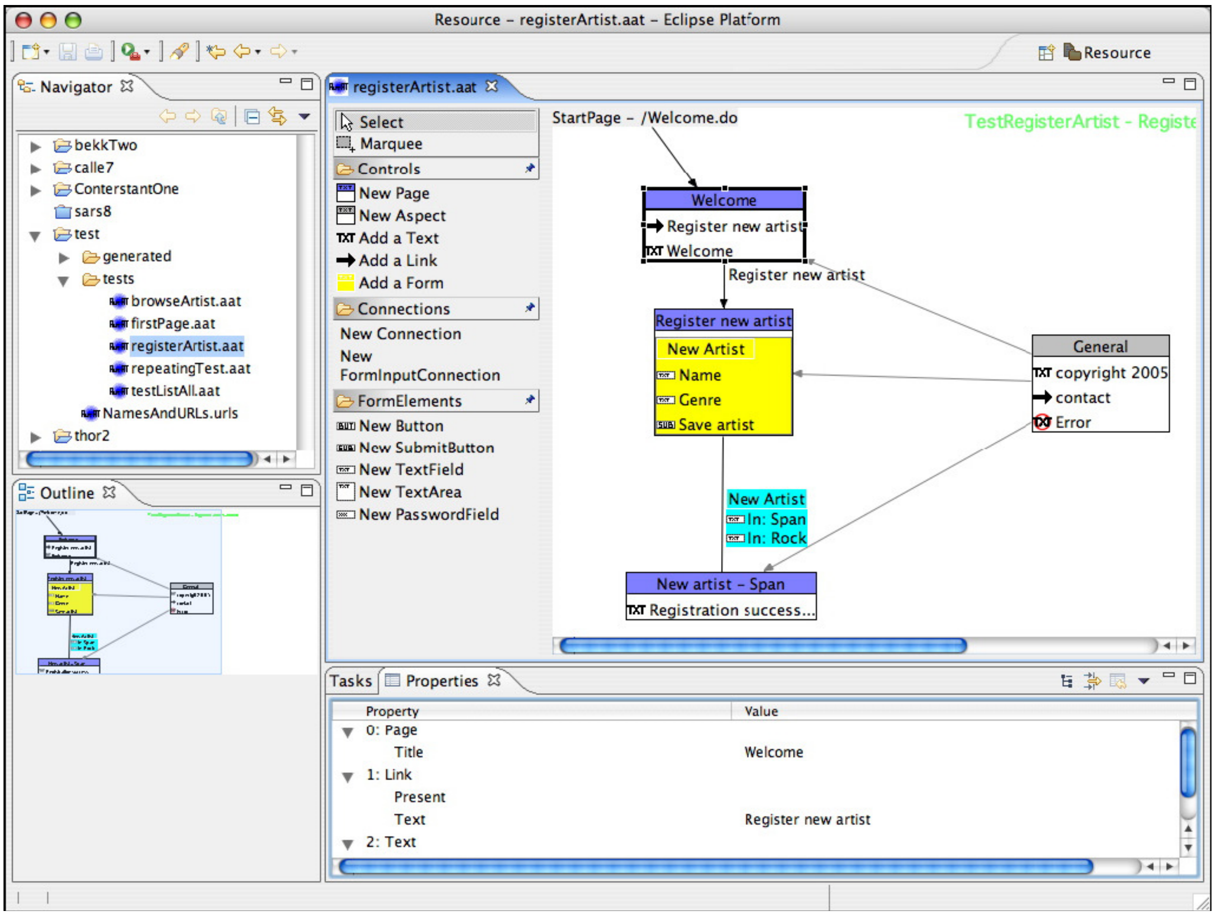
Even though Ruby is the RSpec language of choice, RSpec can be used to specify and test Java code when used in conjunction with JRuby. RSpec is most commonly available in Rubygems and the Ruby on Rails project.

Another BDD framework is JBehave, where you create behavior classes in Java that contain methods that specify the expected behavior. Like RSpec, JBehave also features syntax like "Ensure.that(actualValue, eq(expectedValue))", to emphasize behavior examples.

## *CubicTest*

CubicTest, formerly known as AutAT, is a framework for automatic acceptance testing of web applications which was first introduced as a M.Sc. thesis in 2005 (Skytteren and Øvstetun 2005) at the Norwegian University of Science and Technology (NTNU), in collaboration with BEKK Consulting AS.

It was distributed freely as open source software as part of the BEKK Open Source Software (BOSS) site[1]. It has been developed as an Eclipse plug-in and evolved further in 2006, to CubicTest, after a new M.Sc thesis on NTNU (Loe and Olsen 2006). The intention of developers involved in CubicTest is to make it easier for non technical users as well as technical users to make tests for web applications. Also, the goal is "to make it possible to replace a detailed requirements specification and manual test scripts with tests designed in CubicTest".



| Property | Value |
|---|---|
| 0: Page | |
| Title | Welcome |
| 1: Link | |
| Present | |
| Text | Register new artist |
| 2: Text | |

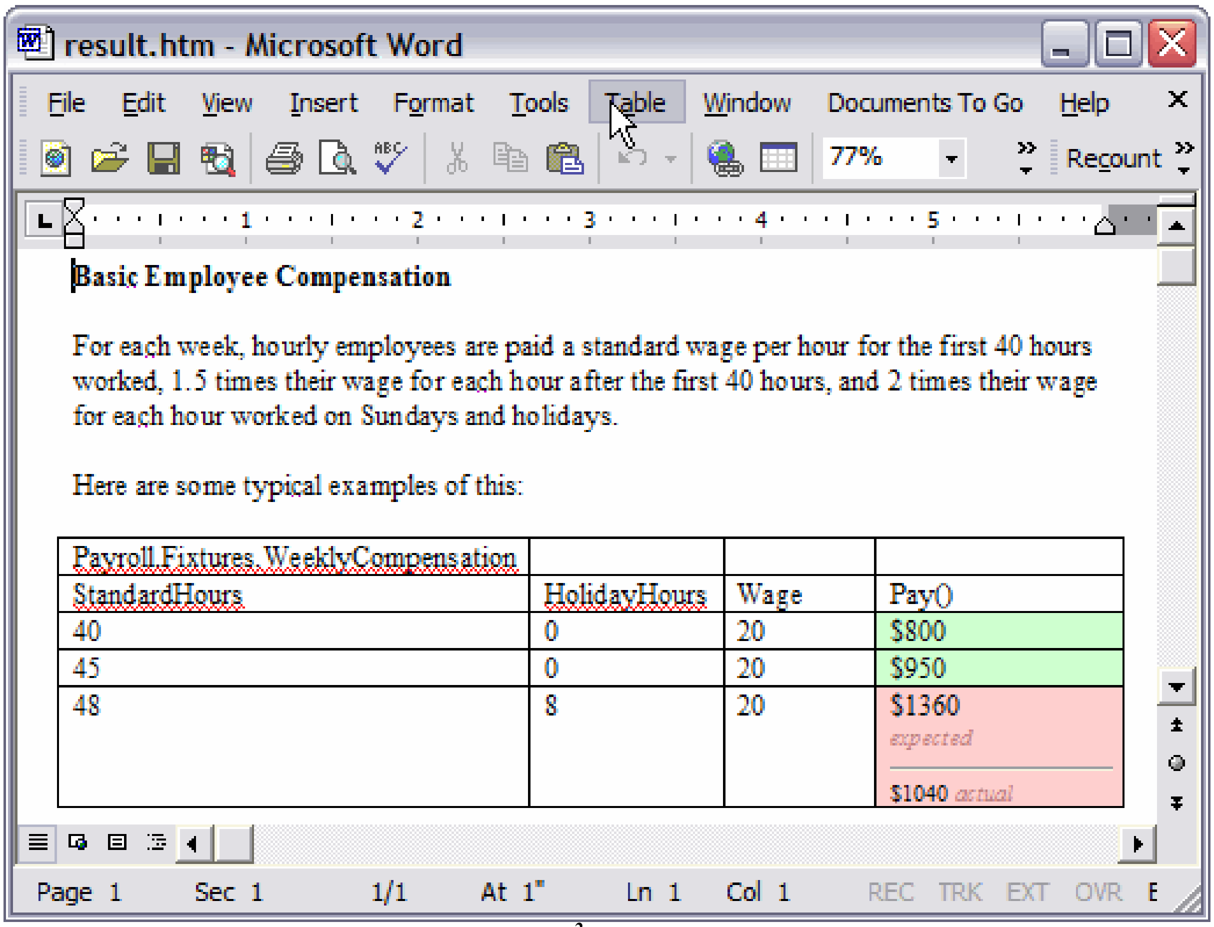
**Figure 2: CubicTest in Eclipse**

CubicTest uses a state approach, which means that a CubicTest page will be seen as a state or snapshot of the current content of a web page (Kalitina and Myhre 2006). There are modeling advantage to the state-concept, which the user doesn't need to be aware of state changes in a web application. The state-concept can be difficult to grasp for a non-technical user, however.

[1] http://boss.bekk.no

## *Fit - Framework for Integrated Tests*

Introduced by Ward Cunningham in 2002, Fit is a way of writing automated acceptance tests. Fit uses JUnit and the test cases are written and displayed in HTML tables. These tables can be made in dedicated Fit environments, like FitNesse that uses a Wiki as front end, or in a familiar work environment like Microsoft Office. This is meant to help customers write their own test cases. The test cases are processed by java classes that programmers would write, called fixtures. The fixtures take input from the HTML tables and run it on the project being tested.


result.htm - Microsoft Word
File Edit View Insert Format Tools Table Window Documents To Go Help
77%
Recount
Page 1 Sec 1 1/1 At 1" Ln 1 Col 1 REC TRK EXT OVR

**Basic Employee Compensation**

For each week, hourly employees are paid a standard wage per hour for the first 40 hours worked, 1.5 times their wage for each hour after the first 40 hours, and 2 times their wage for each hour worked on Sundays and holidays.

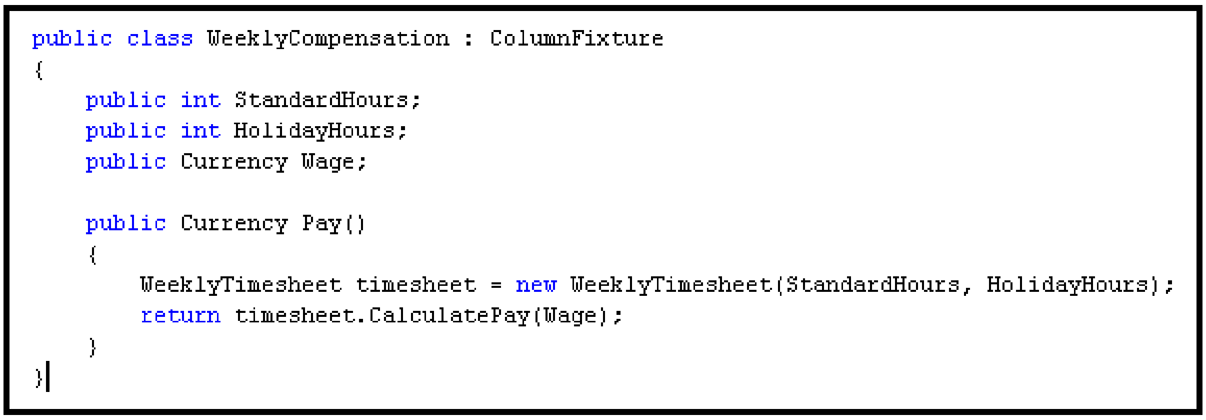
Here are some typical examples of this:

| Payroll.Fixtures.WeeklyCompensation | | | |
|---|---|---|---|
| StandardHours | HolidayHours | Wage | Pay() |
| 40 | 0 | 20 | $800 |
| 45 | 0 | 20 | $950 |
| 48 | 8 | 20 | $1360 expected<br>$1040 actual |

**Figure 3: Fit test case made with Microsoft Office** [2]

The fixtures define the structure of the tables, or test cases. Some columns are input for the system being tested, while other columns are the expected result. In Figure 3, the two green fields are where returned data from fixtures match the expected value in that field. The red field is an error, where actual fixture value differs from the expected value and is displayed in the same field.

[2] http://fit.c2.com/wiki.cgi?IntroductionToFit



```
public class WeeklyCompensation : ColumnFixture
{
    public int StandardHours;
    public int HolidayHours;
    public Currency Wage;

    public Currency Pay()
    {
        WeeklyTimesheet timesheet = new WeeklyTimesheet(StandardHours, HolidayHours);
        return timesheet.CalculatePay(Wage);
    }
}
```

**Figure 4: Fit fixture example**

Fixtures are what runs the data input of the Fit tables on the developed classes, and return and compare the output to the expected output in the same Fit tables. Since the fixtures are what define the Domain-Specific Language, Fit tables cannot be created unless the table structures are clearly defined. Two possible approaches can for instance be:

## Creating fixtures first

Developers need to create these fixtures, since they are in essence programmed classes. They are the link between classes under development and tables that holds input for testing. When fixtures are made first, they define the structures of the Fit tables and enforce them automatically. Non-technical users can then make Fit tables for the DSL. This is the usual way for acceptance testing using Fit.

## Creating Fit tables first

Fit tables do not need to be created by developers, since there is no programming involved. A non-technical person with high domain knowledge may do this, but a big problem remains: no DSL is established yet. The creators of Fit tables will need to be aware of this and in essence define the DSL through creation of tables. Any deviation from that definition will make the developers unable to create functional fixtures.

# Research Agenda

## Experiment Definition

This chapter defines the experiment and lays down a proper foundation to minimize risks and failures during planning and operation (Wohlin et al. 2000). The following framework is used to help the definition of the project to be clear and sound.

### Definition Framework

*Object of study*
The object of study is the process of using Domain-Specific Languages, or DSLs, as specification tools. In the study, the DSLs are two frameworks for acceptance testing: CubicTest and Fit. RSpec, which will be the third DSL, is not an acceptance testing framework, but a BDD framework.

*Purpose*
The three frameworks work in very different ways. Fit uses input-data and compares with expected output data via HTML forms, RSpec is used by programming an example of a desired behavior and CubicTest is used by modeling a system at different states. The purpose of the study is to evaluate the object based on the subjects' ability to perform a set of tasks with the framework they are given to work with.

*Quality focus*
The study will evaluate the understandability and perceived effectiveness of using the frameworks as specification tools rather than testing or verification.

*Perspective*
The point of view will be from students with limited experience in software engineering, but have knowledge of specification methods such as Use Case.

*Context*
The experiment was run using M.Sc. students in a university computer laboratory on NTNU campus, where the students have access to one computer each. The students participated in the experiment between 16:00 and 18:00. A textual case with a set of goals was handed out, where the goals were to be made into formal specifications for the case context.

### *Summary of definition*

Analyze the use of DSLs for specification,
for the purpose of evaluation based on subjects limited experience,
with respect to their perceived effectiveness and understandability,
from the point of view of the students playing the role of non-technical users,
in the context of a university computer laboratory.

## *Planning*

### Context selection

The experiment was set to run as an off-line exercise, performed by students. The location was a computer lab accessible by students on NTNUs university campus. The problem case used in the experiment was a fictional case scenario describing a company that needed a web portal system for selling their products. This scenario was chosen based on the domain-knowledge the subjects were perceived to have about web shops.

The subjects were asked to make functional requirements based on a selection of goals that the problem case described. This toy problem was closely related to the general functionalities that should be available in any web shop portals.

The experiment context characterization is “Multi-test within object study”, since we have one object of study, using DSLs for the purpose of specification, and 10 subjects participating. The subjects applied their assigned treatment to the factor of study individually.

### Hypothesis formulation

$H_{01}$:There is no significant difference in perceived effectiveness:
$H_{01} = \mu_1 = \mu_2 = \mu_3$
Alternate $H_{11} = \mu_i \neq \mu_j$ for at least one pair (i, j)

Metric: Level of case completion, Level of programming expertise
Scale: Ordinal

$H_{02}$: There is no significant difference in understandability
$H_{02} = \mu_1 = \mu_2 = \mu_3$
Alternate $H_{12} = \mu_i \neq \mu_j$ for at least one pair (i, j)

Metric: Ease of framework overview, Ease of syntactical use, Desired level of help
Scale: Ordinal

$H_{03}$: DSLs cannot be used by non-technical users for specifications, without assistance from technical expertise. To reject this hypothesis, $H_{01}$ and $H_{02}$ must be rejected.

