Individual Professional Practice in the Company
VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Bachelor Thesis Assignment

Student: Aleksi Antero Stenholm
Study Programme: B2647 Information and Communication Technology
Study Branch: 2612R025 Computer Science and Technology
Title: Absolvování individuální odborné praxe
Individuional Professional Practice in the Company

Description:
1. The student shall undergo individual practice with the limited liability company ABB s.r.o.
2. The final report structure shall be the following:
   a) Specification of the professional orientation of the firm where the student underwent his/her professional practice, specification of his/her occupational category.
   b) The list of tasks the student was assigned in the course of his/her professional practice including their time spans.
   c) Methods chosen when dealing with the given tasks.
   d) Theoretical and practical knowledge and skills acquired in the course of his/her professional practice.
   e) Knowledge and skills the student were missing in the course of his/her professional practice.
   f) Results achieved in the course of his/her professional practice and the general assessment of them.

References:
In accordance with the tutor guiding the student's professional practice.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: MSc. Donald David Davendra, Ph.D.
Consultant: Jan Mináč

Date of issue: 01.09.2013
Date of submission: 07.05.2014

doc. Dr. Ing. Eduard Sojka
Head of Department

prof. RNDr. Václav Snášel, CSc.
Dean of Faculty
I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

In Ostrava 6th of May 2014

Aleksi Stenholm
Abstract

This thesis is written about my tasks in practical training at ABB Ostrava’s software department. I had two major tasks. These were: Programming Coded UI Tests with C# in Visual Studio 2012 and programming minor tasks with ASP.NET MVC 4 using C#. Introductions to Coded UI tests and MVC will be given. All the tasks were focused in to one project called Motor Service Portal and I was placed to the team of this project.

Key words

UI, Testing, Refactoring, C#, ABB, MVC
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ajax</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>ASP.NET</td>
<td>Microsoft's framework for making dynamic web pages.</td>
</tr>
<tr>
<td>C#</td>
<td>Microsoft's object-oriented programming language</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, Read, Update, Delete</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>MSP</td>
<td>Motor Service Portal</td>
</tr>
<tr>
<td>MVC</td>
<td>Model, View, Controller</td>
</tr>
<tr>
<td>ORM</td>
<td>Object-Relational Mapping</td>
</tr>
<tr>
<td>RFQ</td>
<td>Request For Quotation</td>
</tr>
</tbody>
</table>
Contents

1 Introduction .................................................................................................................. - 7 -

2 Motor Service Portal .................................................................................................. - 8 -
   2.1 Database ........................................................................................................ - 8 -

3 Software testing ......................................................................................................... - 9 -
   3.1 Coded UI tests ................................................................................................. - 9 -
   3.2 System testing ................................................................................................. - 9 -
   3.3 Testing in MSP .................................................................................................. - 9 -

4 Building Tests ......................................................................................................... - 11 -
   4.1 Coded UI Test Builder ..................................................................................... - 11 -
   4.2 Coding the tests .............................................................................................. - 11 -
   4.3 Assertions ....................................................................................................... - 13 -
   4.4 Running the tests ............................................................................................ - 13 -

5 Problems with Coded UI Tests .............................................................................. - 15 -
   5.1 Compatibility .................................................................................................. - 15 -
   5.2 Sharing tests ................................................................................................... - 15 -
   5.3 Coded UI Test Builder .................................................................................... - 15 -

6 Programming ........................................................................................................... - 16 -
   6.1 ASP.NET MVC ................................................................................................. - 16 -
      6.1.1 Controllers ............................................................................................... - 16 -
      6.1.2 Views ...................................................................................................... - 17 -
      6.1.3 Models ..................................................................................................... - 18 -
   6.2 My tasks .......................................................................................................... - 18 -
      6.2.1 Research ................................................................................................. - 18 -
      6.2.2 Refactoring ............................................................................................. - 19 -

7 Conclusions .............................................................................................................. - 21 -

References .................................................................................................................... - 22 -
1 Introduction

This thesis is about my practical training that took place in ABB Ostrava’s software department from October 2013 to May 2014. I was part of the development team that developed web application called Motor Service Portal.

My main task was to make tests for MSP. Only GUI tests were made, because of client’s request. Tests were Microsoft’s Coded UI Tests done in Visual Studio 2012. These tests were new thing for everyone in the team, so I had to do some research about it by myself. I also conducted manual testing for MSP.

