Industrial Programming

Coursework 1

Developing a Simple Web Browser

F21SC

Boris Mocialov
(H00180016)

Heriot – Watt University, Edinburgh
October 2014
# Table of Contents

1  Introduction  6  
1.1.  Purpose  6  
1.2.  Task  7  
1.3.  Assumptions  8  
2  Requirements  8  
3  Design considerations  11  
3.1.  UI design  11  
3.2.  Class design  15  
3.3.  Choice of data structures  16  
3.4.  Use of advanced language constructs  20  
3.5.  Other performance-relevant choices  21  
4.  Guides  21  
4.1  User Guide  22  
4.1.1.  Request URL  22  
4.1.2.  Save URL  24  
4.1.3.  View saved URLs  25  
4.1.4.  Going back  27  
4.1.5.  Going forward  29  
4.1.6.  'Jumping' around history  29  
4.1.7.  Available shortcuts  32  
4.2.  Developer Guide  33  
4.2.1.  Requesting URL  33  
4.2.2.  Saving URL  36  
4.2.3.  Viewing saved URLs  37  
4.2.4.  Going back  37  
4.2.5.  Going forward  38  
4.2.6.  Saving the state / Restoring the state  39  
4.2.8.  XML structure  40  
4.2.7.  'Jumping' around history  40  
5  Testing  41  
6  Conclusion  47  
7  References  49  
8.  Appendix A  50
### Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Web browser design inspiration</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>Current version design</td>
<td>15</td>
</tr>
<tr>
<td>5</td>
<td>Use-cases diagram</td>
<td>22</td>
</tr>
<tr>
<td>6</td>
<td>User opens BuggyBrowser</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>User opens a new tab</td>
<td>23</td>
</tr>
<tr>
<td>8</td>
<td>User loads a web page (google.com)</td>
<td>24</td>
</tr>
<tr>
<td>9</td>
<td>Save favourite popup box</td>
<td>25</td>
</tr>
<tr>
<td>10</td>
<td>Favourites list</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>2 pages currently loaded in one tab</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>State after back button is pressed</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>State after forward button is pressed</td>
<td>29</td>
</tr>
<tr>
<td>14</td>
<td>Loading few web pages in one tab window</td>
<td>30</td>
</tr>
<tr>
<td>15</td>
<td>Performing long click on back button (3 seconds)</td>
<td>31</td>
</tr>
<tr>
<td>16</td>
<td>After ‘jump’ is performed</td>
<td>32</td>
</tr>
</tbody>
</table>
Table of Tables

Table 1 Requirements table................................................................. 10
Table 2 UI design decisions................................................................. 15
Table 3 Interesting data structure/data type choices................................. 20
Table 4 Client-server communication constructs.................................... 36
Table 5 Test cases................................................................................. 46
Table of Code Snippets

Code 1 User presses Enter key in address text box................................. 33
Code 2 method that starts request and awaits for reply from a server........ 33
Code 3 Actual request to server with response handling.......................... 34
Code 4 ResponseObject class structure.................................................. 35
Code 5 Add favourite label clicked....................................................... 37
Code 6 Favorites icon clicked.................................................................. 37
Code 7 Back button click event.................................................................. 38
Code 8 PopLastHistory method................................................................. 38
Code 9 Forward button clicked.................................................................. 38
Code 10 PopLastPoppedHistory method...................................................... 38
Code 11 Read stored file to restore state.................................................... 39
Code 12 Write current state....................................................................... 39
Code 13 XML file....................................................................................... 40
Code 14: 'Jumping' around history............................................................. 40
Code 15 Reduce history method................................................................. 40
1 Introduction

1.1 Purpose
The purpose of this report is to describe the implementation of the simple web browser, written in C# programming language. The choice of programming language was specified in advance as it was one of the requirements for the application. As it is agreed that choice of a programming languages affects the structure of the final product as well as places some constraints on the development, C#, in particular, has a very wide range of libraries that facilitate the development process. The choice of this particular programming language did not cause any unanticipated difficulties in developing the final application.

This report will guide its reader through the development process of the application and point out some of the difficulties that were faced during design, implementation and testing as well as explicitly state choices that were made and agreed upon with the product owner (in this case – course coordinator) and laboratory assistants. Sections, namely, user guide and developer guide, will give an overview of the application using screenshots and textual description where necessary, as well as deeper description (using code snippets) for software developers to be able to get more information of how application works. The structure of the report will allow reader to view the screenshots, read descriptions and consult the code for a particular use case. At the end, in the testing section, reader will be presented with results from testing phase and will be able to see what was achieved based on the requirements specification, and what was not. Report will explicitly tell the reader about shortcomings of the current version of application and what are the existing bugs and errors of the application. Before considering testing, requirements section will explicitly state all the requirements, both functional and non-functional that were either specified explicitly or had to be deduced based on textual description of the desired product.

Before going any further, current state of the application requires versioning in order to be able to trace its development process as well as give some starting point for anyone who would want to expand functionality for the current application. Current version is set to v.0.7 (~74% of requirements met, each requirement corresponding to a minor release), since, in the light of
identified requirements (39 – given in the requirements table), most of the points were achieved and tested (29 achieved and tested). Possible bugs still exist, as application was tested on empirical evidence of existence of bugs/errors. Such evidence is not adequate to be able to imply that the application is free from run-time errors. To further test application, unit tests are necessary that will consider all possible effects on the existing code. Only then it will be possible to consider upgrading version to v.1.0. Due to this fact that the product possibly not error-free, application was given a name: “BuggyBrowser” (aka BB).

