Mestrado em Engenharia Informática
Dissertação/Estágio
Relatório Intermédio / Final

CUE: Cambridge Usability Evaluator

Ricardo Freitas
rfreitas@student.dei.uc.pt

Orientador:
Bruno Cabral

Orientador:
Ian Hosking

Data: 3 de Junho de 2013

FCTUC DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

Abstract

As part of the Cambridge Engineering Design Centre (EDC), the author of this report details his experience and work, while developing the Cambridge Usability Evaluator (CUE). The CUE allows to create automated usability experiments on web applications. The experiments can be done both locally and remotely. Mixed methods, quantitative and qualitative, are used for data gathering. The focus of the development was to deliver a fist version of the application, based on a flexible and extensible architecture.

*

Contents

*

List of Figures

Chapter 1

Introduction

What is Usability?

According to the ISO 9241 [W3C02], usability is "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use.".

In shorter words, usability is the ease, the joy and the efficacy of using a product.

Products have to be designed for their audience. Therefore, understanding ones audience is of great importance.

Everyday, business man, designers, developers, make assumptions on how people work, how people think.

But even the most experienced of designers can easily be surprised on how people end up using his design.

There is a known anecdote that the designer of the long slim sugar packets committed suicide, after witnessing the general public use his creation. The purpose of the design was to tear the middle of the packet by bending it on itself. However, most people unknowingly ignored the designer's intention and just ripped the end of the packet like it was any another normal packet.

The story is probably not entirely true, but it does illustrate a classic example of design expectations not being met by real world usage.

Testing, even if with a small sample, is the only way to really know how usable a design is.

How to test Usability?

Opinions vary on how usability testing should be done. Some see it as exclusively quantitive, others as qualitative, other as a mix of the two.

The approach taken by the CUE, the Cambridge Usability Evaluator, is to try and support the two approaches as best as possible.

The CUE is a platform to create automated usability experiments.

Imagine for example, one wants to test how quick users are in finding the first Harry Potter book on Amazon.com. The CUE allows to create an experiment for the website that will display to the subjects the task: "Can you find the book Harry Potter and the Philosopher's Stone?". Relevant information pertaining to the experiment, such as: the time it took to complete the task; the web pages the subject visited before reaching the destination; where the user clicked (or touched) on the page. These are all pieces of information that can be tracked and later processed in order to give a better insight to the quality of the design.

What is the purpose of the CUE?

Testing usability has a purpose: to generate feedback that will hopefully shape the design towards a better usability.

The ultimate goal of the CUE is to improve the usability of a software application.

Originally the scope of the CUE included both the prototyping and the testing of the design.

However, it was realised that such a scope would be to large for a project that would take only one year, like this one did.

Therefore, the choice was made to make the CUE web based and able to support as many different web applications as possible.

The choice was passed to the developer to implement his application in the way he sees fit. Knowing that the CUE will support it regardless of the development tools he decides to use.

What was expected out of the developement?

Unlike the slim sugar packet design, the development of the CUE was meant to be always evolving. To be shaped by real world usage of the platform. To be

driven by authentic need, rather than an assumed one.

Therefore, more than trying to develop a product packed with features, a lot of time was spent on laying down an architecture that could support the evolution of the design of the CUE.

The minimum viable product of the CUE was defined and delivered in the end, together with a flexible and extensible architecture.

How is this report structured?

This report is structured as follows:

1. **Requirements** The desired functionality together with the desired properties of the architecture are detailed.

2. **Architecture** An high-level overview of the technical design of the system and its different components is made.

3. **Implementation** The tools used to implement the architecture, together with components developed from scratch by the author are described.

4. **Risk Analysis** Risks in the architecture are discussed, as well as their mitigation strategies.

5. **Testing** The testing performed is described. Recommendations for the future testing is also stated.

6. **Planning and Work Process** The way the development was performed, managed and planned is detailed.

7. **Conclusion** A personal overview of the experience is depicted by the author, along with future plans and recommendations.

8. **Appendix A: State of the Art** Usability testing tools are analysed and compared

Chapter 2

Requirements

The requirements always evolved together with the development of the project, so it became evident that the development had to be based on an architecture that would allow to easily modify the system.

The requirements have been divided into functional and non-functional. This is a common industry practice. The separation is needed because both have different implications in the development, architecture and testing.

Some future functional requirements are mentioned. Even tough they were not planned to be implemented, the architecture of the system was still influenced by them.

In order to prioritise work MOSCOW[1] was used for both categories of the requirements.

2.1 Actors

Before going into detail about the functionality of the system, it is important to point the two actors of the CUE.

In the figure 2.1 the actors and use cases of the CUE are detailed.

A tester is who operates on the data of the system. He defines experiments, initiates them and visualised the data that came out of the runs of the experiments. The tester can also be called the supervisor, for also supervising the

[1] MOSCOW's definition can be found in section 6.1.5.2 of [IIB09]

experiments, remotely or locally.

A subject is who undergoes an actual experiment. He has no control over the data of the system. His only data contribution to system is when his actions are tracked and collected.

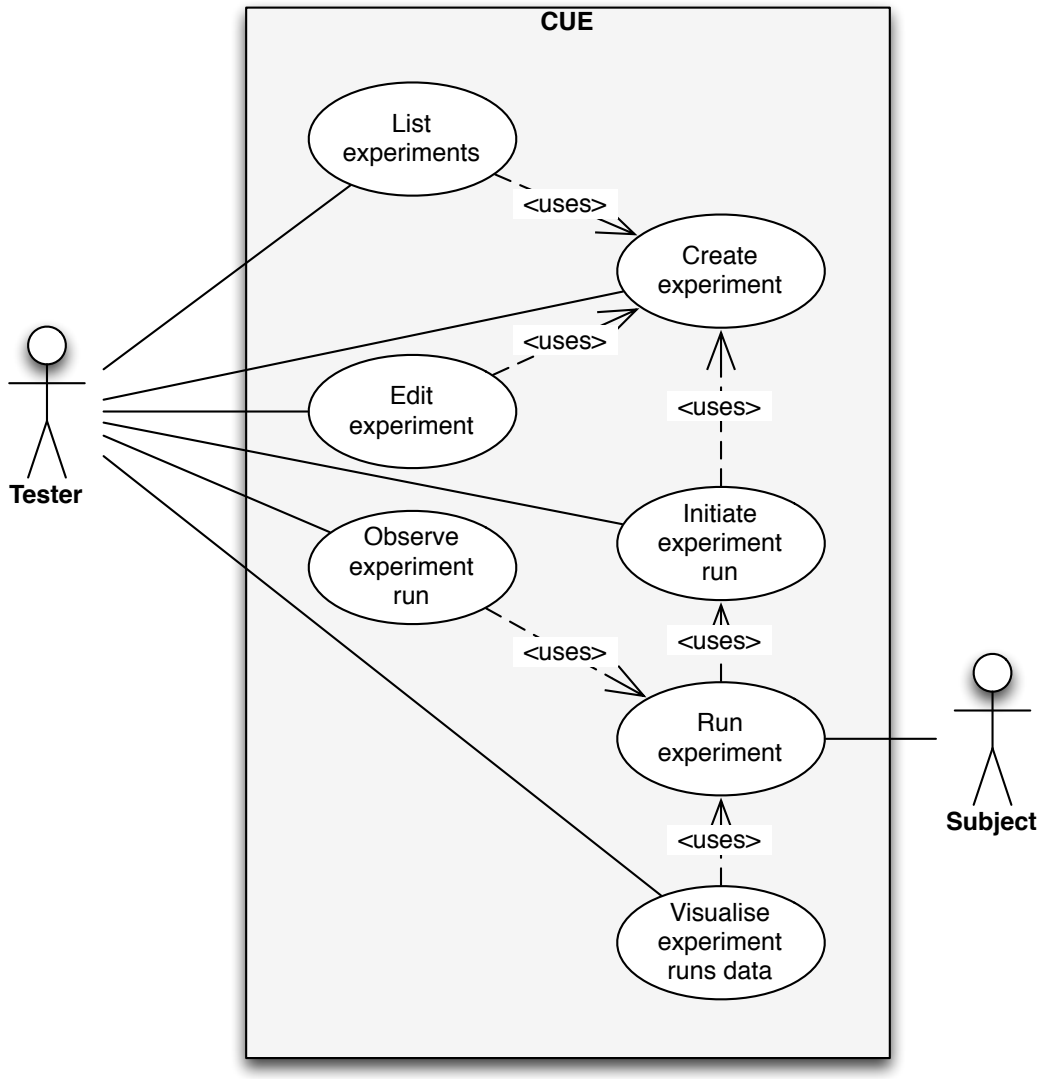

Figure 2.1: UML diagram detailing how the different use cases depend on each other and which actors take part in them.