I also made some minor programming tasks and refactoring to the code to make MSP perform better and be more user friendly. For this I needed to learn MVC 4 programming model.

All the shown code in this thesis is imaginary and not part of MSP. They are just to show how Coded UI Tests are made in Visual Studio 2012 and how MVC works.
2 Motor Service Portal

MSP is web application that is being developed to be only used internally in ABB Ostrava’s motor services. MSP focuses on keeping track of contracts and RFQs. Before MSP this all was done using Excel sheets. Excel sheets grow very fast in size and become hard to read. With MSP everything is saved in to database and accessed through web portal. MSP is developed using C#, ASP.NET MVC 4 programming model and some Javascript/jQuery.

2.1 Database

Microsoft SQL Server was the used database management system in MSP. Object Relational Mapping was done using Micro ORM called PetaPoco.

PetaPoco is an open source micro ORM for .NET framework. PetaPoco is only one C# file. PetaPoco was chosen to be used in MSP project because after some tests it was found to be faster than LINQ.

Database creation was automated by using RoundhousE database deployment system. RoundhousE makes database creating easy, because with one BAT file it is possible to drop database and create it directly after it, according to scripts provided to RoundhousE. This way dropping and creating database using database management system becomes unnecessary.
3 Software testing

Software testing is crucial part of software development. By software testing developers check if code they are working on is working as intended. Testing is also crucial from customer’s point of view. Through testing customer can be sure that correct end product is being build. [1]

There are two methods of software testing: White-box and Black-box. In white-box testing tester has the knowledge of the code behind the GUI, so white-box testing also tests this code. In black-box testing tester has no knowledge of the code behind the GUI. These don’t test internal parts of the application, but external parts. In MSP only black-box testing was done. [1]

There are many different type of test levels. Unit testing focuses on only on smallest testable pieces of software that have no dependencies to other pieces. Integration testing tests a group of pieces that are dependent on each other. System testing tests the application as a whole. This is also known as acceptance testing. Then there is GUI testing. Tests are usually automated to make them easy and fast for developer to run. This way, after modifications to code, programmer can check if the modifications really work by running the test.

3.1 Coded UI tests

Coded UI Tests test your application’s GUI by making inputs that have been coded to the test. It is also possible to make asserts that verify that properties of HTML elements are correct. Coded UI Tests are completely automated. Microsoft Visual Studio provides environment for making and maintaining tests. Microsoft Visual Studio Premium or Ultimate versions only include Coded UI Tests, so with lesser versions running tests is impossible.

Coded UI Tests are being mapped from HTML elements. Tests find elements by their properties, usually by ID or class, but if ID or class is not available can other properties also be used.

3.2 System testing

System testing is usually one of the last steps of software development. System testing focuses on testing the software functionality and reflecting it to the technical requirements given by customer. System testing is done by testing specialist unlike Unit, GUI and integration testing which are done by programmer. [2]

3.3 Testing in MSP

At the start of the project both UNIT tests and Coded UI Tests were supposed to be made, but customer decided that Coded UI Tests are enough, so UNIT tests were not made.

Coded UI Tests were made using the Coded UI Test Builder at first, because of its ease of use. This method seemed to be alright, but when there started to be more and more tests, maintaining all the tests became very hard because of the size of UIMap files, which hold...
generated code. This made us move in to programming tests by ourselves. In MSP one UIMap
file generated by Coded UI Test Builder, which contains one test, is 377 lines long. One
manually programmed C# file, which contains three tests, is 188 lines long.

We decided to make one test class for each Controller in our application. One test class
holds tests for all the methods inside its corresponding controller. Before each test, database is
being prepared for the test. This is done with a class that first deletes all records from database
and after it inserts data back to database. This way before each tests database is in its correct
state and tests won’t fail because of some problems with database. Before each test, browser is
being prepared also. This is also done with a class that opens Internet Explorer with right web
page. After each test browser is closed, so next test will have fresh instance of it.

Before every demo and meeting with customer, MSP was also manually tested to make
sure that no bugs were left on the code. This testing was done like system testing. I went
through the application and checked that functionality worked like it should. After this I tried to
find some vulnerabilities that might cause the application to crash because of unhandled
exception, such as variable overflowing and script injection. This kind of testing overlaps with
GUI testing, because with Coded UI Tests it is possible to test the same things, but we felt that it
is necessary, because there were some problems with Coded UI Tests.
4 Building Tests

This section shows the two ways how Coded UI Tests can be made in Visual Studio 2012.