1.2. Task
The task was to develop a simple web browser given a set of requirements that needed to be fulfilled in order to deliver the final product. Key points from requirements were taken as a starting point in deducting functional and non-functional requirements for the simple web browser. Requirements did not carry detailed specifications of which libraries/tools were to be used during the construction of the simple web browser, therefore some assumptions regarding the appropriate choice of libraries, structure of classes and relationships between objects in the application had to be made. One explicit requirement/constraint was placed on the development of the simple web browser, which explicitly stated that C# class, known as WebBrowser, could not be used for this particular project, neither WebRequest class. This requirement placed a considerable limitation on the desired product, since both of the mentioned classes would make development process much easier and therefore less time could be spent on the implementation, since named classes are sophisticated wrappers around low-level request-reply communication methods used in computer communication scheme using network as a medium. On one hand, this can be seen as a certain shortcoming for desired functionality, since interconnected components, needed for this particular application, would have to be developed separately in order to facilitate lower-level communication between client and a server to deliver desired functionality. On the other hand, more control of the interoperability of different components would become
available to the developer. In this case this particular requirements/constraint was not perceived as a drawback but rather as an opportunity to make final product more custom.

An overview of the task can be described as a high-level use case, where a user should be able to use application in order to make requests to any web resource including all basic main functionality of any commercial web browser, such as IE, Firefox, Chrome, Safari, etc. In fact, most of the design choices or functionality was inspired by listed web browsers.

1.3. Assumptions
As requirement/constraint explicitly advocated against using both WebBrowser and WebRequest classes, an assumption was made that desired functionality could be achieved by going lower in the hierarchy of classes that facilitate the server-client communication as it was obvious that the client (product owner) wanted to have a customized web-communication facilities for the application. Another crucial assumption was made that the graphical user interface should resemble that of any commercial web browser in order to ease the learning process and not to impose any new views on the mental construct of a user. In other words, it should be easy and intuitive to use the final product. Yet another assumption was made, which influenced the preceding assumption: potential user of the application should have some basic understanding of the web browser concept and how it operates as final product might be not as intuitive for the beginner user. There were no explicit requirements for the product to support unexperienced user.

2 Requirements
Requirements table listed below presents requirements that were extracted from textual description of the task and which were stated explicitly in the task description. Distinguished points are separated into functional and non-functional requirements. Green checkboxes symbolise the achievement of listed requirements.
<table>
<thead>
<tr>
<th><strong>Functional</strong></th>
<th><strong>Non-functional</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application should be able to:</strong></td>
<td><strong>Application should be:</strong></td>
</tr>
<tr>
<td>✔ Display raw contents of the HTTP messages</td>
<td>✔ Accessible</td>
</tr>
<tr>
<td>✔ Handle response HTTP status codes:</td>
<td>Efficient*</td>
</tr>
<tr>
<td>200 OK</td>
<td>Fault tolerant*</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>Operable*</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>Recoverable*</td>
</tr>
<tr>
<td>404 Not found</td>
<td>Resilient to failures*</td>
</tr>
<tr>
<td>✔ Display response error messages</td>
<td></td>
</tr>
<tr>
<td>✔ Load favourites list on start-up</td>
<td>Robust*</td>
</tr>
<tr>
<td>✔ Load history list on start-up</td>
<td>Safe*</td>
</tr>
<tr>
<td>✔ Load Home page URL on start-up</td>
<td>Scalable*</td>
</tr>
<tr>
<td>✔ Maintain a list of requested URLs</td>
<td>Secure*</td>
</tr>
<tr>
<td>✔ Perform client-server communication on the background</td>
<td>Stable*</td>
</tr>
<tr>
<td>✔ Receive HTTP response messages</td>
<td>✔ Usable</td>
</tr>
<tr>
<td>✔ Request more than one URL simultaneously</td>
<td>✔ Extensible</td>
</tr>
<tr>
<td>✔ Send HTTP request messages</td>
<td></td>
</tr>
</tbody>
</table>

**Users should be able to:**

✔ Associate a name with each favourite URL
✔ Create Home page URL
☑ Delete a favourite from favourites list

☑ Edit Home page URL

☑ Jump to any previously requested page

☑ Load favourites URL from favourites list by clicking favourites associated name

☑ Modify favourites in favourites list

☑ Navigate to next pages

☑ Navigate to previously requested pages

☑ Save loaded URL to favourites list

☑ See all pages requested by the user

☑ Use buttons

☑ Use menus

☑ Use shortcut keys

<table>
<thead>
<tr>
<th>Functional:</th>
<th>25/25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-functional:</td>
<td>4/14</td>
</tr>
</tbody>
</table>

Table 1 Requirements table

As it can be seen from the table above, most of the requirements were met for discussed application. Requirements, marked with asterisk (*), were not explicit, albeit these points are the ones that are crucial for today’s software products and this is the main reason for including them in the table. Most of the points in non-functional requirements are not marked as resolved. To be able to state that following non-functional requirements are met, extensive testing: unit-testing, user-testing and testing in environment (considering external factors) has to be performed and extensive additional...
research conducted in order to be able to show how certain aspects were tested to support customers confidence of the software. Certain non-functional requirements are so superficial that it will never be possible to state with full confidence that they were entirely tested and met during the system development life cycle.

This report will give an overview of to what extent non-functional requirements, marked with asterisk (*), were met. The reader will be able to evaluate self the quality of application when reading testing section.

