Milestone 3 : Narrative and Reflective Document

Matthew Pike, 523355@swansea.ac.uk
Supervisor: Dr Max Wilson

Swansea University
Prifysgol Abertawe

Swansea University
Computer Science Department

Contents

1 Introduction

In this document we will provide a narrative and reflective account of the development process for the User Study System. The primary purpose of this document is to describe the problems we encountered during the development, and the steps we took to overcome these obstacles. Additionally, we looked at the other reflective aspects, such as lessons learnt, future development and successes.

Term Definition

As with any technical project, there exists numerous technical abbreviations that are used in place of long descriptors. We present a table below of the various abbreviations used in this document, which should serve as an aid to the reader.

Term	Definition
Visualiser	The part of the system responsible for displaying the data visualisation.
Document	The web site or web based document being investigated in the user study.
Experiment	The study that is being conducted. Typically will contain a User and a Conductor.
Data Source	A device/software/location that is recorded by the Web Browser.
Recorded Instance	Since a single experiment on a single user can contain numerous conditions and tasks, we must distinguish what one unique combination of these properties are called. We have chosen 'Recorded Instance' to represent this. A Recorded Instance is an identifier for: Experiment -> User -> Condition -> Task.
Stack	A Stack is a part of the Visualiser UI that represents a single Recorded Instance. A Stack is therefore a UI component.
WPF	Windows Presentation Foundation - The visual framework used to develop the user interface(s) in this project.
ORM	Object Relational Mapping - a way of linking entries in a database to a usable format in application code.
SQLCE	SQL Compact Edition, the embedded database product we use to store application data.

Personas

In addition to technical abbreviations, the project features unique individuals that partake or feature somehow in the system. Below is a table documenting these individuals and their role in the system. Again, this table is meant as an aid to the reader.

Individual	Role
Conductor	The person responsible for performing the user study. This is typically a researcher who is aiming to prove a particular hypothesis.
Researcher	The person who is responsible for gaining insight from the user study. The Researcher and Conductor can typically be the same person, however this is not always the case, especially in a large research group. The Researcher is therefore a member of the research team who is wishing to use the data collected from the study.
User	The person who is sitting the user study. Their responsibility is to perform tasks provided to them by the conductor.
Client	The person who will be receiving the finished software project. In this case it is Pingar.

2 General Problems

As with any project, the development of the USS had some general / logistical problems. In this section we will detail these general problems and describe our approach to solving them.

Code Management

We knew prior to beginning the project that some sort of version control and backup system should be used in order to safeguard our work. Having not used version control before, we based our decision on the availability of resources (tutorials, books, forums, etc.) and the number of services available. We decided upon using Git, a distributed version control and source code management system. Since the system is distributed, source code can be committed both locally (on a local machine) and to a remote, central server. For the hosting of our source code repository, we used BitBucket (https://bitbucket.org/), as it allowed us to set up a private repository at no cost. Additionally backup came from developing out of a Dropbox (https://www.dropbox.com) folder.

We found Git to be a very useful and unobtrusive system. We were especially fond of the ability to roll back to previous points in development of the project, when a prototype or approach did not work out. We found that Git complemented our Prototyping development methodology also, allowing us to create and work out of testing branches, without affecting the "Stable" (i.e. working) version of the system.

Hardware

The USS system incorporates two hardware devices (apart from the PC upon which it runs), namely the Emotiv EPOC brain scanner and the microphone based recording device. While the latter of the two proved to be extremely straightforward, thanks to the support for the device in windows, the EPOC brain scanner proved to be less compliant. We experienced numerous difficulties with the device, such as the Emotiv software not detecting the device, the device itself not functioning as expected and intermittent connection issues. It was a difficult problem to solve since the hardware is outside the control of the developer. We noted however, that upgrading to the latest software provided by Emotiv seemed to reduce, but not completely remove, some of the difficulties we experienced with the device.