Alternate $H_{13}$: Non-technical users can use DSLs for specifications, without the help of technical expertise.

## Variables selection

The factor: The process of using a DSL for functional requirements specifications.
Treatments: CubicTest, RSpec and Fit

Independent variable:
Subject programming expertise

Dependant variables:
Perceived effectiveness
Understandability

## Selection of subjects

The subjects were recruited from classes on NTNUs Computer Science and Communication Technology studies by Tor Stålhane, the head supervisor of this study. Volunteers were asked to submit their e-mail addresses. The volunteers were then divided randomly between different experiments, based on simple lottery principle. The subjects of this study were then contacted by their e-mail address and asked to confirm their interest in participation in the experiment. A total of 10 subjects confirmed their interest and was selected as subjects.

## Experiment design

### *General design principles*

*Randomization*
As mentioned previously, selection of subjects was not completely random. The selection of object of the study was not random, but discussed and reasoned until admitted or dropped. Since the subjects volunteered through courses in computer science and the non-random way of selecting objects for the study, the experiment should therefore be classified as a quasi-experiment.

*Blocking*
Two treatments had 3 subjects assigned to them and the third had 4 subjects assigned, but no systematic blocking was performed. Instead, randomly assigning treatments to subjects was thought to be best. See Figure 9 on page 26 in the results section.

*Balancing*
The 10 subjects were divided among the 3 treatments applied to the factor in the study, making CubicTest the treatment with 4 subjects and the other two treatments with 3 subjects each. The study is therefore not balanced, but as close to balanced as possible.

### ***Specific design type***

The design type of this study is based on one factor with more than two treatments. Among many design types, Wohlin (Wohlin et al. 2000) describes the two most common design types in this category as:

*Completely randomized design*
This requires that the experiment is performed in a random order to ensure that treatments are used in an environment as uniform as possible. The design uses one object to all treatments and the subjects are assigned randomly to the treatments.

*Randomized complete block design*
If the variability between the subjects is large, we can minimize this effect on the results by using a randomized complete block design. With this design, each subject uses all treatments and the subjects form a more homogeneous experiment unit, i.e. we block the experiment on the subjects. The blocks represent a restriction on randomization. The experiment design uses one object to all treatments and the order in which the subjects use the treatments are assigned randomly.

This study conforms to the completely randomized design because, as previously mentioned, no systematic approach to blocking was performed. All subjects were assigned randomly to one treatment each and only performed this treatment. See Figure 10 on page 26.

## Instrumentation

The instrumentation used in the experiment was regular desktop PCs with Microsoft Windows XP Professional, Service Pack 2, Microsoft Office XP and Eclipse 3.2 with relevant plug-ins. Eclipse was used for CubicTest, while Microsoft Excel was used for Fit and Notepad for RSpec.

Excel and Notepad was selected as editors because they are familiar working environments for most non-technical people. Notepad may not be familiar, but its simplistic user interface should not be confusing to anyone. Eclipse had to be used in the CubicTest part of the experiment since there were no other options available. Eclipse is not a familiar working environment for non-technical users, but the experience level of the subjects was thought to negate this.

Data collection was done through an online survey system provided by ConfirmIT, “the world's leading survey, community panel & reporting software”[3]. Through ConfirmIT software, two online questionnaires were made, on pre-experiment survey and one post-experiment survey.

[3] http://www.confirmit.com/

## Validity evaluation

There are several things that threat the validity of a study. This chapter will summarize the treats thought to be relevant for this study (Wohlin et al. 2000).

### *Conclusion validity*

**Low statistical power:** For statistical testing of data, suitable statistical tests must be used. The power of these tests is their ability to reveal a pattern in the data. If the power is low, there is a higher risk of drawing erroneous conclusions.

**Violated assumptions of statistical tests:** Statistical tests, such as the Chi-2 test used in this study, are based on assumptions about the data set they are performed on. If these assumptions are violated, the wrong conclusion may be drawn.

**Fishing and the error rate:** By looking for a specific outcome, researchers may influence the results and risk a wrong conclusion based on analysis that is no longer independent. Also, the significance level of multiple analysis increase to the power of the number of investigations.

**Reliability of measures:** Problems such as poor questionnaire wording and unclear questions that leads the subject to misunderstand a question may threaten conclusion validity. Measures that are subjective in nature is less reliable than objective measures.

**Random irrelevances in experimental setting:** The experiment took place in a computer laboratory that other students had access to. Room noise and other students may have disturbed the results. Also, since the subjects were seated in relatively close proximity, they may have disturbed each other when implementing their treatment or when seeking help from the observer.

**Random heterogeneity of subjects:** Having a very heterogeneous group means that the difference between subjects can be so large that it affects the results along with the treatments, or even more than the treatments.

### *Internal validity*

*Single group threats*

**Instrumentation:** If any artifacts used in the study, such as survey forms, problem cases etc. fail or is designed poorly, the experiment results may be affected.

**Selection:** The performance of subjects may vary based on how they are selected from a population. The risk is selecting subjects that are not representative of the whole population.

*Multiple group threats*

**Interaction with selection:** The different groups behave differently and may mature at different speeds due to for example learning abilities.

*Social threats*

**Diffusion or imitation of treatments:** A control group may learn of another treatment in the experiment or even imitate the behavior of the other groups.

### ***Construct validity***

*Design threats*

**Inadequate preoperational explication of constructs:** This threat is based on the constructs of the experiment not being designed god enough before they are translated into measures and treatments. If the theory isn't clear, the results of the experiment may also become unclear.

**Mono-operation bias:** This threatens construct validity if there is only a single independent variable, case, subject or treatment in the experiment. There was only one problem case used in the experiment, and may therefore under-represent the cause construct.

**Mono-method bias:** Using only one type of measure and observation method may result in a data set biased by measurement, since there is no way to cross-check results with data sets made by other measurements or observations.

**Restricted generalizability across constructs:** If a treatment affects a studied construct positively, but also affect another construct negatively. If the negatively affected construct is not also studied, there is a risk of drawing a conclusion that the treatment only has a positive effect.

*Social threats*

**Hypothesis guessing:** Subjects that participate in a survey might try to figure out what the hypothesis of the study is. Depending on their attitude towards their perceived hypothesis, they may even act positively or negatively towards it.

**Evaluation apprehension:** There are several reasons why the subjects would report inaccurate results. Fear of being evaluated and not wanting to seem less able than the other subjects of the study are two very human reasons, and may provide inaccurate results.

**Experimenter expectancies:** By expecting a result from the experiment, the researchers may consciously or unconsciously affect the results based on what they expect

### ***External validity***

**Interaction of selection and treatments:** Having the wrong subject population and not the population we want to generalize.

**Interaction of setting and treatments:** Not having the experiment setting or material representative of the environment we wish to generalize.

**Interaction of history and treatments:** Conducting the experiment on a day or time that may affect the results.

## *Experiment operation*

### Preparation

One week before participation, the subjects were sent e-mails with a link to the pre-experiment survey, to get an idea of their background; what type of study they were in, what year of study and their perceived level of programming skill. A computer laboratory was reserved, so that Eclipse and relevant CubicTest plug-ins could be installed.

The Subjects were randomly assigned to their treatments and received an e-mail, asking them to read up on and gain an overview of the treatment they were assigned to. They then had a week to gain this overview by reading the official web sites of the treatments. They were also told that they were allowed to bring material to use during the experiment. The subjects were only notified of their own treatment, not of anyone else's. The reason introduction to the treatments wasn't done by introduction lessons, was because the lack of manpower to hold 3 introduction courses. Also, this way meant it could be possible to measure how well the documentation worked.

The experiment day had to be postponed over a weekend because the lab engineers failed to remember scheduled power grid maintenance on the exact time of the experiment. Luckily, all but 1 subject was able to attend on the new date, and a replacement subject was found for the drop off.

The experiment started with meeting up and handing out the problem case. The researcher then explained the case by clarifying that this was an exercise in specification, where they were asked to use a specific framework to do the job. They were also reminded that they were not being evaluated, only the frameworks. They were also reminded that all results would be published anonymously.

### Execution

The experiment was performed without co-operation between individual subjects. They were allowed to ask questions, but reminded that the researcher could not answer questions about the use of the frameworks, only with problems understanding the case. Some questions were raised, but about the usage of the treatments, which the researcher were unable to respond to.

The experiment lasted for one and a half hour, and then asked to e-mail their solutions to the researcher before they left. They were told to expect a new e-mail with a link to another questionnaire, the next day.

## Data validation

Thee data collected from the surveys was all received within the week, except from two subjects who experienced errors while submitting their answers to the post-experiment survey. These answers were submitted again, but over a week later and may have been unreliable due to the subjects' memories of the experiment. They were nevertheless included in the datasets, since the number of samples in this study was already low.

The submission of the case solutions was supposed to yield data like number of syntactical errors and wrong use of the framework. This was unfortunately meaningless, since the case solutions differed significantly and contained so many errors as to render any comparison between the treatments meaningless. The only results that were thought to be meaningful, was the degree of completeness, how complete the domain is reflected in the specifications.

## Results

The 10 subjects participating in the experiment each filled out questionnaires before and after the experiment. The questionnaires were made in an online survey system provided by ConfirmIT. An e-mail with the URL to the survey was sent to each of the participants. The pre-experiment survey aimed to determine some relevant information about previous knowledge in computer science, programming and specification practices. The post-experiment survey was designed to uncover difficulties in usage, based on the limited experience levels of the subjects. Also included in the results are the submitted answers for their application of the treatments.

### *Descriptive statistics*

Due to the ordinal scales of the results, median value will be used for measures of central tendency, frequency tables, histograms and pie-charts to show measures of dispersion and the Spearman rank-order correlation coefficient for measures of dependency. The Spearman rank-order correlation coefficient can be calculated as follows:

$$r_s = \frac{\left(n \cdot \sum_{i=1}^{n} x_i y_i\right) - \left(\sum_{i=1}^{n} x_i\right)\left(\sum_{i=1}^{n} y_i\right)}{\sqrt{\left(n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2\right) \cdot \left(n \cdot \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2\right)}}$$

**Figure 5: Pearson correlation coefficient**

This is called the Pearson correlation coefficient, but the Spearman rank-order correlation coefficient can be calculated in the same manner by using order numbers when samples are sorted, instead of the actual values (Wohlin et al. 2000).

### Pre-experiment survey

The subjects were asked what year of study they attended, if they studied Computer Science, Information Technology or other, their experience level in programming and which of the selected terms they were familiar with. Along with the results shown below, it is worth mentioning that all of the subjects were in their 2nd year of study.

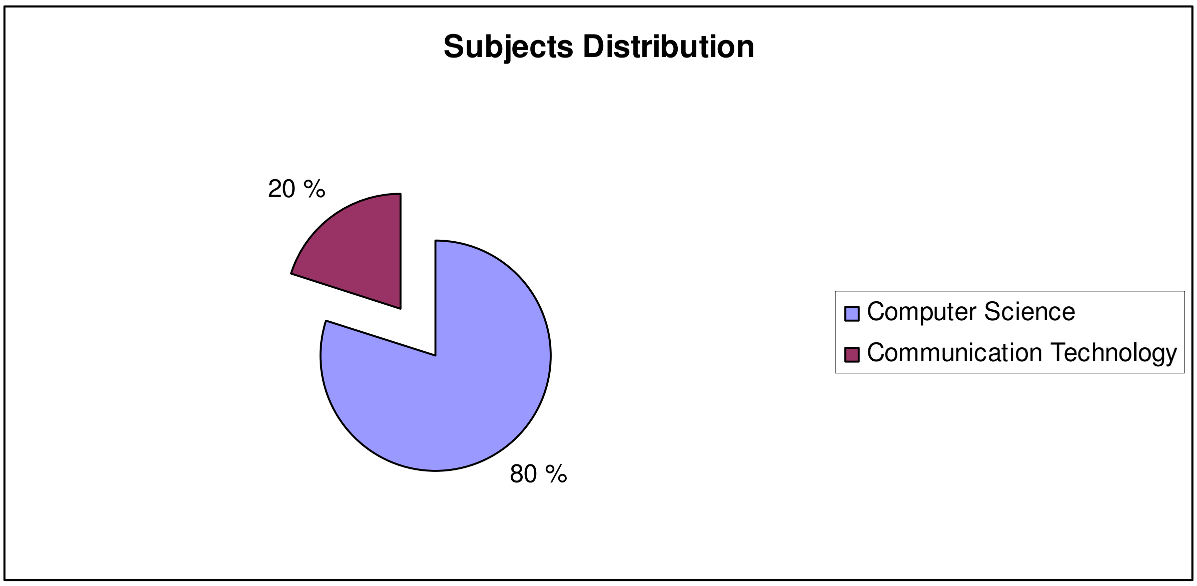


**Figure 6: Class distribution of subjects**

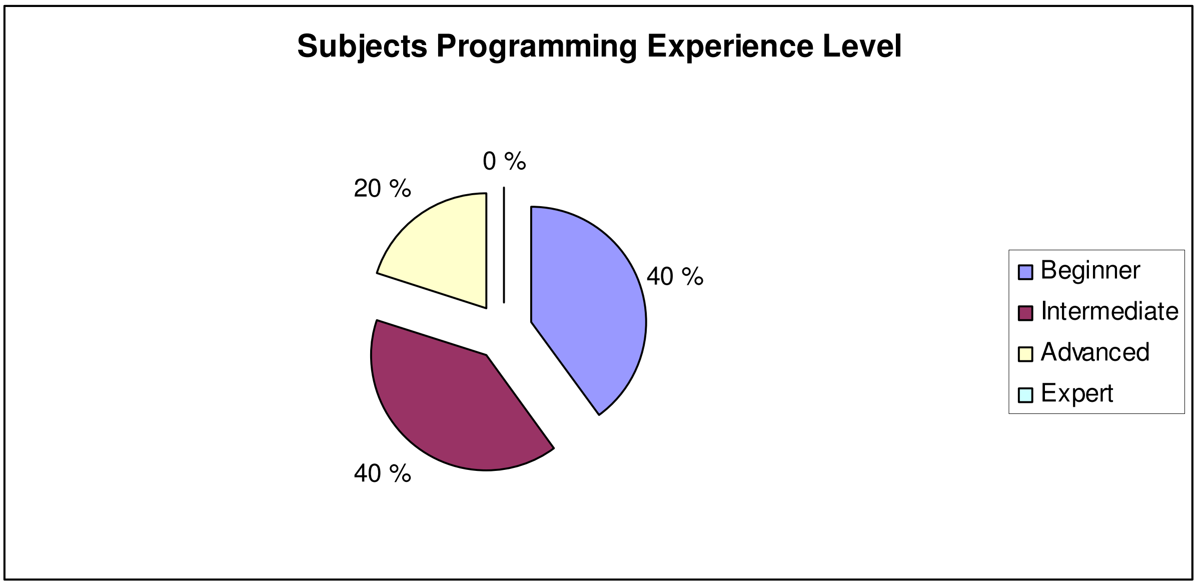


**Figure 7: Relative frequency of subjects programming experience level**

The alternatives to determine the subjects experience level was as follows:

- Beginner – Completed introduction/basic programming courses
- Intermediate – Completed one or more advanced programming courses
- Advanced – Completed several advanced programming courses and do programming on spare time
- Expert – Completed several advanced programming courses, have at least one programming related certificate and do programming on spare time or at work.

| Beginner | Intermediate | Advanced | Expert |
|---|---|---|---|
| 4 | 4 | 2 | 0 |

**Table 1: Frequency table of experience level**

The alternatives of the results can be seen as an ordinal scale, since there is a notion of "better than" between for example intermediate and advanced levels. These are subjective measures, since every subject rated their own expertise levels. The median value of this measure is "Intermediate". The measures of dispersion are shown in Figure 7 by relative frequency and in Table 1 as actual frequency.