4.1 Coded UI Test Builder

With this method programmer records the wanted inputs by using Coded UI Test Builder, as shown in figure 1. Programmer is also able to edit already existing tests with Coded UI Test Builder. After all wanted inputs are done, Coded UI Test Builder generates code in to UIMap files that are included in test project. Test cases still need to be invoked in main test class. Coded UI Test Builder has one major flaw compared to other option and that is length of the code. Generated UIMaps are usually very long, hard to read and have a lot of unnecessary lines of code.

![Figure 1: Coded UI Test Builder](image)

4.2 Coding the tests

By coding tests by yourself code stays in acceptable length and there are a lot less unnecessary code. This method also makes UIMaps not needed, because you can program tests straight in to test classes. It is a good principle to keep tests of one controller in one class, for example. CRUD operations of user are in one controller called UserController and all the tests concerning these methods are in a test class called UserControllerTests.

Figure 2 shows simple example code which demonstrates how to program Coded UI Tests. This test will open browser, navigate to [http://www.vsb.cz](http://www.vsb.cz) webpage, change the language of the page and do a search for “Information technology”. After the search results are shown test will see if the header of the page is correct. At the end browser will be closed.
public static T FindHtmlControl<T>(UITestControl parent, string findBy, string query) where T : HtmlControl, new()
{
    //Method for finding Html controls from page
}

[TestMethod]
public void VSBChangeLanguageAndSearch()
{
    //Open browser and moves to page 'http://www.vsb.cz'
    BrowserWindow browser = BrowserWindow.Launch("http://www.vsb.cz/");
    //Find hyperlink that changes language of the page and click it
    Mouse.Click(FindHtmlControl<HtmlHyperlink>(browser, "id", "language-en"));
    //Find textbox for searching and click it
    Mouse.Click(FindHtmlControl<HtmlEdit>(browser, "id", "pattern");
    //Type 'Information technology'
    Keyboard.SendKeys("Information technology");
    //Press enter
    Keyboard.SendKeys("{Enter}");
    //Check if header of the page is what it is supposed to be
    Assert.AreEqual("Vyhledávání", FindHtmlControl<HtmlDiv>(browser, "class", "heading").InnerText);
    //Wait for playback to be finished
    //Close browser
    browser.Close();
}

Figure 2: Example test method
4.3 Assertions

It is also possible to make assertions in to the test cases. Assertion checks the property of the HTML element you have chosen to be tested and sees if it matches the criteria given to it. So basically assertions test that HTML elements are shown correctly on the screen.

In MSP assertions were used to verify end results of tests, like when testing edit user method, assertion checks if the username really changed in the main user list also. Basically every time something is changed, it is checked also with assertion.

4.4 Running the tests

After all the tests are done, they can be run from Visual Studio’s Test Explorer. If test ran correctly, it will show green mark next to test name as shown in figure 3. If test failed, it shows red cross and from summary you are able to see what went wrong and in what line, like shown in figure 4.

![Figure 3: Successful test](image)
Test has failed on figure 4, because it could not perform action “Click”. This is because there is not HTML control with the searched ID. Test explorer shows where the exception happened. This case it was in CodedUITest1.cs file, on line 27. To fix this exception programmer should change the ID’s on that line to be right, and after that test will work again.
5 Problems with Coded UI Tests

5.1 Compatibility

Coded UI Tests have a lot of compatibility issues with different browsers. Internet Explorer version 8 and higher is fully supported, but 7 or lower are not supported at all. With Google Chrome and Mozilla Firefox recording the actions is not possible, but running the tests is. Opera and Apple Safari are not supported at all. Because of these compatibility issues, Coded UI Tests are not suitable for projects that need to be running in other browsers than Internet Explorer 8 or higher. For projects that need to be running in other browsers, developers need to find something else.

5.2 Sharing tests

Sharing tests between development team is almost impossible due to Coded UI Tests being so sensitive for little changes in environment. Sharing tests through Team Foundation Server is easy, but most of the time Coded UI Tests, which have been coded on some other computer than yours, won’t work. Sometimes Coded UI Tests don’t find HTML controls of the application. This might be causes by difference in browser version or Visual Studio version, but we were not able to locate the real cause for it.