Time Constraints

After drawing the initial requirements in Milestone 1 of the project, we knew that

the functionality we were intending to build in the available time-frame was very ambitious. We set ourselves a challenging schedule which was unrealistic at times, but proved to be largely successful in general. Our primary way of overcoming this problem was through careful planning and design. A primary example of a significant time-saving comes from the DAL component of the system. Without this reusable data access component, we would have had to create readers and writers for each entity in the system, causing massive duplication and consuming a lot of time. There are numerous design features like this that have contributed to the project being completed on time.

Communication

The final, and possibly most prevalent problem, was communication between the developer and the client - Pingar. Since we are working with Pingar who are based in New Zealand, the time difference meant that we were waking as they were going to sleep and vice versa. Email became our primary form of communication and we found that regular updates were crucial in ensuring that the project remained on target. We also found that videos demonstrating some new functionality were extremely helpful to convey progress. We produced many videos that provided a visual representation of the progress. We found that the client was especially fond of the videos and we would continue this practice with future projects.

3 Methodology Review

For the development of this project we chose to use the prototyping methodology. The methodology is Agile based, incorporating unit testing, disposable prototypes and an iterative development cycle[1]. We found that the methodology had both positive and negative attributes when applied to the development of the USS system.

We found that being test driven was particularly important during the development of the USS. We developed an iterative cycle that centred around testing. When adding a new feature, we would follow these steps:

1. Create a new branch in Git so that we always have a working version of the USS to fall back to. (The feature was developed in its own branch.)
2. Create a new set of tests targeting this new functionality explicitly.
3. Implement the new functionality in code.
4. Run all USS unit tests. This ensures that the added functionality does not break existing (working) code.
5. Review the results.
    a) Fails Tests - Investigate why and fix if possible. (Possibly the approach is incorrect.)
    b) Passes Tests - Continue to next feature.

This cycle also conforms to the prototyping methodology. We see at stage 5 that we review the implementation. If it fails the tests then we have to consider whether or not this approach is correct. If not, then the methodology states that the prototype should be disposed, but the knowledge gleaned from the exercise of constructing the prototype should be applied to future versions.

One final benefit of this approach is that we always have something to show the client. Since the prototypes are quick and rough versions of the finished product, the client is free to criticise and explore various avenues without having wasted a large portion of the developer's time. We found that this was especially true for this project, which evolved significantly from the feedback that was received on prototype versions of the project.

[1]For further detail on the Prototype methodology, please refer to "Requirements and Methodology" document in Milestone 1 of the project.

There was one particular negative with the chosen methodology however, and that was the tendency on our behalf to overdevelop a prototype. In a project that was severely constrained for time, having a methodology that does not enforce a definitive schedule was definitely missed on this project. We found that the freedom to develop a prototype, that could ultimately be dropped, was difficult to constrain, since the developer wanted to have as functioning a prototype as possible. We would improve future projects by using a methodology that allows prototypes, but also incorporates a form of governance on the effort invested upon the prototype. Rapid Application Development is perhaps a suitable candidate for future projects.

4 Web Browser

In this section we will reflect on the development process of the Web Browser section of the USS. We will reflect on the actual stages of development that occurred and discuss the issues we encountered. Additionally we will look at the lessons we have learnt as a result of this process. We will also describe the future work and successes of this section of the USS.

4.1 Development Process

In this section we will detail the stages of development that occurred during the construction of the Web Browser. For each part of the Web Browser's functionality, we followed the iterative cycle described in chapter 3. In the proceeding paragraphs we describe both the process and the order in which the sub-components that form the Web Browser were developed.

Data Abstraction Layer (DAL)

The USS project began by developing the DAL which would hold the system's User Study data. This data includes everything from Participants partaking in the User Study, conditions and tasks for a given study, as well as the data sources being recorded during the study. The DAL therefore was the backbone of the entire USS system. We were conscious however of the fact that the DAL does not perform anything new or "interesting", it is simply a database. Therefore we wanted to develop the DAL, quickly but in a manner that allowed for code reuse, reliability and most importantly flexibility.