2.2 Functional requirements

A functional requirement is a feature of a system that is defined not on how well it is implemented, but on the simple fact that it is implemented. It is better defined by what it is not and a functional requirement is not a quality requirement. It does not specify performance or user ability to understand the system.

2.2.1	Create Experiment Definition	Must
2.2.1.1	Task Description	Must
2.2.1.2	Task Start URL	Must
2.2.1.3	Previous State as Task Start	Must
2.2.1.4	Checkpoints	Must
2.2.1.4.1	Data Steps	Must
2.2.1.4.2	Task Completion (by URL)	Must
2.2.1.4.3	Task guidelines	Would
2.2.2	Edit Experiment Definition	Would
2.2.3	Play an Experiment	Must
2.2.3.1	Display Task description	Must
2.2.3.2	Display Task Confirmation Dialog	Must
2.2.4	Record Experiment Data	Must
2.2.4.1	Task Execution Time	Must
2.2.4.2	URL changes	Must
2.2.4.3	User inputs	Could
2.2.5	Observe Running Experiment	Should
2.2.5.1	Take notes	Should
2.2.5.3	Remotely Observe	Would
2.2.6	Data processing and Visualisation	Should
2.2.6.1	Histogram of Task Times	Should
2.2.6.2	User Path	Could
2.2.7	Data Protection	Must
2.2.7.1	Password Protect Website	Must
2.2.7.2	User accounts	Would

Table 2.1: Priority of functional requirements based on MOSCOW

2.2.1 Create Experiment Definition

The tester should be able to define tasks for the test subjects to perform. Imagining an e-commerce website as the application under test, an example of a task

would be: "Find X and purchase it".

In the following sections are the requirements for what the user should be able to define about an experiment.

2.2.1.1 Task Description

The description of the task needs to be defined, said description will then be displayed to the user during the run of the experiment, see requirement 2.2.3.1 for more details.

2.2.1.2 Task Start URL

A task can have a URL on where the task starts. The previous example task could, for example, have the homepage of the e-commerce website has the start URL, so that when the task is initiated, the CUE will display, to the subject, the e-commerce homepage together with the description of the task to perform.

2.2.1.3 Previous State as Task Start

The idea is to not define a start URL, but to rather use the last state of the previous task as the starting point for the task.

2.2.1.4 Checkpoints

The checkpoint idea was one that was actually introduced very late into the development, but that thanks to the flexible architecture and a careful implementation planning, it was quick to implement.

For this project, the concept of a checkpoint has been defined as a condition that can be triggered by subject while under going an experiment. Such trigger will have an action as a consequence.

Concretely, checkpoints can have different types of triggered actions, amongst them are the following desired requirements:

2.2.1.4.1 Data Steps Checkpoints are able to create different data steps within a task, eg. if a task is Find A and then B, the data collected for that task could be split between the two exclusive user paths: Start-to-A and then A-to-B.

2.2.1.4.2 Task Completion (by URL) Checkpoints are also able to determine whether a task has been completed or not, and if it has, the experiment moves to the next available task.

To determine whether a task has been completed or not, an URL condition is used.

For example, imagine a task is find Google Maps starting on Google's homepage. The url condition used to check whether the task has been completed can be "https://maps.google.com".

2.2.1.4.3 Task guidelines In order to minimise experimental bias, the experiment supervisor should be as absent as possible. Therefore, there should be a way to define guidelines that can assist the subject as he performs the task. Checkpoints with a URL condition are used to the trigger these guidelines.

2.2.2 Edit Experiment Definition

Editing an experiment is a requirement that needs a lot of design considerations. The problem faced is on what to do with already collected data after an experiment is edited. The data collected is influenced by the experiment definition, so mixing data from different versions of an experiment should be a matter for debate.

2.2.3 Play an Experiment

Based on the definition of an experiment, its tasks and parameters must be used to run it so that a subject can execute its tasks.

2.2.3.1 Display Task description

As the subject performs a given task, the task description is to be always visible together with the application which is under test. Using the previous example task, the description display would bear the text, "Find X and purchase it", all through out the experiment.

2.2.3.2 Display Task Confirmation Dialog

When a task is complete, a confirmation dialog box appears so that the subject has time to take a break, before moving on to the next task.

2.2.4 Record Experiment Data

As the subject performs the experiment, information about the run must be tracked.

2.2.4.1 Task Execution Time

Record the time a subject takes to perform a task. Divide it in time intervals specified by the task's checkpoint, in case requirement 2.2.1.4.1 is implemented.

2.2.4.2 URL changes

Every URL changes within the aplication being tested must be tracked, together with the time stamp of said event.

2.2.4.3 User inputs

Mouse and touch inputs are to be tracked, so that for example heatmaps can be calculated.

2.2.5 Observe Running Experiment

2.2.5.1 Take notes

As an experiment is being run, the tester should be able to take notes about the performance of the subject of the different tasks he executes.

2.2.5.2 Pause Running Experiment

For if an experiment is interrupted by unpredictable events, the tester can pause the experiment, effectively stoping the timer of the task. The subject receives a message on the screen that the experiment has been paused.

2.2.5.3 Remotely Observe

A tester could be able to remotely observe the subject performing the experiment, either in real-time or after it is finished. This way the tester does not need to be present in the same room to take notes, or even be available at the same time the experiment is performed.

2.2.6 Data processing and Visualisation

All the raw information collected is important, but more important is extracting relevant information from it, so that it can help the stakeholders to identify possible problems with their prototype or application.

2.2.6.1 Histogram of Task Times

The average and standard deviation of the completion time of each task is to be displayed on an histogram.

2.2.6.2 User Path

With the data of all the web pages that the user visited on an experiment, it is possible to draw the path that the user took in the form of a graph.

2.2.7 Data Protection

Experiments and their associated data must be protected in some manner. The following requirements represent possible ways of achieving it.

2.2.7.1 Password Protect Website

The entire CUE website access is to be restricted to those without a password.

2.2.7.2 User accounts

The next step of data protection is to have experiments be associated to user accounts and lock that data to the users and their passwords.

2.3 Non-Functional Requirements

Non-functional requirements are concerned with the properties of the system. They are the ones that influence the architecture the most.

2.3.1	Applications Compatability: Developement Tools Support	Must
2.3.2	Open Source	Must
2.3.2.1	Community Contribution	Would
2.3.3	Extensible	Must
2.3.4	Flexible	Must
2.3.6.1	Performance: Responsiveness	Should
2.3.5	Scalability	Could
2.3.6	Usability	Must

Table 2.2: Priority of non-functional requirements based on MOSCOW

2.3.1 Applications Compatability: Developement Tools Support

Initially the CUE was meant to include a Prototyping Toolkit, but through out the development it became clear that there was no set of tools that could satisfy all the desired needs; nor the time to develop such tools. The reason was that the toolkit had to be friendly enough for a non programmer to use, but also had to be powerful enough to model complex interactions. Therefore, the choice was made to make the CUE as compatible as possible with different types of development tools available. Effectively giving the freedom to the developer to choose his own set of tools.

Realistically, it was not feasible to make the CUE completely agnostic to all the development tools available. Since this would mean that the CUE had to be compatible with all the application development platforms existing today.

This limitation gave strength to the idea of having the web as the platform for the CUE.

The web itself is cross-platform. It is a platform which is supported by many devices' software stack. If not all, these devices support at least a sizeable subset of the entire web functionality.