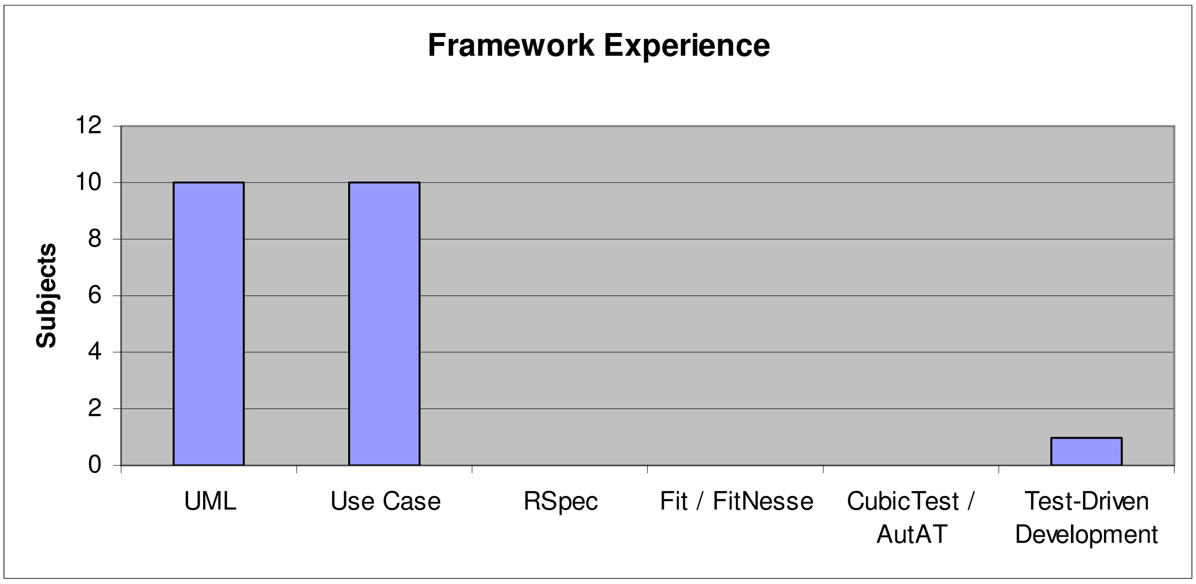


**Figure 8: Subjects previous framework experiences**

This graph shows the results based on the subjects' knowledge of a small selection of frameworks. Only one of the subjects rating themselves as advanced programmers reported hearing about TDD.

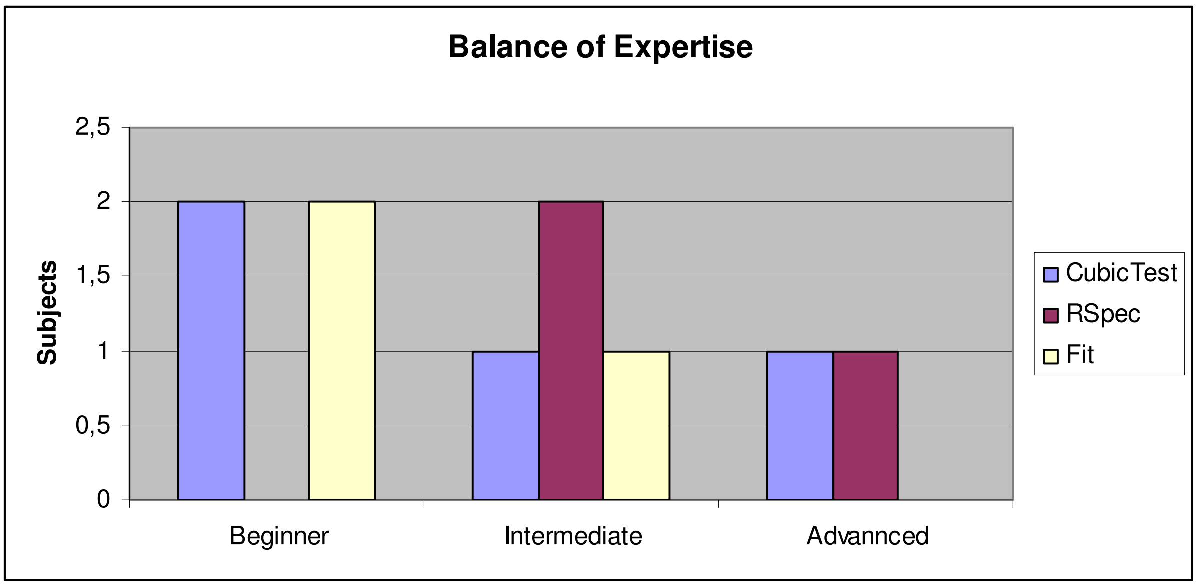


**Figure 9: Distribution of expertise between frameworks**

These results do not show what level of expertise the subjects perceive they have in the various frameworks, but rather what level of programming expertise was assigned to the three treatments. This balance of expertise graph was made by combining the data from Figure 7 and Figure 10.

## Post-experiment survey

This survey was performed by the students after completing the experiment. An e-mail with the URL to the survey was sent to the participants in the same way as the pre experiment survey. The results were as followed:

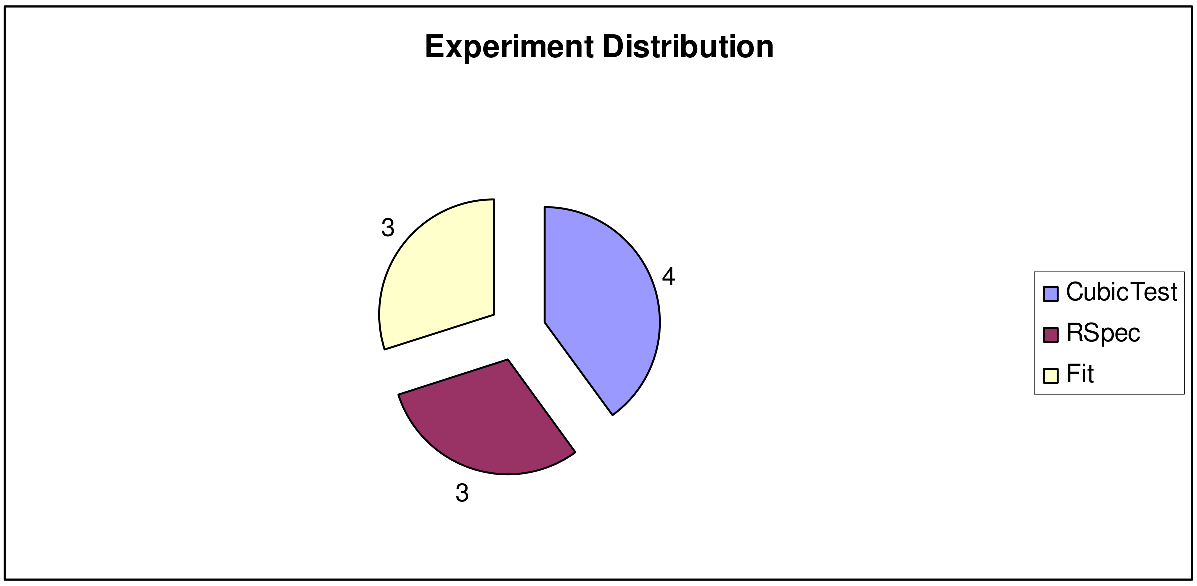


**Figure 10: Experiment Distribution**

This distribution of subjects assigned to treatments was made at random, but as balanced as possible.

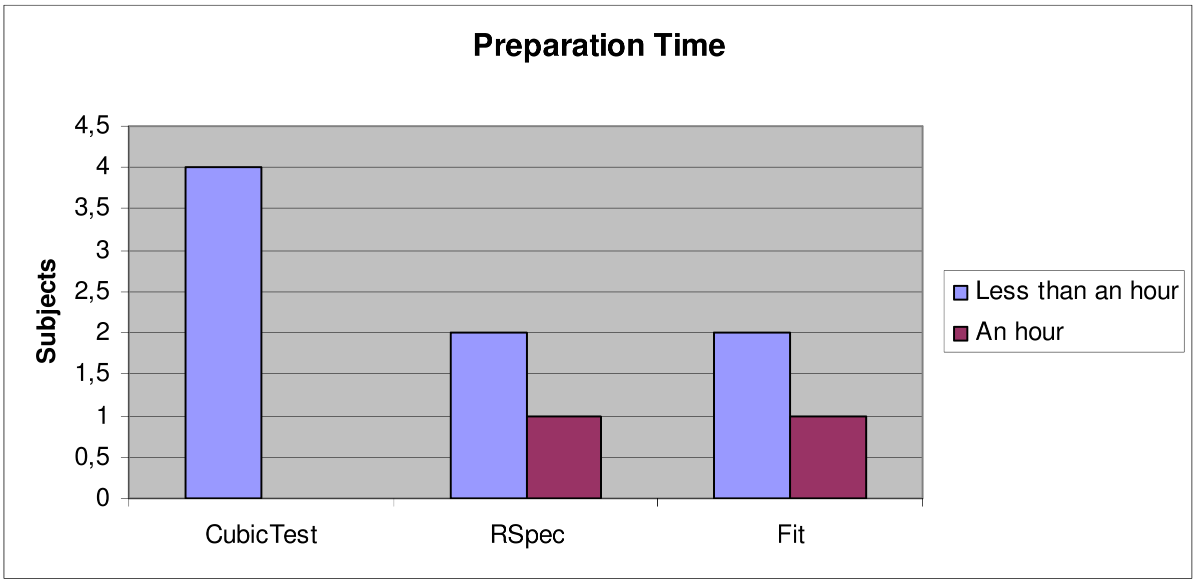


**Figure 11: Subjects Preparation Time**

This is the result of asking the subjects to prepare themselves for the experiment by reading about the framework they were asked to test. It was done in this manner due to the lack of introduction holders, since expertise in the area was unavailable and manpower limited. It is also noteworthy to mention that 90% of the subjects did not feel adequately prepared for the experiment.

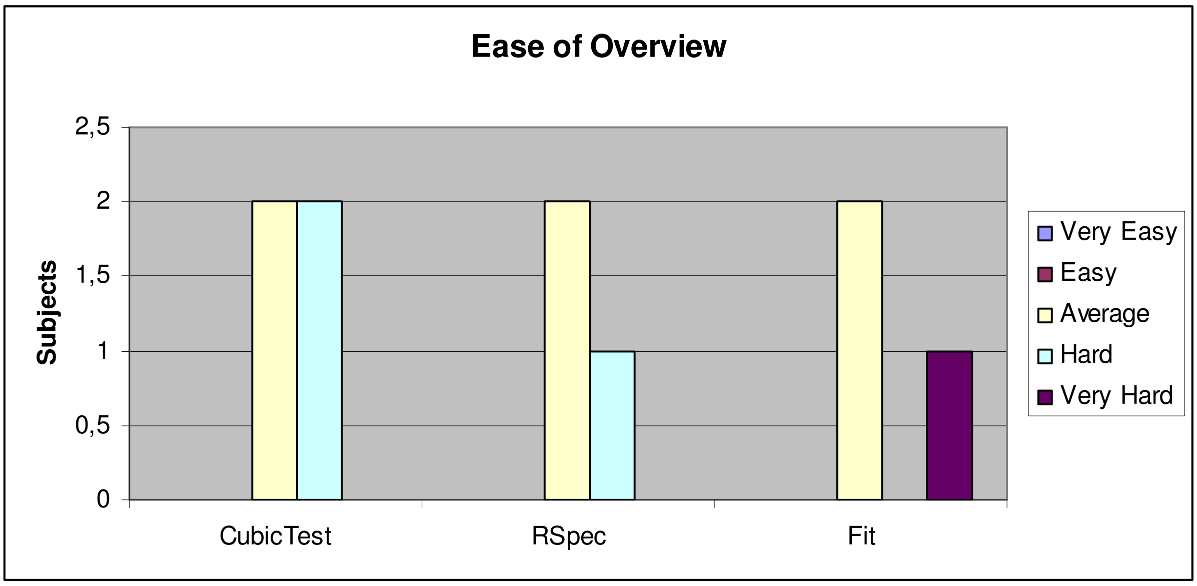


**Figure 12: Ease of framework overview**

| | CubicTest | | RSpec | | Fit | |
|---|---|---|---|---|---|---|
| Value | Frequency | Relative frequency | Frequency | Relative frequency | Frequency | Relative frequency |
| Very Easy | 0 | 0,0 % | 0 | 0,0 % | 0 | 0,0 % |
| Easy | 0 | 0,0 % | 0 | 0,0 % | 0 | 0,0 % |
| Average | 2 | 50,0 % | 2 | 66,7 % | 2 | 66,7 % |
| Hard | 2 | 50,0 % | 1 | 33,3 % | 0 | 0,0 % |
| Very Hard | 0 | 0,0 % | 0 | 0,0 % | 1 | 33,3 % |

**Table 2: Frequency table for Ease of Overview**

These results are the basis of questions to try to determine the perceived ease of framework overview. The aim is to uncover weaknesses in documentation. When asked to comment, all of the subjects found it hard to do the problem case based on the limited and simplistic examples provided on the official websites of their assigned framework. An RSpec tester also mentioned that the documentation assumed knowledge of Ruby programming language.

The measures of central tendency for CubicTest is tricky, since both median values "Average" and "Hard" are equally valid. The mean values of both RSpec and Fit are "Average".

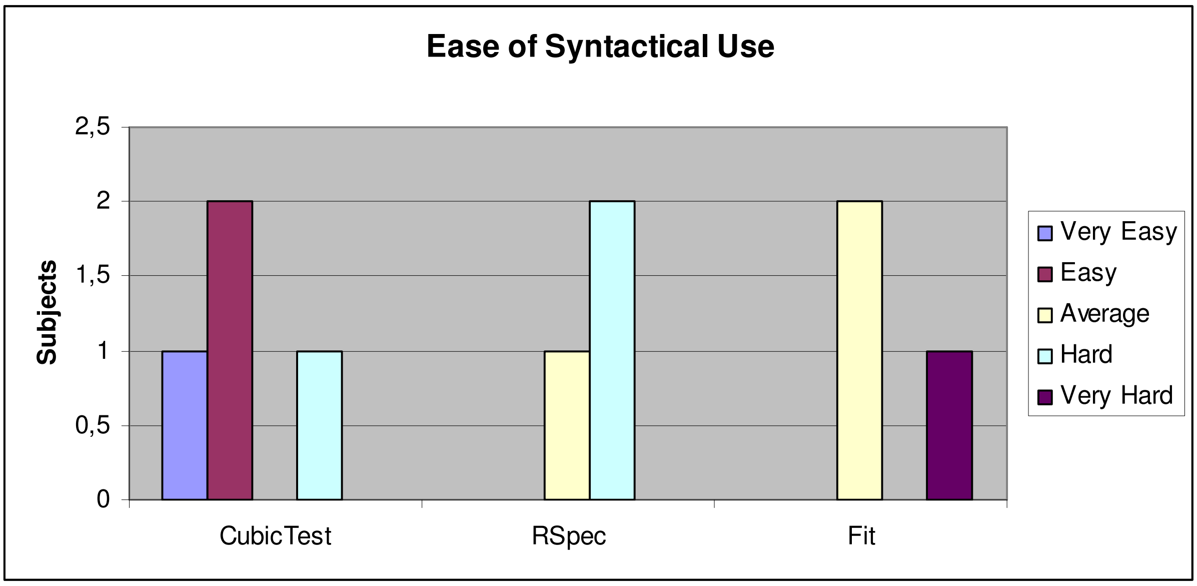


**Figure 13: Ease of syntactical use**

| | CubicTest | | RSpec | | Fit | |
|---|---|---|---|---|---|---|
| Value | Frequency | Relative frequency | Frequency | Relative frequency | Frequency | Relative frequency |
| Very Easy | 1 | 25,0 % | 0 | 0,0 % | 0 | 0,0 % |
| Easy | 2 | 50,0 % | 0 | 0,0 % | 0 | 0,0 % |
| Average | 0 | 0,0 % | 1 | 33,3 % | 2 | 66,7 % |
| Hard | 1 | 25,0 % | 2 | 66,7 % | 0 | 0,0 % |
| Very Hard | 0 | 0,0 % | 0 | 0,0 % | 1 | 33,3 % |

**Table 3: Frequency table for Ease of syntactical use**

By assigning the values 1 through 5 to the difficulty levels, 1 being Very Easy and 5 being Very Hard, we can calculate the measures of dependency by comparing the ease of syntactical use data set with ease of overview.

The Spearman rank-order correlation coefficient between the two previous data sets:
For Fit the coefficient $r_s = 1$, for RSpec $r_s = 0{,}5$ and for CubicTest $r_s = 0{,}688$. For all the samples, disregarding the division into treatments, we find a coefficient $r_s = 0{,}59$ between the ease of overview and the ease of syntactical use.