Key to fulfilling these properties was the Object Relational Manager (ORM) framework - Entity Framework. The framework allowed us to completely ignore the issue of actually storing the data, we simply needed to focus on defining what the data was. The framework is designed such that developers produce entities, which are simple POCO (Plain Old CLR Objects which means just a simple class) classes. We can define relationships by referring to other classes through fields in a class. This simple approach meant massive productivity in a short space of time. The real benefit however of the framework, was that it creates the database for us and handles all communications between the application and DB. This approach follows the Repository Pattern and proved extremely successful for this project. The DAL

was the fastest component to develop and proved reliable throughout the various stages of development.

Setup User Interface

Having completed the DAL, we then needed to produce a graphical interface for Conductors to setup and run user studies. We experimented with the Model View View Model (MVVM) design pattern when developing the User Interface (UI). The pattern separates the data and view logic from one another, and "glues" the two together via the View Model. We made a particular effort in ensuring that the visual appearance of the interface was of a high quality, and modelled it on the Cosmopolitan UI demonstrated by Microsoft in its Zune software. Creating a visually attractive interface took considerable effort and required a significant amount of time to perfect. However, the Setup component's development was fairly straightforward, helped in no small part by the DAL which made interfacing with the system database a straightforward task.

Web Browser and Collectors

Finally came the Web Browser component. The Web Browser was the component responsible for collecting the various data during a user study. Development began by developing a simple web browser using the Windows Presentation Foundation (WPF) user interface library. This was fairly straightforward thanks to an existing control in WPF, which provides a standard Web Browser rendering engine. We simply needed to add the additional UI components, such as back and forward buttons and an address bar. The difficult part came in collecting the data. We developed the collectors in order of easiest to most difficult, and as such began by collecting the participant's mouse position on screen, before progressing to collect the data from the brain scanner. Whilst progress on developing the collectors was smooth, we soon discovered that with all collectors running simultaneously, the application was severely impeded in terms of performance. We spent significant periods of time optimising all collectors, looking especially at time consuming operations such as IO and processing of data.

4.2 Problems

In this section we will detail the problems we encountered during the development of the Web Browser and the steps we took to overcome these issues.

Entity Framework and SQLCE

Despite the Entity Framework being a massive aid to the project as a whole, there were some initial configuration issues in using the framework with our chosen database - SQL Compact Edition (SQLCE). The database is embedded, meaning it does not require a server to be installed on the user's machine in order to run, it simply operates out of a single "flat" file. The difficulty however was interfacing the Entity Framework to target the embedded database. After significant research and experimenting, we discovered that a particular connection string worked regardless of the machine the application was running upon.

Designing the Setup User Interface

We stated that a goal for the Setup User Interface was to appear modern and aesthetically pleasing. We had decided upon the style which we wanted to pursue on the interface fairly easily, the difficulty however came in scaling the design to work at various sizes and resolutions. Our first attempt worked perfectly on our development machine, but when resized or run on a different system, text would begin to overlap and certain interface features would be hidden behind others. We discovered that our initial approach was flawed. We had based our approach on a low level container called **Canvas.** This container allows a developer massive freedom to add features to an interface without any practical restrictions. The problem with **Canvas** however is that the appearance is completely unmanaged. That is, when a user resizes the window or the application is run at a different resolution, then the container does not scale the interface accordingly. To overcome this issue, we re-engineered the interface so that the styling occurred within a managed container (**ViewBox**), and thus scaling was managed automatically, solving this issue.

Collector's Performance

The major issue encountered during the development of the Web Browser component was the performance of the data collectors at runtime. In Milestone 1, we identified this as a potential problem and added the requirement that during a User Study, a Participant's interaction with the Web Browser could not be impeded in anyway whatsoever. Solving this issue was simply a case of identifying the parts of code that were "expensive" and optimising them to reduce their burden on the application's performance.