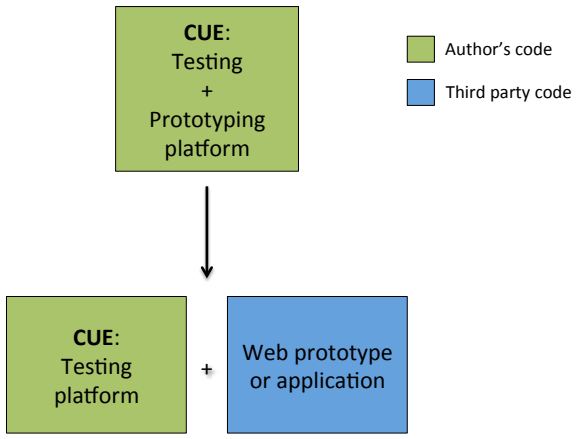

Figure 2.2: Evolution of the CUE's vision.

This set of functionality is also expanding as the standard and the standard adoptions grows. Therefore, the potential for developing and testing new design ideas will keep increasing.

More importantly, the number of development tools available for the web will keep growing as well. From programming libraries, to full fledge graphical tools, the set of tools available for both experts and beginners seems to be expanding.

It is in part due to this good support of tools that the Web was chosen as the development platform.

The better the CUE supports different types of web applications, the better it will support the tools that can create them.

A way of measuring the degree of compatibility of the CUE to different web development tools is to test it with as many different existing web applications as possible.

2.3.2 Open Source

The project had to be open source due to a funding requirement. However, since the application only had 6 months of solo development, it was of the interest of all the stakeholders for the project to be completely open to contributions.

2.3.2.1 Community Contribution

According to some well known open source developers [1] [2], for a project to be successful with the open source community, it is recommended to have easy to read documentation, detailing and explaining the vision of the project, together with examples and demos.

A way to minimise this documentation is to reuse other popular open source projects when possible, in order not just to leverage their functionality, but also their possible good support and approachable documentation.

Github offers a lot of metrics that can help determine the popularity of an open source project, from a favourites counter, to the number of contributions it has. If the project is published in Github, the CUE can then be measured in terms of popularity in the open source community.

2.3.3 Extensible

New possible future requirements were always coming up on meetings. Many of them were disregarded as too complex to implement in a feasible time. However, the architecture should accommodate the possibility to implement such requirements.

One way to ensure a degree of extensibility is to imagine a possible way to implement a desired future requirement and then accommodate the existing code to make sure that such implementation would not require much refactoring in the future.

Many other design principles also ensure safe extensibility, such as: modularity, by separation of concerns; loose coupling of different modules; refactoring with automated unit testing.

[1] http://coding.smashingmagazine.com/2013/01/03/starting-open-source-project/
[2] http://thechangelog.com/top-ten-reasons-why-i-wont-use-your-open-source-project/

This is a requirement that is not simple to measure. However, if a new functional requirement breaks other distinct functionality, as it is being implemented, then it's safe to assume that the system was not prepared for such extension. If such situation happens too often, and long refactoring sessions have to be employed, then the architecture should undergo some revision.

2.3.4 Flexible

The user interface design (UI) of the CUE changed through out the development considerably. A prototypal approach was preferred, in order to iterate this UI design with functional code.

HTML, and it's supporting technologies[1], is flexible enough to evolve from a prototype stage to a full implementation stage. Prototyping in HTML is an approach which is used by some successful companies, including 37 signals [2] . The idea is that, for an expert web developer, mockups take almost as much time as a prototyped HTML document to develop. The time spent developing mockups can be considered lost, compared to the time prototyping in HTML, since the mock-up won't be used in the final code of the system, whereas a prototype, with some refactoring, as much higher chances of being used.

Not just the UI was expected to change through out the development, but also data-oriented features such as what data to track, how to segment such data, etc.

Therefore, the system needed to be easy to modify, so that revised ideas could be quickly tried out, both in the user interface facet as well as in the data facet of the CUE.

2.3.5 Scalability

Scalability is usually a necessity in any web service, however it was not be a priority requirement in this initial development. Estimates of number of users was non-existent and the focus of the project was on defining the design and the architecture of the system, rather than how well that same system performs with

[1] JavaScript and CSS

[2] http://blog.handcraft.com/2010/07/how-37signals-does-html-prototyping/

a large number of simultaneous users.[1]

This did not exclude the requirement completely. Technical decisions should still be made with scalability in mind, although, due to time constrains, the system was not to be tested in this regard.

2.3.6 Usability

The CUE, as a tool to create usability experiments, needs to have an acceptable usability as well. In order not to distract the subject from the experiment and negatively impact the results.

The usability focus was therefore put on the subject of the experiment itself, rather than on the the tester of said experience.

The usability of the CUE for its experiment subjects, can be assessed qualitatively, by observing a subject undergo an experiment run; and noting how much the CUE is interfering with the realisation of said experiment's tasks. Quantitatively, a tool similar to the CUE could also be employed to test the usability of the CUE itself. However, due to possible confusion about the nature of this meta experiment such approach was dismissed.

2.3.6.1 Performance: Responsiveness

Response time is defined as the time it takes for the system to respond to the a user action. Responsiveness can be defined as the user tolerance to that delay. The tolerance can depend on different users, however there is an agreement when it comes to acceptable response times.

According to [Nie93], there are three response time intervals, all with different agreed upon user tolerance.

The excerpt from the book:

> The basic advice regarding response times has been about the same for thirty years [Miller 1968; Card et al. 1991]

[1] Many popular web services were initially developed with an architecture that scaled poorly. Twitter and Youtube both had troubles scaling their systems. They intially focused more on the product design rather than on the quality of the implementation.

1. 0.1 second is about the limit for having the user feel that the system is reacting instantaneously, meaning that no special feedback is necessary except to display the result.
2. 1.0 second is about the limit for the user's flow of thought to stay uninterrupted, even though the user will notice the delay. Normally, no special feedback is necessary during delays of more than 0.1 but less than 1.0 second, but the user does lose the feeling of operating directly on the data.
3. 10 seconds is about the limit for keeping the user's attention focused on the dialogue. For longer delays, users will want to perform other tasks while waiting for the computer to finish, so they should be given feedback indicating when the computer expects to be done. Feedback during the delay is especially important if the response time is likely to be highly variable, since users will then not know what to expect.

It was not a requirement to time precisely the different user interactions of the CUE, however, the interactions where expected to fall in the 0 to 1 second interval.

However, this analysis on responsiveness has been developed with the mouse and keyboard in mind. Nowadays, with the proliferation of touch screens and their different levels of software imposed sensitiveness and button target areas, responsiveness has to take into account that on occasions the hardware or software may not react at all (or react wrongly) to a user action.

One way to measure the quality of this requirement is to have subjects experiment with the user interface and observe how responsive the system is to their desired actions.

2.3.7 Mobile support

The CUE should support mobile platforms as well as desktops. The target is for the CUE to work at least on iOS devices, iPhone as well as iPad. Android devices are lower priority since the hardware is too diverse to effectively test in a short

time.

Chapter 3

Architecture

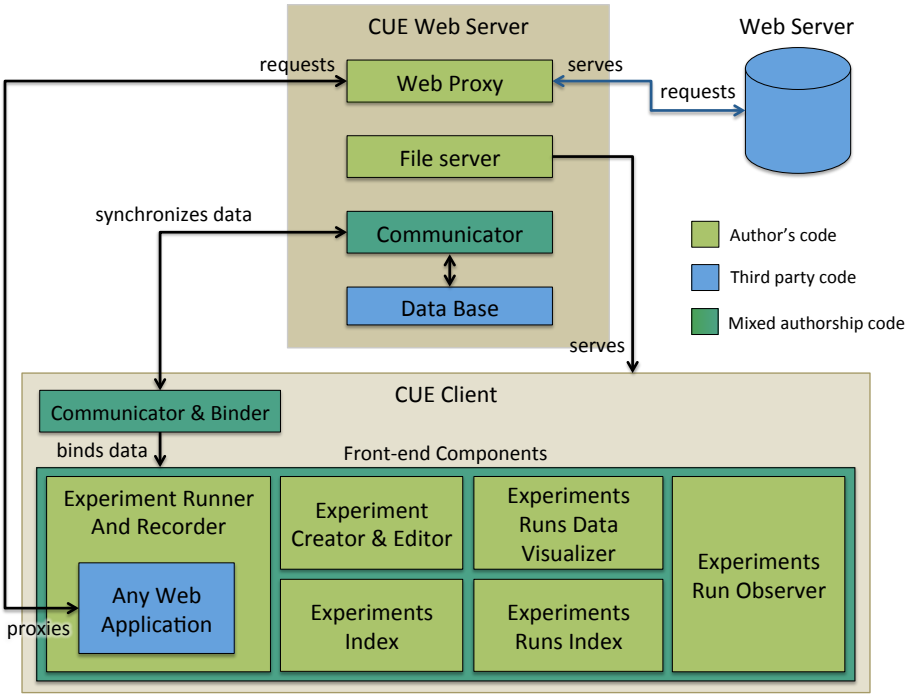

Figure 3.1: High Level Architecture.