5.3 Coded UI Test Builder

Using Coded UI Test Builder it is very easy to make GUI Tests, but it has some flaws. UIMap files grow very big and because of this they are very hard to understand. Generated UIMaps can also be very hard, because if you make some changes to generated code, next time UIMap is being generated, all the changes will be wiped. These changes needs to be stored also somewhere else than UIMap, so you can make these changes again when UIMap is generated.
6 Programming

This section will talk about the ASP.NET MVC and the little programming tasks that I made during my practical training.

6.1 ASP.NET MVC

ASP.NET is a Microsoft technology for creating dynamic and interactive web pages or web applications. ASP.NET supports three different programming models: Web Pages, MVC and Web Forms. In this paper we only speak about MVC. [3]

MVC is one of the three programming models supported by ASP.NET. Abbreviation MVC comes from the words Model View Controller. MVC model has three different layers. First layer is Model, second layer is View and third and final layer is Controller, like you can see from figure 5. [4] Web forms used to be the programming model of choice for web developers, but lately MVC has become as popular as Web Forms. They both co-exist because both have their advantages over the other. MVC gives developer full control of what to render in to view. Web Forms use automatically rendered HTML, so developer has less control over view. Maintaining MVC application is easy application is divided in to three parts

![MVC programming model](image)

6.1.1 Controllers

Controllers are classes that handle user’s requests. These requests are sent through HTTP protocol. Controllers work as a link between Views and Models, so Controller passes requests from View to Model and data from Model to View. Controllers return objects of type
ActionResult. There are many types of ActionResults. ViewResult returns HTML to the browser, FileResult returns binary content, ContentResult returns any type of content and JsonResult returns JSON type content. [5]

6.1.2 Views

Views are the HTML pages that user can see in their browsers. From views, user’s requests are sent to controllers, which forwards requests to models. Views are coded in HTML language, but they can include some server-side code also. To process server-side code, implementation of view engine is needed. View engines provide special syntax to work with server-side elements and render HTML to user’s browser. MVC provides two different kinds of view engines: Razor and ASPX view engines. View engines enable developer to mix C# code with HTML. Figures 6 and 7 show differences in the syntax of these two view engines. [5]

Example of Razor syntax:

```razor
<ul>
    @{
        List<int> numberList = new List<int>();
        numberList.Add(1);
        numberList.Add(2);
        numberList.Add(3);
        numberList.Add(4);
        numberList.Add(5);
        foreach (int num in numberList) {
            <li>@num</li>
        }
    }
</ul>
```

*Figure 6: Razor view engine example*
Example of ASPX syntax:

```csharp
<ul>
  <%{
    List<int> numberList = new List<int>();
    numberList.Add(1);
    numberList.Add(2);
    numberList.Add(3);
    numberList.Add(4);
    numberList.Add(5);
    foreach (int num in numberList) {
      %><li>%<%=num%>%</li>%
    } %>
</ul>
```

Figure 7: ASPX view engine example

First version of MVC was released on March 2009. First and second version still used only ASPX view engine, but third version introduced Razor view engine.

6.1.3 Models

There are three different types of models: Data model, business model and view model. Data models are classes that communicate with database. Business models implement functionality, such as calculating or altering data. Business models usually work together with data models for saving and reading data from database. View models store data which will be rendered in to user’s browser window. In MSP all three kind of models were combined in to one model class for each entity in database. [5]

6.2 My tasks

6.2.1 Research

Because MVC programming was new thing to everyone in the team, there were some functionalities that were a little challenging to make. Main developers were not able to research
these functions because customer wanted to see new functionality, so they had to keep programming. So when these kind of problematic functions were noticed, it became my task to try solving them, because I had more time to do research about them. These functions were always something that are not crucial part of application, so I had enough time to figure them out.

