Abstract

This master thesis presents a new open source framework for functional testing of Flex applications, FunFX. FunFX is a framework that encourages test-driven development. The project was initiated by BEKK due to Flex's lack of a proper tool for functional testing.

This report will focus on testing, and will describe central concepts within the area of software testing. Similar testing frameworks for other technologies will also be described and compared to FunFX. It will try to document the usefulness of such an open source tool, and try to discover what entry level this framework imposes on the user.

During development, test cases using the framework will be created to be able to document both positive and negative aspects of the framework. A usage test was also arranged, to be able to document the framework better. This test session resulted in valuable information about the usability and the reliability of the created tests.

The design and implementation are thoroughly described together with each class created, along with their roles in the framework. The issue of synchronization is handled as its own part. To make the implementation decisions easier to understand, the different design patterns used are elaborated.

The final result is a framework that has the ability to interact with a Flex application programmatically. When used together with any test unit framework for Ruby, it is a fully functional testing tool for test-driven development.

Source code, a deployable library file of the FunFX Flex adapter, and a gem of the FunFX framework, together with a Flash movie showing the framework in use can be found on the attached CD.

Keywords: Framework, Flex, Ruby, Open Source, Software testing, Functional testing, Flash, Test-driven development.

Preface

This report is the result of work performed on the master thesis by Peter Nicolai Motzfeldt at the Institute for Computer and Information Science (IDI), Norwegian University of Science and Technology (NTNU) during Spring 2007.

Acknowledgements

I would like to thank Carl Fredrik Sørensen for his valuable guidance during this project, and Trond Arve Wasskog at BEKK for assigning me to this project, and Børre Wessel for his guidance and help with Adobe products and licenses. I would also like to thank Matt Chotin, Matt Horn and Sreenivas Ramaswamy at Adobe for helping me understand how the automation package works.

The participants of the usage test also deserve a big thanks for their views and pointers on the usability of this framework. Thank you, Erlend Oftedal and Christian Schwarz.

Trondheim, June 1, 2007

Peter Nicolai Motzfeldt

Contents

List of Tables

List of Figures

PART I

Context

CHAPTER 1

Introduction

1.1 Motivation

Computers are becoming a more and more important part of life. Our way of living is relying on computers and computer software. Both the economy and health institutions depend on computers. Due to this, faulty software can not only be expensive, but might threaten human lives. To prevent this, testing the different parts of the software is of high importance.

Even though computer software has been around for half a century, testing is still immature in the industry. It is a fact that writing bug free software is regarded as almost impossible. Testing delimits defects and should thus be an important part of every software development cycle. In Norwegian technological colleges and universities, testing has just recently become a part of the course material.

In the beginning, functional testing was the only way of testing software. It was performed with users clicking around in the application. Since then, there have been other types introduced and applied in software life cycles, as unit, integration, system, and acceptance testing. Functional testing is still an important test form, but today functional tests are automated, which reduces time and cost. The automation is important, because if scheduling slips, it is often testing that suffers.

Test-driven development (see Section 3.1.2) is a rather new development model, where the developer writes tests before writing the code, which the test evaluates. It is a way to write small requirements for the software, and to support agile development[1]. When adding new functionality, only tests for these new functions are needed, while the present tests will act as a regression test suite (see Section 4.3.4).

Adobe Flex (see Section 5) is a framework developed by Macromedia in 2004, which enables creation of Rich Internet Applications, (RIA), based on the Macromedia Flash platform [35]. The Flash player was originally built upon an animation metaphor, which

[1] Agile development emphasize working software as the primary measure of progress, rather than written documents[36]

traditional application programmers found challenging to adapt to. This challenge, together with the increased demand for Web applications with requirements of traditional desktop applications, were the driving factors for developing Flex. It provides a work flow and programming model, which is familiar to the developers. At this time, there exists only one tool for functional testing of Flex applications, Mercury Quick Test Pro. This tool is expensive, does not support TDD, and is also difficult to use because of licensing models that limit the number of users.

Due to the lack of a framework for functional testing of Flex applications, software companies do not dare to use it in production. The motivation is to build an open source framework that can be a free alternative to the Mercury Quick Test Pro. The framework will be based on test-driven development, which means that the framework needs to support writing tests before any code is written.

1.2 Problem definition

Currently there is only one tool that is able to test the functionality of Flex applications as described above. This tool is expensive and does not support test-driven development. Because of this, Flex is a tool many software companies do not dare take in use. They do not want to deploy applications that have not been properly tested[2]. If a free alternative is launched, Flex might become a powerful alternative for creating Rich Internet Applications (RIA).

The objective of this thesis is to build an open source framework for functional testing of Flex applications. The framework will be based on the test-driven development model (see Section 3.1.2), which will enable the developer to write tests before incrementally adding code to verify these tests. Possible benefits using the tool will be documented during development. To enhance usability, a test framework needs to support regression testing, hence this will be an important factor when designing the framework.

1.3 Project context

FunFX is a project undertaken as a master project at the Norwegian University of Science and Technology in cooperation with BEKK Consulting AS. The master project will try to contribute to the field of open source, and hopefully be launched as an open source project for other programmers to use and develop further.

1.4 Readers guide

This readers guide will help different readers with different interests to find what chapters might be worth reading. The chapters will be listed in chronological order with a short description of the content.

Readers interested in the problem domain should read Chapter 1.1 together with Part III. Chapter 1.1 describes why this project was initiated, while Part III describes central concepts and technologies.

[2]BEKK Consulting AS

Readers interested in developing tests should read Chapter 5, Section 7.1, Section 11.3, Appendix B, and look at the tests in Appendix D. Chapter 5 and Section 7.1 will give the user an introduction to the technologies used. While Appendix B will enable the user to add the library file and install the FunFX gem file. Section 11.3 will explain how to write the tests.

Readers interested in improving the framework should read Chapter 2, Part IV, and Part V. These parts will give insights to the research questions, how they were answered, how the framework was designed , and what the findings were.

Part I Context

Chapter 1 Introduction This chapter describes the motivation for this thesis, the problem definition, and the project context. The chapter also includes a short summary of each chapter in this thesis.

Part II Research questions and methods

Chapter 2 Research questions This chapter enumerates the research questions this paper will try to answer. It also describes the evaluation methods used.

Chapter 3 Development methods and tools This chapter will describe the development methods used during this project. It also describes the tools used.

Part III Prestudy

Chapter 4 Central concepts This chapter describes important concepts in the field of software testing. It also describes some important concepts such as Rich Internet Application and what a framework is. This chapter will improve the understanding of the upcoming chapters.

Chapter 5 Flex This chapter describes in detail the Flex technology. Flex is the technology this project wants to functionally test. It will describe the different parts of the technology as well as some of the competitive technologies.

Chapter 6 State of the art This chapter gives an account of the functional testing tools available for Web applications.

Chapter 7 Development technology This chapter describes the technologies used during the development of FunFX.

Part IV The FunFX Framework

Chapter 8 Introduction to FunFX This chapter introduces the implementation part, and describes some thoughts behind the framework.

Chapter 9 Requirement specifications for FunFX This chapter describes both the functional and non-functional requirements. A scenario elaborates on how the framework might be used.

Chapter 10 Design of FunFX This chapter describes the design of the application, and the design patterns used.

Chapter 11 Implementation of FunFX This chapter thoroughly describes how the framework is implemented. It provides some code samples to illustrate important aspects of the framework.

Part V Evaluation of FunFX

Chapter 12 Testing of FunFX This chapter shows how the framework has been tested, both the functional requirements and the non-functional requirements.

Chapter 13 Implementation issues This chapter describes in more detail some problems encountered, and how they were solved.

Chapter 14 Evaluation This chapter will answer the research questions stated in chapter 2.

Chapter 15 Summary This chapter will summarize the results and the further work.

Part VI Appendix

Appendix A Questionnaire The questionnaire given to the test group.

Appendix B User guide This chapter will aid a potential user, with installing and using the framework.

Appendix C Contents of CD-Rom This appendix lists the contents of the CD-Rom.

Appendix D FunFX tests This appendix lists sample tests written with the FunFX framework.

PART II

Research questions and methods

CHAPTER 2

Research questions and methods

This chapter will describe the research questions this thesis tries to answer. If the questions have some kind of requirements on how to be answered, they will be elicited beneath each question. It will also describe the methods that will be used to answer the questions.

2.1 Research questions

This project will develop a framework for functional testing of Flex applications. During the development process, there will be developped small test applications. This will discover weaknesses and strengths in the framework.

The research questions this thesis wants to answer are:

RQ1: What are the possible benefits of using the framework in development?

Evaluation approach: Manual functional testing and automated testing with help of the framework will be compared.

RQ2: What kind of entry level does the framework impose on the user?

Evaluation approach: The framework will be tested on different kinds of potential users to ensure the right level of usability. The test personnel will be asked to fill out a simple questionnaire (see Section 2.1.1), to provide some data for this evaluation.

RQ3: How well does the framework support regression testing?

Evaluation approach: No evaluation approach.

RQ4: How does the framework inform the user where errors happen? And how well does this information help the user to correct the error?

Evaluation approach: The framework needs to inform the user of where the error happened. This ability will be tested with a user test. This will be elaborated more in section 2.1.1.

RQ5: What are the positive effects with open source software? Does the fact that FunFX is supposed to be an open source framework, put any constraints on the implementation?

Evaluation approach: Constraints on the implementation due to open source, will be documented during development.

RQ6: How well did the research methods (see Section 2.2) and the development methods (see Section 3.1) help during this project?

Evaluation approach: The methods limitations and positive sides will be documented and evaluated during the entire project.

2.1.1 Usage test

As a usability test, the framework will be deployed to a group of potential users to see how well it performs, and to see what kind of entry level the framework requires. It will focus not only on how to write tests, but also if there are something missing in the framework.

The group of users will be using the framework as they would if the framework was used in real life, writing the test before implementing code until the test passes.

After the session, the users will be asked to answer a simple questionnaire (see Appendix A). The questionnaire will seek to answer among others the usability, benefits, and possible improvements of the framework.

2.2 Research method

The main objectives of this thesis, as mentioned in Section 1.2, is to develop an open source framework to enable functional testing of Flex applications, and to see how the framework will be in use. The following will describe the research method used in this thesis.

Research in software engineering can be based on several different methods. Basili describes three common research methods that are relevant for software development[10]. The three methods described are:

Engineering method (scientific) In the engineering method the developers observe existing solutions, propose a new and better solution, improve the solution, and repeat until no further improvement is needed.

Empirical method (scientific) The empirical method proposes a model and develops statistical methods to validate a given hypothesis.

Mathematical method (analytical) The mathematical method proposes a formal theory and results derived from the theory, are compared with empirical observations.

This thesis will create a framework for functional testing of Flex applications. There already exist solutions for functional testing of various applications. Because of this, the

engineering method will mainly be used in this thesis. The existing solutions will be studied and they will be a foundation to how the framework will be created.

2.2.1 Models for validating technology

In [44], Zelkowitz and Wallace describe a taxonomy that divides 12 different experimental approaches into three broad categories:

Observational methods: An observational method collects relevant data as a project develops, using the new technology being suited.

Historical methods: A historical method collects existing data from completed projects.

Controlled methods: A controlled method provides for statistical validity of the result, if sufficient instances of an observation are available.

One historical method, *literature search*, and one observational method, *case study* will be used during this project.

Literature search

A literature search analyses existing papers and documents to either confirm an existing hypothesis or to enhance data collected in one project with similar data.

Functional testing is a well-known technique, and there have been written many papers about it. There have also been written frameworks for functional testing of several other types of technologies. Both the papers and the already developed frameworks will be studied and make up the literature search. This search will be initiated early and form the basis for the case study performed later in the project.

Case study

In a case study, the researchers monitor a project and collect data over time. The data collection in a case study is derived from a specific goal for the project. Humans may have influence on the development, and due to this, the case study is regarded as an active method.

The main objective in this thesis is to create a framework for functional testing of Flex applications. A case study will be performed during development to collect data to be able to answer the research questions stated in Section 2.1.

CHAPTER 3

Development methods and tools

This chapter will describe in more detail what development method and tools used during the master project.

3.1 Development methods

There exists many different development methods to be used when creating an application. What kind of method that is most suited for a project, depends on the application and the domain knowledge.

3.1.1 Unified Process - UP

A well know method is UP, the Unified Process. UP is a more agile developing method than the well known *Waterfall* method, but still relies on some documents. This is an iterative development method, which involves early programming and testing of a partial system in repeating cycles [22]. The development starts before all the specifications are elicited. Feedback is used to clarify and improve the evolving specifications. This will often be a time-saving feature in projects that work in domains where requirements might change during implementation.

Iterative development is organized into a series of short, fixed length mini-projects which are called *iterations*. Each iteration includes its own requirements analysis, design, implementation and testing as visualized in Figure 3.1. The outcome is a tested, integrated and executable partial system. The system grows incrementally after each iteration.

3.1.2 Test-driven development - TDD

A problem with software development is that it is too success oriented. Most software is considered working until defects are found. It is not enough to have a good design before

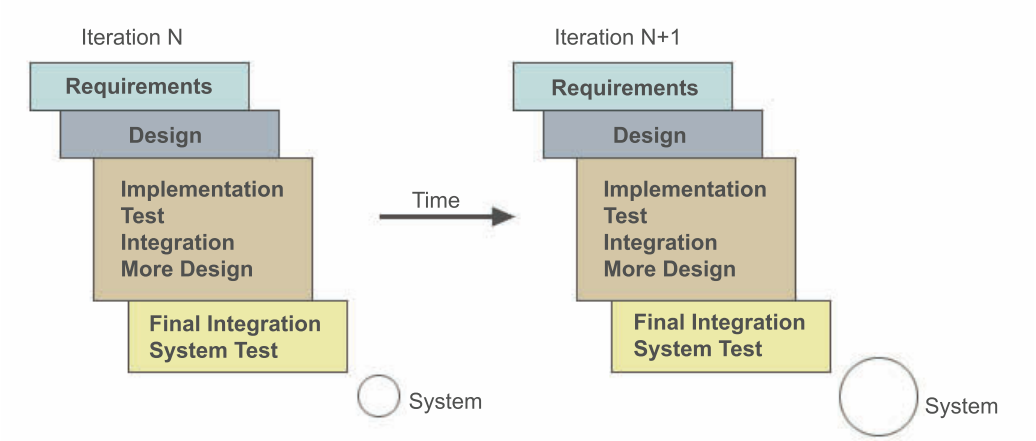

Figure 3.1: Iterative development in UP

implementing a software project because even though you have gone through requirements analysis and created data models and all kinds of theoretical constructs, the time you start coding, a lot of that gets thrown out the window. Thousands of decisions need to be made and you cannot make all these decisions correctly before you start writing code [15].

Test-driven development (TDD) is a method, which is about using tests to create software in a simple, incremental way. TDD relies on two main concepts: *unit testing* and *refactoring*.

This means that a unit-test (see Section 4.3.2) for a method is written before implementing the method. Code is created in small steps, to ensure that the test succeeds. The first step is to create a small test which is certain to fail, then by incremental development, code is created to make the test pass. These steps go in iterations with refactoring until the method is acceptable and the test succeeds [21].

The benefits of using TDD are numerous: incremental development, simpler development process, constant regression testing, and improved understanding of the required software behavior. All code is being developed in small incremental steps, which enables the developer to have working software almost immediately. The focus of the developers is only to get the next test to pass, which makes the developer more productive. Because all written code have corresponding tests, the tests act as regression suites for testing.

3.1.3 Summary

There do not exist other open source alternatives and the requirements are thus likely to change. Because of this, a scaled down version of the UP method will be used in this project, where each iterative cycle is one week. This is the best method by choice because of unfamiliarity with developing frameworks.

Below is an explanation of the steps in each of the iterations in the scaled down version of UP:

Requirements In each iteration, the (10%) most important functional requirements will be fully specified in use cases.

Design After the requirements are specified, the requirements will be analysed and the architecture is specified.

Implementation and Testing During development, a method called test-driven development (see Section 3.1.2) will be used. In short, this means that the test method will be implemented before writing the real method. This ensures that the code is always tested.

Integration At the end of the iteration, the iteration parts will be integrated with the existing system.

Evaluation When the modules are integrated, the entire system is tested and evaluated to see if the system needs any more iterations.

TDD will be used both as a tool during the development of the framework, and as one of the drivers of the framework itself. The framework must be able to be used as a tool in test-driven development.

3.2 Development tools

This section will describe in more detail the different tools that were used to complete this thesis.

3.2.1 Eclipse

The most scalable, modifiable, and free developing environment at the moment is Eclipse. It is an open source platform that is highly modifiable with use of different plug-ins.

Plug-ins

With use of plug-ins, a developer can modify Eclipse to behave a certain way. Plug-ins will give Eclipse support for different kinds of programming languages and development frameworks. The following list explains the plug-ins used during this project:

Ruby Development Tools - RDT Ruby Development tools is a complete Ruby IDE for the Eclipse platform. It features syntax highlighting, on-the-fly syntax check, graphical outline, support for unit testing, debugger, and Ruby application launching [32].

Adobe Flex Builder 2 Flex Builder 2 is a development environment for Flex (see Chapter 5) within Eclipse. It provides the developer with a way to quickly create rich client-side logic that integrates with XML or Web services. To be able to easily develop user interfaces with customized look and fell, the builder comes with design and layout tools [3].

PART III

Prestudy

CHAPTER 4

Central concepts

This chapter will describe central concepts in the problem domain.

4.1 Rich Internet Application - RIA

Rich Internet Applications (RIA) are Web applications that have the features and functionality of traditional desktop applications. RIA usually puts the processing work needed for the user interface to the Web client, but keeps business logic and the program state at a server. RIA was introduced by Macromedia in *Macromedia Flash MX-A next-generation rich client*[7] in 2002.

Traditional Web applications are normally client-server architectures with thin clients. With such systems, all the processing are done at the server, and the client is only displaying static content. Every action on the client must go through the server, thus requiring some response time. By using some client-side processing, RIA circumvents this slower loop.

Methods and techniques for RIA[40]:

JavaScript was the first major client side language, and is still widely used. It is a scripting language and is based on prototype-based programming. JavaScript is an important part of AJAX.

AJAX , short for Asynchronous JavaScript and XML, is a technique that combines JavaScript and XML to build interactive Web applications.

Flash is another way to build RIAs. It is based on an animation metaphor that creates a movie to be played on a Flash player.

Flex (see Chapter 5) is a framework for developing RIAs based on the Flash player. It enables the developer to build Flash user interfaces by compiling MXML, a XML based interface description language.

Java Applet is an applet delivered in java byte code, and can be run in any browser using a Java Virtual Machine.

4.2 Framework

In software development, a framework is a defined support structure in which a software project can be organized and developed. It is an extendable set of objects for related functions. A software framework is similar to class libraries. The main advantage of development frameworks is the potential high level of reuse.

The signature quality of a framework is that it provides an implementation for the core and unvarying functions, and includes a mechanism to allow a developer to plug-in varying functions, or to extend functions [22].

Some well known frameworks are e.g., *Ruby on Rails*, *Hibernate*, *.NET Framework*, and *Spring Framework*.

4.3 Software testing

It is regarded as nearly impossible to write defect-free software, but it is still important to strive to limit the defects to a trifling number. To be able to do this, testing is an essential activity. Software testing is a process or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended [27]. A more precise definition is:

> **Testing is the process of executing a program with the intent of finding errors and to ensure that a program does perform as intended[27].**

Testing needs to be performed at different levels in software, and there exist different ways of testing software. The following sections will describe different methods used in this thesis and their intentions and benefits.

4.3.1 Test case

A test case is a set of conditions or variables under which a tester will determine if a requirement in an application is partially or fully satisfied [42]. To be sure that the requirement is satisfied, the test must try to detect errors.

There must be at least one test case for each requirement. Multiple test cases are often collected into test suites.

Since it is exhaustive and impossible to test for any possible value [27], it is a clue to find the subset of all possible test cases that has the highest probability of detecting most errors.

4.3.2 Unit testing

Unit testing is the procedure where individual modules or units of code are validated if working properly. A unit is the smallest testable part of an application.

A unit test is a test for a specific unit, and each test case is independent of other tests. Mock objects[1] can be used to be able to test a module in isolation.

Unit testing is the backbone of test-driven development as described in Section 3.1.2.

Listing 4.1 displays a simple unit test class written in JUnit, which is Java's version of unit testing. There is a simple setup method where the developer can enter information that will be used by multiple test cases. The test case `testCreate` is testing the creation of a `Student` object (instance of the class `Student`).

Listing 4.1: Unit test

```
1  public class StudentTest extends TestCase
2  {
3    private String studentName;
4
5    public void setUp()
6    {
7      studentName = "Jane Doe";
8    }
9
10   public void testCreate()
11   {
12     Student student = new Student(studentName);
13     assertEquals(studentName, student.getName());
14   }
15 }
```

Unit testing can act as regression testing, which will be described in Section 4.3.4.

4.3.3 Functional testing

Functional testing, also called acceptance testing or black box testing, is designed to verify that an application works according to functional, non-functional, and other stakeholder requirements. It differs from other testing methods like unit testing, as it tests the system as a whole, not just a class or method.

Web applications

There exist two different methods to functionally test Web applications [14]:

- Tools that simulate browsers by implementing the HTTP request/response protocol and parsing the resulting HTML. HttpUnit, WWW::mechanize and WebUnit are examples of such tools.
- Tools that use COM calls to drive the Internet Explorer browser. Watir (see Section 6.1), Samie, and JSSh are example tools that drive the browser.

To avoid tests that break, it is important to consider how to perform actions on the application. Many functional testing applications use the x and y position on a screen to perform actions. This approach is weak if the positions of the graphical objects are

[1] Artificial objects that mimic the behavior of real objects in controlled ways

changing. To be able to reach the actual objects, instead of using the positions on the page, makes it much easier and safer to perform the actual testing.