In this chapter the different components seen in the diagram 3 are described on a higher level. Albeit, the two first sections purpose is to both describe the

untraditionally of the architecture and some technical concepts that influenced its design.

3.1 Comparision to Classical Web Server

On a very high-level the CUE is similar to a traditional web server. Data can be accessed and manipulated by the clients. And HTML document, together with CSS and JavaScript files are served by the server, in order for the clients to present the data to the user in an understandable manner.

Although, there are some aspects to the architecture that set it apart from the traditional web server. Data is actually synchronised in real-time between the client and the server. This mean that if a client A modifies a piece of data that the client B is accessing as well, the client B will get the update automatically, without needing to request for it.

The choice for such solution was to align with the non-functional requirement of needing to have the system to be both flexible and extensible. With this approach, data-oriented features that may be requested in the future can be much more easily implemented, since the process of modifying data and propagating its changes is complete and simplified for the developer.

The fact that the CUE is also a web proxy, also sets it apart from the conventional web solutions.[1].

Another particularity that also sets this architecture apart is the fact that much of the construction of the HTML document is done on the client-side rather than on the server. Conventional web servers usually construct the HTML document themselves and then send it to the client, all based on the data that the client requested in the first place. In other words, the client requests data, and from the raw data, the server creates and sends humanly readable data to the client: the HTML document. However, with the increased performance of JavaScript, clients now days are becoming the HTML constructors themselves. Getting from the server only the templates, the scripts, other relevant resources and the raw data.

[1]The web proxy is necessary to circumvent a security policy of iFrames, this problem is discussed in more detail in sections 3.2 and 4.1.4

Yet another aspect of the CUE that sets it apart, is the fact that the server code is also written in JavaScript. This allows for the entire stack of the system to be written and maintained in one language. It also opens the possibility of sharing code between the client and the server, adding flexibility to the system.

In the following sections the client and server side are discussed, together with the components that make them.

3.2 Evolution of the Architecture: Looser coupling

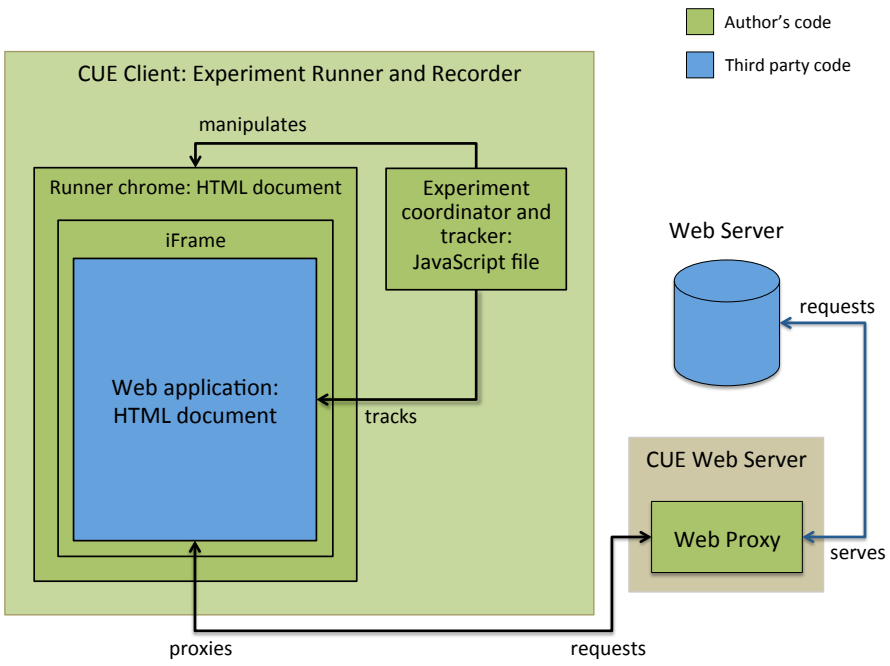

Figure 3.2: Diagram explaining how the iFrame's proposal fits within the architecture.

One of the requirements of the architecture was to be as compatible as possible with different web applications.

Originally a solution was tried out by the author which relied on a dormant script inside of the application under test, similar approach is used by Google Analytics. In order for it to work, the application to be tested needed to have an external script that would then take over the application and initiate an experiment when commanded by the CUE server. However, with this solution only applications under the control of the tester could be tried out, since scripts can only be inserted by the webmaster.

A proof of concept for this solution was developed in the first semester. However, many technical challenges were encountered, which made it extremely difficult to maximise compatibility, leading to the belief that a new solution was necessary.

The new proposal would use iFrames, as a means to wrap the test application. iFrames are a web standard that allows to have HTML documents inside of other HTML documents. With this solution it is left to the CUE to coordinate the experiment and track relevant information from the iFrame's document, leaving the document inside of the iFrame unaware of what is being performed, see picture 3.2.

One advantage of this solution, besides fixing the technical problems found in the previous one, was the fact that it would allow for full decoupling. No dormant script would be necessary to make an application testable by the CUE, effectively removing any trace of testing code from it. As a consequence, applications not owned by the tester could also be tested.

However, for this architecture to work, a Web Proxy has to be used, in order to bypass the same domain policy that iFrames are restricted to by standard.

The degree of compatibility falls then on the Web Proxy on how able it is to proxy different types of web sites.

3.3 CUE Webserver

The CUE server is divided into multiple high-level modules, in this section, a description of their purpose and functionality is detailed.

3.3.1 Web Proxy

The Web Proxy serves one purpose: to proxy http request to and from a CUE client and the external server where the test application resides.

The need for a proxy is to bypass a security restriction: the same-domain policy. This policy applies to iFrames, which are essential to the technical design of the CUE.

iFrames are used when an experiment is being run. They offer a clear separation between the chrome of the CUE, meaning the information about the task being performed and other specific operations, and the application under test itself.

More importantly, this approach separates the code of the test application and the code of the CUE. They are two different pieces of software that can coexist independently, they are completely decoupled.

The design motivation behind this choice was that an application should not have testing code within it. Also, ideally, any application, even the ones not owned by the tester, should also be able to be tested.

3.3.2 Communicator

The Communicator serves as the way for the client and the server to synchronise data in real-time. It also ensure the persistence of the data by handling the communication with the database.

Such an approach allows for the system to be easily extended with data-oriented features, since the process of exchanging data with the server and the client is complete and in real-time.

3.3.3 Database

Data is stored in documents.

A document-oriented solution was chosen since the data structure is not enforced by a scheme. Hence it facilitates the process of modifying it, aligning with the flexibility requirement.

There are individual documents for all the different experiments. Each experiment has references to its runs, the runs themselves are separated documents. The reason for the document separation between the experiment and its runs, it's to allow for better scaling. If the data of the runs was inside of the experiment document, every time an experiment run was initiated by the client, all the data of the other runs would be sent as well, which would serve no point; and for a large number of runs it could easily become expensive.

3.3.4 File server

The file server provides the static resources requested by the client, such as HTML document, CSS, JavaScript files, images, etc. Some HTML documents can be dynamically generated by the server, however in this architecture most of the manipulation of HTML documents is done in the client.

3.4 CUE Client

The CUE client is defined as all the code that is ran by the web browser.

The JavaScript code, the data and other resources that the client uses are all served by the CUE server.

The client is divided in to two different components.

1. The front-end, which is where the data can be accessed and transformed by the end-user. It is, in other words, the user interface.