3 Design considerations

Before writing the actual code, following decisions were made regarding various aspects of application:

3.1. UI design

As mentioned before, GUI design was inspired by major commercial web browsers. Still a simple design choice had to be made, since every existing browser differs in one or another way and has some of its own particular features that are different from any other existing web browser. After a short search on the internet, following screenshot was found:

![Figure 1 Web browser design inspiration](image_url)
The found screenshot resembles Chrome web browser OS X version in its early years. The design of the desired product was influenced by this screenshot, but not too much.

The aim was to minimise the design as much as possible and use shortcuts to increase productivity for common functions. Existing UI design decisions are presented in the UI design decisions table below for more convenient lookup for the reader.

<table>
<thead>
<tr>
<th>UI design decision</th>
<th>Purpose</th>
<th>Reason/justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back and forward buttons</td>
<td>Navigate through the history of the browser</td>
<td>To support users existing metaphors and metal constructs. Resembles many available commercial web browsers.</td>
</tr>
<tr>
<td>(top-left)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address line</td>
<td>Accept user URL input</td>
<td></td>
</tr>
<tr>
<td>(top-middle)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Favourites icon</td>
<td>Show list of saved favourites</td>
<td></td>
</tr>
<tr>
<td>(top-right)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘+’ label/button</td>
<td>Add current URL to favourites</td>
<td></td>
</tr>
<tr>
<td>(between favourites icon and address line)</td>
<td></td>
<td>• Back and forward buttons must be thought of as one group – navigation.</td>
</tr>
<tr>
<td>Status ribbon bar</td>
<td>Display current status of particular web request</td>
<td></td>
</tr>
<tr>
<td>(bottom)</td>
<td></td>
<td>• Address line transforms user input to HTTP request on Return key pressed</td>
</tr>
<tr>
<td>Menu bar</td>
<td>Display various choices in</td>
<td></td>
</tr>
<tr>
<td>(top of the first tab when Alt key is pressed)</td>
<td></td>
<td>• ‘+’ label/button shows dialogue, where user has to specify a name for the new favourite item</td>
</tr>
</tbody>
</table>
the application

- Status ribbon bar shows when page is loading, when it finished loading and when error is received from the server.

- Menu bar allows user to choose between exit the application option and to show information about the application. Extra buttons could be added to this bar as more functionality will become available.

<table>
<thead>
<tr>
<th>Edit label/button</th>
<th>Create/Edit URL for home page</th>
</tr>
</thead>
<tbody>
<tr>
<td>(between back/forward buttons and address line)</td>
<td>(visible only on the first tab)</td>
</tr>
</tbody>
</table>

The first tab is purposely made separated from the rest of the tabs in the application. First tab, home tab, resembles a static home page tab, which would always load saved home page when the browser starts.

This concept is taken from many IDEs commercially.
available (i.e. RAD Studio, Eclipse, Visual Studio, Qt Creator, etc.), where home page is loaded up when user opens application to show welcome page, possibly with news from developers community or some RSS feed.

<table>
<thead>
<tr>
<th>Tabs</th>
<th>Browse through opened URLs and be able to add new ones</th>
</tr>
</thead>
<tbody>
<tr>
<td>(on the top of the middle component)</td>
<td>Tabs metaphor was user for this particular web browser since availability of UI component in Visual Studio allowed for sophisticated management of different instances of web pages loaded in the browser and used very little of the overall space on the available form area. Some additional control features were introduced to be able to navigate tabs. Such features include 'close button' represented by ([x]) character on every tab, which also copies behaviour of the most commercial web browsers available. Another feature included showing the URL directly on the tab for users convenience to be able to distinguish loaded URLs in different tabs and have</td>
</tr>
</tbody>
</table>
Design choices made: 8

Table 2 UI design decisions

Based on design choices discussed over, the current state of the application is shown below. Reader will be able to distinguish some of the components listed in Table 2.

![Current version design](image.png)

**Figure 2 Current version design**

### 3.2. Class design

(Class diagram is given in Appendix A.)

Some particular class design choices will be mentioned explicitly as it might be difficult to see the presented diagram.

First, main *WebBrowser* class, which extends *Form* class of C# framework and which contains all the design components, mentioned in the
Table 2. Structure of application was made so that all the graphical components used would lie inside this class and most of the important functionality would be splitted into accompanying classes, extending functionality of basic UI components.

Second, there is another class in the program structure, which extends Form class, namely, PromptBox. This class is represented as an abstract class, since it contains one abstract method – initialize(…); which is meant to be implemented in the subsequent classes, extending PromptBox subclass. The purpose of the PromptBox class is to be a base class for prompt dialogues, that should be used as communication medium between application and a user. That is why there are other three classes used in the application that extend functionality of the PromptBox class: ListPromptBox, InputPromptBox and InfoPromptBox that are used for various purposes in the application, based on their functionality (names of classes might suggest some knowledge of their use).

Third, once application starts, WebBrowser initializes two other classes: BrowserManager and RequestManager that encapsulate information and functionality that is, in case of former, such information as home page, favourites list, histories list and popped histories list are encapsulated with functionality operating on appropriate to them data. In case of latter, request-response information is processed and returned to the callee that has an instance of this class (in the case of web browser – WebBrowser class)

Fourth, supporting classes are used in the application, namely, WebPage class that contains all needed information about returned web page and additional information about the browser current situation and ResponseObject class that carries the data about response from the server.