Many new Web technologies use an XML based language to build the graphical user interface (GUI). This puts the display objects in a hierarchical way. For tests to be enduring, the framework should be able to reach objects in both an absolute path and a relative path. If the GUI is changed with additional containers, a test must not break as long as the functionality of the application is the same.

Acceptance testing

Acceptance tests are run by the customer or the end user to ensure that the system works as anticipated. It generally involves running a suite of tests on the completed system. Acceptance tests are done before deployment.

4.3.4 Regression testing

Regression testing is not a separate testing method, but more a result of other testing methods. It means that the tests always check whether code is still working after changes in the software. If the development process follows TDD (see Section 3.1.2), the unit tests work as a regression test suite. High test coverage[2] is a requirement for regression testing.

4.3.5 Keyword-driven testing

Keyword-driven testing is a relatively new way of writing tests. It has also been called *action words*, *test frameworks*, or *third-generation test automation*.

A keyword-driven test tool consists of multiple keywords, which correspond to methods in the tool. They perform the corresponding method with the supplied attributes. The result of the action is compared to an expected value. The test passes if the values are equal.

Table 4.1 shows an example of a keyword-driven test written in Fitness (see Section 6.4).

Table 4.1: Keyword-Driven test written in Fitness

Action	**Fixture**	
start	fitness.fixtures.CountFixtures	
check	counter	0
press	count	
check	counter	1
press	count	
check	counter	2

[2] All code have a corresponding test

4.4 Component Object Model - COM

Component Object Model (COM) is a Microsoft platform for software components introduced by Microsoft in 1993. It is used to enable inter-process communication and dynamic object creation in any programming language that supports the technology. The term COM is often used in the software development world as an umbrella term that encompasses the OLE, OLE Automation, ActiveX, COM+, and DCOM technologies [37].

The essence of COM is a language-neutral way of implementing objects so that they can be used in environments different from the one they were created in. It provides high level of reusability because the developer is forced to provide well-defined interfaces that are separate from the implementation.

CHAPTER 5

Flex

Flex is a framework developed by Adobe Systems Incorporated for developing RIA for the Flash platform. It was a result of the increased demand for more desktop-like applications for the Web.

Traditional application programmers found it challenging to adapt to the animation metaphor upon which the Flash Platform was originally built. Flex seeks to minimize this problem by providing a work flow and programming model that is familiar to application developers [35].

Flex allows Web application developers to quickly and easily build RIAs. In a multi-tier model, Flex applications serve as the presentation tier. It enables enterprises to create personalized, multimedia-rich applications that dramatically enhance user experience [3].

Flex is currently in version 2.0, but the version 3.0 beta is available for download from the Adobe site. Adobe has shipped Flex with a majority of its source code for a while, but on the 25th of April 2007, Adobe announced that they will release Flex under a Mozilla license. This will make Flex more popular within developing communities.

Flex lacks a proper open source testing tool for developers concerned with functional testing that support TDD. This is therefore a motivation for this master project.

5.1 MXML

MXML is an XML language used to create layouts of user-interface components for Flex applications. It is also used to declaratively define non visual aspects of an application, such as access to server-side data sources and data bindings between user-interface components and data sources [19]. All components are nested in a hierarchy of parents and children as a regular XML file. This hierarchy is something that need to be taken into account when developing a framework that must interact with these components. Figure 5.1 shows what the hierarchy of a simple application might look like.

Listing 5.1 shows a simple Flex application written in MXML.

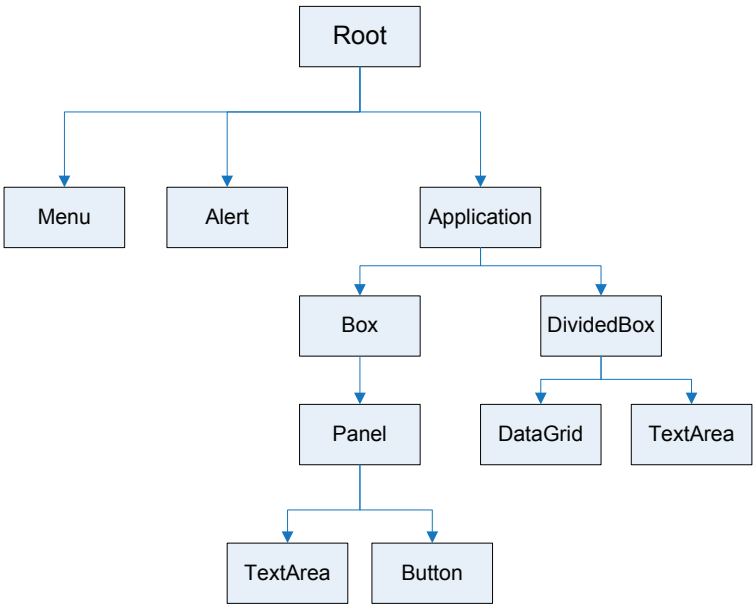

Figure 5.1: Example hierarchy of an Flex application

Listing 5.1: Sample MXML

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
3      <mx:Array id="sampleArray">
4          <mx:String>Sample Label 1</mx:String>
5          <mx:String>Sample Label 2</mx:String>
6      </mx:Array>
7      <mx:Panel title="Example Panel">
8          <mx:ComboBox dataProvider="{sampleArray}"></mx:ComboBox>
9      </mx:Panel>
10 </mx:Application>
```

Figure 5.2 shows how the application shown in Listing 5.1 looks like when built.

5.2 ActionScript

ActionScript is a scripting language based on ECMAScript, primarily used to develop software for the Adobe Flash Player. Together with MXML capabilities with user interfaces and binding to data sources, ActionScript can be used to write business logic.

ActionScript is a prototype-based language, which means that class definitions can be changed at runtime. This makes it easy to modify and add functionality.

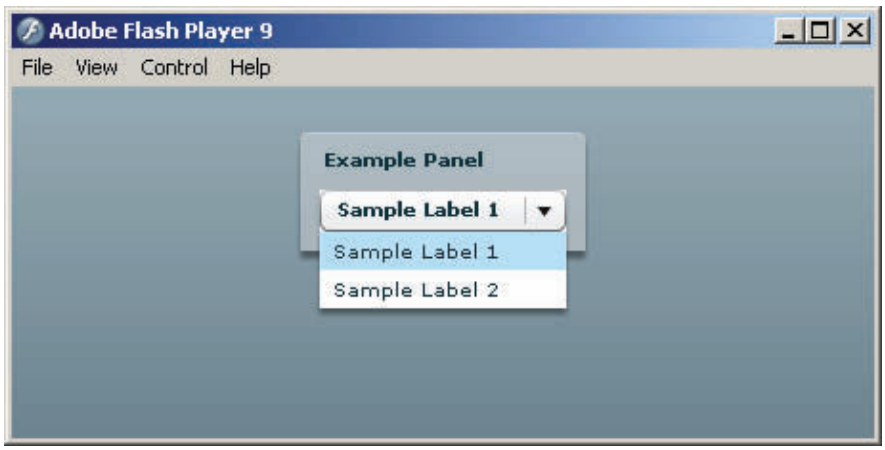

Figure 5.2: Sample Flex application

5.2.1 Mixin

ActionScript does not support multiple inheritance. However, it provides something called *mixin*, which provides an easy way to dynamically add methods of an existing class to a custom tailored class without using inheritance [4].

A mixin is an atomic unit in an object oriented language that adds functionality to another class. The word mixin is attributed to a 1970's MIT hangout, where the owner made his own ice cream. He had basic flavors like vanilla and chocolate, and added extra ingredients like nuts, cookies, or candies. These extra ingredients where called mixins. In this way, the nuts are similar to mixins in ActionScript, you do not order just a bowl of nuts in an ice cream-bar [4].

A mixin is a powerful tool, but it is necessary to be aware that the use of mixins generally violates the principles of good object-oriented practices.

5.2.2 ExternalInterface API

ExternalInterface is an API, which enables developers to access ActionScript methods within a Flash-movie. A developer can either use JavaScript within a browser or use an ActiveX control to call the defined ActionScript methods.

The ActionScript methods need to be defined with an `addCallback` method in the respective ActionScript file in the Flex application as shown in Listing 5.2. This method makes it possible for JavaScript and an ActiveX control to access this method on the flash object embedded in a HTML file.

Listing 5.2: Define a callable method with ExternalInterface

```
1 import flash.external.ExternalInterface;
2 ExternalInterface.addCallback("callname", methodname);
```

The ExternalInterface provides a two-way communication, and it has the ability to call methods provided by JavaScript or an ActiveX control. Listing 5.3 shows how to call a method called `method`.

Listing 5.3: Call a method with ExternalInterface

```
1 import flash.external.ExternalInterface;
2 ExternalInterface.call("method");
```

With the ExternalInterface, various data types (Boolean, number, String) can be passed between ActionScript code and JavaScript or a ActiveX Control [17].

5.3 Flex Builder 2

Flex Builder 2 is an Eclipse-based IDE for developing RIA using the Flex framework. There exist two versions, one standalone version and one plug-in for Eclipse. With this tool, it is easy to design user interfaces and debug applications.

Flex Builder 2 is not freeware, but priced per license.

5.4 Flex Data Services 2

Flex Data Services 2 (FDS) is a server-side complement to the Flex 2 SDK and it is deployed as a Java EE application. It enhances the client-side Flex framework by providing high-performance connectivity with existing server-side data and business logic. Based on a robust messaging architecture, it integrates with existing middleware, add support for real-time data push and publish/subscribe messaging, and enables collaborative and occasionally disconnected applications [2].

The simpler way to work with data in Flex is with the use of *data providers*. A data provider is a collection of objects that contains data required by a component. Because the data provider is separate from the component, it can be used by multiple components and is in this sense a "model" that can be used by many views[1]. Data providers often require it's information from Web services.

FDS2 for a single CPU is free of charge, but does not include the Flex Automation Package (see Section 5.4.1). The full version is priced per CPU and includes the Flex Automation Package.

5.4.1 Flex Automation Package

To drive a Flex application programmatically, Adobe has developed an automation package. This package is only available when buying Flex Data Services 2. They have no relationship other than that it was practical for Adobe to do it this way. Adobe lets developers develop testable applications without the FDS and the automation package, but to actually drive the applications through the automation API, requires a license. The package contains two library files: *automation.swc* and *automation_agent.swc*. These files makes it possible to reach a class called `AutomationManager` that implements the `IAutomationManager` interface, which is the central point in the automation environment, a façade class[1].

The package has methods for creating an `AutomationID` of a object, and methods for easily resolve the object back from the `AutomationID`. This makes it easy to get back

[1] Façade is a design pattern (see Section 10.1.3)

to an object. It can also extract information from the display components, and report if the current event is finished or not. These abilities will play an important role during development of FunFX.

Automation Package is built for the purpose to support recording of events. This is noticeable with the `AutomationID`. It is easy to create an id of an object found and to get back to this object with the id. It is more difficult to find this object by other means than clicking on it and recording the event. To support TDD, another way is needed to locate the object that will be automated.

5.5 Flex Charting 2

Flex Charts is an additional package to the Flex 2 SDK, and provides a rich library of interactive charts and graphs that enable rich data dashboards and interactive data analysis.

5.6 Competitors to Flex

This section describes some of the competitors to Flex, and will try to document their strengths and weaknesses compared to Flex.

5.6.1 Silverlight

Silverlight formerly known as Windows Presentation Foundation Everywhere, or short WPF/E, is the Microsoft solution for delivering rich, cross-platform, interactive experiences including animation, graphics, audio, and video for the Web [26]. This Microsoft technology is a direct competitor to Adobe Flex, but has not yet been released as a final version. Silverlight is based on the Windows Presentation Foundation technology, which is the graphical subsystem feature of the .NET Framework 3.0. It uses XAML[2] and JavaScript.

Silverlight is preinstalled with Vista, and using Vista together with Internet Explorer does not require any plug-ins. But using Silverlight in any other browsers on any other OS, will require a lightweight browser plug-in freely available from Microsoft.

Since XAML is scriptable with JavaScript, Silverlight is supposed to work together with AJAX.

Silverlight was unveiled by Microsoft in Las Vegas April 15th, 2007, but only as a Community Technology Preview (CTP) [25]. A beta version of Silverlight was released during the Mix07 conference in Las Vegas April 30th, 2007.

Summary

Silverlight is Microsoft's clone of Flex. They provide the same services to the developer and end user, and the users will not be able to tell the difference. The fact that Flex has been around for nearly three years now, must be an advantage for Adobe and Flex,

[2] eXtensible Application Markup Language

but Microsoft with its significant position within enterprise development may seize market shares.

The beta of Flex 3.0 is now available, and it will be interesting to see how Silverlight will be received when the final version is launched.

5.6.2 AJAX

AJAX, shorthand for Asynchronous JavaScript and XML, is a new way to use old technologies together. Google started using this technology and made it immensely popular within Web development. AJAX is a way to make regular Web pages rich and interactive, and makes Web applications as expressive as desktop applications.

AJAX is a composition of several technologies[13]:

- Presentation using XHTML and CSS
- Dynamic display and interaction using the Document Object Model (DOM)
- Data interchange and manipulation using XML and XSLT
- Asynchronous data retrieval using XMLHttpRequest
- Using Javascript to bind everything together

AJAX enables Web developers to update only parts of a page, and making the continuing post backs obsolete.

Summary

AJAX is a great opponent to Flex. Since it is open source and uses already known Web technologies, makes AJAX the preferred choice by many Web developers. The development time of a Flex application is much faster than a full AJAX Website, and thus might be a possible advantage for Flex.

5.7 Summary

Flex is an easy framework to use, and the entry level is low. With few minutes spent, an application with drag and drop capabilities is created. The fact that Adobe will release it as open source is positive. We hope the negative aspect that Flex Builder is not freeware, will be solved with an open source version in the near future.

CHAPTER 6

State of the art

Acceptance-, and functional-testing is designed to put manual tasks through their paces[16]. Doing so manually is both time consuming and error prone. Automating these types of tests achieves higher quality software and lowers the cost of development. There exists several solutions for automating functional tests of a wide variety of applications, and this section will describe the ones most frequently used.

6.1 Watir

Watir (Web Application Testing In Ruby) is an open source framework for performing automated functional testing, automated acceptance testing, and large-scale system testing of Web applications [20]. It uses Ruby as the language to write tests in, which is an advantage since Ruby is an interpretable language and has built in OLE[1] capabilities. This enables Watir to drive Internet Explorer programmatically with OLE. OLE is implemented above the Component Object Model (COM) architecture [43]. With the use of Ruby, it also has the ability to check database states and retrieve text from HTML code [29].

Listing 6.1 shows how to write a simple test to ensure the right functionality of a Web page. The first line is needed for Ruby to import the Watir functionality. Watir creates an instance of an Internet Explorer window with `Watir::IE.new`, which enables the tester to perform tasks on the window programmatically.

Listing 6.1: Simple Watir example [43]

```
1 require 'watir'
2 test_site = "http://www.google.com"
3 ie = Watir::IE.new
4 ie.goto(test_site)
5 ie.text_field(:name, "q").set("pickaxe")
```

[1] Object Linking and Embedding - A protocol developed by Microsoft. OLE allows an editor to "farm out" part of a document to another editor and then re-import it (see Section 4.4).

```
6  ie.button(:name, "btnG").click
7  if ie.text.include?("Programming Ruby")
8    puts "Test Passed. Found the test string: 'Programming Ruby'."
9  else
10   puts "Test Failed! Could not find: 'Programming Ruby'"
11 end
```

6.1.1 Summary

Currently Watir only supports testing of Web applications in Internet Explorer, and it does not support testing of Flex applications, hence it is no competitor to FunFX. Due to its implementation in Ruby, it drives IE with great precision, and it is a great tool for testing traditional Web applications. In the engineering method (see Section 2.2) existing solutions are studied before a new and better solution is proposed. Watir will be used as one of the existing solutions.

6.2 Selenium

Selenium is another framework for functional testing of Web applications, and it is similar to Watir (see Section 6.1). Selenium tests run directly in the browser and is supported in Internet Explorer, Mozilla, and Firefox on Windows, Linux, and Macintosh [28].

The tests are written as a collection of commands like `click`, `open`, and `type` and assertions such as `verifyValue`. These commands and assertions form a mini language called *Selense* [14].

Selenium drives the browser via JavaScript to run testing scripts, which means that Selenium has the ability to test client-side functionality implemented in JavaScript. The JavaScript engine is called BrowserBot and it translates the tests written in Selense into JavaScript, which it sends to the browser.

Selenium is deployed in three different versions, *Selenium Core*, *Selenium IDE*, and *Selenium Remote Control*.

6.2.1 Selenium Core

Selenium Core, also known as Test Runner, is written in JavaScript/DHTML that enables it to run in any browser which is JavaScript enabled.

The Selenium tests are written as HTML tables, similar to tests written in FitNesse (see Section 6.4). Table 6.1 shows how a simple Selenium test is written in test runner mode.

Figure 6.1 shows how the Selenium Test Runner is running the test-script detailed in Table 6.1. Whenever a mouse gesture is performed on any of the controls, the event is written into the textbox. At the end, the test verifies that all the events have happened as they should. If the test-steps (commands) pass, they get a light green color, and a light red color if they fail. The assertion `verifyValue` gets the color green if passed and red if failed.

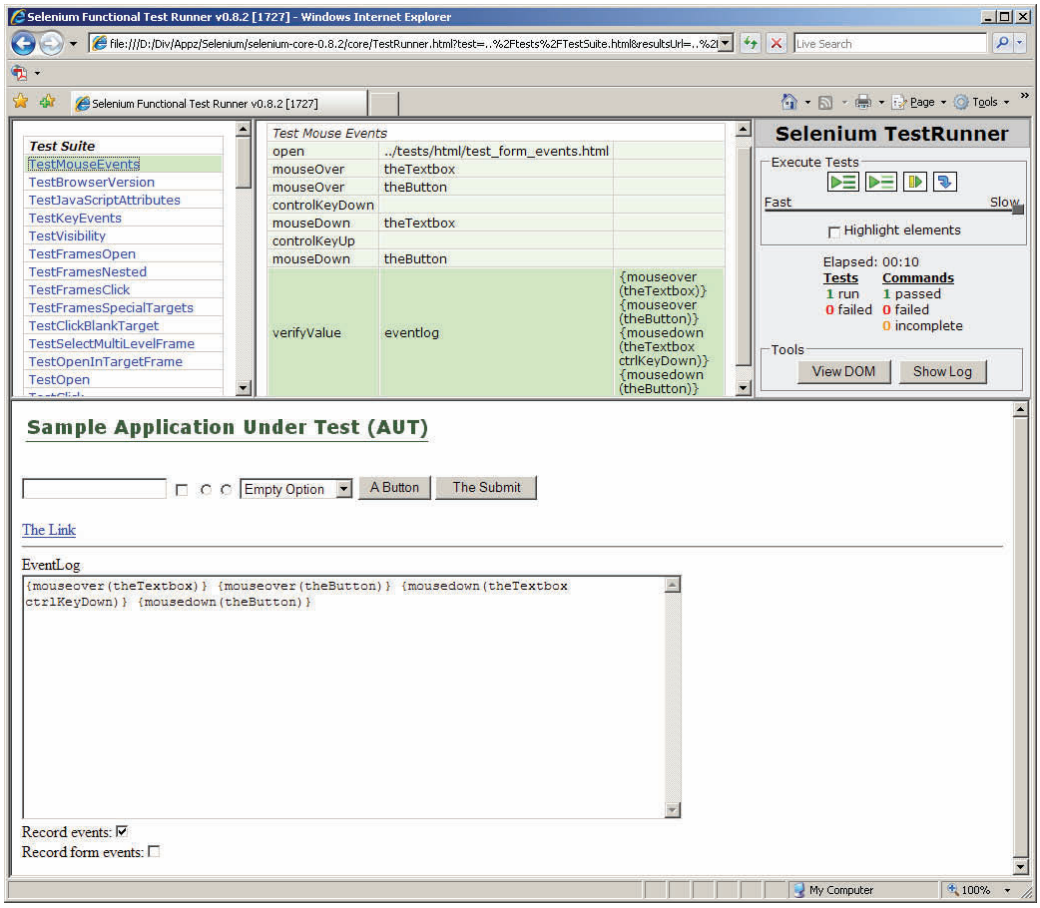

Figure 6.1: Selenium test runner

Table 6.1: A Selenium test

Test Mouse Events		
open	test_form_events.html	
mouseOver	theTextbox	
mouseOver	theButton	
controlKeyDown		
mouseDown	theTextbox	
controlKeyUp		
mouseDown	theButton	
verifyValue	eventlog	{mouseover(theTextbox)} {mouseover(theButton)} {mousedown(theTextbox ctrlKeyDown)} {mousedown(theButton)}

6.2.2 Selenium Remote Control

In the Selenium Remote Control, also known as Driven mode, tests are written in one of the supported programming languages, Java, Ruby, or Python. The scripts run as a separate process outside the browser. The driver executes the script and drive the browser by communicating with the BrowserBot[2], which runs inside the browser.

Driven scripts are more powerful and flexible than test runner scripts, and do easily integrate with xUnit frameworks.

Listing 6.2 shows a part of a test written in Ruby. It is simplified to show how to drive the browser with the `open` method, and then assert the task afterwards.

Listing 6.2: Selenium Driven mode ruby code [43]