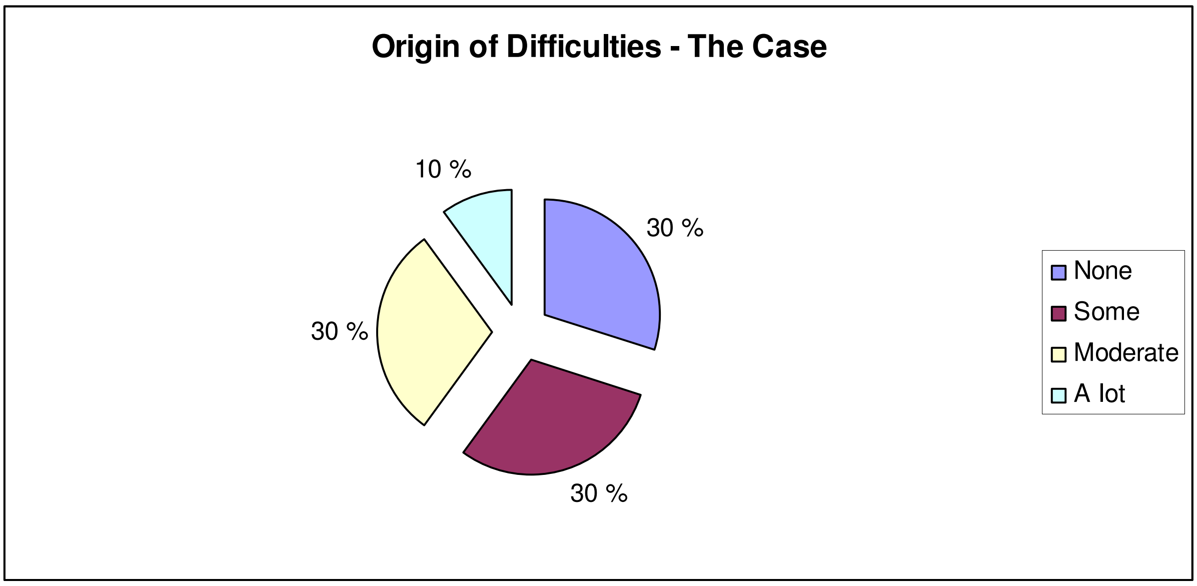


**Figure 14: Origin of Difficulties - The Case**

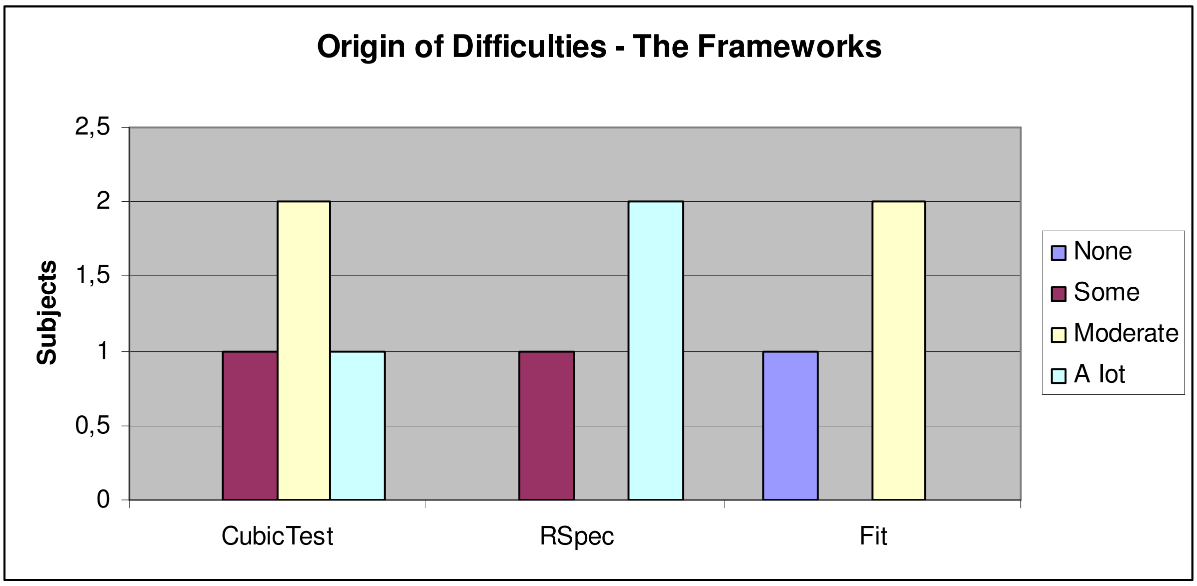


**Figure 15: Origin of Difficulties - The Frameworks**

The two previous results are from questions that try to uncover where the main difficulties were found, a problematic case or problems when using the frameworks. If an experiment case is formed in an undesirable way, the results of the experiment will be influenced by this.

The measures of central tendency with respect to problems associated with the case have a median value of “Some”. The median value of the problems associated with the framework is “Moderate”.

Measures of dependency between subject programming expertise and framework difficulties, where ranking of the data sets was done, resulted in the following dependencies:

Expertise rank: Beginner equals 5, expert equals 4
Problems with framework rank: None equals 1, a lot equals 4

The resulting $r_s = -0{,}028$.

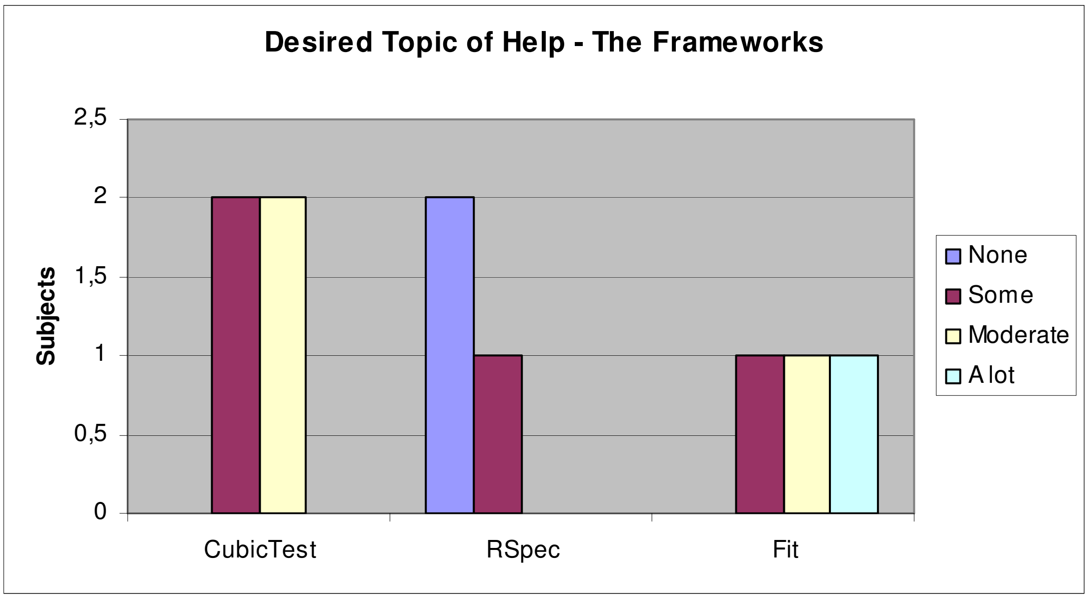


**Figure 16: Desired help from observer**

By using the same measures on dependency between expertise data set and desired help from observer resulted in $r_s = -0{,}038$.

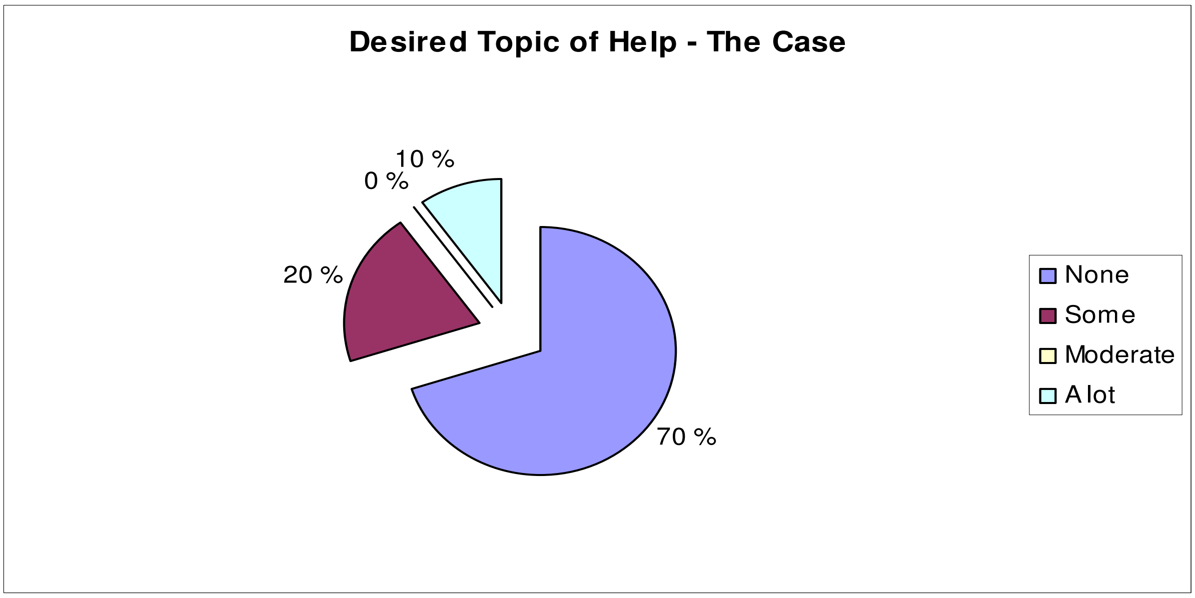


**Figure 17: Desired topic of help - The problem case**

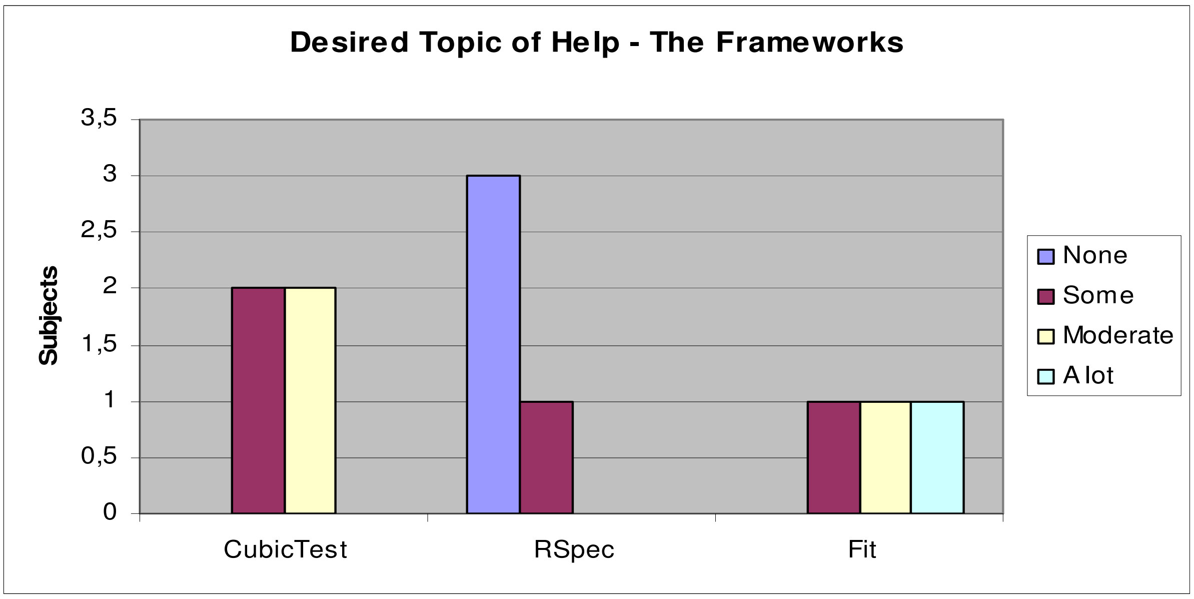


**Figure 18: Desired topic of help - The framework**

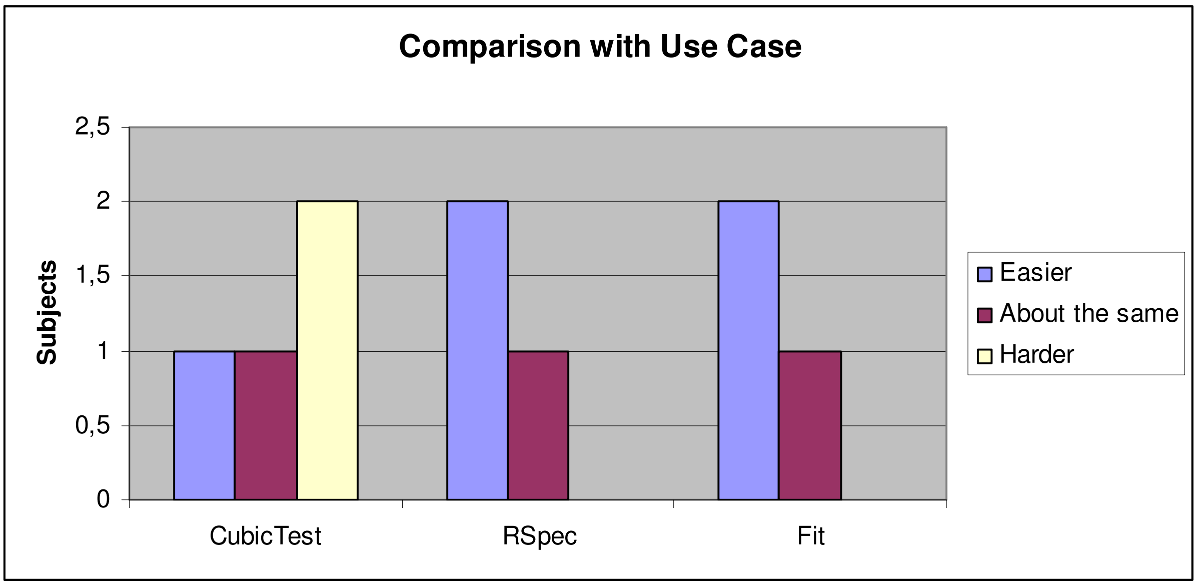


**Figure 19: A comparison with Use Case**

Since 100% of the subjects had previously had experience with Use Cases, the subjects were asked about the perceived ease of use of Use Cases, compared to the framework they tested.

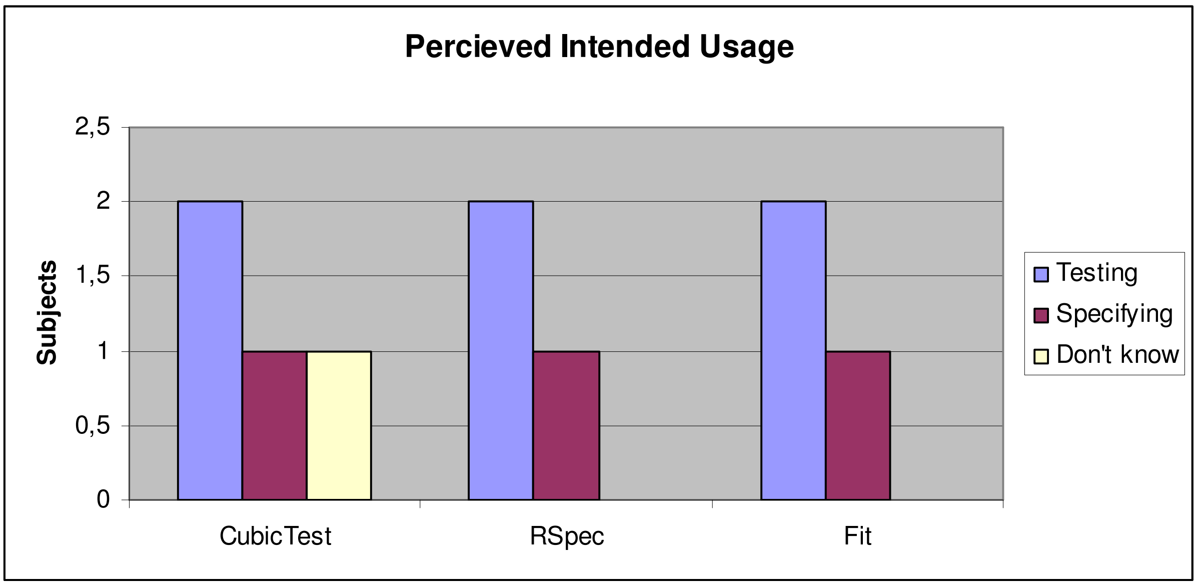


**Figure 20: Perception of intended usage**

Among the questions asked, some tried to capture the subjects opinions on what they thought the intended usage of the frameworks were. This was done in an effort to find out what effects the language usage in the frameworks had. It is interesting to note the similar results between RSpec and the other frameworks, since RSpec is meant to be a specification framework rather than acceptance testing, like the other two.

