KTH Computer Science
and Communication

Automatic Testing of Modern Web Applications in an Agile Environment

A case study of testing a Google Web Toolkit Web Application using Selenium

SIMON LUNDMARK

Bachelor's Essay at CSC
Supervisor: Alexander Baltatsis
Examiner: Mads Dam

TRITA xxx yyyy-nn

Abstract

Software testing is generally considered difficult but necessary. With the growing popularity of agile software development methodologies it is becoming increasingly necessary to automate this difficult task. Modern web applications using AJAX technology require a slightly modified testing process.

The primary aim of this study is to highlight some problems regarding automatic testing of modern web applications. The secondary goal is to evaluate these suggested problems by performing a case study, comparing the process of testing a traditional JSP implementation without AJAX technology, with the process of testing a modern implementation implemented using Google Web Toolkit. The automatic testing tool Selenium will be used to record and play the test suite.

Testing the traditional web application implementation was very straight forward. Testing the modern AJAX web application implementation was not. Work-arounds do exist and once that lesson is learned Selenium is a welcomed tool used together with Google Web Toolkit.

Sammanfattning

Testning av programvara anses generellt vara svårt men nödvändigt. Med den växande popularitet av agila mjukvaruutvecklingsmetoder är det allt viktigare att automatisera denna svåra uppgift. Moderna webbapplikationer byggda med AJAX-teknik kräver en något modifierad testprocess.

Det primära syftet med denna studie är att belysa några problem som berör automatisk testning av moderna webbapplikationer. Det andra målet är att utvärdera dessa föreslagna problem genom att utföra en fallstudie som jämför att testa en traditionell JSP implementation utan AJAX-teknik, med att testa en modern implementation implementerad med hjälp av Google Web Toolkit. Det automatiserade testverktyget Selenium används för att spela in och spela upp testsviten.

Att testa den traditionella webbapplikationes implementation var rakt på sak. Att testa den moderna AJAX-webbapplikationens implementation var inte det. Sätt att kringgå problemen existerar och när den lektionen är lärd är Selenium ett välkommet verktyg att användas tillsammans med Google Web Toolkit.

Contents

Chapter 1

Introduction

1.1 Background

1.1.1 Software Testing

Software testing has always been an important aspect of computer science. It has at the same time been a very neglected aspect of computer sience and software development. There has even been claims that computer science is not even a science, partly due to the low level of testing. In an article titled "Is Computer Science Science?" written by Peter J. Denning in 2005 he discusses and defends these claims but in the last section he states the following:

> "In a sample of 400 computer science papers published before 1995, Walter Tichy found that approximately 50% of those proposing models or hypotheses did not test them. In other fields of science the fraction of papers with untested hypotheses was about 10%. Tichy concluded that our failure to test more allowed many unsound ideas to be tried in practice and lowered the credibility of our field as a science. The relative youth of our field—barely 60 years old—does not explain the low rate of testing." [Denning, 2005]

The theory of software testing has largely been the same for more than three decades. This can be seen by reading the preface of the second edition of "The Art of Software Testing" published [Myers et al., 2004]. The first edition was published in 1979 and the major changes in the second edition covers topics that no one knew about like web programming and agile programming methodologies[1].

Testing web applications can therefore be considered a relatively new field of study. To make matters more interesting, web applications in general has already branched into what is defined in this essay as *traditional* and *modern* web applications. The following sections will state the difference between them.

[1] E.g. eXtreme Programming and Test-Driven Development

1.1.2 Traditional Web

The building blocks of traditional static web sites are *pages* (web pages). The user navigates between these pages using hyperlinks. Following a hyperlink on one page brings the user to another page. The web site is considered static because the flow of information is unidirectional, from server to user.

Moving on to the building blocks of traditional dynamic web sites. They bring in another aspect called a *form*. The user may submit data through filling out this form and submitting it back to the web server. This gives the user a way to interact with the web server and it gives the software developer means of building web sites that can be considered web *applications*. The communication is bidirectional. A web page request from the user carries user data, the server receives this data and then responds with another complete web page of information.

This way of receiving complete web pages over and over again has a few drawbacks. Some content on each web page received may be static (e.g. the header, footer and menu). The user will receive the same content over and over again. Reloading this information puts unnecessary load on the server and wastes bandwidth. The result is web applications feeling slow and becoming less usable from the user's point of view.

Ambiguity 1. *How can software developers create large dynamic web applications with a high level of bidirectional communication, and still keeping the application responsive from the user's point of view?*

The next section proposes a solution.

1.1.3 Modern Web

The backbone of the web, `HTTP`[2], has practically been the same since the World Wide Web was born. Although, in recent years the web development community has witnessed the rapid development of many advanced web application frameworks such as jQuery [jQuery Project, 2011] and Dojo Toolkit [The Dojo Foundation, 2011]. The framework of choice for the purposes of this essay is Google Web Toolkit [GWT Development Team, 2011a], first released in 2006 [GWT Development Team, 2011b]. A common goal of these frameworks is to further improve the interaction between the user and web server.

Web developers have always been struggling to make the web more dynamic. These frameworks help web developers break the "user request → page reload → user request" cycle, making it possible for the user to both send and receive data without reloading the complete web page. They do this by leveraging advanced features of the `XMLHttpRequest` API [W3C, 2010], supported by many modern web browsers. The use of this technology in web applications is often refferred to using

[2]HyperText Transfer Protocol

its catchier name AJAX[3], which has become a buzzword in the web development community.

Buzzwords does not usually appear alone. Something that often goes hand in hand with AJAX is "agile" development.

1.1.4 Agile Development

The vast availability of the World Wide Web makes it a very good distribution channel. Not only for distributing static content, but also for distributing new and improved versions of dynamic web applications. The length of the web application's development cycle is the crucial factor for making optimal use of this distribution channel.

Static web sites could in theory (and often does in practice) have the shortest possible release cycle. Just edit the live web page. The worst that could happen is introducing broken hyperlinks. There is simply no application logic to write or test.

Making good use of this distribution channel for dynamic web applications is much harder. You are using the same distribution channel as the previous example, but there is a problem. You have actual application logic. This could be written as any generic computer software, thus any generic software development and testing methodology could apply.

An example of a traditional software development cycle is following the ESA[4] Software Engineering Standards PSS-05-0 [Mazza, 1994a,b]. See table 1.1 for an overview of this traditional software development cycle.

The agile development methodologies opposes these traditional methods, defining new principles which are redefining what is considered important and contradicts the phases of table 1.1. One decade ago, seventeen people (each of them with many years of experience in the software industry) met at a ski resort and wrote what they call their "Manifesto for Agile Software Development" [Beck et al., 2001]. Quoting the manifesto states quite clearly what they think about traditional development methodologies:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> 1. Individuals and interactions over processes and tools
> 2. Working software over comprehensive documentation
> 3. Customer collaboration over contract negotiation
> 4. Responding to change over following a plan
>
> That is, while there is value in the items on the right, we value the items on the left more."

[3] Not Ajax the Greek hero of the Trojan War, son of Telamon, king of Salamis. AJAX is shorthand for Asynchrounous JavaScript and XML.