```
1 puts selenium.open('/logout.html')
2 puts selenium.verify_location('/index.html')
```

6.2.3 Selenium IDE

Selenium IDE is an integrated development environment for Selenium testing implemented as a Mozilla Firefox extension. It contains the Selenium Core (see Section 6.2.1), which enables the user to record and play back tests in the actual environment that they will run in. It is a complete IDE where the user can record, edit, and debug tests.

The only drawback of Selenium IDE is that it is currently only supported by Mozilla Firefox, but the tests created/recorded are runnable in Selenium Core in Internet Explorer.

6.2.4 Summary

Selenium is one of the best functional testing tools for Web applications that are built with regular HTML and JavaScript. Its ability to perform tests within multiple browsers on multiple platforms makes it an excellent and versatile tool.

[2] A JavaScript engine translates the tests written in Selense into JavaScript

Unless the Selenium team decides to implement support for Flex applications[3], Selenium will not be a competitor to FunFX.

6.3 Mercury QuickTest Professional

Mercury QuickTest Professional (QTP) is an advanced testing solution for building functional and regression test suites. It is an automated testing tool, and uses a *keyword driven* approach (see Section 4.3.5), which simplifies test creation and maintenance. Mercury QTP supports testing of .NET, Web services, Java, ERP/CRM, Flex, and Windows applications [24].

With QTP, testers have the ability to create test cases by capturing flows directly from the application screens. The tester can record actions, and afterward play the test. QTP also provides the ability to create tests by simply declaring the steps using the script-free Keyword View [23].

6.3.1 Functional testing of Flex application

Mercury QuickTest Professional uses a keyword driven approach. This approach has great potential and makes writing tests easy. QTP performs functional tests using recording of a set of actions and replaying these actions. The user has the ability to assert the results of each action.

This record and replay method is an easy way for non-programmers to be able to use this test tool for acceptance testing of functional requirements. But it does not help developers during development of new applications, since QTP can only be used on already created applications.

Figure 6.2 shows the QTP application during the recording of a test. Each time the user initiates an event on an Internet Explorer window, the action is recorded into QTP. Listing 6.3 lists the test caught in Figure 6.2. Due to a lack of space, line number 1 is supposed to be in front of each of the other lines. QTP writes the tests in VB script, and supports both Internet Explorer and Mozilla Firefox. With Internet Explorer, QTP is able to reach objects in the Flex application by name, but with Firefox it uses the WINOBJECT in VB script to create events with a specific position on the page.

Listing 6.3: Recorded test case in QTP

```
1 Browser("Browser").FlexApplication("AutomatinTest").
2   FlexToggleButtonBar("_ToggleButtonBar1").Change "Home"
3   FlexToggleButtonBar("_ToggleButtonBar1").Change "Products"
4   FlexCanvas("Products").FlexSlider("priceSlider").Change 190
5   FlexCanvas("Products").FlexSlider("priceSlider").Change 200
6   FlexCanvas("Products").FlexCheckBox("Camera").Click
7   FlexCanvas("Products").FlexCanvas("productList").Check CheckPoint(
        "list")
```

When the recording of a test is finished, the user has the ability to replay the recorded test. The application is programmatically driven by QTP, with the use of the Automation

[3]Flex applications are packed into one Flash movie, a swf file, and cannot be accessed with regular HTTP requests and JavaScript

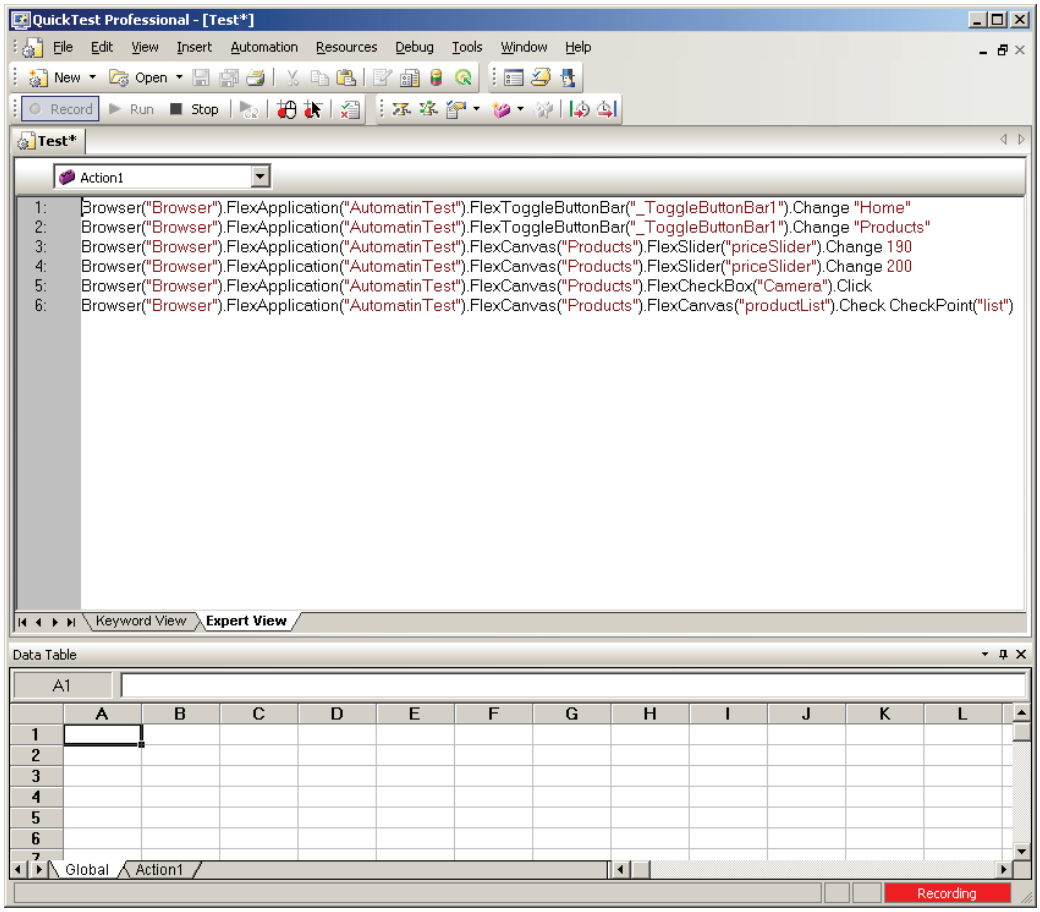

Figure 6.2: QuickTest Professional environment

Package (see Section 5.4.1), and the recorded events are performed. Figure 6.3 shows the window that is shown after running the stated tests.

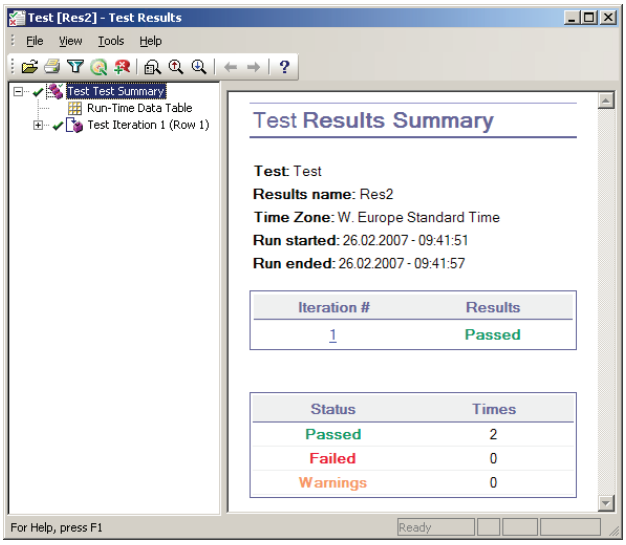

Figure 6.3: QuickTest Professional test result window

6.3.2 Summary

Mercury QuickTest Professional (QTP) is currently the only testing solution that supports testing of Flex applications. QTP is not open source and is expensive in use. The positive thing with QTP is that it is extremely easy to use for non-technical users to perform acceptance tests. The two most negative aspects with QTP are that it is an expensive non-open source tool and that it does not support test-driven development very well. This means that it is not a developers tool, but more a tool for Quality Assurance (QA).

QTP is a direct competitor to FunFX, but if FunFX is deployed correctly into the open source environment, it will have the potential to be the preferred tool. FunFX will support TDD (see Section 3.1.2).

During the development of FunFX, QTP will be used as a model for how to access the hierarchy of Flex components intuitively.

6.4 FitNesse

FitNesse is a software development collaboration tool. It enables customers, testers, and programmers to learn what their software should do and automatically compare that to what it actually does [12].

FitNesse is a testing tool, a wiki, and a Web server. It is a lightweight, open source framework for developing and running acceptance tests for Web applications. It has a keyword driven approach (see Section 4.3.5).

In the test shown in Table 6.2, the lines with `press` are lines where some action is performed and the lines with the keyword `check` are assertions where the tests pass or fail.

If such an assertion line fails, it turns red, and if it passes it turns green. The user thus gets an immediate feedback on how the test performed.

Table 6.2: Keyword-Driven test written in Fitness

egActionFixture		
start	fitness.fixtures.CountFixtures	
check	counter	0
press	count	
check	counter	1
press	count	
check	counter	2

FitNesse is also a wiki, which means that every page can easily be edited with a simple markup language. This means that it is very easy to create the test tables, and edit them if needed.

6.4.1 Summary

FitNesse does not support testing of Flex applications, but it has a smart way of writing tests, which might be interesting to consider when developing FunFX.

6.5 TestComplete

Another commercial automated testing tool is TestComplete from AutomatedQA. TestComplete is a full-featured environment for automated testing of Windows, .NET, Java, WPF (XAML) applications, and Web pages and servers. It has been designed to free developers and QA departments from the massive drain on time and energy required by manual testing [8]. It also supports multiple languages.

When it comes to testing Flex applications, TestComplete is capable to record and replay actions. This is done with coordinates[4], and the tests recorded are not intuitive like the tests recorded by QTP (see Section 6.3.1). It uses Microsoft's Accessibility Interface (MSAA) to identify and interact with controls inside a flex application. This makes it possible to test Flex applications residing in Internet Explorer [9].

6.5.1 Summary

TestComplete is an application that enables testing of multiple type of applications. This might be the reason why the testing of Flex applications is not done well. The focus is on supporting multiple technologies, rather than providing quality testing functionality. Because it uses coordinates when interacting with the Flex application, the tests written

[4]Tests that uses coordinates are more likely to break than tests that interacts with the display object directly

are not enduring and might break if a display object is moved slightly. It cannot be regarded as a competitor to QTP and thus it will not be given any more attention.

6.6 Summary

This chapter has described some of the most popular and used functional testing tools for Web applications. FitNesse was added as one of the tools even though it is not a functional testing tool because it writes tests in a clever way. Table 6.3 shows a summary of the tools. The comparison emphasize what kind of technology they support, if it supports TDD, and if it interacts with the browser programmatically[5].

Table 6.3: Summary of the state of the art functional testing tools

Tool	**HTML support**	**Flex support**	**TDD support**	**Programmatic browser interaction**	**Free**	**OSS**
Watir	Yes	No	Yes	Yes	Yes	Yes
Selenium	Yes	No	Yes	Yes	Yes	Yes
Mercury QTP	Yes	Yes	No	Yes	No	No
FitNesse	No	No	Yes	*	Yes	Yes
TestComplete	Yes	Yes	No	No	No	No

* Since FitNesse is not a functional testing tool it does not drive any browser.

During this comparison, Watir was found to be a very good testing tool, but it lacks support for Flex applications. The use of Ruby as language will be a model for developing FunFX. Selenium is also a quality tool, but was not found relevant as a model in the case study. Mercury QTP is the only tool that has the ability to actually drive a Flex application programmatically, but it lacks support for TDD. QTP will be a model during development of FunFX, especially regarding how it writes the test lines. In the early phase FitNesse's way to write tests was thought to be the best way to write tests with FunFX. During the prestudy it was decided to follow Watir and QTP instead. TestComplete is a huge testing tool, that was not easy to manage. Together with no support for TDD and no support to drive the Flex application programmatically, TestComplete came last in the comparison. To summarize, Watir and QTP will be the models in our engineering method (see Section 2.2).

[5]Programmatically interaction means that it interacts with the display objects, not using coordinates

CHAPTER 7

Development Technology

This chapter describes in more detail the technology used in this project.

7.1 Ruby

Ruby is a object-oriented programming language initially created by a Japanese programmer, Yukihiro Matsumoto, in 1995 [33]. Ruby is a single-pass interpreted language, and has become more and more popular among developers.

When Yukihiro developed Ruby, he wanted to create a language he himself enjoyed, which emphasizes human rather than computer needs [33].

Everything manipulated in Ruby is an object, and the results of those manipulations are themselves objects. Even primitives like integers are objects in Ruby.

Ruby does not support multiple inheritance, but classes can import modules as mixins. It also features dynamic typing, which means that variables are not required to be explicitly declared before they are used. The variable type is decided at runtime. Ruby relies less on the type (or class) of an object and more on its capabilities. Hence, Duck Typing: This means an object type is defined by what it can do, not by what it is [30]. Duck Typing can be defined in a single sentence;

> **If it looks like a duck, swims like a duck and quacks like a duck, then it is indeed a duck [38].**

It is an extremely powerful language, which has combined the semantics of Smalltalk with the syntax of Perl. This makes it compact, yet readable and maintainable. Ruby is a general-purpose language, and can either be used to build small scripts, build middle-tier server processes, write GUI applications, database interfaces, or generate dynamic content [33].

Ruby is a clean and simple language, which makes it highly readable and maintainable. There is no semicolon at the end of each line. One line is one statement of code. Ruby does

not use brackets as most other languages to denote the body of a compound statement, but rather an `end` statement at the end.

A method in Ruby is defined as in Listing 7.1.

Listing 7.1: Ruby method

```ruby
def sayGoodnight(name)
  result = "Goodnight, " + name
  return result
end
```

Listing 7.2 shows a simple Ruby class. Instance variables are like any other variables, but are written: `@nameofvariable`. This class overrides the method `to_s`, which is the method converting an object to `String`. The last two lines create a new `Song` object and call the `to_s` method to display its content.

Listing 7.2: Ruby class[33]

```ruby
class Song
  def initialize(name, artist, duration)
    @name     = name
    @artist   = artist
    @duration = duration
  end

  def to_s
    "Song: #{@name}--#{@artist} (#{@duration})"
  end
end

aSong = Song.new("Bicylops", "Fleck", 260)
aSong.to_s     >>     "Song: Bicylops--Fleck (260)"
```

Even though Ruby is very intuitive, there are some aspects that need to be addressed:

Names with capital letters are regarded as constants. Local variables should begin with a lower case letter, since Ruby will treat variables with capital letters as constants.

Must denote floating point numbers, because numbers are susceptible to method syntax, it is not possible to denote a floating point with a dot at the end (`99.`). It must be denoted `99.0` or `99.to_f`.

7.1.1 Method missing

Whenever a Ruby object receives a method it does not implement, it goes up the hierarchy of parents to see if some of them implement this method. If no one does, it raises an error. This is where the genius of the `method_missing` shows. In Ruby, the developer has the ability to implement a method called `method_missing`, which makes the developer able to support multiple methods dynamically. It is up to the developer to make use of the arguments passed to the method.

Listing 7.3 shows a simple example of how to use the `method_missing`. This forces the class that implements this method to return any object. This functionality will be used

to enable return of any type of Flex display object, due to the *parent-child* model Flex uses.

Listing 7.3: Use of the `method_missing` method in Ruby

```
1 def method_missing(object_name)
2   Flex.const_get(object_name).new
3 end
```

7.1.2 Ruby conventions

All programming languages have their own conventions and best practice, and Ruby is no different. In Ruby only class names and constants start with an upper case letter. All others should start with lower case letter. Method names are written in all lower case letters with an underscore dividing multiple words. FunFX will follow these conventions.

7.1.3 Interaction - irb

Interactive Ruby (irb) is an interactive command-line interpreter, which can be used to quickly test code. It is useful when discovering new functionality to test without creating a new file.

Listing 7.4 shows a simple irb session, where the developer is using `puts` to write `Hello world!` to the console.

Listing 7.4: Irb session

```
1 irb(main):001:0> puts ‘‘Hello world!’’
2 Hello world!
3 => nil
4 irb(main):002:0>
```

With irb it is also possible to create new objects and classes, which will be reachable within the current session.

7.1.4 Win32OLE

`Win32OLE` is a library, which provides Ruby developers an interface to OLE automation. With this library, it is easy to take control of any application that supports Microsoft's COM platform.

In this project, `Win32OLE` will be used to programmatically drive Internet Explorer, and access the Flash movie that is embedded within the HTML file.

Listing 7.5 shows a simple example using `Win32OLE` to create a new Internet Explorer window, and programmatically drive it. In the last line, a new `Win32OLE` object is created with the embedded Flash file named “flexapplication”.

Listing 7.5: Using `Win32OLE` to drive IE

```
require 'win32ole'

@ie = WIN32OLE.new("InternetExplorer.Application")
@ie.visible = true
@ie.navigate("http://localhost/flex/testApplication.html")
embeddedObject = @ie.document.getElementsByName("flexapplication")
```

7.1.5 Test::Unit

`Test::Unit` is the standard unit testing framework for Ruby development. It is basically three facilities wrapped into a neat package[34]:

1. It provides a way of expressing individual tests.
2. It provides a framework for structuring the tests.
3. It provides flexible ways of invoking the tests.

When using the framework, a Ruby class which extends `Test::Unit::TestCase` must be created. Within such a test-case, multiple tests can be created. All the test methods created need to be named with `test` as a prefix, for instance `test_initialize`. This is important because `Test::Unit` uses reflection to find tests to run [34]. To be able to check whether the tests are correct or not, `Test::Unit` uses methods called assertions. These methods checks whether the assertion is true or not. Test-cases can be collected into test-suites.

To be able to run the tests, `Test::Unit` comes with a test runner that invokes all the tests.

Listing 7.6 shows a simple test case, which tests the creation of a class that converts numbers to roman numbers.

Listing 7.6: Simple test-case written with `Test::Unit`