We immediately identified one aspect of the Collectors that could not be changed - the incoming data. The data being collected was voluminous, but it all needed to be recorded. Knowing this, we began to investigate the sections of code responsible for writing the collected data to file. We knew that this would be the likely source of the performance hit, since writing to disk and IO in general is expensive (in terms of performance). The majority of the data was being stored in XML format, so to

achieve the most performance gain, we focussed our attention on optimising XML writing to file.

Our original implementation for writing XML used the ***XDocument*** library. The library was chosen since it provided many useful utility functions, which greatly reduced the amount of code that needed to be written. However the cost of this ease of use was performance. After researching and performing our own performance testing, we switched to the lower level ***XmlWriter*** library. This greatly improved write performance (~30x improvement), but also required a lot of additional coding to compensate for ***XmlWriter*** lack of utilities/helper methods. This was ultimately achieved and improving the XML write performance benefited the entire Collectors component, making collection unnoticeable during a User Study.

Full-Page Screen Capture

Another tricky problem encountered during the development of the Web Browser was capturing full-page screen shots of a web page. We hoped that such functionality would be built into the Web Page renderer already, and it would simply be a method call to save the rendered page to an image file. This was not the case however and a significantly more complex solution was required. After researching the issue, we discovered that the only possible approach was to programmatically scroll a web page part by part, capturing each individual part as a separate image. Finally these individual parts would be "stitched" together to form a complete image. We developed this functionality and all seemed to work well. However, we had not covered all eventualities. During the development, we had presumed that a web page was simply "long" and not "wide", that is we presumed that we only needed to scroll vertically. However, during testing we discovered that pages did indeed need scrolling horizontally as well as vertically. The solution to this essentially required the same approach as we had taken in our original implementation, but it needed to systematically scroll horizontally as it moved vertically.

4.3 Reflection - Lessons Learnt

In this section we will reflect on the lessons that we have learned as a result of developing the Web Browser.

Importance of a Good IDE

Throughout the development of the USS we used Visual Studio 2010 Ultimate Edition with the following plugins:

- ReSharper
- SQLCE Explorer

- Productivity Power Tools
- Git Extensions

We found the IDE extremely helpful and in certain situations indispensable. A primary example from the development of the Web Browser component was the performance analysis functionality built into VS2010. Using the performance monitor we were able to identify where our application was incurring its performance penalty. Additionally, after making the necessary changes, we were able to confirm the success and suitability of the alterations by running the application through the performance monitor once again.

Time Invested on Setup Application

Upon reflection we realised that too much time was spent on making the Setup application visually appealing. The lesson learnt here is to complete the functionality of a system before concentrating on the visuals of the application.

Document During Development

During the earlier stages of development, as developers we failed to fully document the code we were developing, instead focussing on developing the next bit of functionality. Of course we ultimately paid the price (in time) when it came to revisiting the undocumented code and attempting to comprehend its purpose/function. We learnt this lesson early on in the development of the system, and did not and shall not attempt this short-cut again.

4.4 Future Developments

Given the opportunity to continue working on this project, or projects similar to the Visualiser, our general approach would be very similar to that taken here. In addition to this approach, using the knowledge and experience we gained through this project, we would make the following changes:

Improve Screen Capture and Renderer

We noted in section 4.2 that our current approach for capturing an entire web page is far from optimal. We are currently limited by the IE9 rendering engine we are using, since it does not allow for full-page capture. We have however researched alternative rendering engines, such as Awesomium (http://awesomium.com/), which does allow this functionality. This would require a significant re-engineering of the current system, but is something that is achievable.

Using an alternative renderer such as Awesomium would also add additional functionality that is not present in IE9, such as standards compliant rendering and modern HTML5 capabilities.

Improve Plugin Architecture

When designing our current plugin architecture, the emphasis was placed on simplicity. We wanted plugins to be simple, to develop and to integrate into the system. We believe that this has been achieved. However, the architecture does not support advanced plugin functionalities, such as dynamic loading and runtime integration. To incorporate this, we would need to build the architecture upon existing plugin frameworks such as MEF (http://mef.codeplex.com/).