In addition, RequestManager class contains an enumerator that is used to communicate information of the response type more effectively.

3.3. Choice of data structures

Not all chosen data structures will be mentioned in this report. Such information would take up a lot of unnecessary space and would not be interesting for the reader. It is worth mentioning that the most appropriate data
structures were considered in the development process, so that no unnecessary conversions between data types would have to be made, which has potential hazards of losing important information during conversions. This was considered a bad practice and very few of such instances can be found in the current implementation.

It is still worth listing some important choices made regarding data types that are not intuitive or may result in different choices among different developers. Reader is invited to make his own evaluation and, possibly, not agree with one or another choice of data type. Data type choice in the table below has certain structure to make lookup easier: [dataType][className]::[variableName]

<table>
<thead>
<tr>
<th>Data type choices</th>
<th>Purpose</th>
<th>Reason/justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>List&lt;WebPage&gt;</td>
<td>Holds list of currently loaded web pages</td>
<td>To be able to keep track of loaded web pages and reference them appropriately</td>
</tr>
<tr>
<td>BrowserForm::webPages</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ResponseObject</td>
<td>Keeps information about returned web page information returned after request has been processed</td>
<td>To be able to extract needed information, such as response code and response text</td>
</tr>
<tr>
<td>BrowserForm::response</td>
<td></td>
<td></td>
</tr>
<tr>
<td>InputPromptBox</td>
<td>References popup box that allows user to enter some text in it and this text will be returned to the caller</td>
<td>Custom data type was needed to be able to construct a custom form for the application</td>
</tr>
<tr>
<td>BrowserForm::prompt</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataTable</td>
<td>This datastructure is used to be able to create a data source for UI components that require such data type</td>
<td>At first, simpler ListView was considered for this purpose, but this choice lacked columns in its representation (which</td>
</tr>
<tr>
<td>Component</td>
<td>Description</td>
<td>为什么要这样做</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ListPromptBox</td>
<td>Popup box that contains a list of items presented to the user</td>
<td>Custom data structure was needed to be able to show a list of items in a popup box</td>
</tr>
<tr>
<td>InfoPromptBox</td>
<td>Static popup box with static text</td>
<td>Needed to be able to show a popup box with static text. Alternative solutions were possible, but due to availability of existing custom data structure, code was reused to produce desired result</td>
</tr>
<tr>
<td>RichTextBox</td>
<td>Displays loaded page content</td>
<td>Sophisticated component that allowed some additional properties, like enabling/disabling text, scrolling, text selection, etc.</td>
</tr>
<tr>
<td>XDocument</td>
<td>Used for storing information about homepage, favourites list and history list</td>
<td>Allowed to work directly with xml file, which can be found in C# framework. This resulted in shorter implementation of</td>
</tr>
<tr>
<td>Class/Method</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>List&lt;WebPage&gt;</td>
<td>Used for keeping lists of currently available history, favourites and popped history. List class has a very sophisticated interface with a range of very useful functionality, which was used in developing web browser. (i.e. Sorting, sub lists, where clauses, etc.)</td>
<td></td>
</tr>
<tr>
<td>Guid WebPage::UID</td>
<td>Used for mapping which tab would display which page (tabs are given Guid identifiers which are stored as tags for every tab). This design decision puts some restrictions on the implementation, but it is a safe choice to use, since it is an accepted fact that there is an extremely low chance to be able to cause collisions between two generated Guid values.</td>
<td></td>
</tr>
<tr>
<td>enum RESPONSE_CODES::RESPONSE_OK</td>
<td>Static representation for response codes was needed to be able to categorize responses from server. Enumerators are common for this kind of problem, when different categorisation of some uncertain data is needed in the application. Since it is certain that response codes are unlikely to be changed soon,</td>
<td></td>
</tr>
</tbody>
</table>
comparison can be used between response codes and static anticipated representations of possible return codes (i.e. RESPONSE_OK=200)

<table>
<thead>
<tr>
<th>Interesting data type choices made:</th>
<th>11</th>
</tr>
</thead>
</table>

Table 3 Interesting data structure/data type choices

3.4. Use of advanced language constructs

In order to be able to deliver a complex application, which would be efficient, resilient to failures, operable, robust, safe, stable and maintainable system, developer must utilise a range of advanced language constructs in development stages. C# presents its user with a wide range of possibilities and alternatives when thinking of constructs that are accumulated from different programming languages, like Java, C++ and other, to deliver the best practices of software programming. If user is not utilizing the full power of C#, then maybe it could be a sign that another language might be required for implementation of a particular application.

In implementation of presented application, a lot of basic and advanced language constructs were used in order to deliver desired functionality for the application. Some of the used advanced language constructs used include:

1. Overloading
   • Functions
2. Inheritance
   • Popup box
3. Overriding
   • Abstract methods
4. Abstract methods/classes
5. Exceptions handling
   - Processing web request
6. Encapsulation
   - Supplementary classes that are used by BrowserForm

Some of the important advanced language constructs were left out, namely, polymorphism and specialisation. This had been done deliberately, because this particular project did not require these features. The only possibility could have been to extend WebPage class to deliver histories and favourites, but since neither histories, nor favourites have any additional information, there were placed under WebPage class. Otherwise derived additional classes would have been merely copies of WebPage class, which would have been considered as a bad design choice.

3.5. Other performance-relevant choices
Threads are used as another performance-relevant feature. Every request is processed in a separate, independent thread, which merges with the UI thread once response is received and updates the user interface with the result from that thread. This allows user to be able to navigate application instead of waiting for response to arrive.