```
require 'test/unit'
class TestRoman < Test::Unit::TestCase
 def test_simple
  assert_equal("ix", Roman.new(9).to_s)
 end
end
```

Since the objective in this project is to create a testing framework using Ruby, `Test::Unit::TestRunner` will be reused to run the written tests.

PART IV

The FunFX Framework

CHAPTER 8

Introduction to FunFX

Flex applications lack the possibility to be functionally tested with an open source testing framework. At the moment, the only way to functional test Flex applications is with the expensive enterprise application Quick Test Professional (QTP) from Mercury (see Section 6.3). Not only is QTP expensive, but it does not support TDD (see Section 3.1.2).

This project will try to fill this gap with a new testing framework based on TDD. There are several aspects to consider when developing tools for TDD, e.g., a developer writes tests for code that has not yet been implemented, and therefore needs to inform the user properly. The results of this project will address a different type of audience than QTP, namely the developers.

The Automation Package will be utilized to the extent it is possible. The Automation Package has not implemented support for test-driven development, but still there are many aspect of this library the framework can use.

This part will elaborate on the requirements and implementation of the framework, called FunFX. It will show how the framework is designed, how this design is implemented, and how it works.

CHAPTER 9

Requirement specifications for FunFX

There are several requirements for successful usage of a test automation tool. Primarily, the tool should be widely available within the project. Developers and testers must be able to create, execute, change, and debug tests at any time. Additionally, the tests should be included in the continuous integration system, providing feedback from the automated test executions.

This chapter will elaborate the requirements for the FunFX framework. Both functional and non-functional requirements will be elicited. The following list explains in more detail how this chapter is outlined.

- Section 9.1 portrays a scenario, which describes how the application is supposed to work.
- Section 9.2 states different goals of the application.
- Section 9.3 describes what functions the application need to support.
- Section 9.3.1 shows how the functional requirements and users relate to each other, with the use of use case tables.
- Section 9.4 describes the non functional requirements, like performance, modifiability, usability, testability, availability, and security.

9.1 Scenario

The setting for this scenario is the development of a new Flex application.

The use of FunFX will follow the test-driven development approach (see Section 3.1.2). Before the developer implements functional requirements, tests are written for the requirements. This is done by the use of Ruby and the FunFX framework. The developer implements code incrementally to make the tests pass.

The tests will be stored in separate Ruby files, and can be run every time there has been a change in the code, hence they will act as regression tests as well. When the developer creates the Flex project a library file must be included to enable the application to be tested.

9.2 Goals

The FunFX framework will try to reach the following goals:

- The framework must support any Flex application compiled with the FunFX library file.
- The framework must be able to drive a Flex application programmatically.
- The framework must be able to support test-driven development.
- The framework must support writing tests in Ruby.
- The framework must be able to run multiple test suites, and provide useful results from the tests.

9.3 Functional requirements

Table 9.1 enumerates each of the functional requirements of the application. The functional requirements describe what functionality the application needs to support.

Table 9.1: Functional requirements

F1	The framework must be able to interact with Internet Explorer
F2	The framework must be able to define and run tests
F3	The framework must provide useful results from the tests
F4	The framework must be able to perform actions on a Flex application
F5	The framework must provide the possibility of test suites
F6	The framework must be able to reach Flex objects by name, id, or label
F7	The framework must be able to make assertions on Flex objects
F8	The framework must be able to wait for slow data providers
F9	The framework must be able to both run the steps visually and not visually
F10	The framework must be able to have different speeds of interaction
F11	The framework must be able to set paths in the tests for better readability
F12	The framework must be able to take advantage of regular unit test runners

9.3.1 Use Cases

The functional requirements are verbalized and visualized as use cases, since UP is the development method (described in Section 3.1). The use case diagram shows use case relationships and how they relate to the user of the application.

The use cases described are of *sub function* level, which means they are small steps of the entire application.

Use Case - Initialize FunFX

Table 9.2 shows the use case for initializing the framework.

Table 9.2: Use Case - Initialize FunFX

Use Case: Initialize FunFX	
Scope:	FunFX()
Level:	Sub function
Primary Actor:	User
Stakeholders and Interests: • User: Wants to initialize the framework to be able to use its capabilities to test a Flex application.	
Preconditions:	None
Success Guarantee:	The framework is initialized, and has created an instance of Internet Explorer. The framework is ready to be used by the user.
Main Success Scenario: 1. User initializes the framework 2. System creates all needed classes dynamically from a configuration file 3. System creates an instance of Internet Explorer	
Extensions: **2a** Configuration file not found 1. System will notify the user	

Use Case - Write test

Table 9.3 shows the steps a developer needs to go through to write a simple test.

Use Case - Create test suite

Table 9.4 shows what a developer needs to do to create a test suite of multiple tests.

Use Case - Create and run tests

Table 9.5 shows the entire process for developing and running a test suite.

Table 9.3: Use Case - Write test

Use Case: Write test	
Scope:	FunFX()
Level:	Sub function
Primary Actor:	User
Stakeholders and Interests: • User: Wants to build a test with an assertion	
Preconditions:	None
Success Guarantee:	A test with steps and an assertion at the end is created
Main Success Scenario: 1. User creates a test 2. User enters wanted steps into the test 3. User enters an assertion into the test	
Extensions: No extensions	

Table 9.4: Use Case - Create test suite

Use Case: Create test suite	
Scope:	FunFX()
Level:	Sub function
Primary Actor:	User
Stakeholders and Interests: • User: Wants to build a suite of multiple tests.	
Preconditions:	The tests must be written
Success Guarantee:	A suite with multiple tests are written
Main Success Scenario: 1. User creates a test suite class 2. User adds pre-existing tests to the suite	
Extensions: No extensions	

Table 9.5: Use Case - Create and run tests

Use Case: Create and run tests	
Scope:	FunFX()
Level:	User goal
Primary Actor:	User
Stakeholders and Interests: • User: Wants to build and run the tests. The user wants useful information back about the results of the tests if something went wrong.	
Preconditions:	None
Success Guarantee:	A suite of tests have been written and run
Main Success Scenario: 1. User performs Initialize FunFX (see Table 9.2) 2. User performs Write test (see Table 9.3) 3. User performs Create test suite (see Table 9.4) 4. User runs the test suite 5. System performs the tests visually 6. System provides the results of the tests	
Extensions: No extensions	

9.4 Non-functional requirements

Functional requirements are usually the focus when developing applications, but it is important to remember the non-functional requirements [11]. Non-functional requirements are requirements that are not directly attached to the functionality, but may have great impact on the architecture. They are also called *quality attributes* and are normally defined in the following sections: *performance*, *availability*, *modifiability*, *security*, *testability*, and *usability* [11].

Since FunFX is a tool for testing applications, we do not find the security attribute very relevant. Development projects often use development integration systems to build the solutions and run the tests.

This section will describe the non-functional requirements of FunFX.

9.4.1 Usability

Usability is concerned with how easy it is for the user to use the application. It is also concerned with what kind of user support the system delivers. To make the application as usable as possible, the flow of the system will be designed from user-feedback (see Section 2.1.1).

Q1.1 - Give user informative response after test is finished
To inform the user what went wrong during a test run it is important to provide high usability. The user must receive information concerning the test run immediately after the test is complete.

Source	User
Stimulus	User run a test
Artifact	FunFX
Environment	Runtime
Response	The user should get a response on what went wrong
Response measure	When test suite is finished

9.4.2 Testability

The testability is concerned with how easy it is to test the application. To support this quality attribute, test-driven development (see Section 3.1.2) will be used, which uses unit testing to ensure that every method is working correctly.

Q2.1 - Test a new feature
When a developer has added a new feature, the application will be suited with tests for the remaining code. The developer needs to write tests for the new code.

Source	Developer
Stimulus	Have added a new feature
Artifact	FunFX
Environment	Design time
Response	Writes a test for the feature
Response measure	1 hour

9.4.3 Modifiability

The modifiability of a system is about how easy it is to change parts of the system. It is important to keep a system modular so that modules can easily be changed, to ensure modifiability at a high level.

Q3.1 - Add a new component
If a developer creates a new custom component in Flex, the only change should be in a configuration file, which describes all Flex components with their events and properties.

Source	Developer
Stimulus	Adds a new custom component to Flex
Artifact	Configuration file
Environment	Design time
Response	Need to add a node to the configuration file
Response measure	5 minutes

Q3.2 - Add support for other browsers
Currently the only browser supported is Internet Explorer, but additional browsers can easily be added by creating a new browser class for the desired browser[1]. What a developer needs to do, is to create a way to grab the flash object embedded in the HTML page.

Source	Developer
Stimulus	Want to implement support for a new browser
Artifact	FunFX
Environment	Design time
Response	Need to implement a new browser class, which enables the framework to grab the embedded flash object.
Response measure	5 hours

9.4.4 Availability

The availability is concerned with system failure and its associated consequences. A system failure occurs when the system no longer delivers a service consistent with its specifications [11].

Q4.1 - Unable to reach the Internet Explorer browser
Sometimes the Internet Explorer might be slow to be initialized. If this happens, the framework must not fail, but try repeatedly to reach the browser until successful.

Source	Internet Explorer
Stimulus	Unable to reach the browser
Artifact	FunFX
Environment	Normal operation
Response	Continue to try reach the browser
Response measure	Seconds

[1] Flex is supposed to behave equally in all browsers, but the option is added for future purpose

CHAPTER 10

Design of FunFX

FunFX consists of several parts that collaborate to provide a functional testing framework. This chapter will describe in more detail the different parts of the design.

10.1 Design patterns

It is important when developing software solutions to define a base design to get a reliable and maintainable construction. Use of good working tactics and patterns where applicable, will ensure working software that is easy to understand and to maintain.

This section will elaborate in more detail what tactics and patterns used during the implementation of FunFX.

10.1.1 Adapter

Different applications are often used within software development, sometimes written in different languages, or has different APIs or objects. To enable these applications to work together an adapter needs to be built that works with both applications API, and to mediate between them. The adapter object converts the original interface of a component into another interface [22].

In this project, Flex applications will be tested with the use of Ruby, and thus it is needed a way to reach and act upon the Flex objects. An adapter may solve this problem. This will be explained in more detail in Section 11.1.

10.1.2 Factory

The factory pattern is a design pattern, which deals with the problem of creating objects without specifying the exact class of object that will be created [39]. A class called the factory is created to handle the creation of objects.

10.1.3 Façade

The façade pattern is a "front-end" object that is the single point of entry for a larger group of code [22]. With the use of façade pattern, the software library is easier to use and to understand.

This pattern is widely used during the development of frameworks. This is partly because it eases the use, but also this makes the framework more modifiable. Façade patterns are usually accessed via the Singleton pattern (see Section 10.1.4) [22].

10.1.4 Singleton

In software development it is often important to be able to have global visibility of a class, and limit the number of instances of this class to only one. It is useful when exactly one object is needed to coordinate actions across the system. This is easily implemented with the pattern called *Singleton* [22]. As shown in Listing 10.1, the class is accessed by its static method `instance`. If an instance has not yet been created, one is created, or else the earlier created instance is returned.

Listing 10.1: A simple way to implement singleton (not thread safe)

```
1 def self.instance
2     return @@instance if defined? @@instance
3     @@instance = new
4 end
```

Listing 10.2 shows the new *thread safe* way to implement singleton. This is the way this pattern has been used in this project. This code includes the actual code in the singleton file.

Listing 10.2: The thread safe way to implement singleton

```
1 require 'singleton'
2 class SingletonClass
3   include Singleton
4
5   # Some other code
6 end
```

This approach can be useful when creating a testing framework, since all test classes will be using the same instance of the framework FunFX.

10.2 Overall design

Figure 10.1 shows how the overall design of the framework looks and works. Whenever a Flex application is compiled, a flash object is created, which is embedded in a HTML page or similar to be displayed in a browser. The FunFX Flex adapter is compiled into the flash object, enabling the FunFX framework to access the Flex objects via `WIN32OLE` (see Section 7.1.4).

The FunFX framework is built dynamically with the help from a file called AutomationGenericEnv.xml, which describes all the possible Flex objects and their events. This makes the framework easily modifiable.

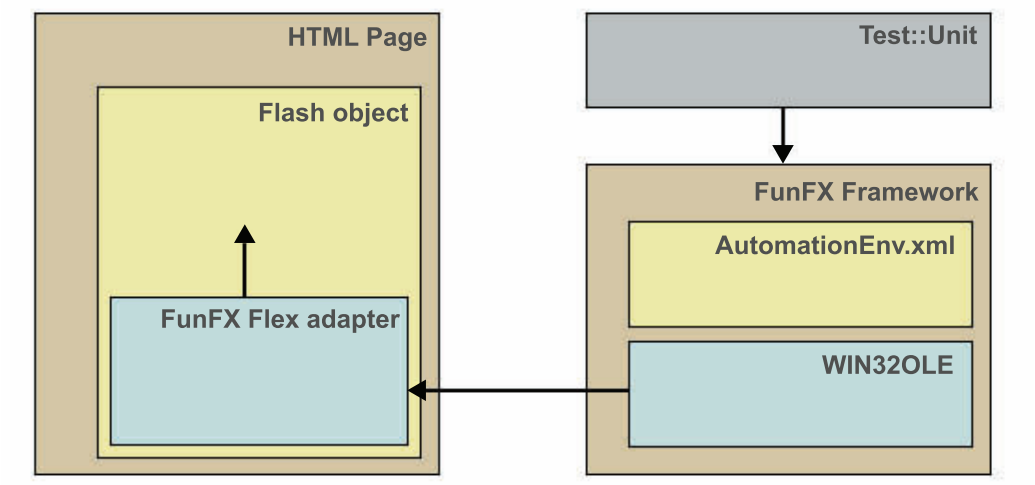

Figure 10.1: Overall design

With the use of `Test::Unit` (see Section 7.1.5), it is possible to create test cases and test suites of FunFX objects and methods, and assert the actions.

All these parts of the design will be described in more detail in the following sections.

10.3 The FunFX Flex adapter

The FunFX Flex adapter is an ActionScript library file, which is compiled into the Flex application. This makes it possible to access the Flex display objects within the application. It consists of one adapter class file, which was developed in this project, and a couple of files received from Adobe[1].

FunFX This class extends `flash.display::Sprite`, which is a base class for a Flex application. It is also built as a mixin (see Section 5.2.1) making it possible to register the library for the application complete event[2]. When the application is complete, the adapter will register all the ExternalInterface methods (see Section 5.2.2), so the FunFX framework is able to perform actions on the application. It uses the custom files received from Adobe to set the automation environment.

Classes designed and implemented by Adobe

XMLParser A helper class for parsing the XML file describing the Flex display components.

CustomAutomationEventDescriptor This class holds information about events, e.g., the event type and the event implementation.

[1]The files are under Adobe license, and are allowed to be used as long as it is stated that they were developed by Adobe

[2]The application complete event of the Flex application ready for testing

CustomAutomationMethodDescriptor This class holds information about methods of the display object.

CustomAutomationPropertyDescriptor This class holds information about properties for both events of the display object and the display object itself.

CustomAutomationClass This class holds information about one single display object class, including its events, methods, and properties.

CustomAutomationEnvironment This class contains `CustomAutomationClass` objects of all possible display objects.

10.3.1 Class diagram

Figure 10.2 shows the class diagram of the FunFX Flex adapter. The FunFX class is the adapter and it is used by the FunFX Framework built in Ruby (see Section 10.4) to invoke actions on a Flex application. The adapter invokes the Automation Package (see Section 5.4.1) with the `CustomAutomationEnvironment`.

10.4 The FunFX framework

The FunFX framework contains the classes used for writing tests and programmatically drive a browser. It also builds a class hierarchy of Ruby classes that are equivalent to the display objects found in Flex. The following will explain the different files in more detail:

AutomationGenericEnv.xml: This file contains a description of all the different Flex display objects. This file is used to dynamically build an entire class hierarchy every time a test is run. This has been done due to modifiability, the file is the only thing that needs to be changed when a new custom display component is introduced.

Flex: Flex is a Ruby module, which contains a Base class for all the Flex display objects. It introduces a method to implement custom methods dynamically.

XmlParser: The XmlParser is a static Ruby class that parse the AutomationGenericEnv.xml and builds the class hierarchy based on the Base class in the Flex module. It creates different methods for different types of Flex components.

FunFX: The FunFX class is the façade class the user gets to know. It implements the singleton pattern (see Section 10.1.4) and creates a new browser window. It extends `Flex::Base` and can be regarded as the root of the display hierarchy of the tested Flex application.

Browser: Browser is a module created for the single purpose to possibly support different browsers. The module will hold implementations for different browsers. Currently, the only browser supported is Internet Explorer. The class `InternetExplorer` uses `WIN32OLE` (see Section 7.1.4) to interact with Internet Explorer.

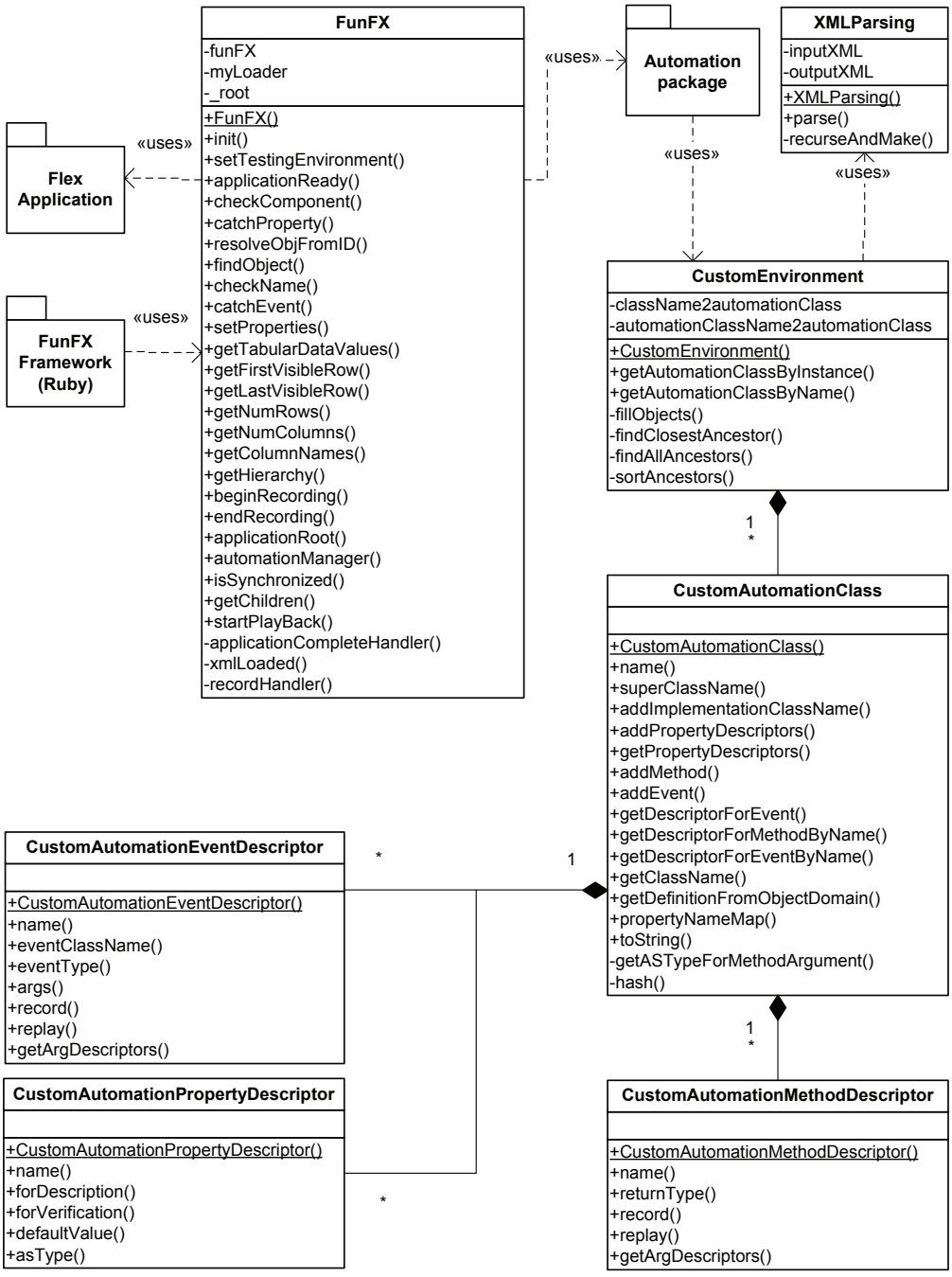

Figure 10.2: Class diagram of the FunFX Flex adapter

10.4.1 Class diagram

Figure 10.3 shows the class diagram of the FunFX Framework. The central part is the `Funfx` class and its super class `Base`. `Funfx` is a singleton class and contains a browser implementation. The `Funfx` class also uses the `XmlParser` to build a class hierarchy of the Flex display objects.

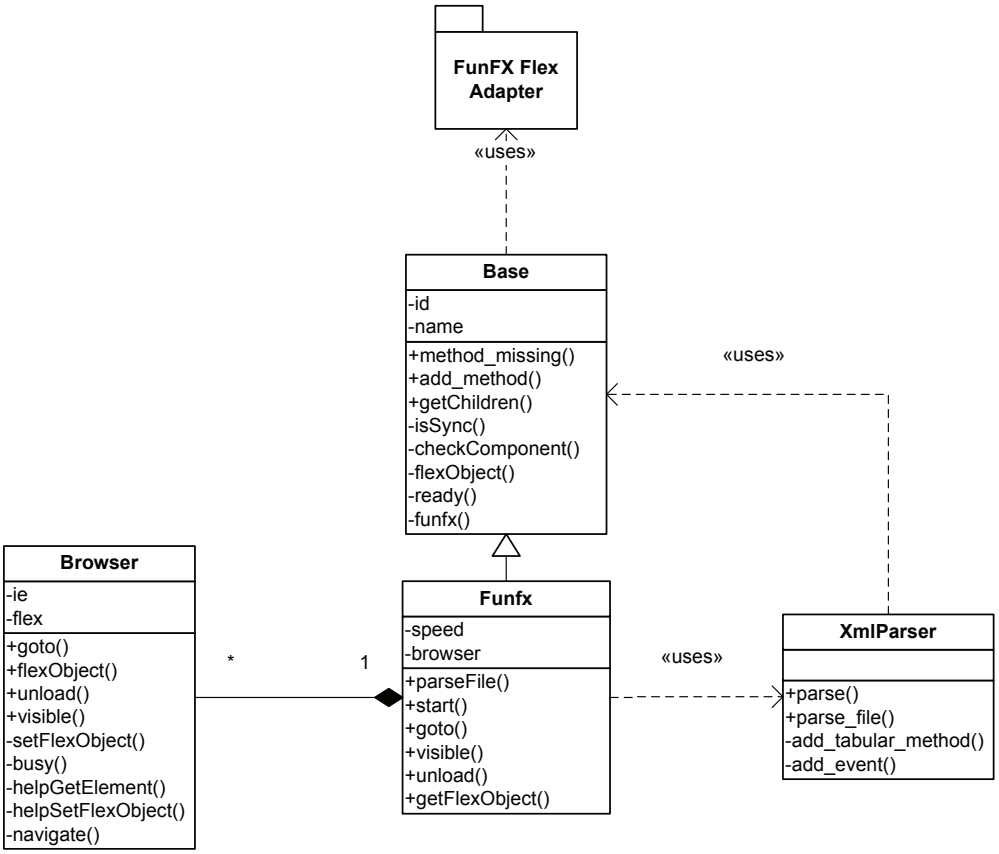

Figure 10.3: Class diagram of the FunFX Framework

CHAPTER 11

Implementation of FunFX

This chapter will describe the implementation of FunFX. It will concentrate on the most important parts of the application, and will show code snippets to be able to give an understandable description of the application's structure and how it works.

The parts the chapter will go into are the FunFX Flex adapter, which enables the Framework to invoke events on the Flex application, the FunFX Framework, and how this Ruby application uses `WIN32OLE` (see Section 7.1.4) to call methods in the adapter. It will also describe how to use the already provided `Test::Unit` framework (see Section 7.1.5) to create test cases and run these cases with the `Test::Unit::TestRunner`.

11.1 FunFX Flex Adapter

A Flex application is compiled to a Flash movie that is embedded into a HTML file or similar. The Flex objects are thus not visible or reachable from the outside of this object. To be able to reach these inner objects, an adapter was built, which can mediate between the Flex application and the FunFX framework[1].

11.1.1 Initialize adapter

The adapter is created as a Flex library file[2], and is meant to be added as a library to a Flex application. The desired functionality of the adapter was to be initialized automatically, without any interaction from the user. This problem was solved with the use of the mixin function (see Section 5.2.1).

Listing 11.1 shows the adapter's constructor and functions used during the automatic initialization. Because the class extends `Sprite`[3], and contains a static `init` method,

[1] The adapter pattern was used, as mentioned in Section 10.1.1
[2] A Flex library file is a group of ActionScript files packed to a file with extension swc
[3] The Sprite class is a basic display list building block

and is built as a mixin, the adapters `init` function is invoked automatically when compiled together with a Flex application. The `init` function invokes the constructor that creates a new adapter object. The constructor also registers an event listener to the Flex application, which makes the `applicationCompleteHandler` function fire when the Flex application is loaded. This function removes the event listener and sets the testing environment with a XML file called *AutomationGenericEnv.xml*[4].

This part of the framework, i.e., setting the test environment, is done by classes provided by Adobe. Adobe was kind and provided this project with some classes under development[6]. These classes are:

- `CustomAutomationClass.as`
- `CustomAutomationEventDescriptor.as`
- `CustomAutomationMethodDescriptor.as`
- `CustomAutomationPropertyDescriptor.as`
- `CustomEnvironment.as`
- `XMLParsing.as`

What they do, are to parse an XML file and set what kind of information the Flex display objects hold.

Listing 11.1: Initialize adapter