Development team wanted to have jQuery modal dialog to pop up when administrator wants to create new user or edit user. There was two problems. One was that we did not want to use two dialogs, one for creating user and one for editing user. Second one was that dialog did disappear when update or insert is being made to database. We wanted to keep dialog open if some error happened and show error message on the dialog. These problems were overcome by using partial view which will work like update panel from ASP.NET. The partial view is shown in the jQuery modal dialog and content of the partial view is being changed depending if administrator is creating user or editing user. And second problem was overcome by changing the normal behavior of submit button. In MVC, forms have submit buttons which automatically call HttpPost method from controller. I altered submit button's behavior so it won't call this method automatically, but call it by using AJAX. This way I was able to force AJAX to have error if some error happened while inserting of updating to database.

There was also problem with search function. We wanted search function to ignore diacritics on the letters, so if someone typed á in to search box, it would return every match with letters á and a. This problem seemed hard to overcome, but at the end it was very easy. Only thing I had to do was add a little command to SQL query. This command tells to database that accent is insensitive, so database ignores diacritics.

I also made same research concerning exception logging. We wanted to use some logging library for taking care of logging the exceptions. We wanted to use Windows own Event Viewer for keeping the log of exceptions. This had to be able to do without administrative rights to computer. I made a little demo showing two different libraries: NLog and Log4Net.

MSP uses resource file, to keep language of the application localized. Contract statuses are shown using the enumerator description attributes, but description attributes cannot be converted using resource file. My task was to figure a solution how status descriptions can also be localized. I solved this by making a new derived class that inherits class called DescriptionAttribute. On the constructor I have base constructor that takes the description of enumerator as parameter. This method gets the right input according to description from resource file. This way also descriptions of enumerators are localizable.

6.2.2 Refactoring

Refactoring means improving applications design without alternating its behavior. In this context design means application’s source code and behavior means application’s functionality. This means that after refactoring application should work the same way as before refactoring. [6]

- 19 -
Changes made in refactoring are usually quite small, but even small change to code can produce significant improvement in code. While one change made is quite small, a dozen changes can transform poorly written software in to well-designed and well written software. [6]

Because of refactoring, first design is not absolute. It is usually thought that initial design is absolute and bad design is hard to fix later in programming process. Because of refactoring it is possible to keep changing design of the application if needed. This means that programming process can be started earlier because design of the program doesn’t need to be perfect up-front. This doesn’t mean that designing can be completely forgotten. [6]

Typical target for refactoring is duplicated code. Duplicated code means that same functionality is programmed more than once in the code. Better practice is to have it done only once and call it every time it is needed. Duplicated code grows the size of the application, which leads in to more poor performance. Second typical refactoring target is variable names and variables as whole. Variable names should be short and the name should tell what value it represents. Usually programmers name variables with just one letter or just with some vague name.

In MSP, refactoring was always ongoing process. Controllers and Models tend to grow very big easily, so splitting Controllers and Models in to smaller and more logical parts were done many times. This way code became easier to maintain and understand.

Models contained a lot of duplicated code. Each model has CRUD methods for database, so there was a lot of duplicated code. This means that every model contained methods for getting all records, relating to this models, from the database. This was refactored by making one base class and using it if possible. This class wasn’t done by me. Variables were also refactored many times to be as short and easy to understand as possible. In MSP, models are named like database entities, so name of the model was not necessary to be used in variable names, for example. Model name is UserModel, so all UserId variables inside that model were changed to be just Id.

Because of these changes code of MSP became much easier to maintain and understand.
7 Conclusions

ABB Ostrava's motor services wanted to have an application that is used to keep track of their contracts and RFQs. Solution for this is MSP. MSP is a web application which is written in C# using ASP.NET MVC 4 with Razor view engine.

I was working in a development team that was developing MSP in ABB Ostrava's software department from October 2013 to May 2014. My part in the development team was to test the application manually and using Coded UI Tests and also to do some minor programming tasks, such as refactoring and researching new functionalities. During the practical training Coded UI Tests turned out to be quite unreliable and problematic. In MSP Coded UI Testing was done by writing the test methods by ourselves, because Coded UI Test Builder turned out to be more unreliable and it had more problems. Refactoring and researching was an ongoing process throughout the development, because we wanted to make our application as user-friendly as possible.

MSP is being piloted at the moment in ABB Ostrava's motor services. It hasn't been taken fully into use yet.
References

[1] Software testing

[2] System testing

[3] ASP
   http://www.w3schools.com/asp/default.asp (10.04.2014)

[4] ASP.NET MVC
   http://www.w3schools.com/aspnet/mvc_intro.asp (10.04.2013)


[6] Refactoring