4. Guides
This section will begin with a use-case diagram that encompasses all the actions that the BuggyBrowser allows, what functionality is made available to the user and where the server comes into play.

   Diagram presented is rather complex, since all possible actions are identified under one boundary. Therefore colors were used for reader convenience. In the diagram, light green color identifies all the actions that the user is able to perform, light red circles show internal actions that are performed by the system without user knowing about them (hidden functionality) and light blue circles show internal resources to the application.
This diagram will be the basis for further discussions in user guide and developer sections.

Figure 3 Use-cases diagram

4.1 User Guide
Following section will describe textually and using screenshots from developed application on how to perform different, but important actions for achieving the desired goal using the application

4.1.1. Request URL
In order to request a URL, user has to open application, activate a new tab, enter address and press Enter key.
Figure 4 User opens BuggyBrowser

Figure 5 User opens a new tab
4.1.2. Save URL

User wishes to save current URL to favourites list. User clicks a ‘+’ symbol and sees a popup box. User enters desired name for current URL and clicks ‘Save’.
Figure 7 Save favourite popup box

4.1.3. View saved URLs

User wishes to look at saved URLs. User clicks a star icon.
Figure 8 Favourites list
4.1.4. Going back

User wishes to go back to previously loaded web site. Precondition: there are at least 2 web pages were loaded in one tab. (For the purposes of this demonstration, another web site was loaded after google.com – vision.hw.ac.uk)

Figure 9 2 pages currently loaded in one tab
User clicks back button and BuggyBrowser follows to google.com web site.

![Figure 10 State after back button is pressed](image-url)
4.1.5. Going forward

Now user wants to go forward to \textit{vision.hw.ac.uk}. This is possible with the use of forwards button.

![Figure 11 State after forward button is pressed](image)

4.1.6. ‘Jumping’ around history

User is able to ‘jump’ in history – not only visit previous page loaded, but any other page loaded during lifetime of a tab. Precondition must hold that there are > 2 web pages that were loaded before in the same tab window.

For demonstration web pages were loaded in the following sequence:

\texttt{google.com \rightarrow facebook.com \rightarrow vision.hw.ac.uk \rightarrow www2.macs.hw.ac.uk/~bm4}

In order to be able to view the list of previously loaded web pages, user must click and hold back button (long click). After three seconds, a popup window comes up and shows previously loaded web pages. User is then able to choose any web page from the list, click it and ‘jump’ to selected URL.
For this demonstration, first URL was chosen (google.com) for user to go ‘jump’ to. After performing ‘jump’, user is able to trace all the way forward to the his final loaded web page (facebook.com -> vision.hw.ac.uk -> www2.macs.hw.ac.uk/~bm4)

Figure 12 Loading few web pages in one tab window
Figure 13 Performing long click on back button (3 seconds)
4.1.7. Available shortcuts

Responses to shortcuts were implemented for user convenience. Such include:

- **Ctrl + R**
  Refreshes web page in currently loaded browser tab

- **Alt**
  Makes top menu visible/invisible

- **Alt + Left**
  Previously opened page within same tab is loaded

- **Alt + Right**
  Next page within same tab is loaded

- **Ctrl + F**
  Adds current page to favorites
4.2. Developer Guide

Based on points described in the user guide section, this section will provide insights into what is happening behind the scenes using some code fragments of interesting functionality used in the project, so that another developer might get more overview over how certain actions are achieved.

4.2.1. Requesting URL

When user requests a new URL, new thread is created and started. When response arrives, UI is updated from the background thread using anonymous method as delegate, which is run on the UI thread

```
private void URLAddressTextBox_KeyUp(object sender, KeyEventArgs e)
{
    if (e.KeyCode == Keys.Enter)
    {
        //try to load web page entered by user
        loadWebPage(false, true, false, URLAddressTextBox.Text);
    }
}
```

**Code 1** User presses Enter key in address text box

```
private void loadWebPage(Boolean refresh, Boolean saveInHistory, Boolean fromHistory, String url)
{
    Thread PageLoadingThread = new Thread(() =>
    {
        ResponseObject response = requestManager.getURL("http://" + url);
        this.Invoke((MethodInvoker)delegate
        {
            if (response.getResponseCode() == RequestManager.RESPONSE_CODES.RESPONSE_OK)
            {
                (RichTextBox)tabControl1.TabPages[tabControl2.Controls[0]].Text = response.getResponseString();
                (StatusStrip)tabControl1.TabPages[tabControl2.Controls[1]].Items[8].Text = "Finished loading" + "http://" + url;
            }
            else
            {
                //handle other than 200 responses appropriately
                ...
            }
        });
    });
    PageLoadingThread.IsBackground = true;
    PageLoadingThread.Start();
    updateGraphics();
}
```