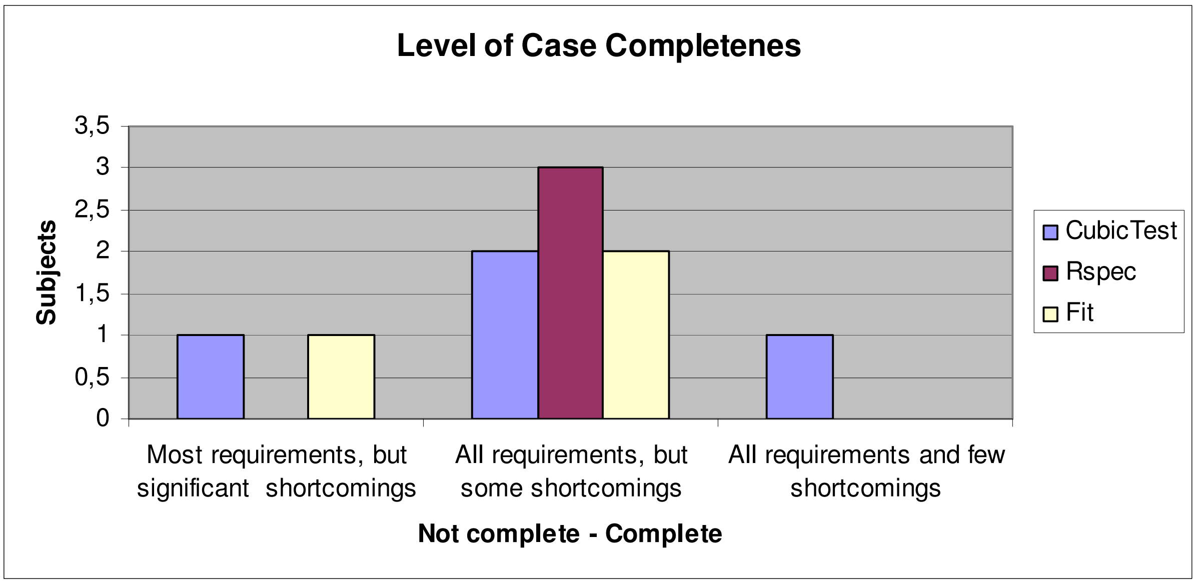


**Figure 21: Level of case completeness**

| | CubicTest | | RSpec | | Fit | |
|---|---|---|---|---|---|---|
| Value | Frequency | Relative frequency | Frequency | Relative frequency | Frequency | Relative frequency |
| 1 | 0 | 0,0 % | 0 | 0,0 % | 0 | 0,0 % |
| 2 | 1 | 25,0 % | 0 | 0,0 % | 0 | 0,0 % |
| 3 | 2 | 50,0 % | 3 | 100,0 % | 2 | 66,7 % |
| 4 | 1 | 25,0 % | 0 | 0,0 % | 1 | 33,3 % |

**Table 4: Frequency table for level of case completeness**

Concerning the problem case solutions that the subjects submitted after the experiment, only level of completeness measure is done. The frequency table adds a value that isn't visible in the histogram, but was included in the judgment of completeness, which is perfect. These values have been given ranks from Perfect, being 4 to lowest being 1.

This measure is made based on the researcher's judgment, and is therefore highly subjective. They are judged based on how many of the requirements were described in their solution. The shortcomings are based on how correctly the requirements reflect the problem domain.

The correlation coefficient $r_s$ = 0,447 between the data sets expertise and level of completeness.

## *Hypothesis testing*

Due to the data collected being on ordinal scale and the frequency-based descriptive statistics, Chi-2 tests is suitable for hypothesis testing. The Chi-2 test can be calculated from the following equation:

$$X^2 = \sum_{i=1}^{r}\sum_{j=1}^{k}\frac{\left(n_{ij} - E_{ij}\right)^2}{E_{ij}}, where\ E_{ij} = \frac{R_i C_j}{N}$$

$E_{ij}$ is the expected frequency if $H_0$ is true, r is the number of variables and k is the number of groups. Where hypothesis $H_0$ is “Measurements from the k groups are from the same distribution”, $H_0$ may be rejected if:

$$X^2 > \chi^2_{\alpha,f}$$

Where f is the degrees of freedom and $\alpha$ is the significance level. The confidence level used is 5%, which enables the use of the Chi-2 test critical values table in (Wohlin et al. 2000).

**Chi-2 test of $H_{01}$:**
$X^2$=1.818
Degrees of freedom: 2
P-value: 0.4029.

**Chi-2 test of $H_{02}$:**
$X^2$=6.967
Degrees of freedom: 6
P-value: 0.3239.

## Interpretation of results

There are a great many things to be said about the results of the experiment. First we discuss the validity threats in this experiment.

### *Validity of results*

The statistical tests used in this study were selected based on the design and scale of the data collected. Since these data has an ordinal scale, non-parametric tests was used. These tests generally have a lower statistical power than parametric tests. The low statistical power threatens conclusion validity and also increases the risk of getting type-I- and type-II-errors during hypothesis testing. Type-I-error has occurred when a pattern is found, when none exist, i.e. we reject $H_0$ when we shouldn't reject it. A type-II-error is when a pattern is not found, when there is an actual pattern, i.e. we don't reject $H_0$ when we should. Type-II-errors are the errors we are most likely to have committed in this study.

Some of the descriptive statistics have involved the spearman rank-order correlation coefficient when determining if there are any correlations between data sets. These measures of dependencies are very much affected by the low statistical power when a non-parametric test is used on data sets that are this small. The measures of dependency that we performed in the results chapter are therefore very unreliable.

Using the Chi-2 test to reject or accept the $H_0$ hypothesis may be reasonable for the type of data collected in the experiment, but the sample size is very small. The Chi-2 test which is used for hypothesis testing on the data of this study is based on a few assumptions. One of these assumptions is if the degrees of freedom f is equal to 0, the test should not be used if any of the expected frequencies are equal to or less than 5. When $f > 1$, the test should not be used if more than 20% of the expected frequencies are less than or equal to 5. Use of the Chi-2 tests is also inappropriate if any expected frequency is 1 or less.

All of the Chi-2 tests done in this study have had more than 20% of the expected frequencies below 5 and some of the frequencies are 1, which means that this test is unsuitable on the data sets. This is a risk which simply has to be accepted, since there are only a very limited amount of samples available. This affects the conclusion validity, since we violate assumptions of statistical tests.

By looking for a specific outcome, the researcher may have chosen a specific set of data and tests that will increase the chance of the desired results to occur. This may have threatened conclusion by the fishing and the error rate threat, but more likely is the error rate part of this threat. Multiple statistical tests at a significance level of 5% may have made the actual significance level increase by power of number of tests.

Data collection was done by having subjects answer questionnaires and researcher studying the subjects case solutions. The nature of these measures is highly subjective and is therefore subject to several validity threats. Also, poor questionnaire wording and unclear questions may cause the reliability of measures to threaten conclusion validity Also, all measurements was subjective in nature, and less reliable than objective measurements.

Since the experiment took place in a computer laboratory that other students had access to, room noise and other students may have disturbed the subjects and affected the results. Also, since the subjects were seated in relatively close proximity, they may have disturbed each other when implementing their treatment or when seeking help from the observer. This may threaten conclusion validity by random irrelevancies in experimental settings.

The pre-experiment survey uncovered a random heterogeneity of subjects. By randomly dividing the subjects between the three objects of study, the resulting distribution of expertise appeared in Figure 9 on page 26. In hindsight, systematic blocking should have been performed, but the researcher believed at the time that randomization was all-important and strived to maximize randomness.

On the other hand, the subjects participating in this study are all 2nd year university students. Their lack of software development in professional context is therefore limited and they may have judged their experience levels too high or too low. It is therefore not unreasonable to assume the level of expertise among the subjects is not as diverse as Figure 7 on page 24 may convey. Figure 8 on page 25 confirms this since only one of the subjects has heard of or had experience with TDD.

Internal validity could have been affected negatively when the online data collection system failed on two occasions, leaving the results of two of the subjects out. These subjects then had to re-submit the questionnaire several days later, and may have forgotten some of the experiences of the experiment. Another instrumentation that may have affected the internal validity, is the problem case used in the experiment. The case may have been unclear and difficult to understand. The questionnaires revealed that most of the problems did however originate from usage of the frameworks, and not the case itself.

The subjects were selected based on volunteers from a course in computer engineering. These volunteers may have been more motivated than subjects selected by other means, and would therefore be less representative of the intended population. The internal validity is therefore threatened by selection.

The selection-maturation interaction may have affected group maturation speeds. One group may have better learning abilities or more expertise than the other groups, and may therefore have matured faster. This is a threat to internal validity by interaction with selection.

Three groups were at close proximity and subjects could have overheard other subjects asking the observer for guidance. Subjects may then have overheard questions that other subject groups have asked the observer. Some of the subjects may also have been friends and told each other about the treatments they were assigned to before the experiment was performed. These things may have influenced the way a subject views the treatments of the study and therefore change their treatment implementation. This diffusion or imitation of treatments may also threaten internal validity.

The post-experiment survey was not designed as well as it could be, since there was a failure to adequately consider the importance of scales and the design types of the experiment. Answers seen in the survey are on an ordinal scale and statistics are therefore limited and the results we can derive from them suffers. This inadequate preoperational explication of constructs is a threat to construct validity and is perhaps the biggest threat to the validity of this study..

The mono-method bias which threatens construct validity may have occurred due to the subjective type of observation. Other threats to construct validity include mono-operation bias, since there is only one problem case used in the experiment. The case described a company needing a web shop portal and then described a set of goals that should be used to make formal functional requirements. This may be insufficient to represent the types of relevant functional requirements.

Using only one problem case for the study may also have affected the construct validity through mono-operation bias, but the small-scale nature of the experiment gives reason to believe that this is not a big threat

The constructs studied are few. Perceived effectiveness and understandability are the studied constructs, while all other constructs are ignored. This may provide an incomplete picture of the object of study, since we won't know what effect the treatments have on other constructs. As a result, the construct validity is threatened by restricted generalizability across constructs.

There are several ways the subjects of this experiment may have guessed the hypothesis of the study. Both pre- and post-experiment surveys may have given hints to what was hypothesized. The experiment design may have given enough information to the subjects so that they have been able to guess the hypothesis and act either positively or negatively according to their guesses. The experiment operation is also a place where the observer unwillingly may have given clues to sufficiently enable hypothesis guessing, which would threaten construct validity. One example of this threat could be shown in Figure 20 on page 32, about what the subjects thought their treatments was really for. They may have guessed that the hypothesis was about frameworks for testing, which prompted them to answer testing. They may have guessed the hypothesis to be about specification, since that's the activity they were told to perform, and answer accordingly. RSpec, which is in fact meant for specification, have a very similar result as the other two.

The subjects of this study were informed about the anonymity of their answers, and the fact that it was not the subjects who were under evaluation, but the frameworks. Still, they may have dressed up their answers to look better. It's a very human reaction, and is hard to explain. One effect of this may be shown in Figure 7 on page 24, where subjects may have overrated their expertise. This evaluation apprehension could threaten the construct validity.

Last of the relevant threats to construct validity in this study is the effect of experimenter expectancies. The researcher may have influenced the results unconsciously by making the questionnaires ask questions in a specific way to make subjects give the expected answers. This threat is high, since there is only one researcher in this study.

The subject population may not be representative of the non-technical population that this study aims to generalize. The subject population is semi-proficient in programming, while the population of interest is "business people" with low to no technical and programming knowledge, but high domain knowledge. The subject population was asked to play the unfamiliar role of customer, rather than developer.

The population chosen for this experiment was the only population available that would not qualify as professional developers. The intention was to test DSL usage for specification by non-technical individuals. Still, part of the population was proficient in computer engineering, and the whole selection had some experience in programming, which puts the external validity of this study in jeopardy through interaction of selection and treatment.

The environment in one of the treatments is not an environment familiar to the non-technical population, namely Eclipse. Also, the context of this experiment was a university computer laboratory, which is not similar to a standard office environment. Also, CubicTest official websites seemed to be offline from the day of the experiment for the duration of a few weeks. Since the subjects were permitted to use all the sources of information they wanted during the experiment, the CubicTest treatment group had a big disadvantage. At the time, CubicTest had a very limited reputation and was not widely known. This lack of information may have influenced the results for the group using CubicTest, since the other two groups still had access to available information. This constitutes a threat to external validity through interaction of setting and treatment.

The experiment was conducted in the late afternoon on a Monday, between 16:00 and 18:00. This may result in a lover motivation for the students to complete the experiment, since it is outside of normal working hours. This interaction of history and treatment may have affected external validity.

To summarize, there are too many threats to validity in effect, and therefore the collected results are questionable. Most of these threats should have been addressed at the experiment design stage, but due to the inexperience of the researcher, the design, operation and data collection suffered serious threats to validity. A single researcher performed this study, which involved a factor with three treatments. A work of this magnitude would benefit by having two or more researchers involved. The results of a larger scale experiment would then be possible, and the results more reliable. Also, a different selection should have been considered that didn't study computer science and information technology. There are many such study programs on NTNU, but convenience and limited time prompted a selection from the students closely available.

## Interpretation of the hypothesis testing

None of the Chi-2 tests achieved the criteria $X^2 > \chi^2_{\alpha,f}$, which allows us to reject a null-hypothesis. In addition, the p-values were in the area of 0.4, which means statistically un-significant results. To achieve statistical significance, the p-value would have to be less than the significance level 0,05, which was chosen for this study.

## Discussion

Acceptance testing, like performed in all too many development projects to day, is something a customer performs at the end of a project to check if the system developed passes their acceptance criteria. This places high risk on a project, since the customer may find out that the system is not up to their standards. By making acceptance tests play the role of specifications through domain specific languages, developers can gradually fulfill the customer acceptance criteria through the implementation phase.

Specification documents are in most cases still written by customers and handed over to developers. These documentations are often saturated with design choices that should be made by developers. Some development teams have already started changing this by having customers write specifications on story cards. Story cards are also a good format for making the specifications into acceptance test. This is an important point in BDD practice, but customers can't always make such acceptance tests without the help of developers.

The vision is to have the customer deliver not a specification document, but a fully functional acceptance test suite based on a domain specific language. For this vision to become real, the domain specific languages must become easier to understand, more publicized, better documented with user guides and examples and also better researched.

Fit is currently a de-facto standard in acceptance testing, and is most commonly recognized through the wiki-system FitNesse. Fit can be used in an environment like Microsoft Excel, an environment which everyone should have some familiarity with. A familiar environment in combination to the readily available documentation for Fit and FitNesse should make any non-technical individual able to learn how to write acceptance tests.

But by using Fit for specification, a domain specific language needs to be established. Fixtures are the elements of Fit which enforces this DSL. Most commonly, developers and "business people" collaborate to make acceptance tests. The "business people" structure their acceptance criteria through tables and developers help formalize the table structures and naming into the DSL that fixtures can work with.

For "business people" to be able to create specifications from Fit as readily as Use Cases on paper, they need to be able to recognize the importance of the DSL. The results of this experiment did not show any evidence of a structured DSL in the Fit tests. It may be that this is very difficult without understanding how fixtures work, or the subjects may just have missed this point completely.

RSpec is the only framework in the study that is meant to be used in specification and not testing. Much of the philosophy and language use of BDD, which RSpec is based on, is meant to draw thought patterns away from testing activities of TDD and similar methods and into specification activities. By using a syntax that resembles the natural way the English language would be used to describe a desired behavior.