```
1  [Mixin]
2  public class FunFX extends Sprite
3  {
4    private static var funFX:FunFX;
5    private var myLoader:URLLoader;
6    private static var _root:DisplayObject;
7
8    public function FunFX(root:DisplayObject)
9    {
10     super();
11     root.addEventListener(FlexEvent.APPLICATION_COMPLETE,
12             applicationCompleteHandler);
13     _root = root;
14
15     //ExternalInterface actions
16   }
17
18   public static function init(root:DisplayObject):void
19   {
20     if(!funFX)
21     {
22       _root = root;
23       funFX = new FunFX(root);
24     }
25   }
26
```

[4]This file is the same as the FunFX framework uses when generating Ruby classes (see Section 11.2.1)

```
27    private function applicationCompleteHandler(event:FlexEvent):void
28    {
29      _root.removeEventListener(FlexEvent.APPLICATION_COMPLETE,
30                  applicationCompleteHandler);
31      funFX = this;
32
33      //Load environment XML
34      var te:String = "AutomationGenericEnv.xml";
35      setTestingEnvironment(te);
36    }
37  }
```

11.1.2 ExternalInterface - Making methods reachable

With the use of the ExternalInterface API (see Section 5.2.2), ActionScript methods are made externally available from either JavaScript or through an ActiveX component.

Listing 11.2 shows how the methods `catchEvent`, `catchProperty`, and `checkComponent` are added as callback methods in the ExternalInterface. The first argument in the `addCallback` method is the call name the Framework uses to call the method. These methods are added as callbacks in the constructor, and thus added during the automatic initialization (see Section 11.1.1).

Listing 11.2: Stating the methods made available through the ExternalInterface

```
1  if(ExternalInterface.available)
2  {
3    ExternalInterface.addCallback("addevent", catchEvent);
4    ExternalInterface.addCallback("checkProperty", catchProperty);
5    ExternalInterface.addCallback("checkComponent", checkComponent);
6    .
7    .
8  }
9  else
10   trace("Error: ExternalInterface not available");
```

11.1.3 Locating objects and replay events

The first contact the framework has with the adapter is to check whether the component is present or not. This is done with the `checkComponent` method. Listing 11.3 show this method. The method uses the `findObject` method to find a child with the name as provided. The name might be the id, name, or label of the display object. If a display object is found, an id is created[5] and returned. If no object is located, the method returns false.

Listing 11.3: The checkComponent method

```
1  public function checkComponent(parentID:String, name:String):*
2  {
3    var obj:IAutomationObject = findObject(parentID,name);
```

[5]The id is created with a class called `AutomationID` provided with the automation package

```
4     if(obj == null) return false;
5     else return createID(obj);
6   }
```

The id returned from the `checkComponent` method can be used with the `resolveObjFromID` method to easily find the corresponding object. Listing 11.4 shows this method. It uses a method available in the automation package to find it.

Listing 11.4: The resolveObjFromID method

```
1   public function resolveObjFromID(objectID:String):UIComponent
2   {
3     var objArray:Array = resolveID(objectID);
4     return UIComponent(objArray[0]);
5   }
```

In Flex applications, the objects are stored as a hierarchy. When locating an object, the adapter needs to know the parent of the wanted object. The framework provides the adapter with this information in a String, which is created with the help from `AutomationID`. Listing 11.5 shows the method `findObject`, which uses this String to find the desired child object. The hierarchy of display objects might become a weak point when writing tests. A GUI might be refactored a bit with an additional container holding some of the display objects, without changing the functionality of the application. When such a change is done, the tests should not break. Hence, a mix of relative and absolute paths is supported in FunFX. The way this is done, is by the use of an breadth-first search algorithm. It first searches all the children of the parent, and then continues down in the hierarchy one level at the time. Figure 11.1 shows how the algorithm searches down a hierarchy. When using a breadth-first search, the requirement of both absolute and relative path is covered. The search is shown in Listing 11.6. The method `startSearch` is a helper method when locating an object from the root.

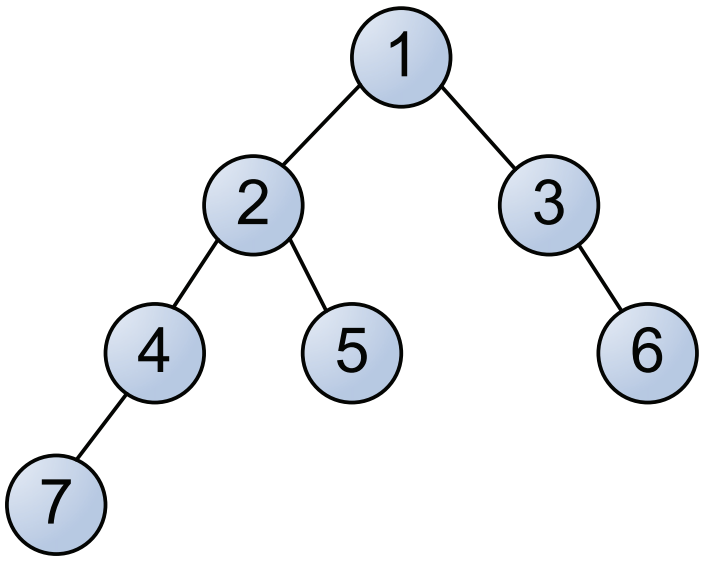

Figure 11.1: Breadth-first search algorithm

Listing 11.5: Method that locates a certain Flex object

```
1  public function findObject(parentID:String, name:String):
       IAutomationObject
2  {
3    if(name == null) return null;
4    if(parentID == null)
5      return startSearch(name);
6    else
7      return performSearch(new Array(resolveObjFromID(parentID)),name)
         ;
8  }
```

Listing 11.6: Search method for locating the desired display object

```
1  protected function startSearch(name:String):UIComponent
2  {
3    var parent:DisplayObjectContainer = DisplayObjectContainer(
       applicationRoot);
4    var openList:Array = new Array();
5    for(var i:int=0; i<parent.numChildren; i++)
6    {
7      try
8      {
9        openList.push(UIComponent(parent.getChildAt(i)));
10     }
11     catch(e:Error)
12     {}
13   }
14   return performSearch(openList, name);
15 }
16
17 protected function performSearch(openList:Array, name:String):
     UIComponent
18 {
19   var closedList:Array = new Array();
20   while(openList.length != 0)
21   {
22     var child:UIComponent  = UIComponent(openList.shift());
23     if(checkName(child, name)) return child;
24
25     for(var i:int=0; i<child.numAutomationChildren; i++)
26     {
27       try
28       {
29         openList.push(UIComponent(child.getAutomationChildAt(i)));
30       }
31       catch(e:Error)
32       {}
33     }
34     closedList.push(child);
35   }
36   return null;
37 }
```

When a test case wants to play an event on the Flex application, it calls the `catchEvent` method within the adapter. Listing 11.7 shows this method. It receives information about an event and initiates a replay of the event. It means that it performs an action on the Flex application. The arguments are:

Event type The event class, e.g., `flash.events::MouseEvent`.

Event type The event type, e.g., if the display object is a button, the event might be `click`.

Object id The object presented as an id string created by a class from Adobe called `AutomationID`. This id contains information about all the parents of the object.

Properties The properties argument, a XML string that describes the additional properties of the event, e.g., the text of an input-text event.

With the use of the method `resolveObjFromID`, which is shown in Listing 11.4, it finds the `IAutomationObject` that the event is being played on. The method has been made as generic as possible, and all types of events can be supplied as a string. It uses this string and the method `getDefinitionByName`, to be able to create an event object from the string supplied. This approach is using the factory pattern (see Section 10.1.2)

If the properties argument is set, it uses the method `setProperties`, which is shown in Listing 11.9, to set the additional properties of the event.

At the end, the method uses the method `startPlayback`, which is shown in Listing 11.10.

Listing 11.7: The method that catches the events created in a test-case

```
1  public function catchEvent(eventType:String, eventClass:String,
       objectID:String, properties:Object=null):String
2  {
3    var object:UIComponent = resolveObjFromID(objectID);
4
5    if(object == null) return "No such object";
6    else if(!Automation.automationManager.isSynchronized(object))
       return "Not synchronized";
7
8    try
9    {
10     var tempEvent:Class = getDefinitionByName(eventClass) as Class;
11     var replayEvent:* = new tempEvent(eventType);
12
13     if(properties != null)
14     {
15       var error:String = setProperties(replayEvent, object,
           properties);
16       if(error == null)
17         startPlayback(object, replayEvent);
18       else
19         return error;
20     }
21     else
22       startPlayback(object, replayEvent);
23   }
```

```
24    catch(eventError:Error)
25    {
26      return eventError.message;
27    }
28    return null;
29  }
```

The method `setProperties` gets the event, the flex object, and the properties as arguments. The `properties` are an XML object, and the method parses this object to set the properties of the event. Listing 11.8 shows how a `properties` XML object might look like.

Listing 11.8: A sample event properties XML

```
<Event Name="Select">
  <Implementation Class="mx.automation.events::ListItemSelectEvent"
            Type="select"/>
  <Property Name="itemRenderer" >
  <PropertyType Type="String" />
  </Property>
  <Property Name="triggerEvent" DefaultValue="flash.events::
      MouseEvent">
  <PropertyType Type="Event"  />
  </Property>
</Event>
```

When selecting or changing the states of a display object, the event often requires a child of the control supplied. The adapter then loops over the children and checks whether their automation value equals the supplied argument. If no such object exists, an error string is returned. The framework picks up this message and posts it to the user.

Listing 11.9: Part of the setProperties method

```
1  public function setProperties(replayEvent:*,
2            object:IAutomationObject,
3            properties:Object):String
4  {
5    var xml:XML = new XML(properties);
6    for (var i:Object in xml.Property)
7    {
8      var propertyXML:XML = xml.Property[i];
9      if(propertyXML.@DefaultValue != "false")
10     {
11       var type:String = propertyXML.PropertyType[0].@Type;
12       var tempClass:Class;
13       .
14       .
15       var check:Boolean = false;
16       var count:Number;
17       tempClass = getDefinitionByName(type) as Class;
18       if(propertyXML.@Name == "itemRenderer")
19       {
20         for (count = 0; count < object.numAutomationChildren; count
             ++)
21         {
```

```
22            if(object.getAutomationChildAt(count).automationValue ==
23                propertyXML.@DefaultValue)
24            {
25              replayEvent[propertyXML.@Name] =
26                IListItemRenderer(object.getAutomationChildAt(count));
27              check = true;
28              break;
29            }
30          }
31          if(!check)
32            return "No such itemRenderer <"
33              + propertyXML.@DefaultValue.toString()
34              + "> on the object <"
35              + object.automationName + ">";
36        }
37        .
38        .
39      }
40    return null;
41  }
```

Listing 11.10 shows the method, which creates an `AutomationReplayEvent` object with the Event and the `IAutomationObject` supplied from the `catchEvent` method. The object is replayed with the use of the `replayAutomatableEvent` method in the `AutomationManager` class.

Listing 11.10: Method that starts to replay an event

```
1  private function startPlayback(obj:IAutomationObject, ev:Event):void
2  {
3    var run:AutomationReplayEvent = new AutomationReplayEvent(
4      "replay", false, false, obj, ev);
5    automationManager.replayAutomatableEvent(run);
6  }
```

11.2 FunFX framework

The FunFX framework is implemented using Ruby (see Section 7.1). This is due to its DOM capabilities and that it is a scripting language. The "no need to compile" makes it great for creating test cases.

This section is divided into three parts, one for the creation of the classes, one for the façade class (`Funfx`), which is the entrance for the use of the framework, and one for the handling of synchronization between the Flex application and the framework.

11.2.1 Creating Flex classes

The central element of the FunFX framework is an XML file with the name Automation-GenericEnv.xml. This file is delivered from Adobe, and it describes all the Flex display objects and their events and properties. This file is parsed, and with the information provided from this file, all the Flex display objects are created as Ruby classes.

Listing 11.11 shows the class `XmlParser` with its static method parse. This method takes the name of the XML file as an argument. An example of a part of the XML file is shown in Listing 11.12. This file is parsed, and for each `ClassInfo` tag a new class is created with the method `generate_new_class` in the module named `Flex` shown in Listing 11.15. For each of the `Event` tags, a method is added with the name of the event to the newly created class (see Section 11.2.1). This new method calls the method called `addEvent` in the ExternalInterface of the FunFX Flex adapter (see Section 11.1) to perform the event.

Listing 11.11: The XML parser

```
1  class XmlParser
2    private_class_method  :new
3
4    def self.parse_file(document)
5      file = File.new(document)
6      doc = REXML::Document.new file
7      doc.elements.each("TypeInformation/ClassInfo") do |element|
8        class_name = element.attributes["Name"]
9
10       new_class = Flex.generate_new_class(class_name, element.
             attributes["Extends"])
11
12       if(element.attributes["SupportsTabularData"] == "true")
13         add_tabular_method(new_class)
14       end
15
16       events = REXML::Document.new element.to_s
17       events.elements.each("ClassInfo/Events/Event") do |event|
18         add_event(new_class, event)
19       end
20     end
21   end
22 end
```

Listing 11.12 displays a part of the XML file used in the framework. Each `ClassInfo` tag defines one Flex object. The `Event` tags is created as methods on the Ruby class so the developer might write: `object.change_focus`.

Listing 11.12: Example part of XML file

```
<TypeInformation>
  <ClassInfo Name="Object" Extends="DisplayObject">
  <Implementation Class="mx.core::UIComponent"/>
    <Events>
    <Event Name="ChangeFocus" >
      <Implementation Class="flash.events::FocusEvent"
            Type="keyFocusChange"/>
      <Property Name="shiftKey"  DefaultValue="false">
        <PropertyType Type="Boolean" />
      </Property>
      <Property Name="keyCode"  DefaultValue="TAB">
        <PropertyType Type="String" />
      </Property>
    </Event>
    </Events>
```

```
    <Properties>
    <Property Name="visible" ForVerification="true">
      <PropertyType Type="Boolean"/></Property>
    </Properties>
  </ClassInfo>
</TypeInformation
```

Add event

Listing 11.13 shows the method `add_event`, which creates a method by the name of an event type in Flex to the Ruby class that simulates the specific Flex control. The method that is created is in charge of calling the FunFX adapter (see Section 11.1) and invoking the event. It also handles the synchronization issues when a Flex application uses external data sources (see Section 11.2.3). Section 11.2.3 will explain in more detail how it handles the synchronization issue.

The properties added by the developer, called `options`, are merged together with the default values stated by the AutomationGenericEnv.xml.

Listing 11.13: Method for adding a new event method

```
1  def self.add_event(class_name,event)
2
3      event_class = event[1].attributes["Class"]
4      event_type = event[1].attributes["Type"]
5      method = event.attributes["Name"]
6
7      class_name.add_method(shift_case(method)) do
8        |*argv|
9        options = argv.shift || {}
10
11       properties = REXML::Document.new event.to_s
12
13       options.each_pair do |key, value|
14         properties.elements.each("Event/Property") do |property|
15           if(property.attributes["Name"] == shift_case(key.to_s,
                 true))
16             property.attributes["DefaultValue"] = value
17           end
18         end
19       end
20
21       if(options[:wait] != nil)
22         count = options[:wait]
23
24         new_event = flex_object.addevent(event_type, event_class,
               @id, properties.to_s)
25         while((new_event != nil || new_event == "Not synchronized")
               && count > 0)
26           sleep(1)
27           count = count - 1
28           new_event = flex_object.addevent(event_type, event_class,
                 @id, properties.to_s)
```

```
29          end
30        else
31          new_event = flex_object.addevent(event_type, event_class,
                @id, properties.to_s)
32          if(new_event == "Not synchronized")
33            count = $wait_sync
34            while(new_event == "Not synchronized" && count > 0)
35              sleep(1)
36              count = count -1
37              new_event = flex_object.addevent(event_type, event_class
                    , @id, properties.to_s)
38            end
39          end
40        end
41
42        raise new_event unless new_event == nil
43
44        is_sync
45
46        sleep(funfx.speed) # To make the interaction slow enough so
              people can watch
47      end
48    end
```

Tabular data

Many of the Flex display objects display information in a tabular way, e.g., `DataGrid` and `List`. Testers often want to be able to assert this information. When the parser hits such a control, it calls the method `add_tabular_method` that generates methods for extracting such information. This method is shown in Listing 11.14. Methods for working with these components are also added: `firstVisibleRow`, `lastVisibleRow`, `numRows`, `numColumns`, and `columnNames`. This makes it possible to assert the right values displayed in the component. The `tabular_data` method takes either zero, or two arguments. They are numbers, and state the starting row and the end row. The returned value is a comma separated list of the information displayed in the selected row.

Listing 11.14: Method for adding a new method for tabular values

```
1  def self.add_tabular_method(class_name)
2      class_name.add_method("tabular_data") do
3        |*argv|
4        options = argv.shift || {}
5
6        if(options[:wait] != nil)
7          count = options[:wait]
8
9          data = flex_object.getTabularData(@id, options[:start],
               options[:end])
10         while(data != nil && count > 0)
11           sleep(1)
12           count = count - 1
13           data = flex_object.getTabularData(@id, options[:start],
                 options[:end])
```

```
14          end
15        else
16          data = flex_object.getTabularData(@id, options[:start],
                options[:end])
17        end
18
19        return data
20      end
21
22      class_name.add_method("first_visible_row") do
23        flex_object.firstVisibleRow(@id)
24      end
25      .
26      .
27  end
```

Flex module

Listing 11.15 shows a part of the Flex module, and the method `self.generate_new_class`. This method creates a class with the name as the provided name, and extends the parent provided.

Listing 11.15: The Flex module

```
1  module Flex
2    def self.generate_new_class(name, parent)
3    if(parent != nil)
4      return const_set(name, Class.new(Flex.const_get(parent)))
5    else
6      return const_set(name, Class.new(Base))
7    end
8    end
9  end
```

Within the Flex module, there is created a class called `Base` that is the base class of all the dynamically created Flex classes. Listing 11.16 shows what this class looks like. Before a new Ruby object is created, the framework must check with the FunFX Flex adapter if it exists. This is due to the test-driven approach (see Section 3.1.2), where the developer needs to get information if the object does not exist. This is done with the help of the `check_component` method. If the object exists, a new Ruby object is created and the value returned becomes the object's id. This id provides the adapter with information to easily find the object.

With the `method_missing` (see Section 7.1.1), the class has the ability to return an arbitrary other instance of a class, or call the `checkProperty` method in the FunFX Flex adapter with an arbitrary property.

The `is_sync` method inquires the adapter if the Flex application is ready. The adapter will let the framework know if the application is working on some event or not, more on synchronization in Section 11.2.3.

Listing 11.16: The Base class which all the framework created Flex objects extends

```
1  class Base
2
3    def initialize(name, id)
4      raise ArgumentError if name == nil
5      raise ArgumentError if id == nil
6      @name = name
7      @id = id
8    end
9
10   def method_missing(method_name, name=nil)
11     is_sync
12     if(name != nil)
13       value = check_component(name)
14       if(value != false)
15         Flex.const_get(shift_case(method_name.to_s)).new(name, value
               )
16       else
17         nil
18       end
19     else
20       property = flex_object.checkProperty(@id.dup, shift_case(
             method_name.to_s,true))
21       if(property != nil)
22         property
23       else raise "No such property or method <#{method_name}> of
             class: #{self.class}"
24       end
25     end
26   end
27
28   #Adds a new method dynamically
29   def self.add_method(name, &block )
30     raise ArgumentError if name == nil
31     define_method(name, &block)
32   end
33 end
```

To make the tests more reliable, it is possible to use both absolute and relative paths as described in Section 11.1.3. Listing 11.17 shows how to write a test with absolute path, and Listing 11.18 shows how to use relative path. There are no difference in how to write the tests, other than that tests are shorter. A developer must be careful using extreme versions of both versions. The negative factor with the absolute path version, is that if the GUI is refactored without changing the functionality, the test will most likely break. When using the relative path, the developer must consider if it is possible to hit another object with the same name.

Listing 11.17: Using absolute path

```
1  @ie.application("app").panel("panel").button("bButton").click
```

Listing 11.18: Using relative path

```
1  @ie.button("bButton").click
```

11.2.2 FunFX class

The framework contains a class named `Funfx`, which is the entrance to the framework. This class acts as a façade class in front of the user (see Section 10.1.3). This is the entry point of the Flex application and it contains the Flex object[6], which the tests are performed upon. This class can be regarded as a root object in the display object hierarchy, and it extends the Base class in the Flex module. Figure 11.2 shows how the `Funfx` class is regarded as the root of the display hierarchy.

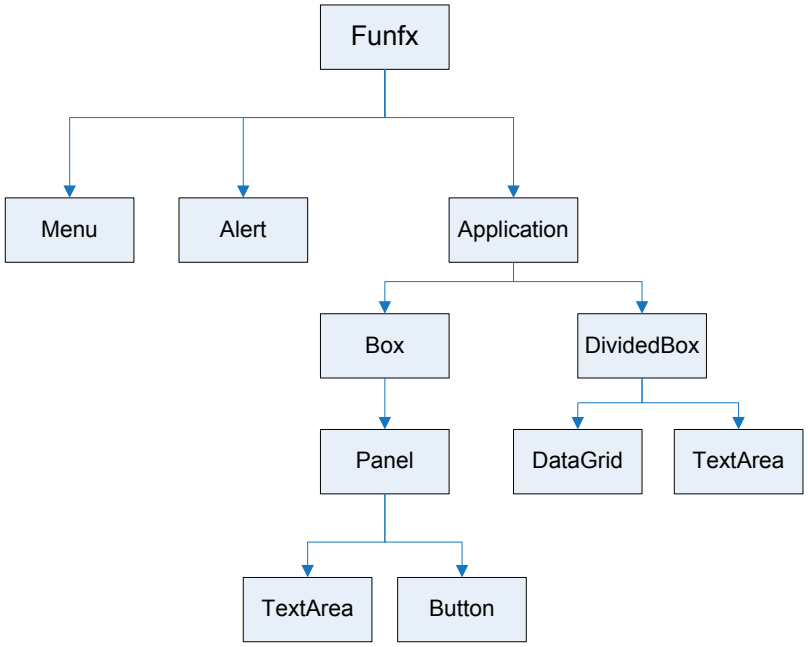

Figure 11.2: Example hierarchy of an Flex application and the relation to the Funfx class

It is also created as a factory class for the implementation for the desired browser. Currently, support is only implemented for Internet Explorer.

The FunFX class is implementing the singleton pattern, which makes it impossible to create more than one instance (see Section 10.1.4), as shown in Listing 11.19.

Listing 11.19: The factory class implements the singleton pattern