[4] European Space Agency

Phase	Description	Major Activities
UR	Definition of the user requirements	Determination of operational environment and identification of user requirements
SR	Definition of the software requirements	Construction of logical model and identification of software requirements
AD	Definition of the architectural design	Construction of physical model and definition of major components
DD	Detailed design and production of the code	Module design, coding, unit tests, integration tests and system tests
TR	Transfer of the software to operations	Installation and provisional acceptance tests
OM	Operations and maintenance	Final acceptance tests, operations and maintenance of code and documentation

Table 1.1. ESA Software Engineering Standards PSS-05-0 development phases

The manifesto also proposes twelve principles to follow. One of them states that the development team should "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale". Since the World Wide Web makes such a good distribution channel, using agile software development methodologies to create web applications can be considered a very productive and seemingly successful combination. This fact brings up another ambiguity.

Ambiguity 2. *How does a software development team keep the software working*[5]*, while responding to change*[6] *so frequently*[7] *?*

An attempt to clarify ambiguity 2 is to identify what activities are required to produce working software. The next step is to determine how to make these activities as fast as possible – thereby agile. While skimming through table 1.1 one will notice that there are basically three major activities a software development team should participate in: definitions, coding and testing. The first two activities are outside the scope of this essay but the third activity leads us into the next section.

[5]Item 2 of the agile manifesto: "Working software over comprehensive documentation"

[6]Item 4 of the agile manifesto: "Responding to change over following a plan"

[7]Third principle of the agile manifesto: "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale"

1.1.5 Test Automation

Referring back to the first section of this chapter, which states that lots of computer software is untested – digging into the literature on the topic gives us a possible explanation why. Software testing is hard to do and even considered practically *impossible* in some aspects. The chapter titled "The Realities of Software Testing" in "Software Testing" [Patton, 2005] lists a few testing axioms[8], the following list shows a few of them:

- It's impossible to test a program completely
- Testing can't show that bugs don't exist
- The more bugs you find, the more bugs there are
- Not all the bugs you find will be fixed
- When a bug's a bug is difficult to say
- Product specifications are never final

This does not look very promising. Although [Myers et al., 2004] states something that gives hope and help software developers focus on how to tackle this seemingly impossible task:

> "Given constraints on time and cost, the key issue of testing becomes: *What subset of all possible test cases has the highest probability of detecting the most errors?*"

Now when we know what to focus on and taking time into account in an attempt to act agile, another axiom is hereby defined:

Axiom 1. *Automatic testing performed by computer programs is faster than manual testing performed by human beings.*

This basically means that the agile software development team always should strive towards automating their testing processes.

1.2 Problems

1.2.1 Testing User Interface Code

When developing modern web applications you can make test coverage very high just by starting out with a well thought out architecture. A very common architecture is the three-tier architecture. The developers create three independent modules. One

[8] A statement or proposition that is regarded as being established, accepted, or self-evidently true.

Common misconceptions:
Dirties design
It doesn't catch bugs
It's slower
It's boring
Hard to change
Too many interfaces
Testing is for QA
Valid excuses:
Legacy code-base
Don't know how
I write UI ← !

Table 1.2. Common misconceptions and valid excuses for software engineers not writing their test code, according to [Hevery, 2011].

module for the data storage and data access. Another module for the application logic, and a third module for the user interface. The two first modules may be unit tested[9] and created using test-driven development. So the actual application logic is quite easily tested, but what about the user interface?

Miško Hevery works as an agile coach at Google, in his recent presentation [Hevery, 2011] he lists some common misconceptions about testing – and more importantly he lists three valid excuses for not writing test code. See table 1.2 where he states that it is a valid excuse to skip writing test code if it regards user interface code[10] – this is a problem.

Problem 1. *Software testing in general is considered very difficult. User interface testing is said to be even more difficult.*

1.2.2 Testing AJAX

Moving on to another problem. Traditional web applications have been around for a relatively long time compared to modern web applications.

Problem 2. *It seems to be a gap in the literature covering this topic and the functionality of automatic testing tools for user interfaces when it comes to testing AJAX technology.*

[9] If you are working in the Java platform, a popular choice of tools is using JUnit as the unit testing framework and EasyMock for mocking out the interactions with the other tiers. A deeper explanation is outside the scope of this essay.

[10] To be fair, he also focuses strongly on writing clean code and moving all application logic away from the user interface code – moving code from the part that is difficult to test into the part that is easy to test.

1.3 Purpose

Automated testing tools are crucial for any agile software development team. The primary aim of this study is to highlight some problems regarding agile development of web applications. The secondary goal is to evaluate these suggested problems by performing a case study.

There are many aspects of testing software and many different tools and methods to use. The purpose of this particular study is to evaluate a specific tool used for automated user interface testing of modern web applications built using `AJAX` technology, compared to testing a traditional web application.

Chapter 2

Method

2.1 Choice of Method

The following list states some different combinations in which this case study could have been performed:

1. Only analyzing previous published studies, no implementation.

2. Writing a small web application with one `AJAX` user interface and test it using multiple different testing tools.

3. Writing a small web application with multiple `AJAX` user interfaces (using different frameworks for implementing them) and test them using a single testing tool.

4. Writing a small web application with multiple `AJAX` user interfaces (using different frameworks) and test it using a multiple testing tools.

The first item in the above list was rejected because this field of study is still relatively new. Methods 2-4 seemed to expand the scope too much for the time frame of this bachelor's essay.

The method of choice: write a small web application with two separate user interfaces. One implemented using a modern `AJAX` framework and another using a traditional web application framework. Thus creating two frontends for the same backend, and testing those two front-ends using a single testing tool. This will give the study a "scientific control" causing the result to be a comparison between testing the two different types of web application frontends. Small enough scope but still has a good chance of delivering interesting results. Also easily reproduced for studying other `AJAX` frameworks and repeating this research.

2.2 The Application Under Test

A study like this must have a well defined application under test. This is the application which will be implemented so there is something to actually test. To keep the focus on what is important in this study – the actual automatic testing – it is crucial to keep the web application simple. The chosen application is a very simple traditional guestbook application. This guestbook has got three[1] features:

- The user may create a new guestbook entry by entering the name of the author and the guestbook message in a form, then submitting this form by clicking a button.
- The guestbook entries will be shown in a list below the form, accompanied by the dates of the guestbook entries.
- This guestbook entry list will only display a maximum of 5 guestbook entries. If more than 5 guestbook entries are stored in the guestbook, links below the list can be used to navigate between the other pages of maximum 5 guestbook entries (this is called pagination). The most recent guestbook entry is always shown as the first entry in the list, right below the form on the first page.

2.3 Test Suite

A set of test cases is called a test suite. The test suite for this study will contain the following two test cases:

1. Assert that after entering values in the form and clicking the submit button, a new guestbook entry is created and shown.
2. Assert that after creating six guestbook entries, the first guestbook entry created is no longer visible. But after navigating to the second page, it becomes visible.

If the two different implementations of the fontend forces the implementation of the test cases to change, of course there will be four test cases in total. Although if the test tool can handle testing different implementations of the same user interface, that is a really good feature of the test tool.