Support Additional Brain Capture Devices

Currently the system only supports the Emotiv EPOC brain scanner. Given additional time, we would look into integrating additional devices. Devices using less sensitive measurement technology, such as fNIRS, could be incorporated into the USS to improve the quality of the output.

Improve Saving Format

Currently the USS saves all data to a Zip file. Zip was chosen because of its relative simplicity, good compression ratio and being a recognised standard. There is a performance issue with the format however, since the contents need to be extracted in order to be used. A custom file format will allow for object serialisation, giving a similar level of compression without the IO operations required when using the Zip file format.

4.5 Successes

In this section we will detail some of the successes of the Web Browser section of the USS.

Test Driven Development

We definitely see the value of introducing tests into the development cycle as early as possible. As a result of using this testing methodology during the development of the Web Browser, we discovered bugs early on, before they became obfuscated by other functionality. This was an extremely valuable property as it undoubtedly saved hours of debugging.

We found that the suite of tests that were developed for the Web Browser also helped with regression testing. On numerous occasions we discovered that a new piece of functionality broke existing functionality. Without the tests in place, we would not have discovered this until later on in the project's development, which would have introduced further complexity to the problem.

Chosen Platform

We believe that the .Net platform was the correct platform for the Web Browser to be developed upon. The platform itself is well documented and has a number of third party libraries available. Additionally the tooling support for the platform, Visual Studio in particular is an invaluable tool which is an incredible aid when developing a complex project such as the USS.

Entity Framework

The Entity Framework proved to be extremely successful in this project. The ability to abstract away from the actual storage of data and concentrate on the data itself saved invaluable hours in development time. The library also improves the reliability of the project, since it is well tested and used by thousands of developers worldwide. We would definitely consider using the framework in future projects.

5 Visualiser

In this section we will reflect on the development process of the Visualiser section of the USS. We will reflect on the actual stages of development that occurred and discuss the issues we encountered. Additionally we will look at the lessons we have learned as a result of this process. We will also describe the future work and successes of this section of the USS.

5.1 Development Process

In this section we will detail the stages of development that occurred during the construction of the Visualiser. For each part of the Visualiser's functionality we followed the iterative cycle described in chapter 3. In the proceeding paragraphs, we describe both the process and the order in which the sub-components that form the Visualiser were developed.

Construct Visualiser Manager

We began by creating the Visualiser manager. This class was responsible for performing the administrative duties required for the Visualiser to operate. This includes loading/saving recorded User Studies, calculating and creating temporary directories and numerous other duties. The class was constructed first as it serves the proceeding classes.

Prototype Each Feature

Now that we have a class that is responsible for loading a recording of a User Study, we can begin to build the functional components of the Visualiser. Following our methodology we created each feature as its own prototype. That is, instead of attempting to integrate each feature as it was built, we simply developed each feature within a sandbox. This allowed us to concentrate on developing the functionality, before attempting to integrate. We developed the Visualiser's functionality in the following order:

1. **Brain and Web Event Data Visualisation** These two features were developed together since they share the same chart.

2.Audio Waveform Visualisation This was adapted from an existing open source project (http://wpfsvl.codeplex.com/). This required significant rewriting, since it was not designed for more than one instance of the component to be executed at the same time.

3.Mouse Trail Visualisation

4.Heatmap Visualisation

Each feature was developed in its own project file, meaning that they were completely independent entities. The next task was integrating into a final application.

Docking and Integration

The final stage required the integration of this functionality into a single application. This stage began by constructing a container window. We used the AvalonDock framework (http://avalondock.codeplex.com/) to provide the necessary docking functionality. Onto this we created a recording display window, which would reside in each tab. We then added each prototype into the application.

5.2 Problems

In this section we will detail the problems we encountered during the development of the Visualiser and the steps we took to overcome these issues.