```
1 require 'singleton'
2 class Funfx
3   include Singleton
4
5   # More code
```

Listing 11.20 shows the `Funfx` class. It has the possibility to make the test visual and to change the speed of the visual impression. Parsing of the XML file happens in the initialize method. This way, the parsing time, a few seconds, only happens the first time. It implements a helper method that provides the ability to parse additional XML files.

[6]The Flash movie embedded in the HTML page

The `start` method creates an instance of the wanted browser, the default (and currently the only implemented) is Internet Explorer. The `goto` method, takes the address and the name of the embedded flash object as arguments.

Listing 11.20: The factory class

```
1  class Funfx < Base
2    include Singleton
3
4    attr_accessor :speed
5
6    def initialize
7      XmlParser.parse
8      @speed = 0.8
9    end
10
11   def pars_file(document)
12     XmlParser.parse_file(document)
13   end
14
15   def start(browser_type = "InternetExplorer")
16     @browser = Browsers.const_get(browser_type).new
17   end
18
19   def goto(address, name)
20     @browser.goto(address, name)
21   end
22
23   def visible(visible)
24     @browser.visible(visible)
25   end
26
27   def unload
28     @browser.unload
29   end
30
31   def get_flex_object
32     @browser.flex_object
33   end
34 end
```

The module Browsers is thought of as the container of the different browser implementations[7]. Listing 11.21 shows this module and a concentration of the methods[8], with the class implementing Internet Explorer. When adding support for another browser, the class has to implement the same methods. Due to the Duck Typing capabilities of Ruby, other browser classes can easily be implemented as long as they provide the same methods.

Because of problems during creation of Internet Explorer windows, a help function for reaching the flex object embedded into the HTML page was created. It will try to reach the element for a defined amount of time. If the time runs out an error is raised.

[7] Currently the only supported browser is Internet Explorer.
[8] Due to space limitations the methods `unload` and `navigate` were left out

Listing 11.21: The browser module

```
require 'win32ole'
module Browsers
  class InternetExplorer

    def initialize
      @ie = WIN32OLE.new('InternetExplorer.Application')
      @ie.visible = true
    end

    def goto(address, name)
      while(busy)
        sleep($sleep_busy)
      end
      navigate(address)
      set_flex_object(name)

      #-- Must sleep so IE is ready
      sleep($sleep_wait)
    end

    def set_flex_object(name)
      while(busy)
        sleep($sleep_busy)
      end

      @flex = help_set_flex_object(name)
      raise "No flash element by name #{name}" if @flex.item == nil
    end

    def help_set_flex_object(name, obj=nil)
      count = $count_set_flex
      obj = help_get_element(name)
      while(obj.item == nil && count > 0)
        sleep(1)
        count = count - 1
        obj = help_get_element(name)
      end
      return obj
    end

    # Helper method to extract the Flash/Flex object
    def help_get_element(name)
      begin
        return @ie.Document.getElementsByName(name)
      rescue
        return nil
      end
    end

    def flex_object
      @flex.item(0)
    end

```

```
54      def busy
55        begin
56          @ie.Busy
57        rescue
58          return true
59        end
60      end
61    end
62  end
```

11.2.3 Handling synchronization

The richness of a Flex application often requires the framework to wait until an operation finishes. To solve this synchronization problem, the method `is_sync` was created. This method inquires the FunFX Flex adapter to check whether the application is ready or still working on some operation. If the application is busy, it initiates a sleep command. The amount of time is stated in a configuration file called `conf.rb`. Listing 11.22 shows this method.

Listing 11.22: The `is_sync` method

```
1  def is_sync
2    while(!flex_object.isSynchronized(nil))
3      sleep($wait_for_synch)
4    end
5  end
```

Several Flex controls take input from a data provider such as an array or a XML object [18]. These XML objects are often retrieved from a Web service or another external source. This may cause a certain latency before the controls are fully initialized with data. The Flex framework and the automation package do not support any synchronization actions for such a scenario, according to Matt Horn at Adobe [5]. It only makes the developer aware if the control is initialized and visible, not if the information from the data provider has arrived. After considering the present framework together with BEKK Consulting AS (see Section 1.3), one important requirement of the framework is to support an ability to wait for the control's data to appear. Because Flex does not provide any method or information, this had to be done with an asynchronous loop, to check whether the information is reachable.

The framework is supposed to be used in TDD (see Section 3.1.2), the framework thus cannot be implemented with this ability as default. If the framework would wait for each control that has not been created, the tests would take too much time. Therefore a wait tag was added. The tag states that the framework should wait for the control to be fully initialized, but not more than the seconds stated in the tag.

Listing 11.23 shows a simple select command on a `DataGrid`, with a `:wait` argument for 10 seconds.

Listing 11.23: Enabling synchronization handling

```
1  @ie.data_grid("dgGrid").select(:item_renderer => "item", :wait =>
       10)
```

Listing 11.24 displays the code part that makes the framework wait until the wanted event is successful or that the wait time has run out. It first examines if the tag `:wait` is not nil. Then the code will run the event until it is successful or until the time stated by the developer in the wait tag is run out. If the time is run up and the event failed, an exception is raised upon which the developer must act.

Listing 11.24: Handling synchronization

```
1  if(options[:wait] != nil)
2    count = options[:wait]
3
4    new_event = flex_object.addevent(event_type, event_class, @id,
         properties.to_s)
5    while((new_event != nil || new_event == "Not synchronized")&&
         count > 0)
6      sleep(1)
7      count = count - 1
8      new_event = flex_object.addevent(event_type, event_class, @id,
           properties.to_s)
9    end
10 else
11   new_event = flex_object.addevent(event_type, event_class, @id,
         properties.to_s)
12   if(new_event == "Not synchronized")
13     count = $wait_sync
14     while(new_event == "Not synchronized" && count > 0)
15       sleep(1)
16       count = count -1
17       new_event = flex_object.addevent(event_type, event_class, @id,
             properties.to_s)
18     end
19   end
20 end
21
22 raise new_event unless new_event == nil
```

11.3 Creating tests

The goal of this thesis and framework is to be able to write tests. To do this, and to reuse as much functionality as possible, FunFXuses the Test::Unit (see Section 7.1.5) framework to create and run tests.

Listing 11.25 shows a simple test-case with the `setup` and `teardown` method together with one test. Since the Flex objects are parents to each other, the developer might go through several objects before an action can be performed on the desired object, as seen with the last action on line 32. But if the id of all objects has been used, it is possible to use relative path. The developer must ensure that the test reaches the desired object and not another object with the same name.

Listing 11.25: A test-case using Test::Unit

```
1  require 'test/unit'
2  require 'funfx'
3
4  class ExampleTest < Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
8      @ie.start
9      @ie.goto("http://localhost/Automation.html", "Automation")
10
11     @test_text = "This is a test"
12   end
13
14   def teardown
15     @ie.unload
16   end
17
18   def test_divers
19     @ie.button("bPress").click
20     assert_equal("Press", @ie.label("lText").text)
21
22     @ie.combo_box("cbCombo").select(:item_renderer => "Test3")
23     assert_equal("Test3", @ie.combo_box("cbCombo").text)
24
25     @ie.check_box("chBox").click
26     assert_equal("true", @ie.check_box("chBox").selected)
27
28     @ie.text_area("tText").input(:text => @test_text)
29     assert_equal(@test_text, @ie.text_area("tText").text)
30
31     @ie.panel("pPanel").button("bOk").click
32     assert_equal(@test_text, @ie.label("lTextBoxText").text)
33   end
34 end
```

The tests can be run with either the regular console test runner or the graphical test runner, which is provided in Eclipse among other IDEs. Figure 11.3 shows a simple test when passed and failed. When the test passes the test runner does not give the developer any more information other than a green bar, but when it fails it both turns red and provides information about what went wrong.

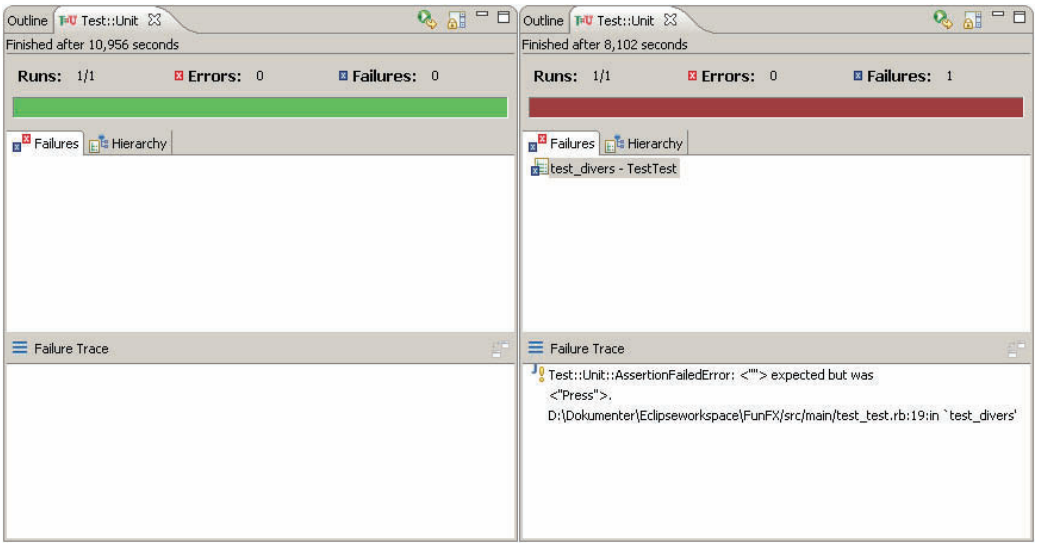

Figure 11.3: Test run with `Test::Unit::TestRunner`

PART V

Evaluation of FunFX

CHAPTER 12

Testing of FunFX

This chapter will elaborate on the testing done during and after development of the FunFX framework. It will try to show how the testing was done, and what parts were covered by the testing.

12.1 Development

TDD (see Section 3.1.2) was used during development. This means that the unit tests are written before writing any code. This ensures that all code is covered by tests.

Listing 12.1 shows an abstract of the test-case for the parsing of the XML file and the dynamic generation of classes.

Listing 12.1: An abstract of the Unit tests for creating Flex classes

```
1  class BaseTest < Test::Unit::TestCase
2    include FlexMock::TestCase
3
4    def setup
5      @parent_name = "FlexObject"
6      @child_name = "FlexButton"
7    end
8
9    def test_generate_class
10     assert_raise(NameError){
11       Flex::TestClass
12     }
13     Flex.generate_new_class("TestClass")
14     assert_not_nil(Flex::TestClass)
15
16     assert_raise(NameError){
17       Flex::TestClassChild
18     }
```

```
19      Flex.generate_new_class("TestClassChild", "TestClass")
20      assert_not_nil(Flex::TestClassChild)
21      assert_equal(Flex::TestClassChild.superclass, Flex::TestClass)
22    end
23
24    def test_a_parse
25      assert_raise(NameError){
26        Flex::FlexDisplayObject
27      }
28      XmlParser.parse("AutomationGenericEnv.xml")
29      assert_not_nil(Flex::FlexDisplayObject)
30    end
31
32    .
33    .
34
35  end
```

12.2 Test of functional requirements

When the implementation was finished, the functional requirements of the framework were tested. A simple Flex project was created, with some functionality implemented and some functionality missing. The following list describes how the functional requirements listed in Table 12.1 were tested.

F1 - The framework must be able to interact with Internet Explorer
This requirement was tested and reached with several of the following tests.

F2 - The framework must be able to define and run tests
A test was created with the existing Test::Unit framework and then ran. The test ran successful.

F3 - The framework must provide results from the tests
The framework utilizes existing test runners, and does provide results if the test fails. When errors happen on the Flex side, the adapter provides expressive messages that will enable the developer to track down the error.

F4 - The framework must be able to perform actions on a Flex application
A test that stated a button click where created and run, and the result was that the button was clicked.

F5 - The framework must provide the possibility of test suites
This requirement is covered by the Test::Unit, which is the test runner used during development. It is also possible to use other Ruby unit frameworks.

F6 - The framework must be able to reach Flex objects by name, id, or label
The button click test was changed to use the button's name, id, and label in turn. In all cases the button was clicked.

F7 - The framework must be able to make assertions on Flex objects
The script was changed to make an assertion if the button was visible. Then an assertion that a label's text was turned into some text after clicking the button was added. First the test failed, since this feature was not yet implemented. A label was added and a method

that made the label show a text when the button was clicked. When running the test again it was successful.

F8 - The framework must be able to wait for slow data providers
For this, a Web service was created with the help of Ruby on Rails, and made it wait for 10 seconds before it sent any information to the call. The test for selecting a row in a datagrid was written, and after the test failed, a datagrid and a data provider talking to the Web service was added. Without a wait command (see Section 11.2.3) in the test phrase the test failed, but with a wait command the test waited as expected and passed.

F9 - The framework must be able to both run the steps visually and not visually
When initializing the framework, one option is to set the visibility to false. The visibility was set to false, and the button test was run again. The browser was not visible and the test passed.

F10 - The framework must be able to have different speeds of interaction
A developer can at any time set the pace of the framework, this can be done at any time, if some tests should go slower than others. With a faster speed, the button click test saved 2 seconds.

F11 - The framework must be able to set paths in the tests for better readability
Due to the hierarchy of objects in Flex, the test phrases can be long and difficult to follow. Since every thing in Ruby and also in this framework are objects, you can easily set a variable to be an instance of an object and use this over and over again.

F12 - The framework must be able to take advantage of regular unit test runners
The framework is built as a standalone framework, and can be used together with any Ruby test runner.

Table 12.1: Functional requirements

ID	Description	Result
F1	The framework must be able to interact with Internet Explorer	Pass
F2	The framework must be able to define and run tests	Pass
F3	The framework must provide results from the tests	Pass
F4	The framework must be able to perform actions on a Flex application	Pass
F5	The framework must provide the possibility of test suites	Pass
F6	The framework must be able to reach Flex objects by name, id, or label	Pass
F7	The framework must be able to make assertions on Flex objects	Pass
F8	The framework must be able to wait for slow data providers	Pass
F9	The framework must be able to both run the steps visually and not visually	Pass
F10	The framework must be able to have different speeds of interaction	Pass
F11	The framework must be able to set paths in the tests for better readability	Pass
F12	The framework must be able to take advantage of regular unit test runners	Pass

12.2.1 Summary

All functional requirements have been tested and reached through both unit testing and functional testing.

12.3 Test of non-functional requirements

This section will describe how well the implementation concurs with the non-functional requirements (see Section 9.4).

Usability

Q1.1 - Give user informative response after test is finished - Implemented Whenever a test fails, the user gets useful information on what went wrong. This feature is implemented with the reuse of `Test::Unit` together with informative messages sent from the FunFX Flex Adapter.

Testability

Q2.1 - Test a new feature - Implemented All the existing code is tested, so a developer only needs to test the extra functionality he/she adds.

Modifiability

Q3.1 - Add a new component - Implemented A new custom component will only be needed to be added to the AutomationGenericEnv.xml file.

Q3.2 - Add support for other browsers - Not implemented This has not been implemented, but the framework has been designed so that such a change is easy to do.

Availability

Q4.1 - Unable to reach the Internet Explorer browser -Implemented The *InternetExplorer* class implements a wait function if the browser is busy or not yet initialized.

12.3.1 Summary

Even though not all of the non-functional requirements have been implemented, the framework has been implemented with the non-functional requirements in mind. Implementation of the non-functional requirements should be done without problems.

12.4 Usage testing

As an extra test of the implementation of FunFX and to point out the usability of this framework, a usage test was arranged. This section will describe how this usage test was performed and what the results were.

The test group was composed of two potential users of the framework, Erlend Oftedal and Christian Schwarz. The session started with a demonstration of how to use the framework. After a bit of discussion, the session went on to look at how things were implemented.

After a quick view on the hierarchy of objects in the tests, Erlend pointed out that this might be a show stopper if a button was moved into a box[1]. Options for using a relative path or a find object method were requested. This request was met, and the current implementation supports the use of relative path. They liked the way the framework used an existing test runner.

One thing they mentioned as a positive factor was FunFX's similarity to Watir (see Section 6.1), but they felt it was a bit confusing with the capital letters of the *event methods*. They would prefer to follow Ruby conventions and write `@ie.button("button").click` rather than `@ie.Button("button").Click`. This requirement was met and the final implementation of FunFX uses the Ruby conventions.

They felt its usability was high due to the similarity to Watir. There was a familiarity when writing the tests.

They saw potential benefits with such a tool, and pointed out that the ability to test the functionality of Flex applications is needed.

Concerning further work, they suggested writing an exporter to the CubicTest framework, which is a tool for making testing of Web applications easier for non-technical personnel [31].

12.4.1 Summary

The usage test discovered that the framework was on the right track, and it was not too difficult to use. A lot of improvements were suggested, but they were more of *nice to have* than *must have*. In a summary, the participants were impressed by the framework, and were excited to see if it was possible to build a community around it. Some of the improvements were implemented to make it a better product.

[1] Flex builds the GUI with different display objects, containing each other (see Section5.1)

CHAPTER 13

Implementation issues

This chapter will outline some of the problems that were encountered during this master project. It will try to describe how and why the problems did arise, and how they were overcome.

13.1 Access a flash movie

A Flash file is a compiled file, which cannot be edited. The Flash player is in a proprietary format, and this makes it difficult to interact with the display components compiled into the movie[1]. There are not much information concerning how to access a display component in a Flex application to perform actions directly on the component. The only way into a Flash movie appeared to be with the ExternalInterface API (see Section 5.2.2). With this API, it is possible to call ActionScript methods within a Flash movie.

The solution was to build a generic structure of reachable methods with the help from the ExternalInterface within the adapter. These methods make it possible for the Ruby tests written with the framework, to access and perform actions on the graphical objects directly.

13.2 Synchronization

Whenever applications shall cooperate with other types of applications, there is always a synchronization problem. This project was no exception. A special consideration had to be made because the framework will be supporting TDD (see Section 3.1.2). This, because the tests will be run without the real implementation available. It was not desirable that tests waited forever for a service that were not yet implemented, but on the other side it should wait for a slow implemented service. A timeout function was provided to be able to support both worlds.

[1] A Flash file is often called a Flash movie

The solution was to make the developer define a wait command that specifies the amount of seconds the test should wait for a single event. This option keeps the control of the testing in the hands of the developer.

13.3 Internet Explorer error

Problems with Internet Explorer were experienced during development. Whenever a test suite contained more than 19 test cases, Internet Explorer raised an exception. This problem is unacceptable considering that development projects are getting larger and larger. An investigation of the problem was initiated, and it was soon linked to memory issues. It seemed that the Ruby test suite created Internet Explorer windows too fast, and that the memory consumption was huge. Figure 13.1 shows the memory consumption of Internet Explorer. The window on the left was captured a few seconds before the error was raised. After 18 test cases, the memory consumption was over 300MB. The problem was that a new browser was created before the old one was destroyed and removed from memory. This made the memory consumption increase for each window.

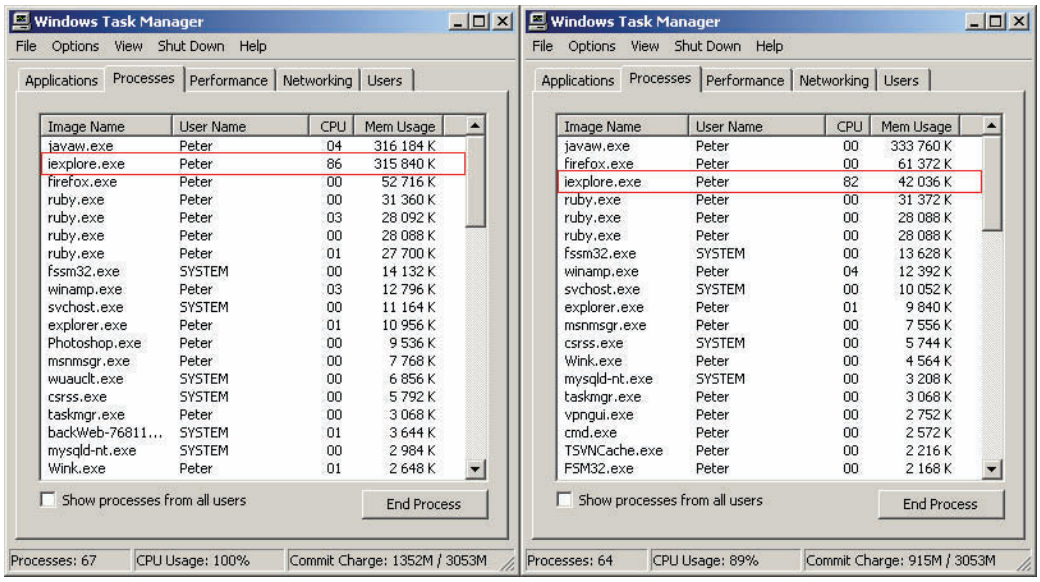

Figure 13.1: Internet Explorer memory usage. The left hand side shows the memory consumption during no sleep.

The solution was to slow down the creation of browsers. A sleep command was added to the initialize method that creates the Internet Explorer window. The amount of seconds it waits is set with a global variable in the `conf.rb` file. The default value is set to two seconds. The window to the right in Figure 13.1 shows the memory consumption after 22 test cases with the solution implemented. The negative aspect with this solution is that the entire test suite will be more time consuming.

Listing 13.1 shows how this solution was implemented in the `initialize` method of the class `InternetExplorer`. It uses the global variable named `$sleep_between_tests` in the `conf.rb` configuration file.

Listing 13.1: Slow down Internet Explorers memory consumption