**Code 2** method that starts request and awaits for reply from a server
public ResponseObject getURL(String url) {
    Stream responseStream = null;
    HttpWebResponse responseObj = null;
    try {
        HttpWebRequest request = (HttpWebRequest)WebRequest.Create(url);
        request.Method = "GET";
        responseObj = (HttpWebResponse)request.GetResponse();
        if (responseObj != null) {
            responseStream = responseObj.GetResponseStream();
            StreamReader responseStreamReader = new StreamReader(responseStream);
            return new ResponseObject((int)responseObj.StatusCode, new StringBuilder(responseStreamReader.ReadToEnd()).ToString());
        } else {
            return new ResponseObject((int)RequestManager.RESPONSE_CODES.RESPONSE_UNKNOWN, "");
        }
    } catch (Exception e) {
        if (e is WebException) {
            if (((WebException)e).Response != null)
                return new ResponseObject((int)((HttpWebResponse)((WebException)e).Response).StatusCode, e.Message);
            else
                return new ResponseObject((int)RequestManager.RESPONSE_CODES.RESPONSE_UNKNOWN, e.Message);
        } else {
            return new ResponseObject((int)RequestManager.RESPONSE_CODES.RESPONSE_UNKNOWN, e.Message);
        }
    } finally {
        if (responseStream != null)
            responseStream.Close();
        if (responseObj != null)
            responseObj.Close();
    }
}

Code 3 Actual request to server with response handling
class ResponseObject
{
    private int responseCode;
    private String response;

    public ResponseObject(int responseCode, String response)
    {
        this.responseCode = responseCode;
        this.response = response;
    }

    public RequestManager.RESPONSE_CODES getResponseCode()
    {
        switch (responseCode)
        {
            case RequestManager.RESPONSE_CODES.RESPONSE_OK:
                return RequestManager.RESPONSE_CODES.RESPONSE_OK;
            case RequestManager.RESPONSE_CODES.RESPONSE_BAD_REQ:
                return RequestManager.RESPONSE_CODES.RESPONSE_BAD_REQ;
            case RequestManager.RESPONSE_CODES.RESPONSE_FORBIDDEN:
                return RequestManager.RESPONSE_CODES.RESPONSE_FORBIDDEN;
            case RequestManager.RESPONSE_CODES.RESPONSE_NOT_FOUND:
                return RequestManager.RESPONSE_CODES.RESPONSE_NOT_FOUND;
            default:
                return RequestManager.RESPONSE_CODES.RESPONSE_UNKNOWN;
        }
    }

    public String getResponseString()
    {
        return this.response;
    }

    public override string ToString()
    {
        return …
    }
}

Code 4 ResponseObject class structure (contains information about the response)

As it can be seen from code snippets above, functionality that deals with client-server communication is separated into 4 distinct parts.

class ResponseObject
Contains information about response from a server

Fields:
    private int responseCode
    response code returned by a server
    private String response
    response text returned from a server

Methods:
    public RequestManager.RESPONSE_CODES getResponseCode()
    Matches response code with predefined enums in enum class and returns enum type
    public String getResponseString()
    Returns response string returned from
public override string ToString()

Returns a combined version, which contains enum name, response code taken from enum and response text

public ResponseObject getURL(…)

Performs actual request to a server and returns a ResponseObject

Parameter:

String url

URL of a web page that is requested by a user

private void loadWebPage(…)

Performs request to a server and updates UI according to response

Parameters:

Boolean refresh

Specified whether user refreshes current web page or not

Boolean saveInHistory

Specifies whether user wishes to save loading web page to history or not

Boolean fromHistory

Specifies whether requested web page is taken from history or not

String url

Specifies desired URL to be requested

private void URLAddressTextBox_KeyUp(…) Listens for users keys being pressed

4.2.2. Saving URL

When user saves a URL, new object of type WebPage is created and added both to the webPages list and to the history list.
4.2.3. Viewing saved URLs

When user views all saved URLs, **DataTable** object is created and its data set is used as a parameter for **DataGridView** UI component. In addition, listeners as set up to be able to capture users actions, like changing data, clicking cell and deleting row within the **DataGridView** object.

**Code 6** Favorites icon clicked

4.2.4. Going back

When user presses back button, the last element of filtered list is popped, returned to the **BrowserForm** and added to the poppedHistory run-time
instance of the list, which is flushed once the main window is closed and all important data is saved into .xml file

```csharp
private void backBtn_Click(object sender, EventArgs e)
{
    loadWebPage(false, true, true, browserManager.popLastHistory((Guid)tabControl1.SelectedTab.Tag).getUrl());
}
```

Code 7 Back button click event

```csharp
public WebPage popLastHistory(Guid tabTag)
{
    WebPage previousPage = this.histories.Where(page => page.getUID().Equals(tabTag)).ToList()[this.histories.Where(page => page.getUID().Equals(tabTag)).ToList().Count() - 2];
    this.poppedHistory.Add(this.histories.Where(page => page.getUID().Equals(tabTag)).ToList()[this.histories.Where(page => page.getUID().Equals(tabTag)).ToList().Count() - 1]);
    this.histories.Remove(this.histories.Where(page => page.getUID().Equals(tabTag)).ToList()[this.histories.Where(page => page.getUID().Equals(tabTag)).ToList().Count() - 1]);
    return previousPage;
}
```

Code 8 PopLastHistory method

## 4.2.5. Going forward

When user presses forward button, the last element from filtered poppedHistory list is popped, returned to the BrowserForm and is appended to the history list.

```csharp
private void fwdBtn_Click(object sender, EventArgs e)
{
    loadWebPage(false, true, false, browserManager.popLastPoppedHistory().getUrl());
}
```

Code 9 Forward button clicked

```csharp
public WebPage popLastPoppedHistory()
{
    WebPage lastPoppedHistoryPage = this.poppedHistory.Last();
    this.poppedHistory.Remove(lastPoppedHistoryPage);
    return lastPoppedHistoryPage;
}
```

Code 10 PopLastPoppedHistory method
4.2.6. Saving the state / Restoring the state

When user closes main window, state of the `BrowserManager` instance is saved into `.xml` file using `XElement` object, which allows to create a structure recursively and thus requires shorter implementation to store desired data.