Experiment Properties Hierarchy

One of the first problems we encountered was how to allow Researchers to select recorded studies and load them into the Visualiser. This was not as much a technical problem, since all of the data was easily accessible from the DAL. It was more of a representational issue - how do we make it easy to select recordings based on the user study properties. After conferring with the client, we decided upon a hierarchical tree structure. We had originally envisaged loading the recording's Zip file into the Visualiser and proceeding from there, but this approach was not pursued as it required the Researcher to remember each recording manually.

Brain / Emotion / Web Event Data Chart

When developing the Brain and Web Event prototype, we soon encountered the problem of Visualising the two sets of data. We quickly learnt that there were no existing third party libraries that deal with this kind of visualisation. Instead we had to adapt a Stock chart component provided by AmCharts (http://www.amcharts.com/). Whilst the chart was fairly reliable for plotting the emotion data, we had great difficulty in plotting the Web Events on the chart. The problem was compounded

by the fact that support for the chart had been discontinued in 2011. The root of the problem was that when a displayed event become invalidated, it was still being displayed on the chart. We eventually found that invalidating all displayed events, followed by the redrawing of the entire chart, would indeed remove the events. Of course this approach is far from optimal, as redrawing the chart each time a new web event needs to be displayed has an associated performance hit, but this is not noticeable at runtime.

Saving Modified Web Event Details

As a requirement from Milestone 1, the Visualiser has to allow Researchers using the application to add or modify events. The Visualiser allows this interaction through a data grid in the centre of the application. However the difficulty came when attempting to save these modifications. Since the Web Events data is stored in an XML file within the recordings ZIP file, we had to figure out a way of updating that file to represent the changes. The solution required the rewriting of the entire XML file, since there was no convenient way of updating only the events that been added or modified.

Visualising Mouse Trails Efficiently

In our prototype version of the Mouse trail visualisations, we quickly developed a working example that produced the desired output. The problem however was efficiency; certain situations caused the prototype to take up to 3 seconds to generate the visualisation. We saw few ways of improving this situation. We believed that sampling the data would improve the efficiency of the prototype, but would also have an unacceptable impact on the data resolution, i.e. it was unacceptable to lose "interesting" data. Analysing the data however, revealed an interesting fact that much of the data was duplicated. When a participant is interacting with a web page, the majority of the cursor positioning remains constant, with only sudden, defined movements occurring. However, when visualising this data, the prototype was drawing all recorded points (the majority of which did not change, but were still being drawn). By filtering this "non-movement" out of the data, the generation of the visualisation was almost instantaneous.

Heatmap Generation

The primary difficulty with generating the Heatmap was knowing where to begin. Having never developed this type of visualisation before, we were uncertain of the approach that should be taken. After researching the problem, we discovered that the visualisation is essentially the successive application of a semi-transparent grayscale image, on top of the target image. The grayscale image is then colourised according to the desired colour scale. The problematic part of this process was the efficiency of successively applying the grayscale image. We found that our initial approach

to drawing was causing the performance bottleneck, and that we should instead be using the ***DrawImage*** method instead. This provided a significant performance saving and was sufficient for inclusion in the USS.

Memory Usage

The major problem we encountered during the development of the Visualiser was memory usage. Using the performance analysis tool in Visual Studio, we noted at one point that nearly 2GB of memory was being used. Being a garbage collected architecture .Net is responsible for reclaiming memory on objects that are no longer being used. Therefore somewhere in our code we were referencing objects which were never used again. We narrowed this memory leak to the graphics components of the visualiser (that is the Heatmap and Mouse Trail visualisations). We discovered that we had failed to invalidate images after they were updated, producing a chain of images instead of only having the most recent one in memory. Adding this invalidation procedure immediately reduced the memory footprint significantly. Additional changes, such as only producing images for the selected visualisation type added additional savings.

5.3 Reflection- Lessons Learnt

In this section we will reflect on the lessons that we have learned as a result of developing the Visualiser.