[1] Actually there's one more feature used to simplify the automatic testing. It's a feature to delete all guestbook entries using one single web request – but this is supposed to be hidden from users when using the guestbook in production mode.

Chapter 3

Study

3.1 Implementing the Application Under Test

3.1.1 Tools and Frameworks Used

Table 3.1 shows a summary of the environment used to perform this study. It's a Java Enterprise Edition web application built using Spring Framework to increase development productivity. When developing in the Java environment, a tool is often used to manage the build cycle and the third party dependencies. Apache Maven is

Description	Software	Version
Operating system	Mac OS X	10.6.6
Web browser	Mozilla Firefox	3.6.16
Automated UI testing tool	Selenium IDE	1.0.10
AJAX framework	Google Web Toolkit	2.2.0
Software development platform	Java (vendor: Apple Inc.)	1.6.0_24
Software project management tool	Apache Maven	3.0.2
Java IDE	Eclipse	3.6.1
JavaEE API version	Servlet	2.5
JavaEE API version	JavaServer Pages	2.1
JavaEE API version	JSP Standard Tag Library	1.2
SQL Database	HSQLDB	1.8.0.10
Object-renational mapping library	Hibernate	3.6.0
Java Bean to Java Bean mapper	Dozer	5.3.2
Java Persistence API library	Hibernate-JPA-2.0-API	1.0.0
Java application framework	Spring Framework	3.0.5
Data access layer enhancements	spring-data-jpa	1.0.0.M2
Servlet Container	maven-jetty-plugin	6.1.26

Table 3.1. This summarizes the environment used to perform this study

a very popular choice for this. A guestbook needs a database to store the guestbook entries. A lightweight in-memory database was used called HSQLDB. To manage the mapping between Java objects and the relational database the Java Persistence API was used, using the Hibernate implementation.

To keep the focus on the web application's user interface instead of heavy back-end code, two important libraries was used. One is Spring MVC, part of the Spring Framework. It helps creating web applications following the Model-View-Controller pattern. The other one is for keeping the actually hand-written database access code to zero[1]. For this the second milestone release of the very impressive and brand new spring-data-jpa library was used.

3.1.2 Data Tier

The data tier in this application shows how very slim the data tier in a Java application can be, and this is good since this study tries to focus on the user interface aspects. It consists entirely of one Java Persistence API entity class (see listing 3.1) and one repository interface (see listing 3.2). Everything else related to the data is solved implicitly and automatically by Hibernate and spring-data-jpa at runtime.

```
@Entity
public class EntryEntity {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String author;

  private String message;

  private Date date;

  /* Public getters and setters omitted */
}
```

Listing 3.1. The guestbook entry Java Persistence API entity

```
public interface EntryRepository
                        extends JpaRepository<EntryEntity, Long> {}
```

Listing 3.2. The guestbook repository

Just for clarification listing B.1 in appendix B shows the methods defined in the `JpaRepository` interface.

[1]The only thing you need to do is create an empty interface (see listing 3.2 for an example) extending another interface which defines all the necessary Create-Update-Delete methods. As a side note, to create domain specific database queries you only need to write the method signature in the interface and the library will implicitly create implementation for you at runtime. Very impressive indeed.

3.1.3 Application Logic Tier

In the application logic tier the data model is represented by a serializable[2] `EntryModel` class (see listing 3.3), and the mapping between the `EntryModel` and `EntryEntity` classes is carried out by a nice little library called Dozer[3]. All of the application logic is contained in an implementation of the `GuestbookService` interface (see listing 3.4 for the interface, or listing C.1 in appendix C for the actual implementation).

```
public class EntryModel implements Serializable {

  private String author;
  private String message;
  private Date date;

  /* Public getters and setters omitted */
}
```

Listing 3.3. The guestbook entry model class

```
@RemoteServiceRelativePath("../spring/gwt-rpc/guestbookService")
public interface GuestbookService extends RemoteService {

  public static final int PAGE_SIZE = 5;

  public ArrayList<EntryModel> getGuestbookPage(int pageNumber);

  public Long getNumberOfPages();

  public Long createNewGuestbookEntry(EntryModel entry);

}
```

Listing 3.4. The GuestbookService interface, this defines the application logic.

The curious reader may have noticed the annotation on the `GuestbookService` interface. It tells the Google Web Toolkit to treat this service as an RPC[4] service used to make asynchronous requests to the web server from the web browser. Communication between the browser showing the traditional web application and web server will go through the `GuestbookController` (see listing 3.5).

```
@Controller
@RequestMapping("/guestbook")
public class GuestbookController {

  @Autowired
  private GuestbookService guestbookService;

  @Autowired
  private EntryRepository repository;
```

[2]The reason for this is explained when describing the GWT communication in section 3.1.4
[3]Dozer copies all the matching Java Bean properties it can find from one instance to another.
[4]Remote Procedure Call

```
  @RequestMapping("/show")
  public ModelAndView show(@RequestParam(defaultValue = "1") int pageNumber)
                                                          throws IOException {
    ModelAndView modelAndView = new ModelAndView("show");
    modelAndView.addObject("formModel", new EntryModel());
    modelAndView.addObject("currentPageNumber", pageNumber);
    modelAndView.addObject("numberOfPages", guestbookService.getNumberOfPages());
    modelAndView.addObject("entries", guestbookService.getGuestbookPage(pageNumber));
    return modelAndView;
  }

  @RequestMapping(value = "/new", method = RequestMethod.POST)
  public String newEntry(@ModelAttribute EntryModel entry) {
    guestbookService.createNewGuestbookEntry(entry);
    return "redirect:show";
  }

  @RequestMapping("/clear")
  public void newEntry(HttpServletResponse response) throws IOException {
    repository.deleteAll();
    response.getWriter().println("Guestbook entries cleared");
  }
}
```

Listing 3.5. The GuestbookController class, this defines the traditional client-server communication

3.1.4 The Dual Presentation Tier

Two screenshots says more than two thousand lines of code examples. See appendix A to see the screen shots of the two separate presentation tiers (figures A.1 and A.2). Notice that they are rendered identically in the browser, except the headline. The only difference is how they communicate with the server and how the data renders in the browser.

Traditional Web Application Framework: JSP

This is simply a very simple JSP[5] implementation of the graphical user interface. The important part to note is that it is using a traditional form post to submit the data to the web server (see listing D.1 in appendix D to read the implementation). This JSP page is served from the `GuestbookController` (listing 3.5) and it communicates back to it when posting the form.