2. The Communicator & Binder, which ensures the synchronisation of the raw data between the client and the server in real-time. However, it also helps to bind that same raw data to the HTML document, so that the user can visualise it in real-time and, if applicable, modify it.

3.4.1 Comunicator & Binder

This component ensures the synchronisation of the raw data between the client and the server in real-time. It also facilitates the binding of the raw data to the HTML document.

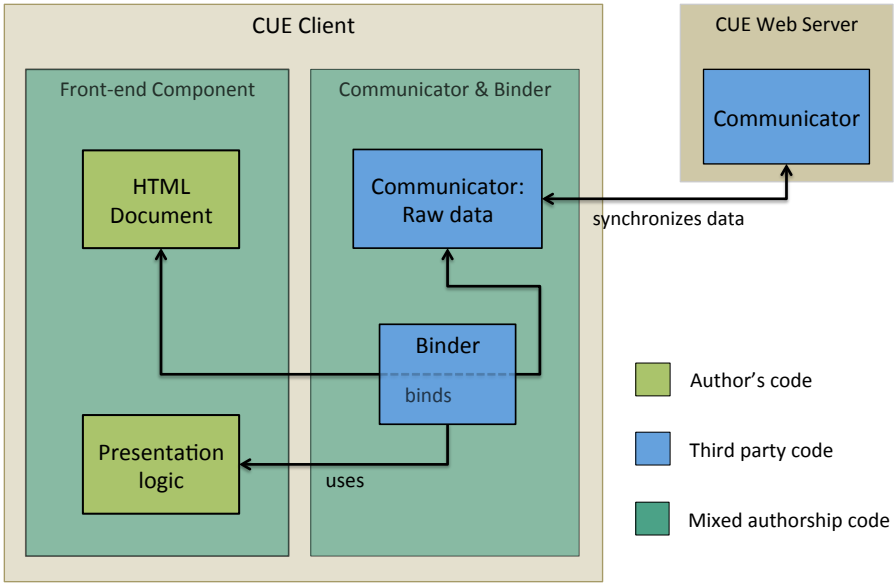

Figure 3.3: Diagram explaining the function of the Communicator & Binder component.

The raw data is synchronised in JavaScript, but the end-user can only read and interact with the HTML document. Therefore, the raw data needs to find a way to and from the HTML document, this is the binding aspect of this component.

This component is used in all the front-end components, since all of them need to exchange data to the server (the synchronisation) and display that same data to the user (the binding).

3.4.1.1 Communicator

Both the flexibility and extensibility of the architecture are increased by having a module that abstracts the developer from having to request, directly to the server, CRUD[1] operation on the data.

[1]Create, read, update and delete

3.4.1.2 Binder

Having a way to declare the binding of the raw data to the HTML document also increases the extensibility and flexibility of the system.

This module allows for the developer to define the presentation logic declaratively rather that imperatively. Stating the "what" rather than the how. It is then up to the Binder to define how the binding is done, based on the declared presentation logic.

A declarative programming [1] approach allows to minimise side-effects, by making code more localised and therefore more easy to extend without affecting already existing code as much. Flexibility is also improved for the same reason.

3.4.2 Front-end Components

The different front-end components divide the functionality available to the user.

All of these different components can be accessed by url, they are the layer of interaction that the end-user has with the system. It it is through them that experiments are created, run, observed and their data visualised.

In the following section the purpose of the different components is explained.

3.4.2.1 Experiment Runner And Recorder

The experiment runner and recorder is where an experiment run actually takes place. This is where a subject goes through all the tasks of an experiment and all the relevant information is tracked and sent to the server.

3.4.2.2 Experiment Creator And Editor

This client module is where a user can create or edit an experiment set up data.

3.4.2.3 Experiment Index

This module displays, in real-time, all of the existing experiments.

[1] http://en.wikipedia.org/wiki/Declarative_programming

3.4.2.4 Experiment Runs Data Visualizer

The data of the runs of an experiment has to be interpreted in some way. Even though the main use case of the CUE was a qualitative experiment, there are many use cases where more objective metrics would be valuable. For a first feature, the average completion time of a task is displayed, together with its mean.

3.4.2.5 Experiment's Runs Index

This modules displays all the runs of an experiment, as well as their status. The different runs link to its experiment run observer, where the tester can observe and take notes about the running experiment.

3.4.2.6 Experiment Run Observer

This part of the application came out of a real need the team encountered while using the CUE. Taking notes about different tasks was crucial for qualitative experiments. This evolved to an idea of observing the experiment itself.

Chapter 4

Implementation

In this section a detailed breakdown of the architecture is presented, together with the libraries used in the software stack of the system.

In the following sections, the different components are broken down in a similar manner as they were presented in the previous chapter, but now with a more in-depth technical description.

4.1 CUE Web Server

In this section the different modules that make the server-side of the CUE are detailed.

In the picture 4.1, the dependencies of the modules with each other are represented.

4.1.1 Node.js

Node.js is a javascript platform used to build servers, mostly web based. Besides being immensely popular with the open source community, meaning there is great free online support, Node.js offers other benefits to the development.

4.1.1.1 Common Language: JavaScript

Due to the fact that Node uses the same language as the browser, JavaScript, it makes it possible to reuse or move code from the client to the server almost

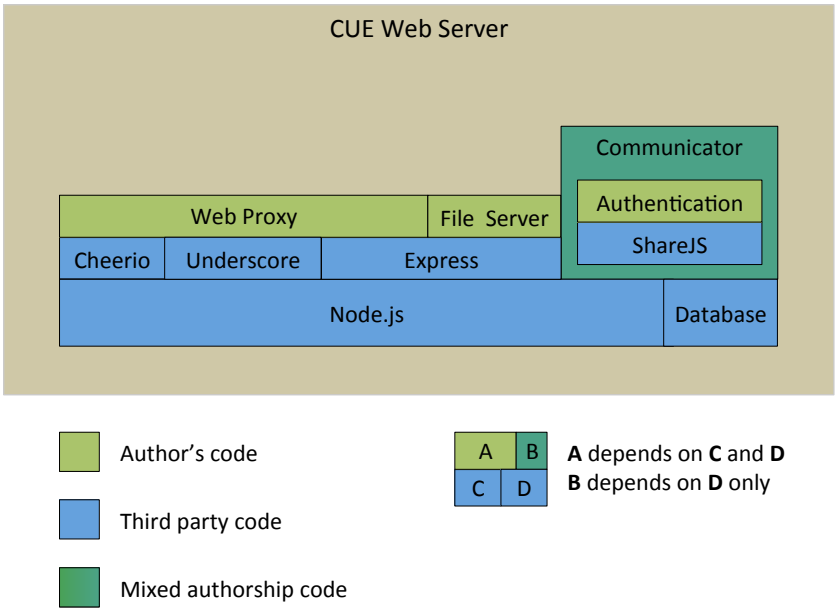

Figure 4.1: CUE's server-side dependencies.

effortlessly. Many times when building a client-server solution, the question is often asked of where the code should be run. Depending on the proficiency of the developers on a particular language, they can be biased to chose a solution that is not exactly the most correct one. Also, oversights may happen and the cost of changing the code from the server side to the client side can be prohibitive, that is if the languages are different in the two contexts.

Even though both contexts use the same language the basic APIs provided in the two are some what different. In the client, for example, an API to change the HTML document is provided, the DOM, whereas in the server, said API is not present, since it would serve no purpose.

Although, the two contexts are different, there are still overlaps in the API, more specifically the JavaScript standard and other APIs that Node borrowed from the browser. However, the community is making this gap narrower by

developing libraries with identical APIs to their counterparts that provide similar functionality, Cheerio is a good example of that. Also, more and more libraries nowadays are not fully depended on having a DOM present, making them also partially or even fully compatible with the Node.js.

4.1.1.2 The Run Loop

Node.js is an event-oriented, non-blocking I/O model. What this means is that Node runs user code in a run loop. While the code is being run, no new requests are being processed, effectively the system is blocked. Incoming requests go to a queue for latter processing by the run loop. The way that Node.js solves the blocking issue is to make costly operations asynchronous. For example, reading a file can be initiated in one loop, but the result is only handled by a callback in a later loop. This is the asynchronous non blocking I/O character of the tool.