Performance Impact of Graphical Operations

We see from section 5.2, that the majority of the problems we encountered during the development of the Visualiser were related to a graphical operation. As a result of developing the Visualiser, we have learned to appreciate the time and detail required to develop fast and efficient graphic routines. In future projects we would give much greater consideration to the time and effort required to add graphical operations to a project.

Research Third Party Libraries in Advance

Another source of problems during the development of the Visualiser were third party libraries. We had numerous difficulties with the charting component we were using to visualise the brain and web data. Additionally, we had to extensively modify the audio visualisation library we were using, because it did not support multiple instantiations. The lesson learnt here would be to thoroughly research any external dependencies and to test that they perform and function as advertised. If we had done this prior to the development of the Visualiser, we could have accounted for the additional time required to get the libraries to work.

Choice of Data Formats

Choosing the correct format to save application data to is vitally important. We learnt during the development of the Visualiser that our approach to saving recordings was perhaps flawed. In future projects, we would consider this aspect in greater detail and weigh the benefits of standards compliance against efficiency and performance.

5.4 Future Developments

Given the opportunity to continue working on this project, or projects similar to the Visualiser, our general approach would be very similar to that taken here. In addition to this approach, using the knowledge and experience we gained through this project, we would make the following changes:

Create a Custom Visualisation Chart

We noted in section 5.2, that the Chart component we were using to Visualise brain and web event data was far from ideal. For future versions of the USS we would like to develop our own chart specifically designed to visualise these two data types. Doing so would allow us to incorporate additional features that are impossible with the current chart, including the visualisation of all 16 channels of EEG data (the chart is limited to the number of data points that can be plotted), alternative chart layouts and reduced memory footprint.

Add Transparency to Visualiser Interface

One of the requirements of the Visualiser interface was that it must allow recorded studies to be compared in an intuitive manner. Whilst the application performs this in its current incarnation, we believe that future versions could be enhanced by allowing visualisations to be stacked on top of one another and the windows be selectively made transparent. This would allow researchers to easily compare two different recordings, without the need to look back and forth (as is the case with side by side comparison).

Export Generated Visualisations

Adding the functionality to selectively export visualisations from the system could be of aid to researchers using the system. The visualisations could be used in reports for example, and so including the functionality to export visualisations would be beneficial in this situation.

Improve Audio Playback

The current implementation of the audio visualisation and playback is somewhat primitive. Future versions of the application could add additional functionality, such as volume selection, fast forward, rewind and other standard audio manipulation operations.

Integration with Existing Tools

Researchers typically use a number of software applications to record and log interesting behaviours and data points during user studies. Future versions of the Visualiser could possibly parse the output of these tools and imports into the Visualiser interface. Doing so would add additional information to a time correlated visualisation, allowing Researchers to draw new and interesting analysis from the USS.

Improve Performance

Whilst the current level processor and memory utilisation is far from high, there is definitely room for improvement. Future versions of the Visualiser could further optimise this aspect of the application. We would focus our attention on the Charting component in future versions as this is the primary resource consumer in the current implementation of the Visualiser.

5.5 Successes

In this section we will detail some of the successes of the Visualiser section of the USS.

Choice of data Docking Library

One of the third party libraries functioned and performed far beyond what we originally predicted it would do and that was the docking library - AvalonDock. The library was responsible for providing the tab and docking interface that the Visualiser utilised. It was seamless to integrate and operated without fail. We would definitely consider using this library in future projects.

Development Methodology

In a component full of new and experimental features, the prototyping methodology was definitely the correct one for developing the Visualiser. The ability to test an approach, without the worry of breaking a working system, was reassuring. The methodology also allowed us to test the feasibility and efficiency of certain approaches, and on numerous occasions it transpired that our initial approach was incorrect or flawed in some respect.

Synchronisation of Visualisations

We feel that the approach taken to synchronise the numerous visualisation displayed in the Visualiser has been successful. We used a simple object which implemented the Observer design pattern, so that listening objects could be notified when the current selected time had been changed by the Researcher. The approach taken was extremely simple, but proved reliable and accurate throughout the development of the Visualiser.