`AJAX` Framework: GWT

Now when it comes to the modern web application frontend, things look a bit different. Listing 3.6 shows the content of `Guestbook.html` from which the GWT (Google Web Toolkit) application is loaded.

```
<html>
  <head>
```

[5] JavaServer Pages

```
    <title>GWT Guestbook</title>
    <script type="text/javascript" language="javascript"
          src="Guestbook/Guestbook.nocache.js"></script>
  </head>
  <body>
    <h1>GWT Guestbook</h1>
    <p>Hello and welcome to my guestbook. Please leave me a message here!</p>
    <hr />
    <div id="form"></div>
    <hr />
    <div id="entries"></div>
    <hr />
    <div id="pagination"></div>
    <hr />
  </body>
</html>
```

Listing 3.6. The `Guestbook.html` file from which the GWT application is loaded

The file looks quite empty, but not when opened in a web browser. Actually the included JavaScript file is a Java program that Google Web Toolkit has compiled into JavaScript. This Java program is then populating the DOM[6] tree with HTML elements. It is also handling the asynchronous communication between browser and server. GWT have automatically generated an asynchronous implementation of the `GuestbookService` (listing 3.4) which is used in the examples of this communication, seen in listing 3.7 and listing 3.8. See how only the serializable java objects are traveling between browser and server.

```
button.addClickHandler(new ClickHandler() {
  public void onClick(ClickEvent event) {
    final EntryModel entry = new EntryModel();
    entry.setAuthor(textBox.getText());
    textBox.setText("");
    entry.setMessage(textArea.getText());
    textArea.setText("");
    entry.setDate(new Date());

    if (entries.size() >= GuestbookService.PAGE_SIZE) {
      entries.removeLast();
    }
    entries.addFirst(entry);

    rerenderEntries();

    guestbookService.createNewGuestbookEntry(entry, new AsyncCallback<Long>() {
      public void onFailure(Throwable caught) {
        rpcFailure(caught);
      }
      public void onSuccess(Long totalNumberOfPages) {
        int newTotalNumberOfPages = Integer.parseInt(totalNumberOfPages.toString());
        updatePageNumbering(currentPageNumber, newTotalNumberOfPages);
      }
    });
  }
});
```

[6]Document Object Model

Listing 3.7. The `ClickHandler` of the submit button of the `AJAX` form. Notice how the text box and text area are cleared after getting the values. This needs to be done since the web browser is not refreshing the whole web page when submitting the form.

What's fascinating about these listings is that this code is actually executed in the user's web browser after being compiled to JavaScript. (Nowadays it is possible to put lots of application logic out in the web browsers, reducing load on the web servers and giving the users a better user experience at the same time).

```
guestbookService.getGuestbookPage(pageNumber,
                                  new AsyncCallback<ArrayList<EntryModel>>() {

  public void onSuccess(ArrayList<EntryModel> result) {
    entries.clear();
    entries.addAll(result);
    rerenderEntries();
  }

  public void onFailure(Throwable caught) {
    rpcFailure(caught);
  }
});
```

Listing 3.8. This shows how the list of five guestbook entries per page is asynchronously fetched from the web server and then rendered directly in the web browser using the helper method `rerenderEntries()`.

3.2 Automated Web Application Testing Tool

This study is about automatic testing of web applications, and the method requires the choice of one testing tool. Here follows a subset of the available tools on the market:

- Sahi <`http://sahi.co.in/w/`>
- Selenium <`http://seleniumhq.org/`>
- SOAtest <`http://www.parasoft.com/`>
- TestComplete <`http://www.automatedqa.com/`>
- Watir <`http://watir.com/`>
- WebUI Test Studio <`http://www.telerik.com/`>

Some research showed that by using Selenium it is possible to make the number of regression bugs released to production essentially drop to zero. That is the experience Chris McMahon got by using Selenium which he explains in detail in his article [McMahon, 2009].

Command	Target	Value
open	/dkand11/spring/guestbook/clear	
assertTextPresent	Guestbook entries cleared	
open	/dkand11/spring/guestbook/show	
assertTextPresent	JSP Guestbook	
type	author	Author using JSP version
type	message	A JSP test message
clickAndWait	submit	
assertTextPresent	Author using JSP version	
assertTextPresent	A JSP test message	

Table 3.2. Selenium test case for the JSP frontend: test writing a single entry.

3.2.1 Recording the Test Cases

Selenium IDE is a Mozilla Firefox add-on, allowing close integration with the web browser while recording and playing the test suite. A very easy way to use the Selenium IDE follows in the next section which describes the work flow of creating the test cases for the traditional JSP web application.

Traditional Web Application Test Cases

1. Open the application under test in Firefox.

2. Start the Selenium IDE add-on from the Tools menu. It will start directly in recording mode, remembering every action you perform on the application under test from now on.

3. Perform the actions you want, for example writing text into form field and clicking the submit button.

4. When the submit is completed and you now see the new content in the web browser, select the text and right click. Selenium IDE have populated this context menu with some Selenium commands, for example "assertTextPresent".

5. Stop the recording and hit the play button in Selenium IDE and watch all of the actions you just performed swoosh by on the screen in blazing speed.

The standard way of representing a Selenium test case is as a three column table[7]. Table 3.2 shows the recording of test case number 1 of section 2.3. Table 3.3 shows the recording of test case number 2.

[7]The default file format is actually a HTML table.

Command	Target	Value
open	/dkand11/spring/guestbook/clear	
assertTextPresent	Guestbook entries cleared	
open	/dkand11/spring/guestbook/show	
type	author	Simon
type	message	First message
clickAndWait	submit	
type	author	Simon
type	message	Second message
clickAndWait	submit	
type	author	Simon
type	message	Third message
clickAndWait	submit	
type	author	Simon
type	message	Fourth message
clickAndWait	submit	
type	author	Simon
type	message	Fifth message
clickAndWait	submit	
type	author	Simon
type	message	Sixth message
clickAndWait	submit	
assertTextNotPresent	First message	
clickAndWait	link=2	
assertTextPresent	First message	

Table 3.3. Selenium test case for the JSP frontend: test pagination.

Modern Web Application Test Cases

This is the most interesting and important part of this study. Is it possible to just perform the same procedure as above for testing the GWT application? The first experiment was to just change the target of the `open` command to `/dkand11/-Guestbook.html` which would open the GWT application instead. Both test cases failed because GWT does not set the ID property on the HTML elements which GWT generates and puts in the DOM.

Not surprisingly this have already been considered in the GWT development team. Developers may take extra measures to ensure ID property are set on elements that are supposed to be part of automatic testing. The base class of all user interface classes is `com.google.gwt.user.client.ui.UIObject` and it defines a method `void ensureDebugId(String id)`, which according to the API "allows it to integrate with third-party libraries and test tools".

So after manually setting the ID properties of each element participating in the test, and reflecting these changes in the test suite (now doubling the number of tests in the test suite) – does the tests succeed when played back? They do not. The `clickAndWait` command fails if the web page has not reloaded within 30 seconds. Since `AJAX` technology allows us to send and receive data without reloading the web page, this causes every `clickAndWait` command to time out.

This too have already been considered, this time by the Selenium development team. The Selenium documentation [Selenium Project, 2011] states that "This is done using `waitFor` commands, as `waitForElementPresent` or `waitForVisible`, which wait dynamically, checking for the desired condition every second and continuing to the next command in the script as soon as the condition is met".

The test cases for the modern web application, which are very similar to the traditional ones, are available in appendix E. Note the user of the `waitForTextPresent` command.

3.2.2 Executing the Test Suite

Previously recorded tests are played back simply by clicking the play button in Selenium IDE. Two screen shots of the Selenium IDE in action can be seen in appendix A. Figure A.3 shows all of the tests succeeding.