This model is more beneficial than the classical threaded model, where a thread is created for every client request. The reason why the classical model does not scale well, even though threads can theoretically maximise the CPU, is because in a multi core system, there is a big cost when one thread has to be paused so that another thread can take its place. This cost is referred to as content switching. The reason why it is so costly is because the entire context of a thread, its memory, has to be stored some where and retrieved later on.

In a multi core machine, the assumption would be that Node.js wouldn't be able to use all of the available computational resources. However, Node does use a thread pool to run the asynchronous operations (anything I/O related can be asynchronous, eg: reading files, request handling, etc.). As mentioned before, user code is ran on the single threaded the run loop, but there are APIs available that allow the user to run code on the thread pool, freeing the main loop from costly user operations.

This model makes it simpler to develop high availability systems. Still, it is possible to block the run loop by either running synchronous blocking operations[1] or by running intensive computational operations on it. If so this can reduce the availability of the system drastically, since there is only one thread to take care

[1]Almost all I/O blocking operations are available by default by Node.js as asynchronous non blocking APIs.

of incoming request. Therefore, is it is important to understand the model, so that it can be leveraged best.

4.1.1.3 Easy deployment

Node.js comes with a package manager, the NPM. NPM simplifies the dependencies problem usually found in deployment. With this tool the developer only has to specify the modules that the project depends on and NPM ensures that those modules are present before the server is run.

4.1.2 Express

Express is a flexible web framework for Node.

It offers a great set of tools that make it easier to do common HTTP operations.

As an example, responding to a (GET) request for an html document with the name "index.html", can be done simply like this:

```
1       app.get("index.html", function(req, res){
2           res.send('<div>hello world</div>');
3       })
```

The WebProxy takes advantage of this framework extensively, since besides routing, like shown above, it also offers other tools that help manipulate and read the different HTTP requests, which, in itself, is integral to the proxy.

A great point of advantage of this tool is that it does not hide the Node.js set of lower level HTTP APIs. It just provides a new set of higher level apis that can coexist with the lower ones.

Express is on its third release already, it has a great community support and has been used in many production applications [1] .

4.1.3 Cheerio

Cheerio is a tool that allows to rewrite an HTML document using the familiar jQuery API.

[1] http://expressjs.com/applications.html

This library is used in the server for the WebProxy. The fact that the API is the same as the (client-only) jQuery was helpful for code sharing between the client and server.

4.1.4 Web Proxy

The Web Proxy was one of the first components to be developed. The reason for this was that the architecture had to be validated first on this point. If the Web Proxy did not work, or revealed itself to be too complex to implement in the given a time, another, less complex and probably less powerful solution had to be found.

4.1.4.1 How to use the proxy

The way to proxy an HTTP request is by appending the URL to proxy to the CUE's domain with the added path ("/proxy/").

Example: "www.cue.com/proxy/http://www.google.com"

4.1.4.2 The Cross Domain Problem.

In order for the CUE to be able to collect information about the experiment, the application under test needs to be probed.

For example, mouser or touch inputs need to be trackable.

Since the test application is inside of an iFrame, what is inside of it needs to be accessed somehow. It's here that the same domain policy interferes.

iFrames can be used even if they have different domains from their host page. However, for their host/parent page to access its content programmatically, the domains must match.

In order for the iFrame to have the same domain has the CUE server, the application has to be either hosted by the CUE or proxied by it.

Hence, in order to bypass the same-domain policy one has to serve the HTML document of the test application through the CUE server.

4.1.4.3 Server Side Document Rewriting

Serving the HTML document through the proxy is not enough From a proxied HTML document, the client can navigate to other documents and those documents also have to be served by the proxy in order to continuously bypass the security policy. Therefore, all the links in a document need to be redirected to the proxy.

The best solution for this problem is to rewrite the client requested document in the server and then deliver the modified version back to the client. [1]

4.1.4.4 Client Side Programming

References and navigations to external resources are not only defined by the Web Server. Client side programming using JavaScript can actually manipulate the HTML document and initiate request with any web servers.

Therefore, to make sure that all remains proxied, any state change that might influence this constrain needs to be monitored and transformed accordingly.

All the code that ensures enforces this behaviour is actually injected in every HTML document that goes through the WebProxy. The code is injected in the very beginning of the head of the document, to make sure it's the first piece to be executed.

JavaScript Navigation Navigation can also be initiated by JavaScript. Therefore all the JavaScript APIs that initiated a navigation had to be wrapped, and a strategy called man-in-the-middle was employed, in order to transform the requested URLs in to proxied ones.

The Ajax Problem Ajax request can suffer from the same domain domain policy as well. The strategy to ensure the request are proxied is the same as

[1] A different alternative was also tried, where the client himself would change all the links in the document after the proxy server delivered the unmodified version of the document. This solution had some techinical problems, leading to halted resourse fetching in Chrome. The reason why this alternative was tried out, was because this way the computation necessary to rewrite the document would have been done by the client, hence reducing the load from the server. Also, due to the fact that Node.js was used together with Cheerio, moving the code from the server to the client was just a matter of cut and paste.

before: man-in-the-middle rewriting urls.

Document Manipulation The HTML in the document can be changed and replaced by JavaScript, the way to track this is by using Mutation Observers API. This API is asynchronous, when the document is changes a callback is called and summary of the changes is passed along. The changed portion of the document is then rewritten using the same code that the server uses to initially rewrite the document.

There is a limitation due to the asynchronotisity of the API, however. Theoretically, in the time that the document was changed and the callback is called by the API, the user is free to interact with the document; and if new links were added to the document, those links will not have been rewritten to go through the proxy.

More testing needs to be done to determine if this can actually be an issue[1].

4.1.5 Communicator: Share.js

The communicator is the component that synchronises the data between the client and server in real-time and also ensures that such data is persisted in a database.

ShareJS was the library chosen to implement such solution.

4.1.5.1 Share.js

ShareJS is an Operational Transform library. It allows to synchronise data between the server and its clients effortlessly. The developer behind the library is an ex Google Wave Engineer, which due to its experience is well versed in this field. Not only is ShareJS able to synchronise text effectively, but other more complex structures like JSON documents are also well supported.

ShareJS is a major component in the architecture. JSON has been chosen as the document format in part due to it.

[1] There is in fact an API that is similar yet synchronous, but it's being deprecated due to performance issues.

The library has yield enormous benefits and it has also infused a great potential in the architecture that classical solutions would not able to provide. Real-time operations can be implemented easily without worrying about data exchange algorithms, since they are all provided by the library. The communication of data between the client and the server is completed abstracted to the developer.

Naturally, with such stake in the architecture the risk of it failing is passed onto the architecture of the CUE. However, very few code is actually depended on ShareJS. The set up code for ShareJS on the server is minimal and on the client, the code was designed to enforce a very loosely coupled dependency, see picture 4.2.

4.1.5.2 Authentication

ShareJS provides the programming hooks that allow the developer to create its own authentication logic and restrict as he chooses the data access.

Information such as the document which is being accessed, the type of operation performed on it and the client who is performing it, are all made available so that authentication rules can be enforced.

For now, the data access is non restricted and data protection is offered with HTTP basic authentication scheme. The entire CUE website is restricted to those who do not know the password.

A more general authentication scheme using user accounts needs to implemented in the future, but for now it has not been. However, the architecture can accommodate such requirement without issues.

4.1.6 Database: Redis

Redis is the technology used for persistence. ShareJS has Redis as one the three forms of data persistence supported, the others being MongoDB and MySQL.

Redis has been chosen since it is one of the most popular key-value store databases [1] and has been used in production in multiple occasions [2] .

[1] http://db-engines.com/en/ranking/key-value+store
[2] http://redis.io/topics/whos-using-redis

Besides scoring well on performance benchmarks [1] , Redis offers a lot flexibility in terms of handling data. Documents are not restricted to a scheme; and with mechanisms such as master-slave replication and Publish/Subscribe it is possible to scale it in various ways, giving more control to the developer on how the data is handled and distributed.

Even though a persistence mechanism had to be chosen, ShareJS makes it straightforward to change the persistence layer to one of the other available options. However, already collected data will have to be migrated by the developer himself.