There are however still issues to deal with if "business people" are to create functional requirements on their own with RSpec. One of the RSpec subjects in the experiment pointed out how currently available information and examples in RSpec assumed a certain familiarity with the Ruby programming language. This would be a major obstacle for non-technical "business people". Users like that have most likely never produced a line of code in their entire life. If "business people" are ever to be expected to use this framework, steps must be taken to further transform programming-like syntax into natural language.

BDD places some emphasis on user stories written on story cards, and aims to use these story cards as a stepping stone to formalize functional requirements in RSpec. This experiment didn't use such stepping stones, but rather asked the subjects to go directly from fairly vague goals, to formal functional specifications. Using story cards may reduce the need for understandability in RSpec, by having the structure of the user story formally transfer to a structured example in RSpec.

CubicTest is a framework that is based on modeling different states of a web based system. This approach to acceptance testing seems promising, since it uses models to convey functional requirements. Models can be easier to understand than tables in Fit and lines of code in RSpec, but the state concept can be difficult to grasp for non-technical "business people". This was also mentioned by a subject assigned to CubicTest.

Also mentioned by the subjects of the experiment, was the inability to find useful information of usage for CubicTest. As luck would have it, all sites containing information about CubicTest and its previous version, AutAt, seemed to be offline on the day of the experiment. Since all helping material was allowed, this caused some difficulties for the CubicTest group, which didn't have access to any.

Another source of questions was whether all the functional requirements should be expressed in the same model like an UML domain model or class diagram, or made into separate models for the separate requirements. The observer was unable to answer these questions, mainly not to influence the result of the experiment, but also because the observer didn't know the answer. 2 out of 3 subjects chose to express all requirements in a single model.

The state of maturation may explain the lack of information and examples for CubicTest. This is still quite new, and in time, developers and "business people" alike may learn of CubicTest and decide to give it a try.

## Conclusions and possible improvements

This study has taken a few twists and turns along the way, and ended up unable to reject any of its null-hypothesis. $H_{01}$, $H_{02}$ and $H_{03}$ are therefore not rejected, but that doesn't mean they are true. This simply means that we cannot conclude anything based on the results of the experiment.

This doesn't mean that the study is a failure, but more a learning experience and perhaps reading material for someone who would like to perform a lager scale experiment on the same topic. The problems encountered in this study may help others steer clear of the many pitfalls and uncertainties that this study experienced.

Probably the only serious concern uncovered in this study, is the lack of documentation for RSpec and CubicTest. Fit is the only framework in this study that has books written about it, and the only one of these three frameworks that is readily known among non-technical users as an acceptance testing framework. RSpec and BDD are starting to get some attention, but it is at this time still only in internet articles and blogs. CubicTest has no other channel of documentation distribution other than its official home site www.CubicTest.org and boss.bekk.no. It is still early in development and CubicTest hasn't had time to receive much publicity yet, but there is great potential in this method, since it relies on modeling, and not anything resembling programming.

We recognize that development methodologies like these will start out and develop on blogs and in informal developer communities, but if non-technical users are to learn how to use RSpec and CubicTest for specifications, the work on documentations and user guides must start. This is due to the fact that non-technical people generally don't frequent technical blogs and forums, or other haunts of developers and gurus.

One of the experiments flaws was to only have a set of goals that was to be formalized into functional requirements. In hindsight, these goals should have been supplied as user stories on story cards. The subjects could alternatively be asked to first write story cards of their own based on these goals, before attempting to produce functional requirements in any framework. In agile software development, story cards play a big role in specifications, and should not have been neglected in this experiment. Further work could possibly involve a framework or interface for formal transformation of story card structure into RSpec examples, Fit/Fixtures or CubicTest models.

## References

Beck, K. (2000). Extreme Programming Explained: Embrace Change, Addison-Wesley.

Canfora, G., A. Cimitile, F. Garcia, M. Piattini and C. A. Visaggio (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. Proceedings of the 2006 ACM/IEEE international symposium on empirical software engineering. Rio de Janeiro, Brazil, ACM Press**:** 364-371.

Conboy, K. and B. Fitzgerald (2004). "Toward a conceptual framework of agile methods: a study of agility in different disciplines." Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research: 37-44.

Erdogmus, H., M. Morisio and M. Torchiano (2005). "On the Effectiveness of the Test-First Approach to Programming." IEEE Transactions on Software Engineering **31**(3): 226-237.

Evans, E. (2004). Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston, Addison-Wesley.

Fowler, M., J. Highsmith, K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, J. Grenning, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas. (2001). "Manifesto for Agile Software Development." from http://agilemanifesto.org/.

Freeman, S., T. Mackinnon, N. Pryce and J. Walnes (2004). Mock Roles, Not Objects. Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications (OOPSLA '04). Vancouver, BC, CANADA, ACM Press**:** 236-246.

Jones, C. G. (2004). "Test-driven development goes to school." Journal of Computing Sciences in Colleges **20**(1): 220-231.

Kalitina, E. and J. R. Myhre (2006). CubicTest Usability. Trondheim, Norway, Norwegian University of Science and Technology (NTNU).

Kaufmann, R. and D. Janzen (2003). Implications of test-driven development: a pilot study. Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA '03). Anaheim, CA, USA, ACM Press**:** 298-299.

Kerenyi, C. (1976). Dionysos: Archetypal Image of Indestructible Life, Princeton University Press.

Loe, K. and S. L. N. Olsen (2006). Automatisert testing av dynamisk HTML. Trondheim, Norway, Norwegian University of Science and Technology (NTNU).

Maximilien, E. M. and L. Williams (2003). Assessing test-driven development at IBM. Proceedings of the 25th International Conference on Software Engineering. Portland, Oregon, IEEE Computer Society: 564-569.

Nonaka, I. and H. Takeuchi (1995). The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation, Oxford University Press.

Schwaber, K. (1995). "SCRUM Development Process." OOPLSA'95 Workshop on Business Object Design and Implementation.

Skytteren, K. and T. M. Øvstetun (2005). AutAT - Automatoc Acceptance Testing of Web Applications. Trondheim, Norway, Norwegian University of Science and Technology (NTNU).

Steinberg, D. H. (2001). The effect of unit tests on entry points, coupling and cohesion in an introductory Java programming course. Proceedings of 2001 XP Universe. Raleigh, NC, USA.

Whorf, B. L. (1956). Language, thought, and reality, MIT Press Ltd.

Wohlin, C., P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, Eds. (2000). Experimentation in Software Engineering - An Introduction. The Kluwer International Series in Software Engineering. Boston, Kluwer Academic Publishers.

# Appendix A – Pre-experiment survey

## INNLEDENDE SPØRRESKJEMA

### Page1

**i1**

Disse spørsmålene er for å få en oversikt over kunnskapsnivået innen programmering, spesifisering og testing. Husk at svarene vil holdes konfidensielt og det er ikke du som deltaker som blir evaluert, men metoder og rammeverk.

**q8**

Velg hvilket studie du tar

❍ Data (1)
❍ Komtek (2)
❍ Informatikk (3)
❍ Indøk (4)
❍ Annet (5)

**q9**

Velg hvilken klasse du går i

❍ 1 (1)
❍ 2 (2)
❍ 3 (3)
❍ 4 (4)
❍ 5 (5)
❍ Stipendiat (6)

**q2**

Hvor høy er din ekspertise innen programmering?

❍ Nybegynner (Hatt innledende fag / introduksjonskurs / grunnleggende programmeringsfag) (1)
❍ Viderekommende (Hatt ett eller flere avanserte programmeringsfag) (2)
❍ Avansert (Hatt flere avanserte programmeringsfag og programmerer på fritid) (3)
❍ Ekspert (Hatt flere avanserte programmeringsfag, ett eller flere sertifikater og programmerer på fritid/jobb) (4)

**q3**

Merk av følgende aternativ som du har kjennskap til og erfaring med

❑ UML (1)
❑ Use Case (2)
❑ RSpec (3)
❑ Fit / FitNesse (4)
❑ AutAT / CubicTest (5)
❑ Test-Driven Development (TDD) (6)

# Appendix B – Post-experiment survey

## FULLFØRT EKSPERIMET

### Page1

**i1**

Rammeverkets oversiktelighet

**q2**

Velg hvilketrammeverk du ble bedt om å bruke

❍ RSpec (1)
❍ Fit / FitNesse (2)
❍ CubicTest (3)

**q13**

Hvor mye tid brukte du på lesing og forberedelser til eksperimentet?

❍ Under en time (1)
❍ 1 time (2)
❍ 2 timer (3)
❍ 3 timer (4)
❍ 4 timer (5)
❍ 5 timer eller mer (6)

**q35**

Følte du at du var godt nok forberedt ved oppmøte til eksperimentet?

❍ Ja (1)
❍ Nei (2)

**q4**

Hvor vanskelig var det å få oversikt over rammeverket?

❍ Svært lett (1)
❍ Lett (2)
❍ Middels (3)
❍ Vanskelig (4)
❍ Svært vanskelig (5)

**q5**

Gi en kort begrunnelse av forige spørsmål.Hva var lett å forstå?Hva var vanskelig?Var dokumentasjonen god nok?

**q34**

Fantes det elementer i dokumentasjonen som virket motstridende, forvirrende eller missledende?

**q29**

Var det forskjeller mellom dokumentasjonen og bruken av rammeverket?

## End of Page1

## Page2

**i6**

Rammeverk i bruk

**q7**

Hvor vanskelig var det å bruke syntaksen i rammeverket?

❍ Svært lett (1)
❍ Lett (2)
❍ Middels (3)
❍ Vanskelig (4)
❍ Svært vanskelig (5)

**q9**

Hva var de eventuelle vanskelighetene med syntaksen?

**q10**

Hvilke funksjonelle krav hadde du problemer å uttrykke, blandt de du forsøkte på?

**q11**

Hvilke problemer opplevde du med å uttrykke kravene?

**q27**

Hvor kom vanskelighetene fra?

| | Ingen | Litt | Moderat | Mye |
|---|---|---|---|---|
| Caset / kravenes utforming (1) | ❍ | ❍ | ❍ | ❍ |
| Rammeverket (2) | ❍ | ❍ | ❍ | ❍ |

**q33**

Fantes det elementer i rammeverket som virket motstridende, forvirrende eller missledende?

**q30**

Hvordan tror du det ville vært å utrykke kravene med Use Case i motsetning til rammeverket?

❍ Lettere (1)
❍ Ca. samme (2)
❍ Vanskeligere (3)

**End of Page2**

**Page6**

**i36**

Test mot spesifisering

**q38**

Hvilke av disse utsagnene ligger nærmest til det du prøvde å oppnå?

❍ Jeg prøvde å teste en internettside som ikke var lagd ennå (1)
❍ Jeg prøvde å spesifisere tester til et system som ikke var lagd ennå (2)
❍ Jeg prøvde å spesifisere en internettside (3)
❍ Jeg prøvde å skrive tester til et system som ikke var lagd ennå (4)
❍ Jeg prøvde å spesifisere oppførselen til et system som skulle lages (5)
❍ Jeg vet ikke helt hva jeg prøvde (6)

**q39**

Hva var rammeverket du brukte mest tilrettelagt for?

❍ Testing (1)
❍ Spesifisering (2)
❍ Vet ikke (3)

**q40**

Hvilke vanskeligheter hadde du med å skrive tester for noe som ikke eksisterer?

**End of Page6**

**Page3**

i12

Observatørens rolle

q21

Hvor mye hjelp ønsket du fra observatøren?

❍ Ingen (1)
❍ Litt (2)
❍ Moderat (3)
❍ Mye (4)

q24

Hva ønsket du hjelp om?

| | Ingen | Litt | Moderat | Mye |
|---|---|---|---|---|
| Om Caset / kravene (1) | ❍ | ❍ | ❍ | ❍ |
| Om rammeverket (2) | ❍ | ❍ | ❍ | ❍ |

q25

I hvilken grad mottok du hjelp?

| | Ingen | Litt | Moderat | Mye |
|---|---|---|---|---|
| Om Caset / kravene (1) | ❍ | ❍ | ❍ | ❍ |
| Om rammeverket (2) | ❍ | ❍ | ❍ | ❍ |

q31

Hvilke typer hjelp ga obsevatøren deg?

❍ Ingen (1)
❑ Om syntaks (2)
❑ Om Caset (3)
❑ Om riktig bruk av rammeverket (4)
❑ Spesifik info om hvordan løse kravene i caset (5)

q42

Hvordan og i hvilken grad påvirket observatøren deg til å løse caset på den måten du prøvde?

# Appendix C – Problem Case

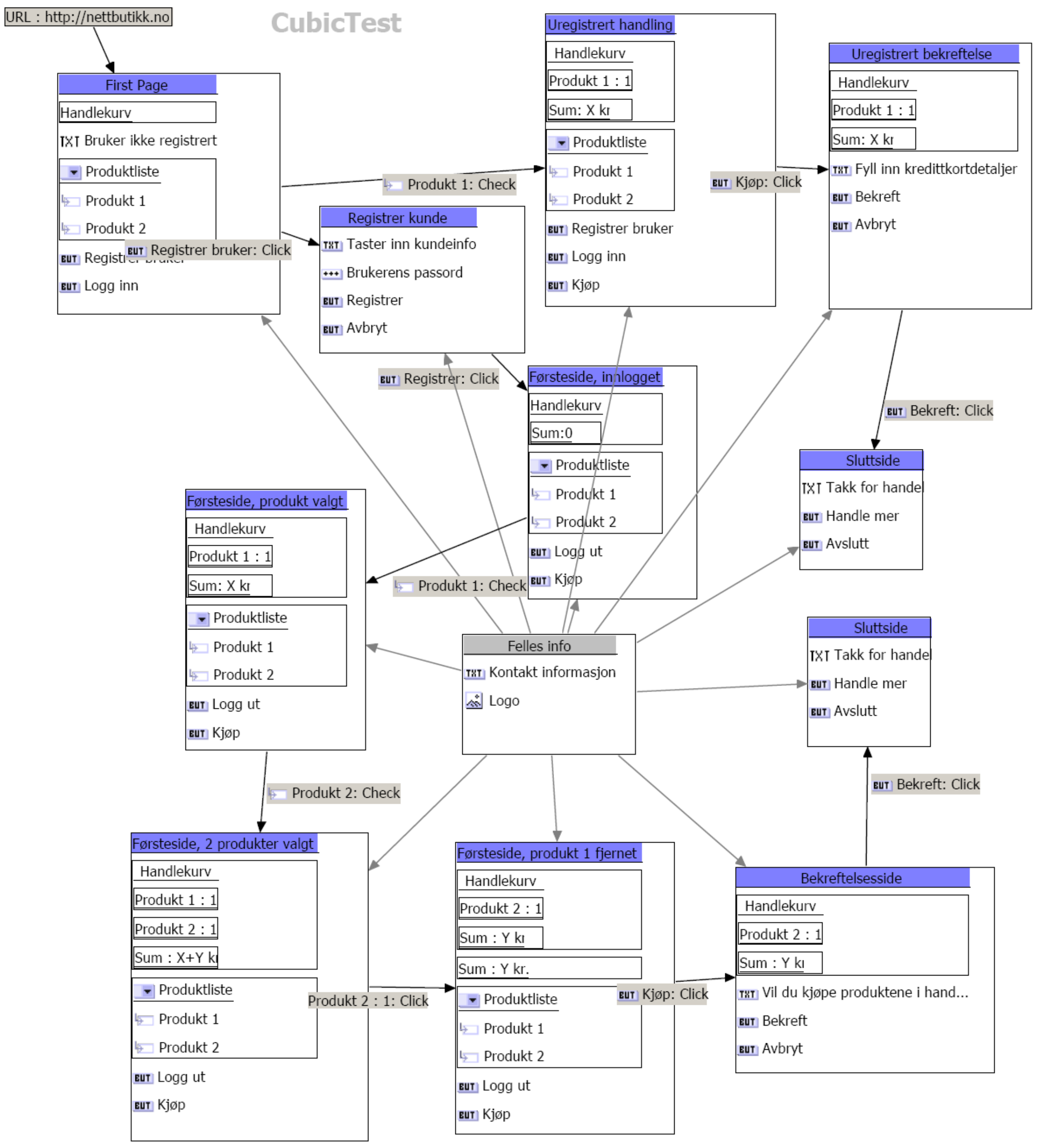
En ny nettbutikk skal utvikles for din bedrift, som selger gadgets og duppeditter. Før du gir oppdraget til systemutviklere og konsulenter, ønsker du å spesifisere funksjonelle krav som utviklere skal implementere. Utviklerne har bedt deg om å spesifisere på en spesifikk måte, slik at de kan kjøre spesifikasjonene som om det var tester. Etter å ha blitt referert til diverse internettsider om rammeverket som skal brukes, vil du nå bruke litt tid på følgende utvalgte forslag til funksjonelle krav:

- Uregistrert kunde skal kunne opprette brukerkonto og registrere betalingsinfo
- Både registrert og uregistrert kunde skal kunne velge blant produkter
- Valgte produkter skal til enhver tid vises med antall og sum i en ”handlekurv”
- Kunde skal kunne fjerne varer fra handlekurven
- Registrert kunde skal kunne betale for valgte varer med betalingsinfo på brukerkonto

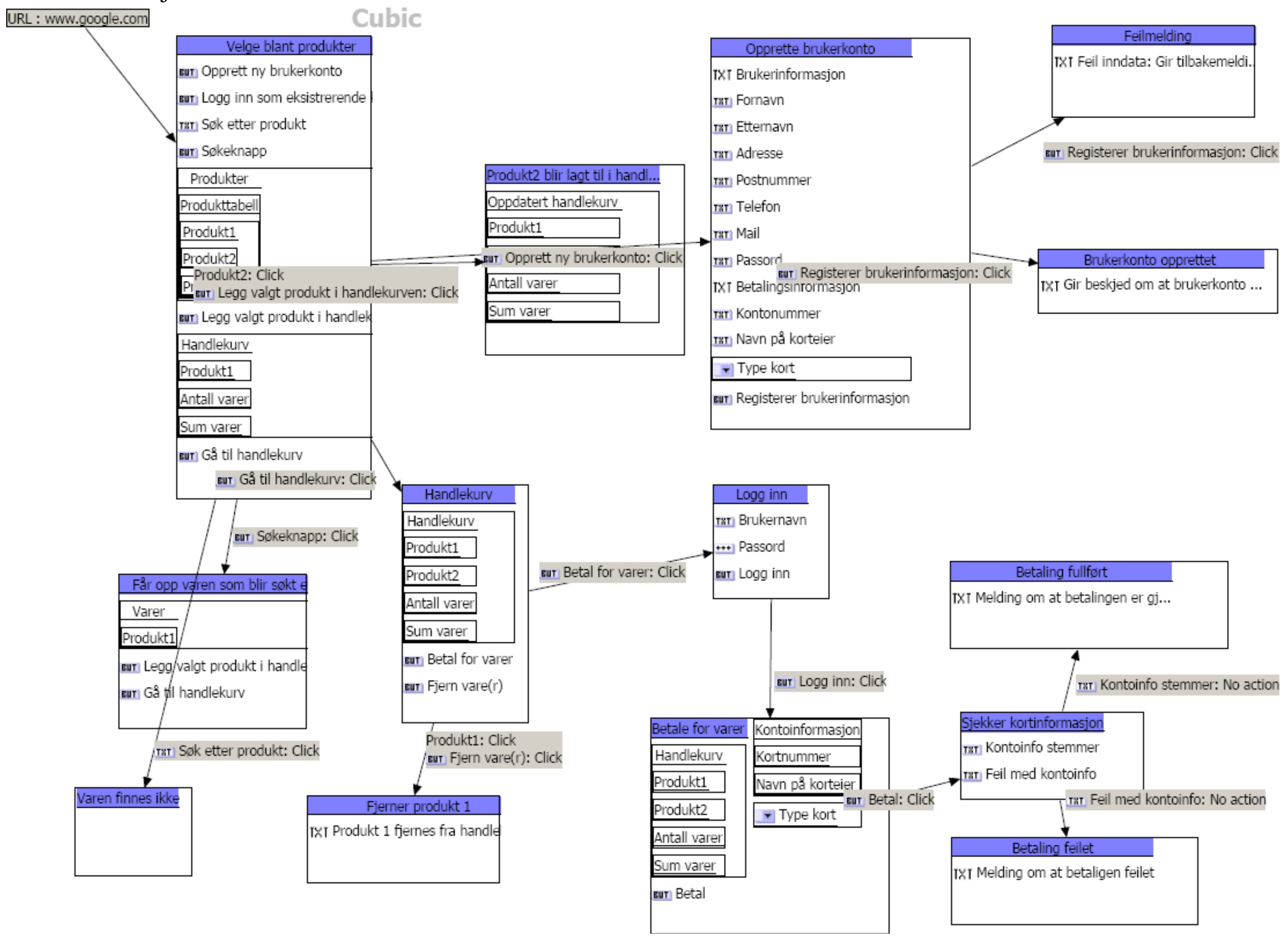
# Appendix D – Problem Case solutions

CubicTest solutions: The last four CubicTest models are representing the different functional requirements of the same case, submitted by the same subject. The Fit answers have to be submitted as a packaged file, since the tables won't fit in this document.

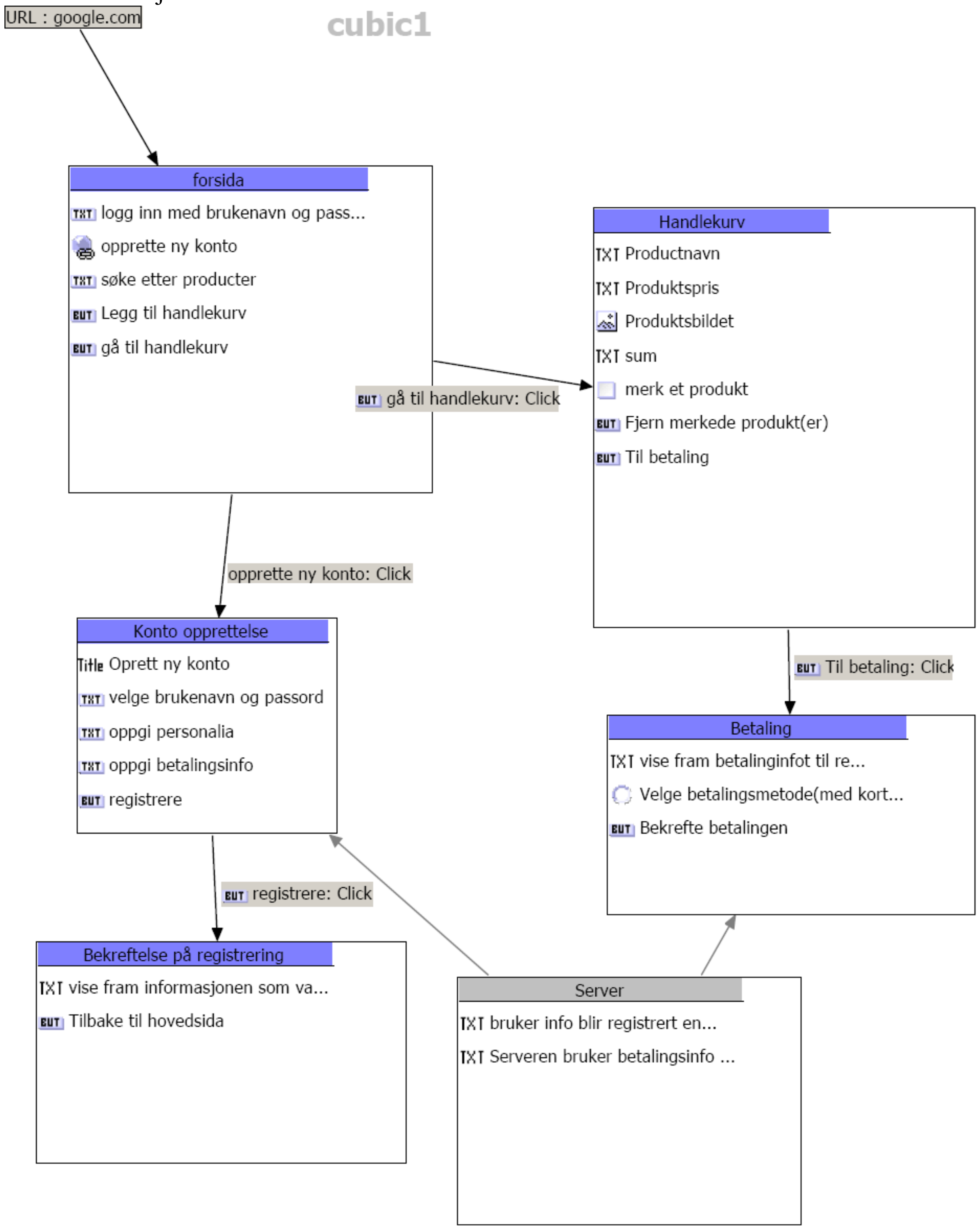
CubicTest Subject 1:



CubicTest Subject 2:

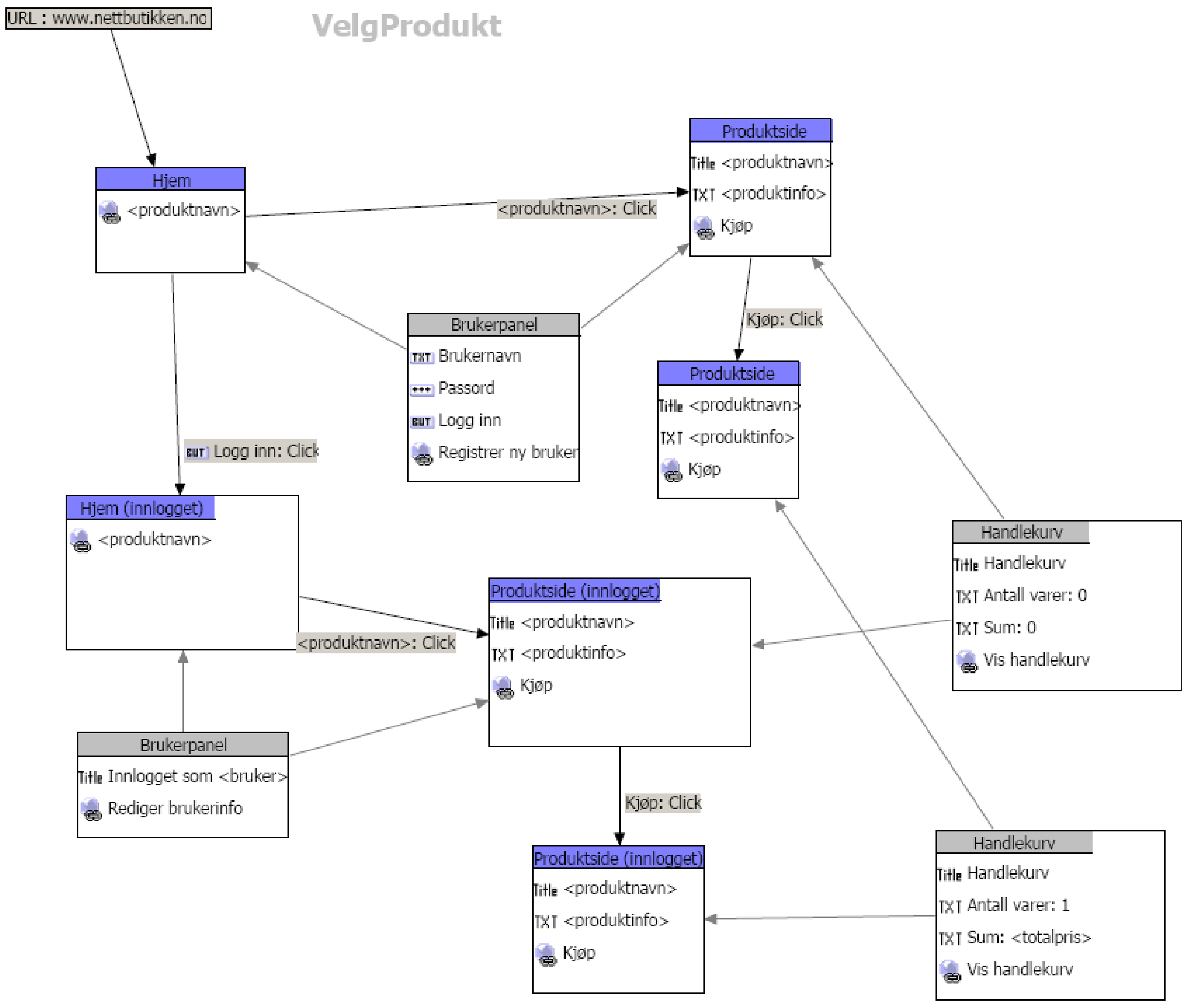


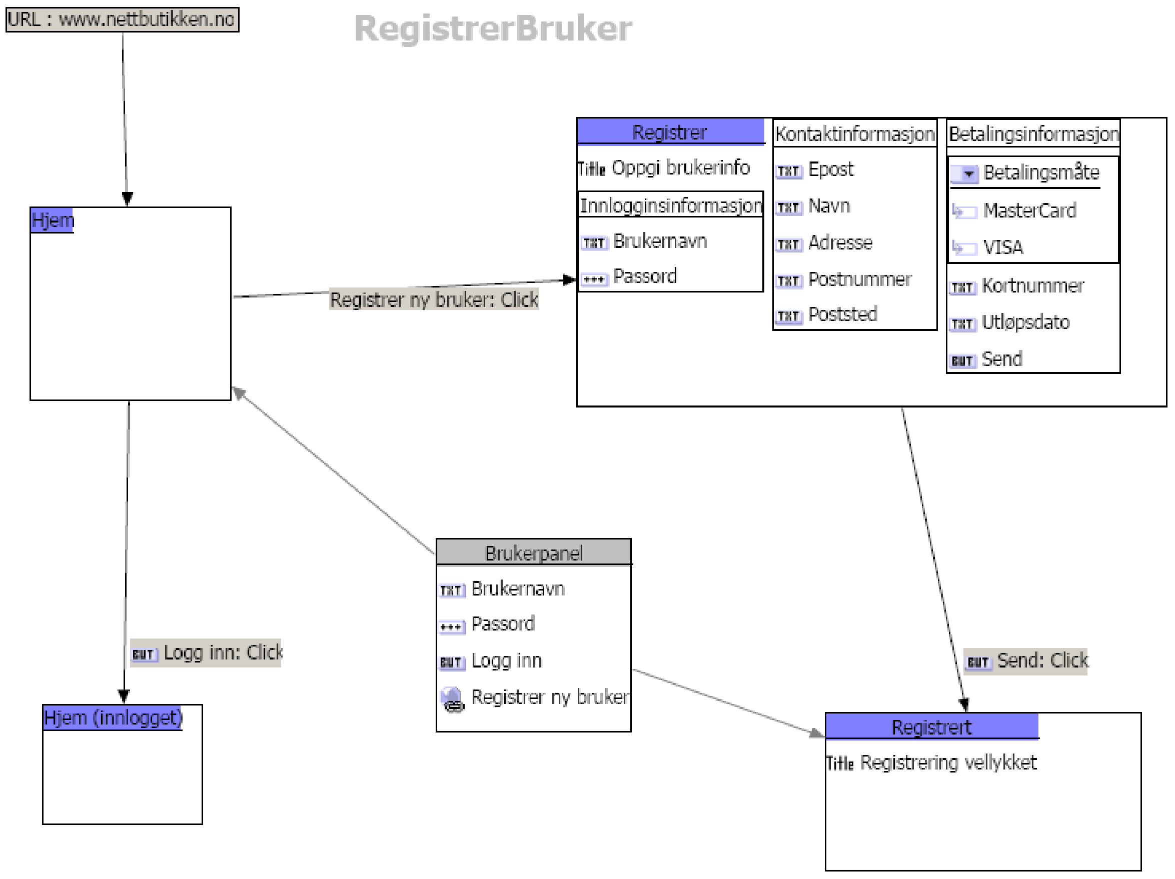
CubicTest Subject 3:



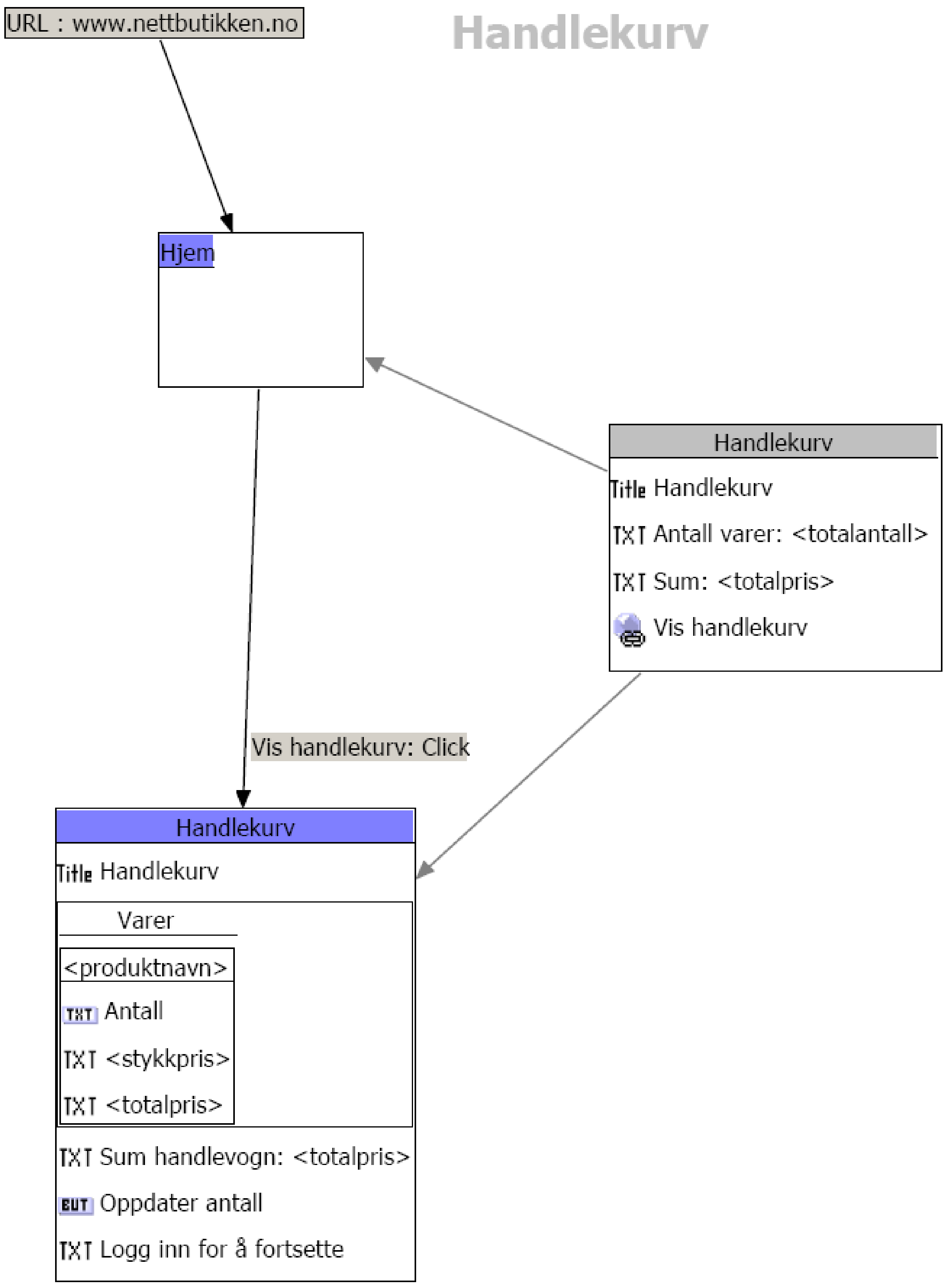
The next four models are by the same subject and represent different aspects of the problem domain:

CubicTest Subject 4:

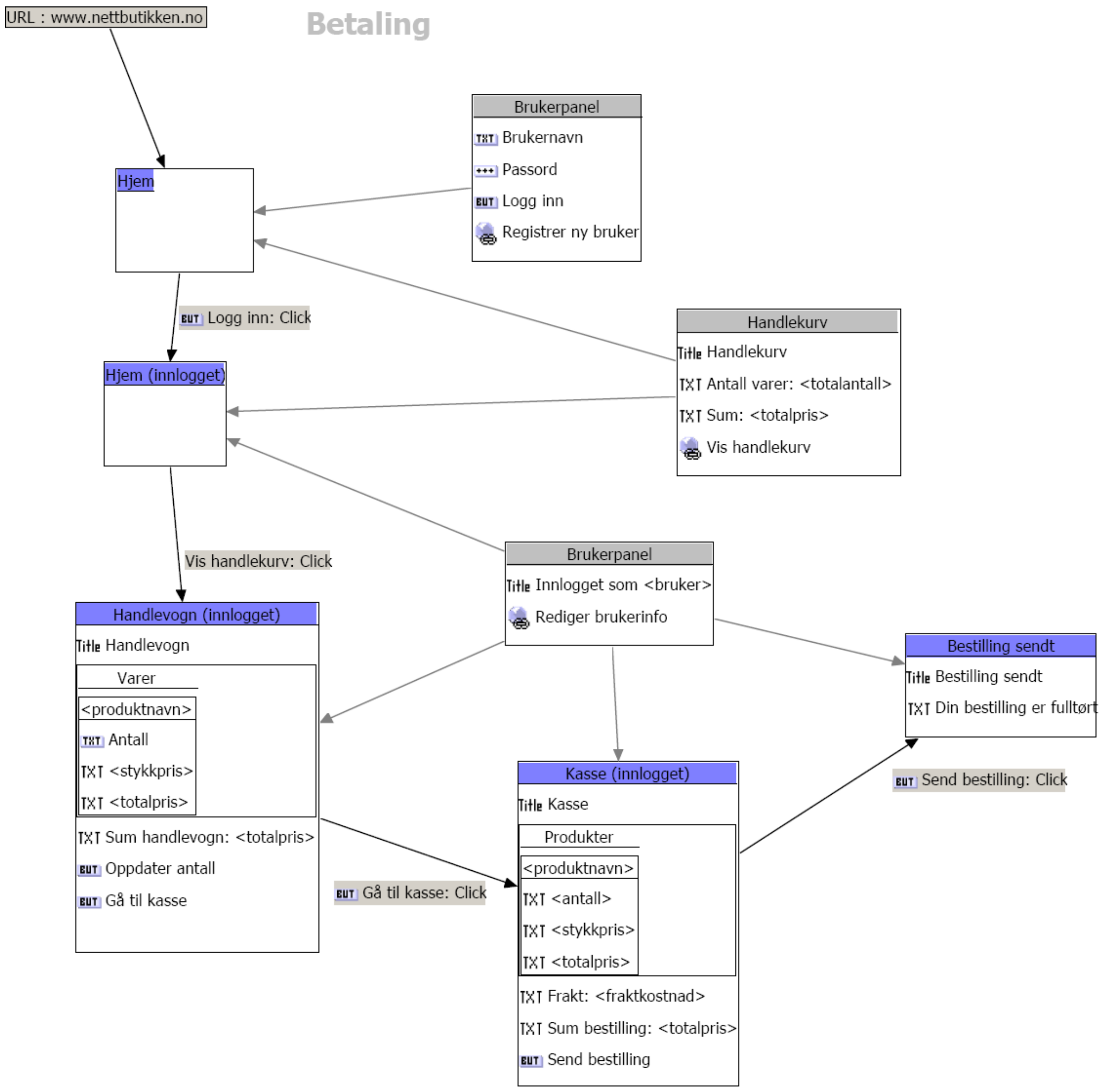



URL : www.nettbutikken.no
RegistrerBruker
Hjem
Registrer ny bruker: Click
Registrer
Title Oppgi brukerinfo
Innlogginsinformasjon
TXT Brukernavn
Passord
Kontaktinformasjon
TXT Epost
TXT Navn
TXT Adresse
TXT Postnummer
TXT Poststed
Betalingsinformasjon
Betalingsmåte
MasterCard
VISA
TXT Kortnummer
TXT Utløpsdato
BUT Send
Brukerpanel
TXT Brukernavn
Passord
BUT Logg inn
Registrer ny bruker
BUT Logg inn: Click
Hjem (innlogget)
BUT Send: Click
Registrert
Title Registrering vellykket

## Handlekurv



URL : www.nettbutikken.no
Hjem
Handlekurv
Title Handlekurv
TXT Antall varer: <totalantall>
TXT Sum: <totalpris>
Vis handlekurv
Vis handlekurv: Click
Handlekurv
Title Handlekurv
Varer
<produktnavn>
TXT Antall
TXT <stykkpris>
TXT <totalpris>
TXT Sum handlevogn: <totalpris>
BUT Oppdater antall
TXT Logg inn for å fortsette


URL : www.nettbutikken.no
Betaling
Hjem
Brukerpanel
Brukernavn
Passord
Logg inn
Registrer ny bruker
Logg inn: Click
Handlekurv
Title Handlekurv
TXT Antall varer: <totalantall>
TXT Sum: <totalpris>
Vis handlekurv
Hjem (innlogget)
Vis handlekurv: Click
Brukerpanel
Title Innlogget som <bruker>
Rediger brukerinfo
Handlevogn (innlogget)
Title Handlevogn
Varer
<produktnavn>
TXT Antall
TXT <stykkpris>
TXT <totalpris>
TXT Sum handlevogn: <totalpris>
BUT Oppdater antall
BUT Gå til kasse
BUT Gå til kasse: Click
Kasse (innlogget)
Title Kasse
Produkter
<produktnavn>
TXT <antall>
TXT <stykkpris>
TXT <totalpris>
TXT Frakt: <fraktkostnad>
TXT Sum bestilling: <totalpris>
BUT Send bestilling
Bestilling sendt
Title Bestilling sendt
TXT Din bestilling er fulltørt
BUT Send bestilling: Click

The next 3 solutions are in RSpec:

RSpec Subject 1:

```
## KILDER
#
http://66.102.9.104/search?q=cache:h1YAqRpmBsoJ:blog.daveastels.com/files/QuickRef.pdf
+rspec+examples&hl=no&ct=clnk&cd=15&gl=no&client=firefox-a
# http://rspec.rubyforge.org
# http://eigenclass.org/hiki/xmpfilter
##
class GeneralCustomer
        def SelectedItem;
        def productCount;

        def allow_to_change
                @generalCustomer.insert "lastProductCount"
                @generalCustomer.insert "ProductCount"
                assertNotEqual(lastProductcount, ProductCount)
        end
end

context "general customer" do
        setup do
                @generalCustomer = GeneralCustomer.new
                def lastProductCount;
        end
        specify "should be able to choose among products" do
                @generalCustomer.selectedItem.should_not_be null
        end
        specify "should be able to remove products from basket" do
                @generalCustomer.productCount.should allow_to_change
        end
end

class UnregisteredCustomer
end
context "unregistered customer" do
        setup do
                @unregisteredCustomer = UnregisteredCustomer.new
        end
        specify "should be treated as a general customer" do
                @unregisteredCustomer.shoule_be_an_instance_of
@generalCustomer
        end
        specify "should be able to create account"
                @registeredCustomer.should_not_be null
        end
        specify "should be able to register payment info" do
                @registeredCustomer.paymentInfo.should_not_be null
                end
end

class RegisteredCustomer
                def paymentInfo;
                def sumPaid;
end
context "registeredCustomer" do
                setup do
                                @registeredCustomer = RegisteredCustomer.new
                end
                specify "should be treated as a general customer" do
                                                @registeredCustomer.should_be_an_instance_of
@generalCustomer
                end
                specify "shoudl be able to pay for products using registered info" do
                                @registeredCustomer.paymentInfo.should_not_be null
                                @registeredCustomer.sumPaid.should_not <= -1
                end
end

class Basket
                def SelectedProducts;
                def totalSum;
end
context "basket" do
                setup do
                                @basket = Basket.new
                end
                specify "should show info about chosen products"
                                                @basket.selectedProducts.should_not_equal 0
                                                @basket.selectedProducts.should_not_be null
                                                @basket.totalSum.should >= 0
                end
end
```

RSpec Subject 2:

```
require 'user'
require 'account'
require 'basket'
require 'item'
require 'order'

context "A user" do
  setup do
    @user = User.new
    @item1 = Item.new
    @item1.prize = Money.new(10, :dollars)
    @item2 = Item.new
    @item2.prize = Money.new(20, :dollars)
  end
  specify "can add items to a basket" do
    @basket = Basket.new
    @basket.owner = @user
    @basket.add @item1
    @basket.should_have(1).things
    @basket.should_include @item1
    @basket.add @item2
    @basket.should_have(2).things
    @basket.should_include @item2
  end
  specify "can get size and cost of the basket" do
    @basket.size.should_equal 2
    @basket.cost.should_equal Money.new(30, :dollars)
  end
  specify "can remove items from the basket" do
    @basket.remove @item1
    @basket.should_have(1).things
    @basket.should_not_include @item1
    @basket.should_include @item2
  end
end

context "An unregistered user" do
  setup do
    @user = User.new
    @user.set_registered false
  end
  specify "can register a new account" do
    @account = Account.new
    @account.register @user
    @account.registered_user.should_equal @user
  end
end
```

```
context "A registered user" do
  setup do
    @user = User.new
    @user.set_registered true
  end
  specify "can enter payment info" do
    @user.payment_info "123"
    @user.payment_info.should_equal "123"
  end
  specify "can pay for a basket with entered payment info" do
    @basket = Basket.new
    @basket.owner @user
    @basket.add @item1
    @basket.add @item2
    @order = Order.new
    @order.set_owner @user
    @order.set_basket @basket
    @order.check_out
    @order.payment_info.should_equal "123"
  end
end
```

RSpec Subject 3:

```
require 'customer'
require 'cart'
require 'product'

context "en ny handlevogn" do
                setup do
                                @cart = cart.new
                                @product = product.new
                end
                specify "skal være tom når den blir opprettet"
                                @card.should be_empty
                end
                specify "skal kunne registrere produkter"
                                @cart.registerProduct(@product).should_not raise_error()
                end

                specify "skal ikke være tom når et produkt er registrert"
                                @cart.registerProduct(@product)
                                @cart.should_not be_empty
                end
end

context "en handlevogn med produkter" do
                setup do
                                @cart = cart.new
                                @product = product.new
                                @cart.registerProduct(@product)
                end
                specify "skal kunne registrere produkter"
                                @cart.registerProduct(@product).should_not raise_error()
                end

                specify "skal kunne fjernes produkter fra"
                                @cart.unregisterProduct(@product).should_not raise_eror()
                end

                specify "skal ikke kunne fjerne et produkt den ikke har i seg"
                                @product2 = Product.new;
                                @cart.unregisterProduct(@product2).should raise_error()
                end

                specify "skal ha et produkt på første plassering"
                                @cart.productAt(0).should_not be_nil
                                @cart.productAt(0).should_eql @product
                end

                specify "skal vise antall varer"
                                @cart.numberOfProducts().should_eql 1
```

```
                        @cart.unregisterProduct(@product)
                        @cart.numberOfProducts().should_eql 0
            end

            specify "skal vise pris på varer"
                        @cart.priceOfProducts().should_not_eql @product.price
                        @cart.unregisterProduct()
                        @cart.priceOfProducts().should_eql 0
            end
end

context "en  kunde" do
            setup do
                        @customer = customer.new
                        @product = product.new
            end
            specify "skal kunne velge blandt produkter"
                        @customer.chooseProduct(product).should_not raise_error()
            end

            specify "skal ha en handlevogn"
                        @customer.cart.should_not be_nil
            end
end

context "En uregistrert kunde" do

            setup do
                        @customer = customer.new
            end
            specify "en ny kunde skal ikke være registeret"
                        @customer.should_not be_registered
            end

            specify "en ny kunde skal kunne registreres og etter dette bli registrert"
                        @customer.register() // skal sikkert registreres med noe input,
dette er utelatt
                        @customer.should be_registered
            end

            specify "skal kunne registrere betalingsinfo"
                        @customer.register()
                        @customer.registerPayment()
                        @customer.should be_registered
                        @customer.should be_registeredPayable
            end

            specify "skal ikke kunne betale for valgte varer"
                        @customer.payForSelected().should raise_error()
            end
```

```
end

context "en registrert kunde" do
        setup do
                @customer = customer.new
                @customer.register() // skal ha indata, men disse er utelatt
                @customer.registerPayment()
        end

        specify "skal være registrert"
                @customer.should be_registered
                @customer.should be_registeredPayable
        end

        specify "skal kunne betale for valgte varer"
                // skal ha inndata, men disse er utelatt
                @customer.payForSelected().should_not raise_error()
        end
end
```