Figure A.4 is more interesting. It shows what happens when the server build and run process was misconfigured for a while during this study. The GWT compilation step was by mistake left out of the process, causing the necessary JavaScript file not being generated. The GWT version of the user interface now only contained the HTML of listing 3.6, thus causing both of the GWT test cases to fail. A closer look at which step of the test case failed shows exactly what was wrong, so the failed test case really helped in correcting this error.

This concludes this study. The results are summarized in the next chapter.

Chapter 4

Results

4.1 Testing the Traditional Web Application

- The test cases were simply recorded in one take.
- They did not need any modifications afterwards, they ran successfully directly when the recording was stopped.
- Testing the traditional web application was a very stable and straight forward process.

4.2 Testing the Modern Web Application

- The previously recorded test cases for a seemingly identical web application needed modification to successfully test this web application user interface.
- GWT brings in another layer of abstraction when it populates the DOM tree as it wishes, forcing the developer to keep an extra eye on the DOM tree when telling Selenium where to find certain elements like form fields and buttons.
- It was definitely not a straight forward process creating these test cases.
- The extra complexity that GWT brings into the user interface code makes the test suite feel more necessary and actually feels very welcomed in the development process.

Chapter 5

Discussion

5.1 Sources of Errors

The testability of a web application, or any software for that matter, depends on how it was implemented. It is always a risk that software engineers create programs that are hard or nearly impossible to test. High level of coupling and lots of dependencies on global state is a recipe for a testability nightmare. When evaluating a specific testing tool, there is a risk that the application under test simply was badly written and was effectively unsuitable for execution using any testing tool available.

5.2 Suggestions for Future Work

The automatic tests features that Selenium provides does not make it a complete user interface testing tool. A user interface is so much more than a functional flow through forms and HTTP requests. An application needs to be *usable*. To be competitive in the software industry the user interface must look good, have a good layout and should follow the latest trend in web design.

A list of suggested future studies follows. It is based on a few things that Selenium is missing in functionality.

- Crawling a web site for automatic spell checking.
- Crawling a web site for automatic broken link checking.
- Automatic testing of esthetics. For example analyzing web pages for commonly known patterns we humans tend to like e.g. the golden ratio.
- Automatic readability checking. E.g. it is hard to read large masses of text in wide columns.
- Automatic testing of more esthetics, there are known color combinations that the majority of people dislike in combination. An extreme example of a detectable web design problem would be white text on white background.

5.3 Conclusion

Today there are lots of tools available to help agile software development teams automate their user interface testing. If the application under test is a traditional web application using Selenium is an elegant solution to problem 1 (see section 1.2).

The process of testing modern `AJAX` web applications still have some polishing and streamlining left before using Selenium can be called an elegant solution, in comparison. Problem 2 still stands.

While testing the GWT application struck some problems, work-arounds do exist and once that lesson is learned Selenium is a welcomed tool used together with Google Web Toolkit.

Appendix A

Screen shots

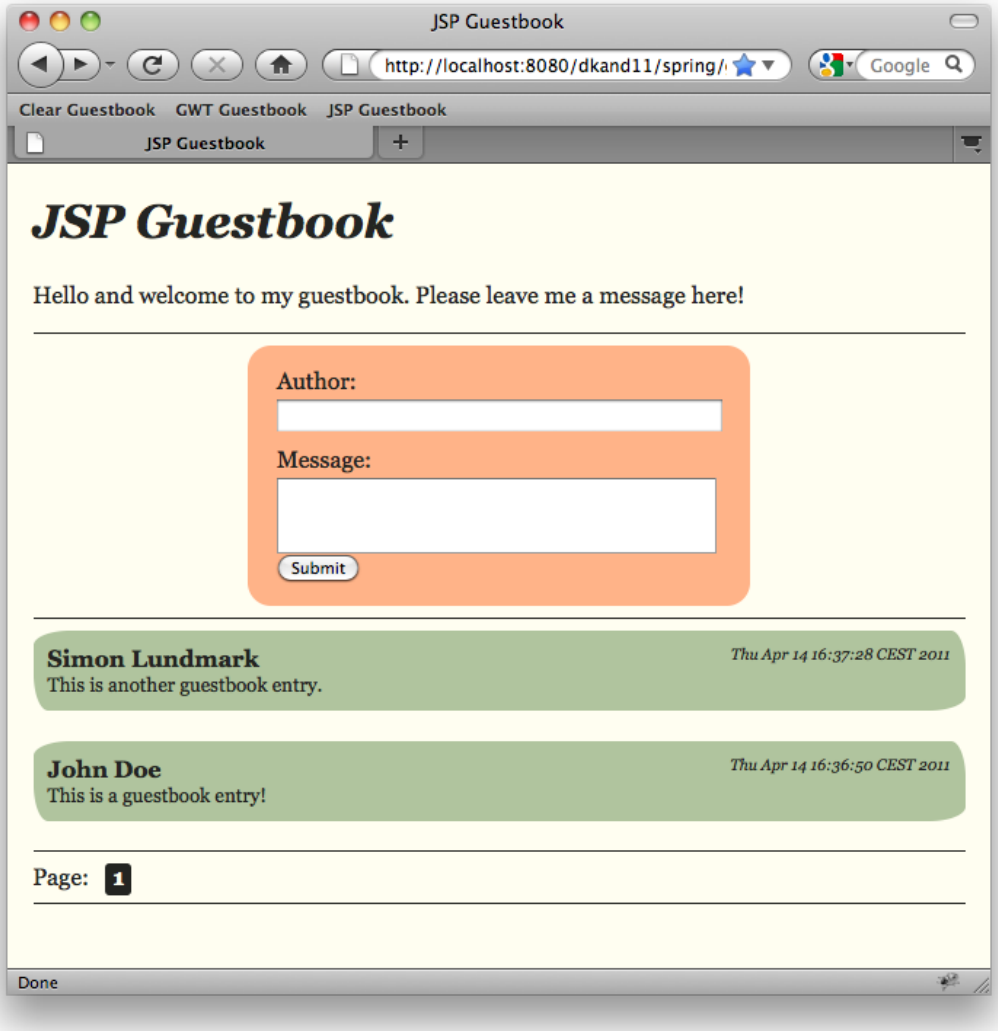

Figure A.1. Screen shot showing the traditional JSP version of the guestbook frontend.