4.2 CUE client

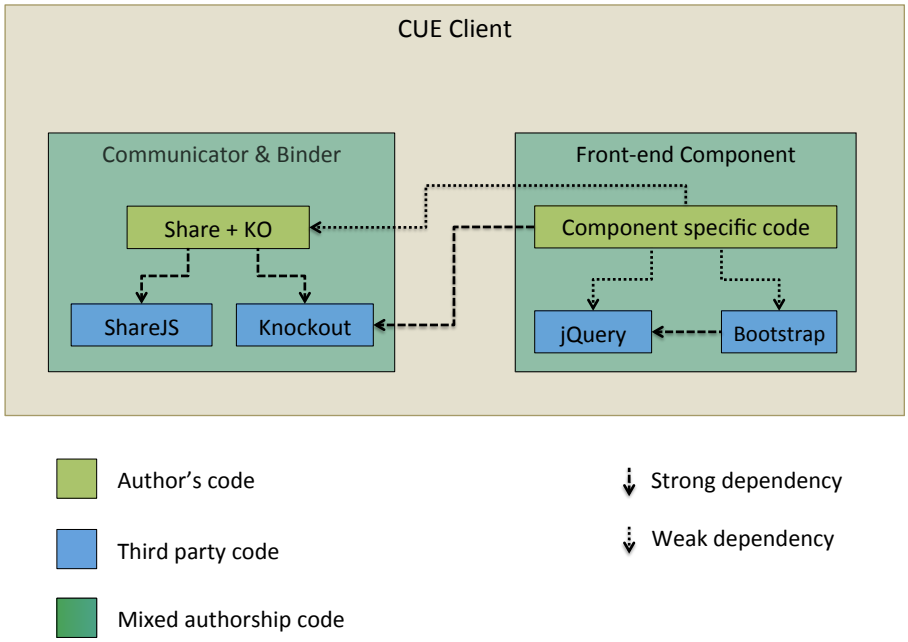

Figure 4.2: CUE's client-side dependencies

[1] http://redis.io/topics/benchmarks

Like the server-side, the client-side of the CUE leverages a lot of available open source libraries.

Rather than choosing libraries that would solve very particular problems, more general tools were preferred, so that they could grow with the system even if the functional requirements changed drastically.

4.2.1 Communicator & Binder

ShareJS is the communicator in the client-side as well.

Knockout is the binder. It helps to bind JavaScript objects to the HTML document.

However, the ShareJS documents, even though being JavaScript objects themselves, are not directly compatible with Knockout.

Share+KO purpose is to bind these two incompatible objects together, mediating the communication between the two. With this approach the two remain loosely coupled.

4.2.1.1 Communicator: ShareJS

On the client side, the synchronisation of data between the server and the client is also handled by ShareJS. The data is updated in real time and any merge conflicts are resolved using Operational Transformation algorithms. More so, ShareJS abstracts the system of any connection issues. For example, if a data update fails, the system will retry the operation until it succeeds.

Any manipulation of data from part of the client is performed by calling a ShareJS specific javascript API. Any updates to the data, coming from the server, can be listened to by using an asynchronous event listening API provided by the library as well.

Even though the document which is being synchronised is a JSON document, a simplified JavaScript object, the way to manipulate it is all done through ShareJS specific APIs. Therefore, the ShareJS document won't work directly with other libraries that expect a regular JavaScript object.

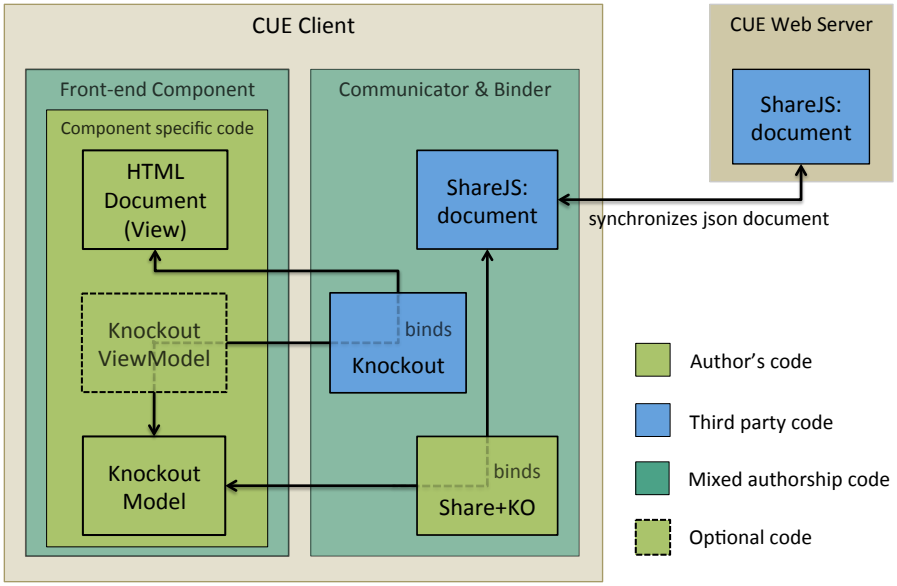

Figure 4.3: Communicator & Binder's diagram of implementation.

4.2.1.2 Binder: Knockout

To create a two way binding between an HTML document (the view) and a Javascript Object (the model), the open source Knockout framework is used.

4.2.1.2.1 MVC vs MVVM In the Model-View-Controller (MVC) pattern, Knockout acts somewhat as an automatic controller, it is the glue between the view and the model.

This is a simplified comparison for those familiar with the MVC pattern.

In reality, Knockout is an implementation of the Model-View-ViewModel (MVVM) pattern.

In the MVVM pattern a data binder is assumed to exists which will help bind the Model to the View, optionally through a ViewModel, in case the binding between the two is not straight forward.

The ViewModel is the presentation logic for the model. It transforms the data in the model to better suit the view, a job done by the Controller in the MVC pattern. However, in the MVC pattern, the controller is also expected to modify the view himself, taking the job of both the ViewModel and the binder.

4.2.1.2.2 Knockout Model A Knockout model is a JavaScript object, however its properties, in order to be bindable by the library, have to be wrapped by a specific function, breaking the default standard for accessing object properties. Due to this constrain, other JavaScript libraries expecting a regular JavaScript object won't be able to use a Knockout model without some sort of mediation.

4.2.1.3 Mediator: Share+KO

To recapitulate Knockout allows for easy synchronisation between the view and the model; whereas ShareJS synchronises a JSON document between the client and the server. However, the ShareJS document cannot serve as the model for Knockout, because their API are incompatible.

To fix this problem, the author created a module called Share+KO, that synchronises a Knockout model with a ShareJS document. This way it becomes trivial to have a model shared between clients and, at the same time, to display it in real time on the view.

With this solution both the Knockout code and the ShareJS code are loosely decoupled. This means that code that is mostly depended on Knockout, can also enjoy the ShareJS functionality, without having to become depended on the library.

4.2.1.4 Alternative Binder: Ember.js vs KnockoutJS

Ember.js is another framework with Model-View binding support. It was consider to be the binder in the architecture, but it was dismissed in favour of Knockout.

In the following section the author compares the two frameworks against the non functional requirements of the architecture. His opinion is based on the experience the author had with both [1].

[1] This opinion was based on the experience that the author had with this framework. He

4.2.1.4.1 Modularity Ember comes with a built-in dynamic class system with mixin support, that facilitates to make the code more modular. However, the model does mix the classical class paradigm and the prototypal inheritance common in JavaScript. This approach leads to some confusion as to what are instance variables and what are variables of the prototype.

Knockout does not come with a class system, opting to leave it as choice to developer on how to construct his object best, with the only requirement that the observable properties must be wrapped with a specific Knockout function.

Winner: Tie Although, the Ember class system would improve the extensibility of the architecture, the default JavaScript prototypal model, which is supported by Knockout, does suffice. Ember's class model is actually built on top of the prototypal model, so a Knockout compatible model similar to Ember's can always be implemented later on if needed.

4.2.1.4.2 Object's Compatability Objects created by Ember's class system can be binded to the HTML document, in a similar fashion as Knockout. However, the method for accessing an object's properties is even more unconventional than Knockout, reducing interoperability with other libraries and making it harder to debug.