When user starts main window, state is restored from `.xml` file using `XDocument` object to recursively select structured data from a file.

```csharp
public BrowserManager Read()
{
    if (File.Exists("browser_data.xml"))
    {
        XDocument xdoc = XDocument.Load("browser_data.xml");
        this.favourites = (from o in xdoc.Element("BrowserManagerData").Element("Favourites").Elements("Favourite")
                            select new WebPage(new Guid(o.Element("UID").Value), (String)o.Element("URL").Value, (String)o.Element("Caption").Value))).ToList();
        this.histories = (from o in xdoc.Element("BrowserManagerData").Element("Histories").Elements("History")
                           select new WebPage(new Guid(o.Element("UID").Value), (String)o.Element("URL").Value, ""))).ToList();
        this.homePage = (String)xdoc.Element("BrowserManagerData").Element("Home");
    }
    return this;
}
```

**Code 11 Read stored file to restore state**

```csharp
public void Write()
{
    if (File.Exists("browser_data.xml"))
    {
        File.Delete("browser_data.xml");
    }
    XElement data = new XElement("BrowserManagerData",
                                new XElement("Favourites",
                                             from p in this.favourites
                                             select new XElement("Favourite",
                                                                     new XElement("UID", p.getUID()),
                                                                     new XElement("URL", p.getUrl()),
                                                                     new XElement("Caption", p.getCaption()))),
                                new XElement("Histories",
                                             from p in this.histories
                                             select new XElement("History",
                                                                     new XElement("UID", p.getUID()),
                                                                     new XElement("URL", p.getUrl()))),
                                new XElement("Home", this.homePage));
    data.Save("browser_data.xml");
}
```

**Code 12 Write current state**
4.2.8. XML structure

```xml
<?xml version="1.0" encoding="utf-8"?>
<BrowserManagerData>
  <Favourites>
    <Favourite>
      <UID>7ab4c3e1-47f3-44be-a678-21dab81c1225</UID>
      <URL>google.com</URL>
      <Caption>Google</Caption>
    </Favourite>
  </Favourites>
  <Histories>
    <History>
      <UID>7ab4c3e1-47f3-44be-a678-21dab81c1225</UID>
      <URL>google.com</URL>
    </History>
  </Histories>
  <Home>vision.hw.ac.uk</Home>
</BrowserManagerData>
```

Code 13 XML file

4.2.7. ‘Jumping’ around history

To make ‘jumping’ in history possible, `reduceHistoriesBy(…)` method was introduced. It filters histories list, and starts removing history items one by one and adding removed items to popped histories list until it reaches the parameter specifying by how many steps user wishes to jump in history.

```csharp
public void reduceHistoriesBy(Guid selectedTabUID, int amount)
{
    if (this.histories.Where(page => page.getUID().Equals(selectedTabUID)).ToList().Count() - (e.RowIndex+1))
    {
        int i = 0;
        foreach (WebPage page in this.histories.Where(page => page.getUID().Equals(selectedTabUID)).ToList().ReverseWebPage())
        {
            if (i < amount)
            {
                this.poppedHistory.Add(page);
                this.histories.Remove(page);
                i++;
            }
        }
    }
}
```

Code 14: ‘Jumping’ around history

`Variable ‘e’ comes from cell click event from DataGridViewCellEventArgs class, which is used to get item number in a grid to be able to deduce by how many items in history user would want to jump.`

---

1 Variable `e` comes from cell click event from DataGridViewCellEventArgs class, which is used to get item number in a grid to be able to deduce by how many items in history user would want to jump.
5 Testing

This section will compare desired results of certain actions with the actual results, which application will provide for this report. Table below identifies most important test cases for the application based on requirements listed in earlier section. Although all the test cases are of equal importance, every test case will be given an importance value in a range between 1 and 3, where 1 will symbolise low impact on the overall application and 3 – high impact. Importance value will give reader some information regarding how much attention was given to every requirement in the development process and the reader would be able to assess how well every requirement was achieved.