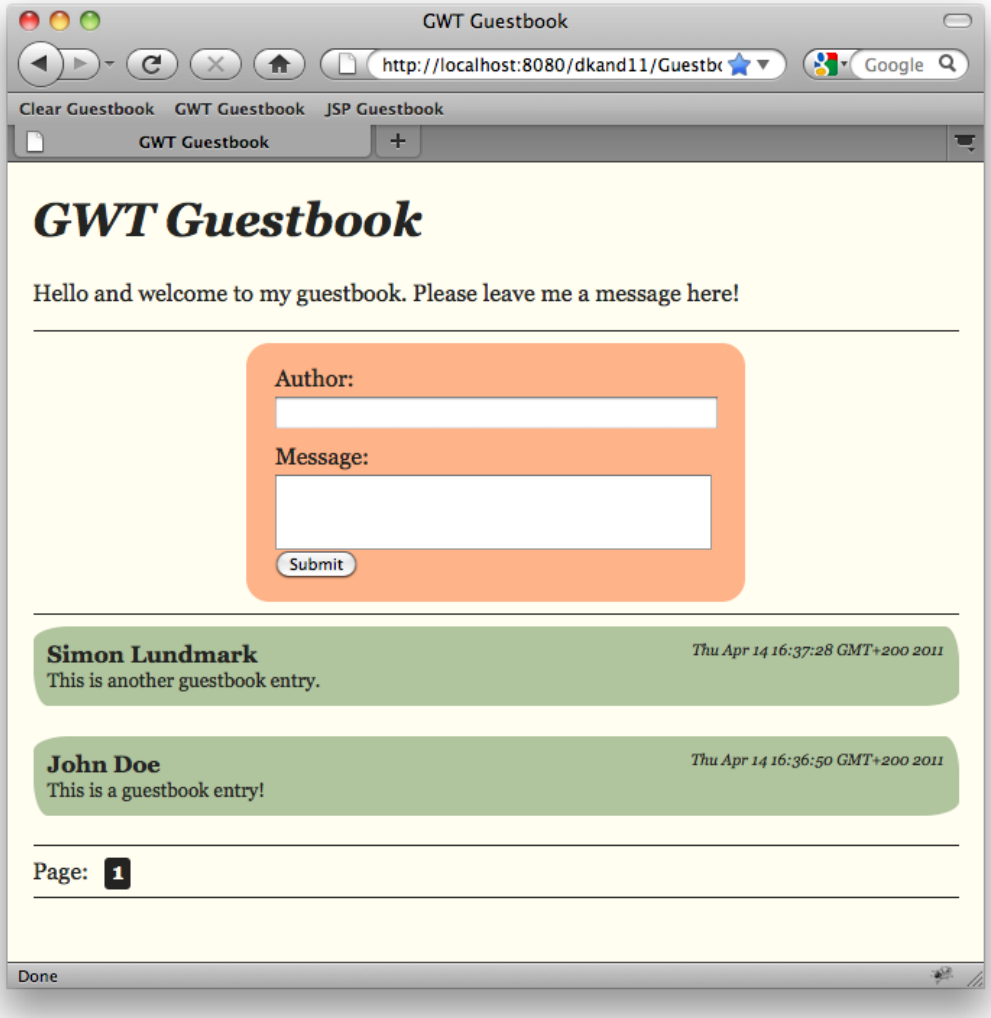

Figure A.2. Screen shot showing the modern GWT version of the guestbook frontend.

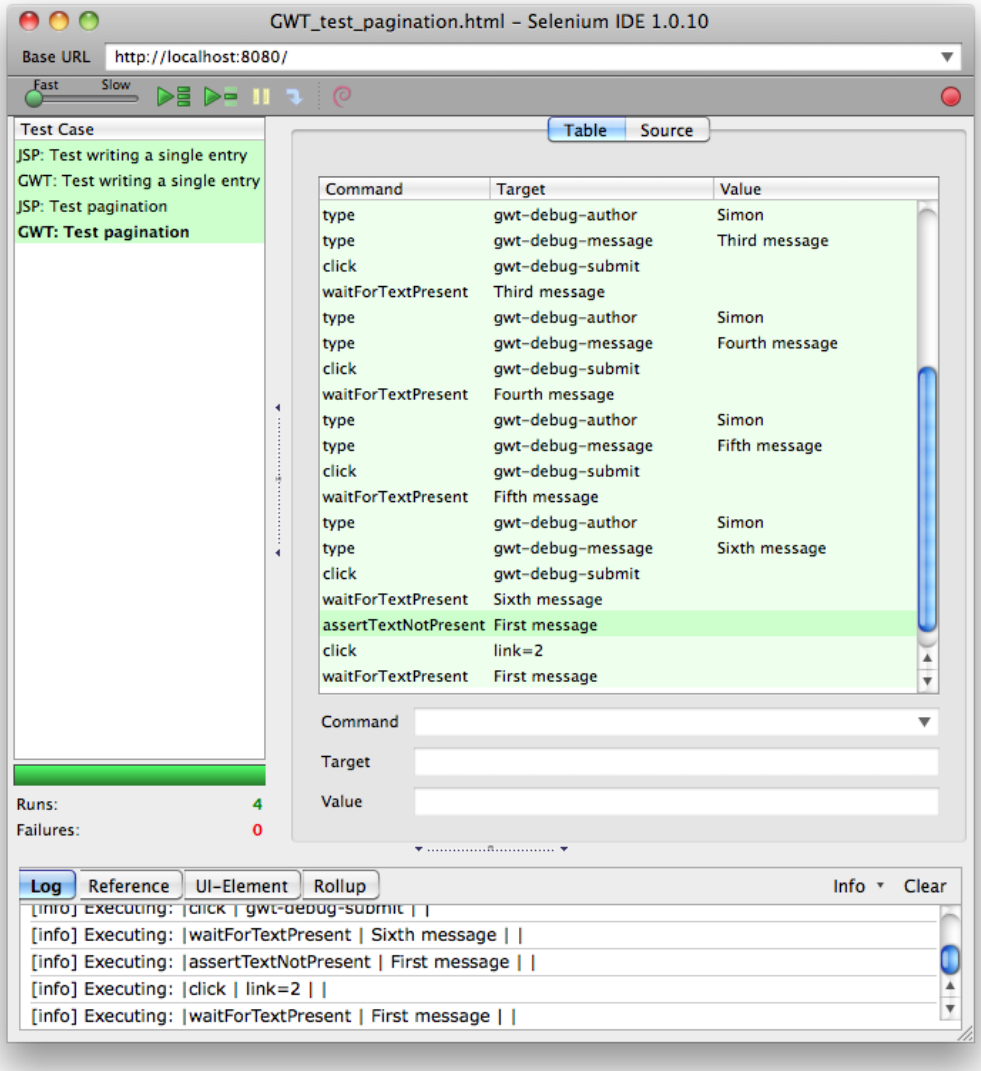

Figure A.3. Screen shot showing a successful execution of the Selenium test suite.