```
1 $sleep_between_tests = 2 # From the conf.rb file
2
3 def initialize(visible)
4   sleep($sleep_between_tests)
5   @ie = WIN32OLE.new('InternetExplorer.Application')
6   @ie.visible = visible
7 end
```

13.3.1 Editing a single AutomationGenericEnv.xml

This project wanted to build FunFX in the way that only a library file needed to be added to the Flex project. With Flash's security sandox it is not possible to access the local file system. This creates a problem when congiguring the AutomationGenericEnv.xml file. With the current version of FunFX, it is required that the AutomationGenericEnv.xml file is added to all projects utilizing the framework. It should be possible to only use one file that is located a single place where multiple projects can reach it. Because of time limitations, this project did not solve this problem.

CHAPTER 14

Evaluation

This chapter will evaluate the results and answer the research questions.

14.1 Comparison to state of the art

The goal of this project was to create an open source framework that was able to functionally test Flex applications. In the prestudy and as a part of the engineering method (see Section 2.2), the most popular testing tools available for Web applications were inspected. Table 14.1 lists all tools including FunFX, and shows that the goal was reached. Watir and QTP were used as the models in the engineering method, and the framework was created after those tools. Watir and FunFX are especially similar. In a project using both HTML and Flex, it would be easy to use Watir and FunFX side by side. Both tools use Ruby and `WIN32OLE` to drive Internet Explorer, and the test commands are very much alike, there are only small differences in the practice of adding arguments to the commands.

Table 14.1: Comparison of FunFX with the state of the art tools

Tool	**HTML support**	**Flex support**	**TDD support**	**Programmatic browser interaction**	**Free**	**OSS**
Watir	Yes	No	Yes	Yes	Yes	Yes
Selenium	Yes	No	Yes	Yes	Yes	Yes
Mercury QTP	Yes	Yes	No	Yes	No	No
FitNesse	No	No	Yes	*	Yes	Yes
TestComplete	Yes	Yes	No	No	No	No
FunFX	**No**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**

14.2 Answer to research questions

This section will answer the research questions presented in Section 2.1.

14.2.1 Research Question - RQ1

What are the possible benefits of using the framework in development?

There are several benefits of using FunFX in development. If the developers use this tool consistently during development, and the tests are run with a continuous integration system, the project will constantly be tested for functional errors. This will be a powerful regression test suite.

Another option that functional testing tools are often used for, is to script a path into a specific part of the application. This way, the developer does not need to click his way through the application before he gets to the part he wants to take a closer look at. This is a time saving ability that can be invaluable during development.

An automated test suite of 19 tests, takes about 3 minutes, depending on the content. If these tests were done manually they would have taken much more time.

14.2.2 Research Question - RQ2

What kind of entry level does the framework impose on the user?

The objective of this project was to build a framework for functional testing of Flex applications. Functional testing is often thought of as done by non-programmers and customers, but it is equally important that the developers create these functional tests during development. Because of this, and the popularity of test-driven development (TDD) (see Section 3.1.2), it was decided to build a tool for namely the developers.

The tests are written as regular Unit tests (see Section 4.3.2), with any existing Unit testing framework for Ruby. Due to this, the user should have a simple knowledge of Ruby. The fact that Flex lays out its display components in a hierarchy is also one thing the user has to have in mind when creating tests.

During the usage test, Erlend and Christian expressed that the framework seemed easy to use. They also noticed the similarity to Watir[1]. FunFX will benefit from the popularity of Watir as a functional testing tool of HTML Web applications.

14.2.3 Research Question - RQ3

How well does the framework support regression testing?

Regression testing (see Section 4.3.4) means that earlier created code is checked whether it still works after changes in other parts of the software. With TDD, unit tests works as a regression test suite for all low level code. The FunFX framework is meant to be used with TDD, and thus the functional tests will act as a regression test suite for the functionality of the application.

[1] Watir has been used as an existing solution in the engineering method

14.2.4 Research Question - RQ4

How does the framework inform the user where errors happen? And how well does this information help the user to correct the errors?

During testing in general, it is important to find out where an error happened. This information should be easy to see and understand. The adapter provides the framework with intuitive messages about what went wrong in the Flex application. With these messages, the framework raises exceptions that are caught by the test runner. Listing 14.2 shows a result after the test shown in Listing 14.1 failed. In this example, the test failed when trying to select an element that did not exist in the datagrid. The error message points out on what line the error happened. In this example, it was line 2 in the file `sync_test` (displayed in Listing 14.1) .

Listing 14.1: Test method selecting an element in a DataGrid

```
1 def test_control
2   @path.data_grid("dg").select(:item_renderer => "nr3")
3 end
```

Listing 14.2: Error message posted by Test::Unit::TestRunner

```
Finished in 34.78 seconds.

1) Error:
test_control(SyncTest):
RuntimeError: No such itemRenderer <blognr3> on the object <dgBlog>
C:/Ruby/lib/ruby/gems/FunFX-0.0.1/lib/xml_parser.rb:143:in `Select'
D:/Eclipseworkspace/flex-object-test/sync_test.rb:2:in `test_control
    '

1 tests, 1 assertions, 0 failures, 1 errors
```

The usage test pointed out that there were informative error messages, and it was easy to find the error.

14.2.5 Research Question - RQ5

What are the positive effects with open source software? Does the fact that FunFX is supposed to be an open source framework, put any constraints on the implementation?

An open source tool provides many advantages over a commercial tool. Being free (as in free beer), the tool is available to everybody in the project at any time, imposing no restrictions with respect to acquirement and usage. A successful open source tool also comes with a great community, providing free access to experience and knowledge (community sites, mailing lists, forums, blogs). Additionally, due to the nature of open source, a lot of the open source tools come with extensions, custom additions and plug-ins. The best example are the various flavours, available tool support and integration of test tools in the xUnit family, the current de facto industry standard for unit testing. Similarly, open source test tools like Selenium (see Section 6.2) and Watir (see Section

6.1) enjoy the highest mind- and market-share in Web application testing. Finally, the ultimate advantage is free access to the source code, being able to change and adjust the test tool to suit particular requirements.

Adobe has announced that they will release Flex under a Mozilla license (see Section 5), and thus it might be easier to release open source Flex projects. But the automation package will not be released and the users of this framework will still need to buy an FDS license from Adobe.

14.2.6 Research Question - RQ6

How well did the research methods (see Section 2.2) and the development methods (see Section 3.1) help during this project?

The engineering method (see Section 2.2) was used during this project. Basili describes it as: *in the engineering method the developers observe existing solutions, propose a new and better solution, improve the solution, and repeat until no further improvement is needed* [10]. After searching for functional testing tools for Web applications, Mercury Quick Test Professional (QTP) (see Section 6.3) and Watir (see Section 6.1) were chosen as models, and this research is based on them. Because QTP is not open source, it was not possible to see how the tool was built, but valuable information was extracted about how it wrote tests. Watir was chosen because it is an open source testing tool written in Ruby. This gave valuable guidance on how to build the framework. The final framework became a mix of QTP and Watir.

During development, test-driven development (TDD) (see Section 3.1.2) together with a scaled down version of the Unified Process (UP) (see Section 3.1.1) were used. The TDD worked really well during development, it helped keep code consistent while refactoring. In spite of the intent to use a scaled down version of UP, the development method tended to be more agile. This was due to the lack of experience with both the Flex automation package (see Section 5.4.1) and the creation of frameworks. To predict how things worked was difficult, and the use cases were to some extent valuable, but in reality the code was created on the go.

Even though we were not able to follow the intended development method, we believe that a more agile approach helped us during development. And the engineering method made an important role in the whole process of this project.

CHAPTER 15

Summary

This chapter summarizes the results of this project, and proposes further work with FunFX.

15.1 Conclusion

The goal of this master thesis was to develop an open source framework for functional testing of Flex applications. When designing the framework, special attention was paid to the ability to support test-driven development (TDD) (see Section 3.1.2). The objective was to build a tool that could be used by the developer during implementation. Since Adobe announced during the project period that they were going to release Flex under a Mozilla license, this motivated the development of an open source project even more.

The problem at hand when commencing this project was how to reach the Flex application within the Flash object that is embedded in a HTML page. The `WIN32OLE` library of Ruby was used to access the Flash object containing the Flex application, and the ExternalInterface API of Flash to act upon the Flex application's display objects. An adapter was made that handles these ExternalInterface methods, and work together with the Automation Package to replay events on the Flex display objects. The adapter is easily added as a library file to a Flex project enabling the Ruby implemented framework to perform and assert actions. With the use of Ruby, the adapter, and `Win32OLE`, the framework has the ability to interact with the display objects rather than coordinates. This makes the tests more enduring, and will not break if a display object is moved to another position in the application.

The usage test provided more intuitive and durable solutions, and at the same time some valuable information about the usability. The use of Watir (see Section 6.1) as a model, was a positive factor since it seemed familiar.

We feel Flex have been a great technology to work with, and will most likely continue using Flex in the future. We believe that the use of Rich Internet Applications (RIA) (see Section 4.1) like Flex will be even more used in the near future of software development.

If more crucial development projects are done with Flex, the need for a free solution for functional testing will be inevitable. And if the work continues on FunFX, we believe that it has the potential to become an important part of a Flex development cycle.

15.2 Further work

Although the FunFX framework works without any appearing problems, the framework is still in an early phase. For it to be a mature version, there are still a lot of work to be done. The current version might be thought of as an early beta, and will need more thorough testing. The following explains in more detail what we believe are the most important parts to consider for further work.

15.2.1 Editing a single AutomationGenericEnv.xml

Due to Flex is built upon Flash, and that Flash uses a security sandbox, it is not possible to access the filesystem. There is one exception, and that is when uploading files. This creates a problem when congiguring the AutomationGenericEnv.xml file. Because of time limitations, this project did not solve this problem. With the current version of FunFX, it is required that the AutomationGenericEnv.xml file is added to all projects utilizing the framework. It should be possible to only use one file that is located a single place where multiple projects can reach it. This is something that needs more work on the current implementation.

15.2.2 Release it as open source project

For an open source project to survive, it is important to build a community around the project. The community does not need to consist of many people, but they should be dedicated. A release of an open source project might not only be a fast way to improve software, but it is also by far the cheapest way. Letting people use it for free, will be a fast way to figure out if the tool is worth to maintain.

FunFX is a project that should be made available to the public. After Adobe announced that they will release Flex under a Mozilla license, the popularity of such applications might grow. To begin with, the project should be released on a smaller site than SourceForge, maybe on OpenQA[1] or BOSS[2].

15.2.3 Recording tool

This project sets out to implement a functional testing tool that supports TDD (see Section 3.1.2). This makes FunFX a developer's tool. In the recent years, many companies are trying to push some of the work with writing tests toward the customer. This is because it is the customer that needs to express what he/she expects, and to make sure that the software does what it is supposed to do. The customers are rarely able to write tests, and thus a recording tool might be a valuable addition to the framework. The adapter implements a recording ability, and the only thing that is missing is a way to publish these

[1] http://www.openqa.org/
[2] BEKK Open Source Software, http://boss.bekk.no/

recordings to a recording tool. The recording tool might be written in Flex and use FDS to publish the recorded tests.

15.2.4 Make tests continue if one part fails

After considering the framework with several employees at BEKK Consulting AS, they pointed out a weak point in every unit testing framework. This weak point is that the test stops after the first error happens in the test method. In projects done by BEKK at the moment, they often have to write their own *try-catch* sentences to avoid the test to break. They want to be able to print a line when a test fails, and summarize all of these errors when all tests are done.

This is more a problem of the test runner, and not of the FunFX framework. But it should be possible to write a higher level test case class that implements such a functionality.

15.2.5 Exporter CubicTest

CubicTest is an open source Eclipse plug-in that tries to make testing Web applications easier to design, understand and run for nontechnical as well as technical users. CubicTest uses a graphical user interface to let users model tests instead of writing test scripts. The tool focuses on enabling test-driven development of Web applications, but also supports testing of existing Web applications. The goal is to make it possible to replace a detailed requirement specification and manual test scripts with tests designed in CubicTest [31]. CubicTest supports testing of Web application with Watir (see Section 6.1), and exports Watir test code from the graphical user interface. As a further work, it might be interesting to build such an exporter for FunFX in CubicTest. This would make the already known CubicTest better, and thus promote FunFX in a good way.

15.2.6 Client for editing configuration file

FunFX uses a configuration file called AutomationGenericEnv.xml to build the class hierarchy of Flex display objects. If a new custom component is introduced this file is the only thing that needs to be altered. To prevent wrong altering of the configuration file, it might be handy to create a client that makes editing easy and safe[3]. This could be a simple Flex application that requires some input about the component, and then updates the configuration file accordingly.

[3]Nothing can go wrong other than an error message, or a test script that does not respond properly

PART VI

Appendix

APPENDIX A

Questionnaire

1. The degree of usability? (1 is not good, 5 is very good)

2. What do you think about the way to reach a specific object in the hierarchy? (`@ie.apllication("app").panel("panel").button("button").click`) **Was it difficult to understand how to reach objects?**

3. What were the positive aspects with the framework?

4. What was missing? What kind of improvements are needed?

5. Do you think development of Flex application will benefit of such a tool (a test-driven functional testing tool)?

6. Did you run into any problems? If yes, please describe the problems.

7. Is this an application you might use during development?

APPENDIX B

User guide

This chapter will act as a recipe on how to use the framework during development. It will describe best practices, and try to show how a developer can utilize this tool.

B.1 What do you need?

First of all, you will need Flex Builder to build a Flex application. You will also need Ruby to be able to run the tests. The other parts that are required for the framework to operate are:

Automation package You must have the automation package installed, and include the files: *automation.swc*, *automation_agent.swc*, and *automation_agent_rb.swc*.

FunFX Flex adapter The FunFX adapter library file must also be included as a library.

AutomationGenEnvironment.xml The XML file describing the automation environment. This file must be added to the project.

FunFX framework To be able to write tests and interact with the FunFX adapter and make the Flex application move, you need the framework implemented in Ruby. How to install is described in more detail in Section B.2.

B.2 Install FunFX

The FunFX framework is distributed as a gem, and is installed by the command *gem install "name-of-gem"*. To see if the framework has been installed properly, you can type *gem list –local*. The list should hold a gem called FunFX.

B.3 Include FunFX library

The FunFX adapter is a Flex library file and must be compiled together with the Flex application. To do this, the FunFXAdapter.swc file must be included as a library to the Flex project. Listing B.1 shows how one can include a library using the *flex-config.xml* file for this. This way makes every Flex project add this file to it's library. The automation files *automation.swc* and *automation_agent.swc* must be included in the same way to utilize the automation package (see Section 5.4.1).

The file *automation_agent_rb.swc* must be added to the folder named locale under the Flex 2 SDK folder.

Listing B.1: How to include the library file

```
<include-libraries>
  <library>/libs/automation_agent.swc</library>
  <library>/libs/automation.swc</library>
  <library>/libs/FunFXAdapter.swc</library>
</include-libraries>
```

If it should only apply for a single Flex project, the compiler option *-include-libraries "path-to-file"* can be added in project properties in Eclipse or Flex Builder 2. Figure B.1 shows how to add a file with the *-include-libraries* tag.

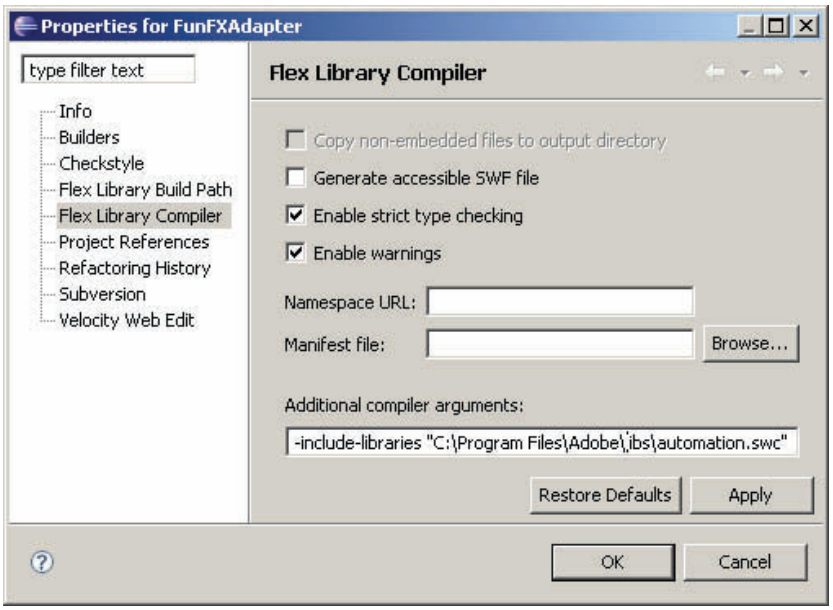

Figure B.1: Include library file

B.4 Writing testable Flex applications

When developing Flex applications that will be tested with FunFX, there are a few aspects to consider. Tests should be understandable and display objects in the Flex application

must provide descriptive names and ids. Using the id on every object, makes it easier to use relative path in the tests, which is a faster way to develop tests.

B.5 Writing tests

Tests are written in Ruby. The only requirement to be able to programmatically drive a Flex application is the FunFX Framework (see Section B.2). But to be able to write tests and assert the results of some functionality, a unit framework is needed. Any unit framework for Ruby works, but this example will use Test::Unit.

B.5.1 Initializing and tear down of the framework

The FunFX framework is built as a singleton, and must be accessed by `Funfx.instance`. The `start` method creates a new instance of Internet Explorer. As an argument to the start method, it is possible to set if the browser should be visible. The default value is `false`. The method `goto` navigates the browser to the provided address and will make the framework ready for accessing the Flash/Flex object by the name provided embedded on the HTML page. Listing B.2 shows how the framework is initialized in the `setup` method of the test case. When using `Test::Unit`, this method will be run before all test methods specified in the class. If the methods use different locations or different names of the Flash/Flex object, the goto method might be moved down into the test method.

Listing B.2: Initializing the framework

```
1  require 'test/unit'
2  require 'funfx'
3
4  class ControlTest << Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
8      @ie.start
9      @ie.goto("localhost/flex/control.html", "name-of-object")
10   end
11
12 end
```

To avoid Internet Explorer to continue to be open, a method called `unload` is used to destroy the instance. This method is used in the `teardown` method of the test case. This ensures that the browser is shutdown after the test method is done. Listing B.3 shows this method added to the test case.

Listing B.3: Destroy the Internet Explorer instance

```
1  require 'test/unit'
2  require 'funfx'
3
4  class ControlTest << Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
```

```
8       @ie.start
9       @ie.goto("localhost/flex/control.html", "name-of-object")
10    end
11
12    def teardown
13      @ie.unload
14    end
15  end
```

B.5.2 Test methods

When using Test::Unit, all test methods must begin with the word `test` because it uses reflection to detect all available test methods. Listing B.4 shows an empty test method called `test_control`. Within this method, actions and assertions are added. An action is any command that performs an action on the Flex application, while an assertion checks whether a value is correct or not.

Listing B.4: A test method defined

```
1  require 'test/unit'
2  require 'funfx'
3
4  class ControlTest << Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
8      @ie.start
9      @ie.goto("localhost/flex/control.html", "name-of-object")
10   end
11
12   def teardown
13     @ie.unload
14   end
15
16   def test_control
17
18   end
19
20 end
```

Figure B.2 shows an example hierarchy of a simple Flex application. The root node is the instance of `Funfx`. When accessing the display objects, the instance of `Funfx` is the starting point. The framework uses a breadth-first search, which enables a mix of absolute and relative paths. Listing B.5 shows two ways of accessing the button named "name". With the first alternative, it is important to ensure that no other Button by the same name exists. With the second option, the test might break if the GUI is modified without changing the functionality.

Listing B.5: Two ways of accessing an object

```
1  @ie.button("name")
2  @ie.application("aname").box("bname").panel("pname").button("name")
```

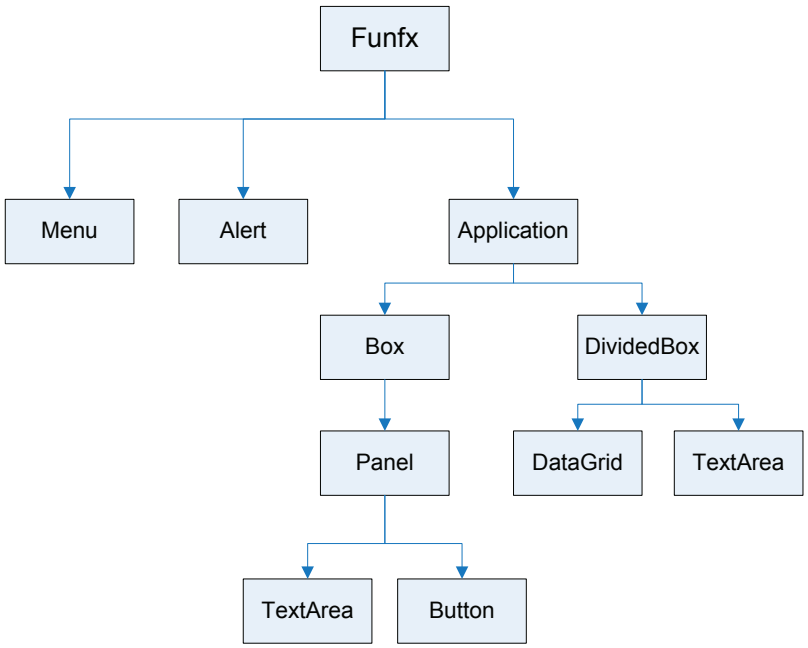

Figure B.2: Example hierarchy of an Flex application

To perform actions on the display objects, like the button in the above example, the objects provide methods corresponding to the events displayed in Section B.6. The events that drive the Flex application might require some arguments, these are also listed together with the events in Section B.6. Listing B.6 shows how to use the methods corresponding to the Flex events. The click event has no arguments. When inputing text in a TextArea, the text must be provided as an argument. The way to do this is with the use of a Ruby symbol that is the name of the argument followed by the value. Notice that FunFX follows Ruby conventions, and writes all names with lower case letters, and divides names with an underscore.

Flex uses something called `itemRenderers` that display parts of some display objects like DataGrids. When selecting a row in a DataGrid the `itemRenderer` must be provided. The String value of the `itemRenderer` is the text in any of the columns, but it must be unambiguous.

Listing B.6: Perform action on the display objects