<table>
<thead>
<tr>
<th>Test case</th>
<th>Expected result</th>
<th>Actual result</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display raw HTML in the application</td>
<td>Be able to see raw HTML in the application</td>
<td>Raw HTML can be displayed in the application in a component on the main window, which takes the most of the space of the overall application</td>
<td>3</td>
</tr>
<tr>
<td>Handle HTTP response messages</td>
<td>Be able to handle any possible response message from server (exhibit adaptive behaviour)</td>
<td>Any possible response message potentially could be received from a server. Application tested receiving various response codes from servers. Timeouts, wrong user input, empty responses were tested as well</td>
<td>3</td>
</tr>
<tr>
<td>Display error response</td>
<td>Be able to cope with variety of responses</td>
<td>Application handles many possible</td>
<td>2</td>
</tr>
<tr>
<td>messages and display meaningful content to a user of application responses from a server and only predefined response codes give meaningful response to a user. Otherwise ‘unknown error’ is returned</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load favourites/history list on application start-up</td>
<td>Application should be able to load favourites/history from a file (precondition: file must contain some favourites/history items) and display them to a user of the application In case if file does not exist, application must not crash, but handle such case gracefully File could be corrupt, so application must be able to cope with such case as well</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load home page URL on start-up</td>
<td>Application must load stored home page URL from a file and display it to</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application loads favourites/history from file. Application does not crash if file does not exist. Application does not load anything if file is empty or if it does not exist. No tests were conducted in case if file that contains favourites/history is corrupt in some way (many possibilities exist for a file to be corrupt – a study must be performed first in order to find out how file could be corrupted to force application to fail)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Application loads stored home page URL from a file and displays it to a user.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feature</td>
<td>Requirement</td>
<td>Implementation</td>
<td></td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Maintain a list of requested URLs</td>
<td>Application must be able to maintain a list of requested URLs to be able to reference them and supply as inputs to functions that process requested URLs to do some other work</td>
<td>Application maintains a list of requested URLs. Correctness of the maintenance of the list was tested empirically (not formally). Application is able to reference list items for other purposes</td>
<td></td>
</tr>
<tr>
<td>Perform client-server communication on the background</td>
<td>Application must be able to perform client-server communication on the background and have control over what is currently</td>
<td>Application processes user requests in separate threads. Threads are run on the background and application is able to stop all background</td>
<td></td>
</tr>
</tbody>
</table>
being processed and which responses were already received and processed by the application. Application must keep its memory under control and not let memory overloads due to huge amount of requests being processed simultaneously. Application must be able to stop/pause/resume background threads when it is needed to free memory or to have control over what is being processed. Application does not have control over which threads are being run at a particular moment since it does not have an overview over running background threads that deal with client-server communication. Application relies on the Garbage Collector in case of freeing memory if such is needed. User has a feedback on executed background process only when UI is updated from a background thread. Application is able to request more than one URL simultaneously. Testing was done by observation and the most of 10 background threads were run at the same time to receive a response from a
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
<th>Test Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associate a name with each favourite URL</td>
<td>Application must be able to associate a name with each favourite saved in the favourites list. Application must be able to cope with any possible character entered.</td>
<td>Application is able to assign a name for every favourite in the list. It is uncertain which characters will possibly cause an error in the application. It is assumed that application must cope with all possible Unicode characters. Such behaviour will have to be tested to gain more confidence.</td>
</tr>
<tr>
<td>Delete favourite/history items from favourites/history lists</td>
<td>Application must be able to cope with deletion of elements from a list and never cause <code>OutOfBounds</code> <code>RunTimeException</code></td>
<td>Application was tested for removal of elements from lists in different occasions, never <code>OutOfBounds</code> exceptions were reached. Since this type of exceptions are very</td>
</tr>
</tbody>
</table>
common in software development projects, application must be further tested for possible occurrences of this type of error in all possible states. This can be done by formal testing.

Jump to any previously requested page and go forward

Application must be able to navigate user to any page selected from history list and be able to forward user. Since application deals with lists in this case and loading of URL, same expectations are required as for test cases described previously

Application is able to navigate user to any web page in the history and next page in the list associated with forward button. Behaviour was tested by observation navigating through forward and back-lists. Formal testing is required further to gain more confidence in functionality

Close tab while page is loading

Exit the loading thread

Thread is still running after tab is closed

| Test cases identified: | 12 |
| Test cases tested: | 12 |
| Errors detected: | 1 |
| Potential errors detected: | 8 |
| Average importance: | ~2.5 |

Table 5 Test cases


6 Conclusion

This report gave an overview over the simple web browser application, developed in C# programming language.

First, task and purpose of the application were discussed; application had been given a name of ‘BuggyBrowser (BB)’ based on its current status. Further assumptions were given explicitly to avoid possible misunderstanding of requirements.

Second, requirements were identified and stated explicitly so that the reader would be able to assess whether application is compliant with the requirements list. Such assessment could be conducted comparing requirements list with design section and guides section, where reader is able to choose those subsections that fit his level of knowledge of C# and software engineering in general.

Third, design considerations breaks down current implementation into components. This is done to support readers’ analysis of the current application.

Fourth, guidelines section is split into two subsequent parts, that is suitable both for a user that wants to learn how to use the application and for a developer that wants to extend current implementation or to learn how different components are implemented.

Fifth, results from testing stage of development life cycle are presented, where current state of the application is tested and results are presented for reader to gain confidence in existing application. In addition, testing section provides possible failures that may occur when using the application. This information is mostly suitable for reader that wishes to extend current functionality. It is strongly advised to perform more testing before adding more functionality to the code.

There are many things that could have been done differently in the application. Child classes to WebPage class could be implemented to serve both history objects and favourites objects. Another improvement would be to
refactor the existing code to make shorter methods. Yet another improvement
would be to reduce amount of flags used in the application for different
purposes, so that the code could be more comprehensible than it is right now.
In addition, instead of using Guids, another application design might have a
better and more robust solution that it is at the moment.

Despite the desired improvement of the application, current application
has a relatively good solution, which meets the requirements and delivers
fairly decent solution, which can be used even by the novice user.

Implementation-wise, current solution contains a moderate amount of
advanced programming language constructs. Considerable amount of time
was paid to implementation of history and favourites, which has a nice touch
to the overall application. In addition, developer is proud of the most of
existing functionality in BrowserManager class that deals with favourites and
history items.

Overall, it is considered that the current application is consistent with
the requirements and assumptions made, that either clarify the requirements
or make additional points that are considered important for this application.
Many hours were used for testing and results revealed that application can be
used without any additional changes, so the reader is invited to try it out!
7 References

Design patterns : elements of reusable object-oriented software (by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides)

http://stackoverflow.com/questions/782274/using-c-sharp-methodinvoker.invoke-for-a-gui-app-is-this-good
Appendix A