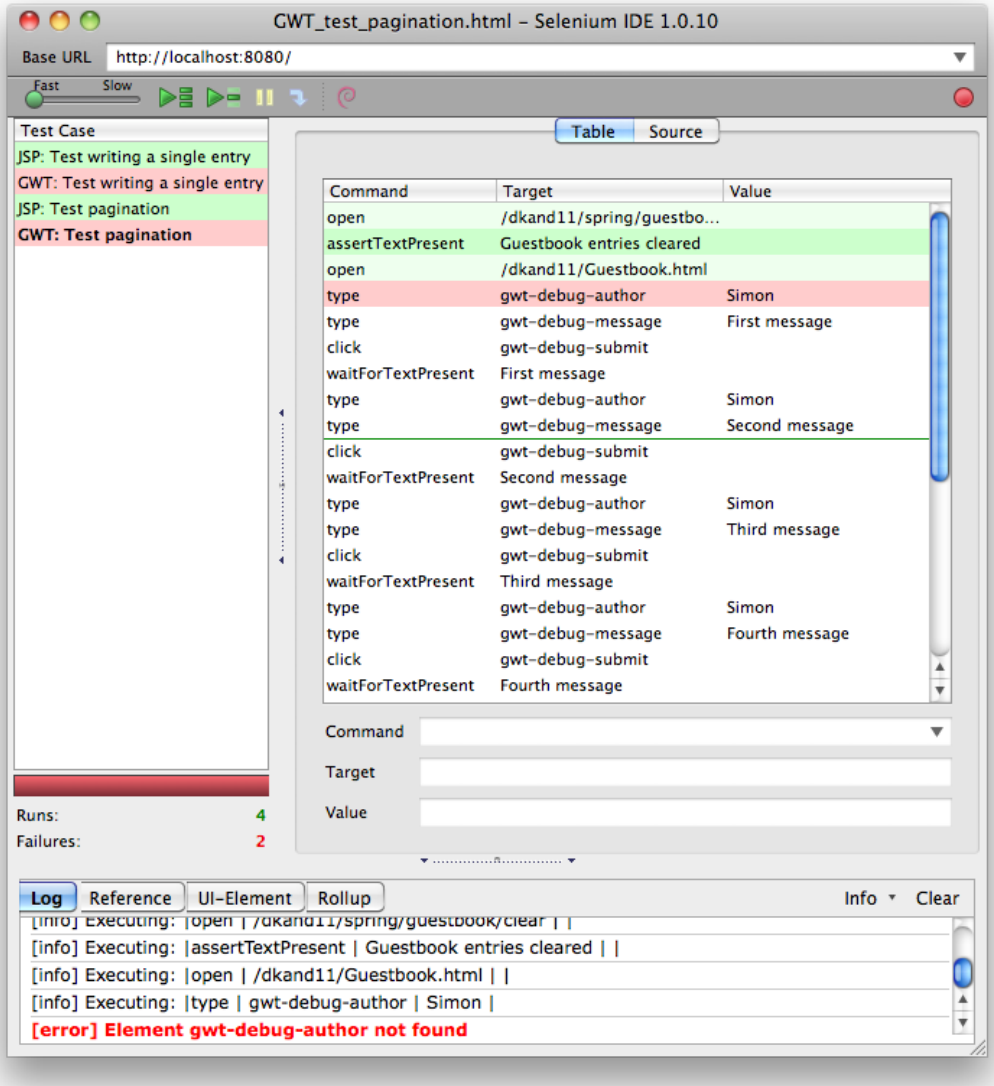

Figure A.4. Screen shot showing a failed execution of the Selenium test suite. This error is related to figure A.5.

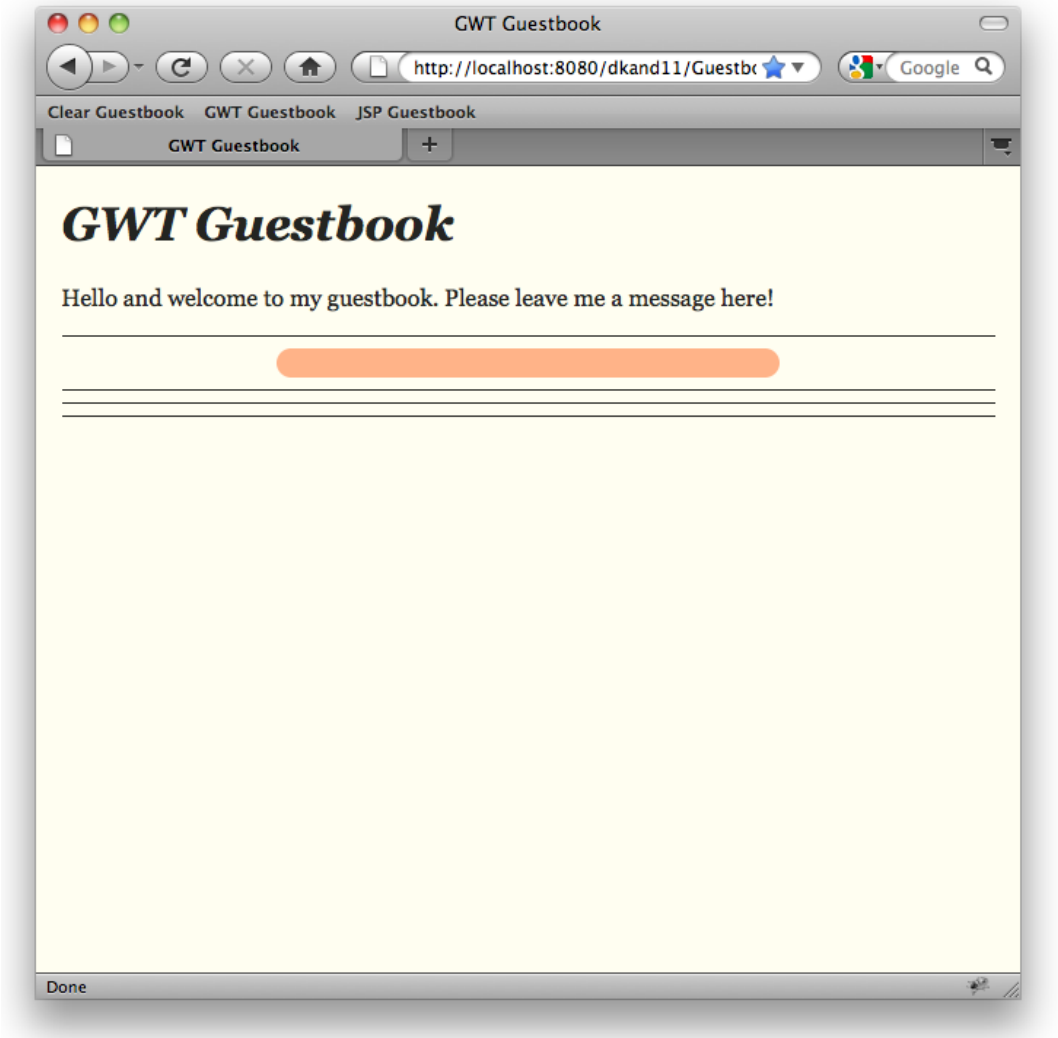

Figure A.5. Screen shot showing the modern GWT version of the guestbook frontend when the included JavaScript file is missing, showing what it looks like when the developer forgot to perform the GWT compilation build step.

Appendix B

The JpaRepository Interface

```
public interface JpaRepository<T, ID extends Serializable> extends
        PagingAndSortingRepository<T, ID> {

    T save(T entity);

    List<T> save(Iterable<? extends T> entities);

    T findOne(ID id);

    boolean exists(ID id);

    List<T> findAll();

    Long count();

    void delete(T entity);

    void delete(Iterable<? extends T> entities);

    void deleteAll();

    List<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);

    void flush();

    T saveAndFlush(T entity);

    void deleteInBatch(Iterable<T> entities);
}
```

Listing B.1. The `org.springframework.data.jpa.repository.JpaRepository` interface. Every method in this interface is also marked `@Transactional`, but that was omitted here.