```
1 @ie.button("name").click
2 @ie.text_area("somename").input(:text => "This is some text")
3 @ie.data_grid("grid").select(:item_renderer => "value")
```

Listing B.7 shows a complete test. This test adds text into two TextAreas and chooses an option in a ComboBox. Then it clicks a button, and selects the value corresponding to the value entered in the TextArea to what is in the DataGrid. Then it asserts that the information is correct. It checks that the information entered has appeared in the DataGrid.

The next thing is to drag the newly entered item to another DataGrid. It does this with the method `drag_start` with the argument `dragged_item`. The value of the `dragged_item` is an unambiguous `String` from one of the columns stating what row to drag. Another argument often used together with drag and drop is `action`. If using `:action ⇒ "copy"`, it will only copy the row, if using `move`, it will move the row from one DataGrid to another. When dropping the element, the only thing necessary is to state a `drag_drop` on the target. The last lines are there to ensure that the correct row has been dragged.

Listing B.7: The test complete

```
1  require 'test/unit'
2  require 'funfx'
3
4  class ControlTest << Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
8      @ie.start
9      @ie.goto("localhost/flex/control.html", "name-of-object")
10   end
11
12   def teardown
13     @ie.unload
14   end
15
16   def test_control
17     @fname = "James"
18     @lname = "Bond"
19     @location = "London"
20
21     @ie.text_area("tFirstName").input(:text => @fname)
22     @ie.text_area("tLastName").input(:text => @lname)
23     @ie.combo_box("cbLocation").select(:item_renderer => @location)
24     @ie.button("bAddEmployee").click
25
26     @grid = @ie.data_grid("dgEmployees")
27     @grid.select(:item_renderer => @lname)
28     row = @grid.selected_index
29     assert_equal("#@fname,#@lname,#@location", @grid.tabular_data(:
           start => row, :end => row))
30
31     @grid.drag_start(:dragged_item => @lname)
32     @drop_grid = @ie.data_grid("dgEngineeringGroup")
33     @drop_grid.drag_drop
34
35     @drop_grid.select(:item_renderer => @lname)
36     row = @drop_grid.selected_index
37     assert_equal("#@fname,#@lname,#@location", @drop_grid.
           tabular_data(:start => row, :end => row))
38   end
39
40 end
```

B.6 Automation environment

This section will show all display objects and their events.

Name	**Extends**
DisplayObject	
Events	**Event properties**
MouseMove	localX(int) localY(int) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Click	ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
Object	**DisplayObject**
Events	**Event properties**
ChangeFocus	shiftKey(Boolean) keyCode(String)

Name	**Extends**
Container	**Object**
Events	**Event properties**
MouseScroll	delta(int)
Scroll	position(int) direction(String) detail(String)
DragStart	draggedItem(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
DragDrop	action(String) draggedItem(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
DragCancel	ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
Form	**Container**
Events	**Event properties**

Name	**Extends**
Canvas	**Container**
Events	**Event properties**

Name	**Extends**
Box	**Container**
Events	**Event properties**

Name	**Extends**
ProgressBar	**Object**
Events	**Event properties**

Name	**Extends**
Accordion	**Container**
Events	**Event properties**
Change	relatedObject(String)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
Button	**Object**
Events	**Event properties**
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
PopUpButton	**Button**
Events	**Event properties**
Open	triggerEvent(Event)
Close	triggerEvent(Event)

Name	**Extends**
CheckBox	**Button**
Events	**Event properties**

Name	**Extends**
RadioButton	**Button**
Events	**Event properties**

Name	**Extends**
ScrollBase	**Object**
Events	**Event properties**
MouseScroll	delta(int)

Name	**Extends**
ListBase	**ScrollBase**
Events	**Event properties**
MouseScroll	delta(int)
DragStart	draggedItem(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
DragDrop	action(String) draggedItem(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
DragCancel	ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Select	itemRenderer(String) triggerEvent(Event) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
MultiSelect	itemRenderer(String) triggerEvent(Event) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Deselect	itemRenderer(String) triggerEvent(Event) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Scroll	position(int) direction(String) detail(String)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
DoubleClick	itemRenderer(String)

Name	**Extends**
DataGrid	**ListBase**
Events	**Event properties**
HeaderClick	columnIndex(int)
ColumnStretch	columnIndex(int) localX(int)
HeaderShift	newIndex(int) oldIndex(int) triggerEvent(Event)
Edit	itemRenderer(String) rowIndex(int) columnIndex(int)

Name	**Extends**
List	**ListBase**
Events	**Event properties**
Edit	itemRenderer(String) rowIndex(int) columnIndex(int)

Name	**Extends**
Tree	**ListBase**
Events	**Event properties**
DragDrop	action(String) dropParent(String) draggedItem(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Open	itemRenderer(String) triggerEvent(Event)
Close	itemRenderer(String) triggerEvent(Event)

Name	**Extends**
Application	**Box**
Events	**Event properties**

Name	**Extends**
ScrollBar	**Object**
Events	**Event properties**
Scroll	position(int) direction(String) detail(String)

Name	**Extends**
NumericStepper	**Object**
Events	**Event properties**
Change	value(Number)
Input	text(String)
SelectText	beginIndex(int) endIndex(int)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
VideoDisplay	**Object**
Events	**Event properties**

Name	**Extends**
Loader	**Object**
Events	**Event properties**

Name	**Extends**
Image	**Loader**
Events	**Event properties**

Name	**Extends**
Slider	**Object**
Events	**Event properties**
Change	value(Number) thumbIndex(int) clickTarget(String) triggerEvent(Event) keyCode(String)

Name	**Extends**
ComboBase	**Object**
Events	**Event properties**
Open	triggerEvent(Event)
Close	triggerEvent(Event)
Scroll	position(int) direction(String) detail(String)
Input	text(String)
SelectText	beginIndex(int) endIndex(int)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
ComboBox	**ComboBase**
Events	**Event properties**
Select	itemRenderer(String) triggerEvent(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Input	text(String)

Name	**Extends**
DateField	**ComboBase**
Events	**Event properties**
Change	newDate(Date)
Scroll	detail(String)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
DateChooser	**Object**
Events	**Event properties**
Change	newDate(Date)
Scroll	detail(String)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
ColorPicker	**ComboBase**
Events	**Event properties**
Open	triggerEvent(Event)
Close	triggerEvent(Event)
Change	color(String)

Name	**Extends**
TextArea	**ScrollBase**
Events	**Event properties**
Input	text(String)
SelectText	beginIndex(int) endIndex(int)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Scroll	position(int) direction(String) detail(String)

Name	**Extends**
Panel	**Container**
Events	**Event properties**

Name	**Extends**
TitleWindow	**Panel**
Events	**Event properties**

Name	**Extends**
Alert	**Panel**
Events	**Event properties**
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
DividedBox	**Box**
Events	**Event properties**
Pressed	dividerIndex(int) delta(int)
Dragged	dividerIndex(int) delta(int)
Released	dividerIndex(int) delta(int)

Name	**Extends**
Menu	**Object**
Events	**Event properties**
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Show	itemRenderer(String)
Hide	
Select	itemRenderer(String)

Name	**Extends**
MenuBar	**Object**
Events	**Event properties**
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)
Show	itemRenderer(String)
Hide	

Name	**Extends**
Repeater	
Events	**Event properties**

Name	**Extends**
Label	**Object**
Events	**Event properties**
Click	ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
Rule	**Object**
Events	**Event properties**

Name	**Extends**
FormItem	**Container**
Events	**Event properties**

Name	**Extends**
ViewStack	**Container**
Events	**Event properties**
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	**Extends**
TabNavigator	**ViewStack**
Events	**Event properties**
Change	relatedObject(String)

Name	**Extends**
NavigationBar	**Box**
Events	**Event properties**
Change	relatedObject(String)
Type	keyCode(String) ctrlKey(Boolean) shiftKey(Boolean) altKey(Boolean)

Name	Extends
LinkBar	**NavigationBar**
Events	**Event properties**

Name	Extends
ButtonBar	**NavigationBar**
Events	**Event properties**

Name	Extends
ToggleButtonBar	**ButtonBar**
Events	**Event properties**

Name	Extends
Chart	**Object**
Events	**Event properties**

Name	Extends
CartesianChart	**Chart**
Events	**Event properties**

Name	Extends
AreaChart	**CartesianChart**
Events	**Event properties**

Name	Extends
BarChart	**CartesianChart**
Events	**Event properties**

Name	Extends
ColumnChart	**CartesianChart**
Events	**Event properties**

Name	Extends
HLOCChart	**CartesianChart**
Events	**Event properties**

Name	Extends
LineChart	**CartesianChart**
Events	**Event properties**

Name	Extends
PieChart	**Chart**
Events	**Event properties**

Name	Extends
ChartSeries	**Object**
Events	**Event properties**
Click	hitSet(int)
DoubleClick	hitSet(int)
ItemRollOver	hitSet(int)

Name	Extends
AreaSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
BarSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
BubbleSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
ColumnSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
LineSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
PieSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
PlotSeries	**ChartSeries**
Events	**Event properties**

Name	Extends
AxisRenderer	**Object**
Events	**Event properties**

Name	Extends
ChartLegend	**Object**
Events	**Event properties**
Click	item(String)

Name	Extends
ListLabel	**DisplayObject**
Events	**Event properties**

APPENDIX C

Contents of CD-Rom

Report

- The report in *pdf* format

FunFX Flex adapter

- Source code
- SWC library file
- Documentation

FunFX Ruby Framework

- Source code
- Gem file
- Documentation

User guide

- A Flash movie showing the framework in use

Tests

- Demo tests

APPENDIX D

FunFX tests

This appendix contains some sample tests, and will provide the reader with information to be able to write tests.

Listing D.1: A sample test for testing a DataGrid with drag and drop

```
1  require 'test/unit'
2  require 'include'
3
4  class DatagridTest < Test::Unit::TestCase
5
6    def setup
7      @ie = Funfx.instance
8      @ie.start
9      @ie.goto("localhost/FlexObjectTest.html", "FlexObjectTest")
10   end
11
12   def teardown
13     @ie.unload
14   end
15
16   def go_to_page
17     test = @ie.link_bar("linkBar").change(:related_object => "
           DataGrid")
18   end
19
20   def test_control
21     go_to_page
22
23     @path = @ie.panel("dgPanel")
24
25     @path.data_grid("dg").select(:item_renderer => "Joanne Wall")
26     pos = @path.data_grid("dg").selected_index
27     assert_equal("Joanne Wall,555-219-2012,jwall@fictitious.com",
           @path.data_grid("dg").tabular_data(:start => pos, :end => pos
```

```
        ))
28
29      @path.data_grid("dg").drag_start(:dragged_item => "
            jwall@fictitious.com")
30      @path.data_grid("dgDrop").drag_drop
31
32      assert_equal("Joanne Wall,555-219-2012,jwall@fictitious.com",
            @path.data_grid("dgDrop").tabular_data)
33    end
34
35  end
```

Listing D.2: A sample test for testing a slow data provider with a wait tag

```
1   require 'test/unit'
2   require 'include'
3
4   class SyncTest < Test::Unit::TestCase
5
6     def setup
7       @ie = Funfx.instance
8       @ie.start
9       @ie.goto("localhost/FlexObjectTest.html", "FlexObjectTest")
10    end
11
12    def teardown
13      @ie.unload
14    end
15
16    def go_to_page
17      @ie.link_bar("linkBar").change(:related_object => "SyncTest")
18    end
19
20    def test_control
21      go_to_page
22
23      @path = @ie.view_stack("vStack").box("SyncTest").panel("SyncTest
            ")
24
25      @path.data_grid("dg").select(:item_renderer => "nr3", :wait =>
            20)
26
27      pos = @path.data_grid("dg").selected_index
28      assert_equal("nr3", @path.data_grid("dg").tabular_data(:start =>
             pos, :end => pos))
29    end
30
31  end
```

Listing D.3: A sample test for testing a menu bar

```
require 'test/unit'
require 'include'

class MenuBarTest < Test::Unit::TestCase

  def setup
    @ie = Funfx.instance
    @ie.start
    @ie.goto("localhost/FlexObjectTest.html", "FlexObjectTest")
  end

  def teardown
    @ie.unload
  end

  def go_to_page
    @ie.link_bar("linkBar").change(:related_object => "MenuBar")
  end

  def test_control
    go_to_page

    @path = @ie.view_stack("vStack").box("MenuBar").panel("MenuBar
        Control")

    assert_not_nil(@path.menu_bar("menuBar"))

    menu = "Menu1"
    item = "MenuItem 1-B"
    item_data = "1B"

    @path.menu_bar("menuBar").show(:item_renderer => menu)
    @ie.menu(menu).select(:item_renderer => item)
    assert_equal("Label: " + item + "\nData: " + item_data, @ie.
        alert("Clicked menu item").text)

    @ie.alert("Clicked menu item").button("OK").click
    assert_nil(@ie.alert("Clicked menu item"))

    menu = "Menu2"
    item1 = "MenuItem 2-B"
    item2 = "SubMenuItem 3-A"
    item_data = "3A"

    @path.menu_bar("menuBar").show(:item_renderer => menu)
    @ie.menu(menu).show(:item_renderer => item1)
    @ie.menu(item1).select(:item_renderer => item2, :wait => 2)
    assert_equal("Label: " + item2 + "\nData: " + item_data, @ie.
        alert("Clicked menu item").text)
  end

end
```

Glossary

FunFX	The framework developed during this project to functional test Flex applications
ActiveX Control	A Microsoft term that is used to denote reusable software components that are based on Microsoft Component Object Model (COM)
Agile Method	A suite of development methods with focus on code and fast development
COM	Component Object Model, a Microsoft platform for software componentry. The essence of COM is a language-neutral way of implementing objects such that they can be used in environments different from the one they were created in.
CPU	Central Processor Unit
CTP	Community Technology Previews, a Microsoft pre-release of software [41]
Flex	A framework for developing RIAs based on the Flash player (see Section 5)
GUI	Graphical User Interface
IDE	Integrated Development Environment
irb	Interactive Ruby, an interactive command-line interpreter
OLE	Object Linking and Embedding (see Section4.4)
OSS	Open Source Software
QA	Quality Assurance
QTP	QuickTest Professional, a testing tool from Mercury
RIA	Rich Internet Application (see Section 4.1)
TDD	Test-Driven Development (see Section 3.1.2)

UP	Unified Process, a development method (see Section 3.1.1)
Wiki	Wiki is a piece of server software that allows users to freely create and edit Web page content using any Web browser. Wiki supports hyperlinks and has a simple text syntax for creating new pages and cross links between internal pages on the fly.

Bibliography

[1] Adobe. About working with data in Flex Builder. http://livedocs.adobe.com/flex/201/html/wwhelp/wwhimpl/common/html/wwhelp.htm?context=LiveDocs_Book_Parts&file=data_046_2.html. Retrieved May 14th 2007.

[2] Adobe. Adobe Flex Data Services 2. http://www.adobe.com/products/flex/dataservices/. Retrieved February 26th 2007.

[3] Adobe. Flex 2 - Product overview. http://www.adobe.com/products/flex/productinfo/overview/. Retreived January 23th 2007.

[4] Adobe. Flex Documentation Introduction to mixins. http://www.adobe.com/support/documentation/en/flex/1/mixin/mixin2.html#118542. Retrieved February 27th 2007.

[5] Matt Horn (Adobe). Flex synchronization. E-mail. Retrieved April 25th 2007.

[6] Matt Horn (Adobe). License for using custom classes. E-mail. Retrieved May 4th 2007.

[7] Jeremy Allaire. Macromedia Flash MX-A Next-generation Rich Client. Technical report, Macromedia, 2002.

[8] AutomatedQA. TestComplete 5. http://www.automatedqa.com/products/testcomplete/index.asp. Retrieved May 7th 2007.

[9] AutomatedQA. TestComplete 5 FAQ. http://www.automatedqa.com/products/testcomplete/faqs/tc_faq_web.asp. Retrieved May 7th 2007.

[10] Victor R. Basili. The Experimental Paradigm in Software Engineering. Technical report, University of Maryland, 1992.

[11] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison Wesley, second edition, 2003.

[12] Fitnesse. What is Fitnesse? http://fitnesse.org/FitNesse.OneMinuteDescription. Retrieved February 26th 2007.

[13] Jesse James Garret. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/essays/archives/000385.php. Retrieved March 26th 2007.

[14] Grig Gheorghiu. A look at Selenium. *Better Software*, 2005.

[15] Russell Gold, Thomas Hammell, and Tom Snyder. *Test-Driven Development: A J2EE Example*. Apress, 2004.

[16] Christian Hellsten IT Specialist IBM. Automate acceptance tests with Selenium. `http://www-128.ibm.com/developerworks/java/library/wa-selenium-ajax/index.html?ca=drs-`. Retreived January 29th 2007.

[17] Adobe Systems Incorporated. ExternalInterface API. `http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/external/ExternalInterface.html`. Retrieved Mars 29th 2007.

[18] Adobe Systems Incorporated. Adobe Flex 2 Developer's Guide. `http://livedocs.macromedia.com/flex/2/docs/Part2_DevApps.html`, 2006. Retreived January 30th 2007.

[19] Adobe Systems Incorporated. Coding with MXML and ActionScript. `http://www.adobe.com/devnet/flex/quickstart/coding_with_mxml_and_actionscript/`, 2006. Retreived January 17th 2007.

[20] Jonathan Kohl and Paul Rogers. Watir works. *Better Software*, 2005.

[21] Jeff Langr. *Agile Java Crafting Code with Test-Driven Development*. Prentice Hall, 2005.

[22] Craig Larman. *Applying UML and Patterns*. Prentice Hall, 3. edition 2005.

[23] Mercury. Mercury Functional Testing: Mercury QuickTest Professional. `http://www.mercury.com/us/website/pdf-viewer/?url=/us/pdf/products/datasheets/DS-0985-0306-qtp.pdf`. Retreived January 25th 2007.

[24] Mercury. Mercury QuickTest Professional. `http://www.mercury.com/us/products/quality-center/functional-testing/quicktest-professional/`. Retreived January 25th 2007.

[25] Microsoft. Microsoft Unveils Silverlight. `http://www.microsoft.com/presspass/press/2007/apr07/04-15WPFEPR.mspx`. Retrieved April 26th 2007.

[26] MSDN. Component Object Model. `http://msdn2.microsoft.com/en-us/asp.net/bb187358.aspx`. Retrieved March 26th 2007.

[27] Glenford J. Myers, Tom Badgett, Todd M. Thomas, and Corey Sandler. *The Art of Software Testing*. John Wiley and Sons Inc, 2. edition 2004.

[28] OpenQA. Selenium. `http://www.openqa.org/selenium/`. Retreived January 26th 2007.

[29] RubyForge. Watir - Web Application Testing in Ruby. `http://wtr.rubyforge.org/`. Retreived January 19th 2007.

[30] rubylearning.com. Duck Typing. `http://rubylearning.com/satishtalim/duck_typing.html`. Retrieved May 7th 2007.

[31] Christian Schwarz, Stein Kåre Skytteren, and Trond Marius Øvstetun. CubicTest. `http://boss.bekk.no/cubictest/`. Retrieved May 9th 2007.

[32] RDT Development Team. Ruby Development Tool. `http://rubyeclipse.sourceforge.net/`. Retreived January 23th 2007.

[33] David Thomas and Andrew Hunt. *Programming Ruby - The Pragmatic Programmer's Guide*. Addison-Wesley Professional, 2001.

[34] David Thomas, Andrew Hunt, and Chad Fowler. *Programming Ruby - The Pragmatic Programmer's Guide*. Addison-Wesley Professional, second edition, 2005.

[35] Wikipedia. Adobe Flex. `http://en.wikipedia.org/wiki/Adobe_Flex`. Retreived January 22th 2007.

[36] Wikipedia. Agile software development. `http://en.wikipedia.org/wiki/Agile_software_development`. Retrieved May 14th 2007.

[37] Wikipedia. Component Object Model. `http://en.wikipedia.org/wiki/Component_object_model`. Retrieved March 1st 2007.

[38] Wikipedia. Duck Test. `http://en.wikipedia.org/wiki/Duck_test`. Retrieved May 7th 2007.

[39] Wikipedia. Factory pattern. `http://en.wikipedia.org/wiki/Factory_method_pattern`. Retrieved April 13th 2007.

[40] Wikipedia. Rich Internet Application. `http://en.wikipedia.org/wiki/Rich_internet_application`. Retrieved April 12th 2007.

[41] Wikipedia. Software release life cycle. `http://en.wikipedia.org/wiki/Software_release_life_cycle`. Retrieved April 26th 2007.

[42] Wikipedia. Test case. `http://en.wikipedia.org/wiki/Test_case`. Retreived January 30th 2007.

[43] Wikipedia. Watir. `http://en.wikipedia.org/wiki/Watir`. Retreived January 24th 2007.

[44] Marvin V. Zelkowitz and Dolores R. Wallace. Experimental Models for Validating Technology. *IEEE Computer*, May 1998.