Appendix C

The GuestbookServiceImpl Class

```
@Service
public class GuestbookServiceImpl implements GuestbookService {

  @Autowired
  private EntryRepository repository;

  @Autowired
  private Mapper mapper;

  @Override
  public ArrayList<EntryModel> getGuestbookPage(int pageNumber) {
    ArrayList<EntryModel> models = new ArrayList<EntryModel>();
    PageRequest pageRequest = new PageRequest(
        pageNumber - 1, PAGE_SIZE, Sort.Direction.DESC, "date");

    for (EntryEntity entity : repository.findAll(pageRequest)) {
      EntryModel model = mapper.map(entity, EntryModel.class);
      models.add(model);
    }
    return models;
  }

  @Override
  public Long getNumberOfPages() {
    long numberOfPages = (long) Math.ceil(
        repository.count() / (double) PAGE_SIZE);
    return new Long(numberOfPages == 0 ? 1 : numberOfPages);
  }

  @Override
  public Long createNewGuestbookEntry(EntryModel entry) {
    EntryEntity entity = mapper.map(entry, EntryEntity.class);

    if (entry.getDate() == null) {
      entity.setDate(new Date());
    }

    repository.save(entity);
    return getNumberOfPages();
  }

}
```

Listing C.1. The guestbook service implementation – the application logic.

Appendix D

show.jsp

```
<html>
<head><title>JSP Guestbook</title></head>
<body>
  <h1>JSP Guestbook</h1>
  <p>Hello and welcome to my guestbook. Please leave me a message here!</p>
  <hr />
  <div id="form">
    <form:form commandName="formModel" action="new">
      <span>Author:</span><form:input cssClass="textfield" path="author" />
      <span>Message:</span><form:textarea path="message" />
      <input type="submit" value="Submit" id="submit" />
    </form:form>
  </div>
  <hr />
  <div id="entries">
  <c:forEach var="entry" items="${entries}">
    <div class="entry">
      <span class="author">${entry.author}</span>
      <span class="date">${entry.date}</span>
      <span class="message">${entry.message}</span>
    </div>
  </c:forEach>
  </div>
  <hr />
  <div id="pagination">
  <div>Page:<%
  int numberOfPages = Integer.parseInt(request.getAttribute("numberOfPages").toString());
  int currentPageNumber = Integer.parseInt(request.getAttribute("currentPageNumber").toString());
  for (int i = 1; i <= numberOfPages; i++) {
    if (i == currentPageNumber) {
      out.print("<span class=\"currentPageNumber\">" + i + "</span>");
    } else {
      out.print("<a class=\"pageNumberLink\" href=\"?pageNumber=" + i + "\">" + i + "</a>");
    }
  }
  %></div>
  </div>
  <hr />
</body>
</html>
```

Listing D.1. JSP implementation using JSTL tags to implement some minor view logic. Note that some uninteresting parts are omitted.

Appendix E

GWT Test Cases

Command	Target	Value
open	/dkand11/spring/guestbook/clear	
assertTextPresent	Guestbook entries cleared	
open	/dkand11/Guestbook.html	
assertTextPresent	GWT Guestbook	
type	gwt-debug-author	Author using GWT version
type	gwt-debug-message	A GWT test message
click	gwt-debug-submit	
waitForTextPresent	Author using GWT version	
assertTextPresent	A GWT test message	

Table E.1. Selenium test case for the GWT frontend: test writing a single entry.

Command	**Target**	**Value**
open	/dkand11/spring/guestbook/clear	
assertTextPresent	Guestbook entries cleared	
open	/dkand11/Guestbook.html	
type	gwt-debug-author	Simon
type	gwt-debug-message	First message
click	gwt-debug-submit	
waitForTextPresent	First message	
type	gwt-debug-author	Simon
type	gwt-debug-message	Second message
click	gwt-debug-submit	
waitForTextPresent	Second message	
type	gwt-debug-author	Simon
type	gwt-debug-message	Third message
click	gwt-debug-submit	
waitForTextPresent	Third message	
type	gwt-debug-author	Simon
type	gwt-debug-message	Fourth message
click	gwt-debug-submit	
waitForTextPresent	Fourth message	
type	gwt-debug-author	Simon
type	gwt-debug-message	Fifth message
click	gwt-debug-submit	
waitForTextPresent	Fifth message	
type	gwt-debug-author	Simon
type	gwt-debug-message	Sixth message
click	gwt-debug-submit	
waitForTextPresent	Sixth message	
assertTextNotPresent	First message	
click	link=2	
waitForTextPresent	First message	

Table E.2. Selenium test case for the GWT frontend: test pagination.

Bibliography

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development. http://agilemanifesto.org/, 2001. Date of access 2011-04-11.

Peter J. Denning. Is computer science science? *Commun. ACM*, 48:27–31, April 2005. ISSN 0001-0782. doi: http://doi.acm.org/10.1145/1053291.1053309. URL http://doi.acm.org/10.1145/1053291.1053309.

GWT Development Team. Google web toolkit official website. http://code.google.com/webtoolkit/, 2011a. Date of access 2011-04-11.

GWT Development Team. Google web toolkit release archive. http://code.google.com/webtoolkit/versions.html, 2011b. Date of access 2011-04-11.

Miško Hevery. Video recording and slides: Psychology of testing at wealthfront engineering. http://misko.hevery.com/2011/02/14/video-recording-slides-psychology-of-testing-at-wealthfront-engineering/, 2011. Date of access 2011-04-11.

The jQuery Project. jquery official website. http://jquery.com/, 2011. Date of access 2011-04-11.

C. Mazza. Esa software engineering standards issue 2. ftp://ftp.estec.esa.nl/pub/wm/wme/bssc/PSS050.pdf, 1994a. Date of access 2011-04-11.

C. Mazza. *Software engineering standards*. Prentice Hall, 1994b. ISBN 9780131065680. URL http://books.google.com/books?id=0NZQAAAAYAAJ.

Chris McMahon. History of a large test automation project using selenium. In *Proceedings of the 2009 Agile Conference*, AGILE '09, pages 363–368, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3768-9. doi: http://dx.doi.org/10.1109/AGILE.2009.9. URL http://dx.doi.org/10.1109/AGILE.2009.9.

G.J. Myers, T. Badgett, T.M. Thomas, and C. Sandler. *The art of software testing*. Business Data Processing: A Wiley Series. John Wiley & Sons, 2004. ISBN 9780471469124. URL http://books.google.com/books?id=OeDMuVI79IoC.

R. Patton. *Software Testing*. Safari Books Online. Sams, 2005. ISBN 9780672327988. URL http://books.google.com/books?id=MTEiAQAAIAAJ.

Selenium Project. Selenium documentation. http://seleniumhq.org/docs/book/Selenium_Documentation.pdf, 2011. Date of access 2011-03-10.

The Dojo Foundation. Dojo toolkit official website. http://dojotoolkit.org/, 2011. Date of access 2011-04-11.

W3C. Xmlhttprequest w3c candidate recommendation 3 august 2010. http://www.w3.org/TR/2010/CR-XMLHttpRequest-20100803/, 2010. Date of access 2011-04-11.