Winner: Knockout

4.2.1.4.3 Model-View Binding Ember uses a binding syntax for the HTML document (the view) that is much more devoid of logic, off loading the presentation logic to the JavaScript, leading to a better separation of concerns.

Knockout, on the other hand, expects much of the presentation logic to be present on the HTML document itself. This choice makes it harder to debug, since the browser debugger works better in evaluating JavaScript code. Although, Knockout does manage to compensate this issue by presenting user friendly error messages for most presentation code errors.

used it in the development of the SmartPhone test case, and in another side project that involved developing an online questionnaire that could run solely on the client.

Winner: Ember

4.2.1.4.4 Dependencies Tracking Both Knockout and Ember support computed properties. Computed properties are properties which are generated on other properties, computed or not. When then their dependencies change, so can their value.

The only difference between the two is that in Ember the dependencies of a computed property have to be explicitly declared, whereas is Knockout an algorithm is employed that detects the dependencies implicitly.

Winner: Knockout

4.2.1.4.5 Overral Even with many hours of experience on Ember, its debugging sessions can remain relatively frustrating through out.

On top of the already mentioned hard to debug functionality of Ember, the framework also comes with its own Run loop. The JavaScript environment already has a run loop, what Ember does is create a wrapper around it and extends its functionality. The down side is that errors in code initiated by the Ember's run loop are really hard to trace.

Ember.js is in many ways over engineered. It's extended functionality does not compensate the distance that it creates with the underlying web layer, making it, at times, very hard to debug.

Ember is only now preparing to release its first public version, contrasting with Knockout which is already on its second public version.

For all this reasons Knockout was preferred.

4.2.2 Front-end components

In the figure 4.2 the dependencies of a regular front-end component[1] are depicted.

[1]By regular front-end component, the author means a component that deals with information in a similar manner to the others, those who do not, are referred to as irregular.

4.2.2.1 Component specific code

The component specific code, is all the code that is part of the logic of a front-end component, but which does not include external libraries. It is work on the user interface exclusive by the author .

Knockout offers a framework that helps to bind the data to the HTML document. However, the data presentation logic is still defined by the developer. This logic is part of the component specific code.

Therefore, much of the component code is heavily depended on Knockout, so if Knockout is removed, a large portion of the code will need to be rewritten.

Share+KO is used in the the front-end components as well, but it is a weak dependency, since it is only used to declare what ShareJS document should be binded to what Knockout object. This way if ShareJS along with Share+KO are removed from the project, most of the front-end code will not need to be rewritten.

4.2.2.2 Experiment Runner And Recorder

This component stands out from the rest, both in terms of dependencies and development process.

Irregular This is the only irregular front-end component, the modules Knockout and Share+KO are not used since the binding of the model and view was deemed not necessary. Therefore their inclusion would add unnecessary complexity.

However, ShareJS is still used to download the experiment document, in the beginning of the run, and to upload the tracked data to the server.

In the code, the calls to the ShareJS API have been compartmentalised, so that if ShareJS is replaced, the refactoring is somewhat encapsulated.

Development Lead This component of the CUE client was the one that suffered the most development. Every other component of the client did not start before this one reached a satisfying level.

The reason for this is because the data structure was going to suffer plenty of revisions, as the requirements of this component were expected to change with time, and eventually did.

Whereas, the other client components, even though dependent on the data structure, did not necessarily have much influence on the requirements of said structure.

Meaning, if these components were to be developed before, they would have had to be changed later on multiple times, as the data structure suffered multiple iterations through out the development of the runner.

4.3 Data Structure

Data is stored in JSON documents.

A JSON document can be seen as dictionary of keys and values. Keys can be almost any string of numbers and letters[1]. The values can be a another JSON document, numbers, character strings or an ordered list of values.

Documents can only be retrieved as whole and not partially. It's for this reason that experiment runs are on separate documents from the experiment document itself. Otherwise, every time an experiment was downloaded by the client, all the runs would be downloaded with it, leading to poor scalability with a high number of runs.

A document oriented database is schemeless by nature, meaning it does not enforce a fixed structure on the data.

Even though the structure is not enforced by the database, the documents still need to follow some sort of structure in order to be usable.

Since the classical entity relationship (ER) diagrams do not fit the document-oriented approach, and since documents resemble simplified object graphs, a UML class diagram was chosen to represent the different document structures, as seen on figure 4.3.

There is however a short sight of using a class diagram in this case, which is the fact that the keys to retrieve some documents are inside of other documents,

[1] What constitutes an accepted key string is defined in the JavaScript standard, which JSON is based on.

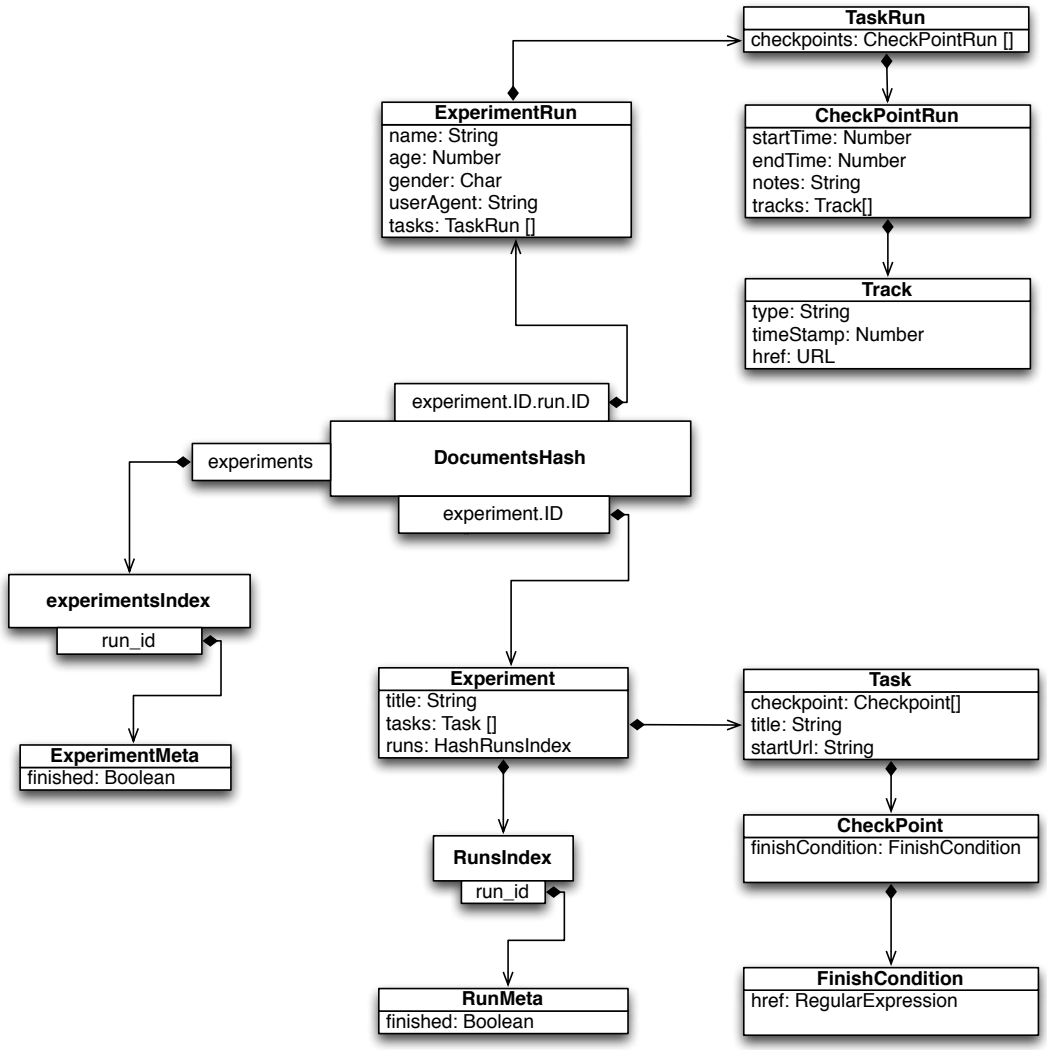

Figure 4.4: Data Structure Diagram

therefore they have an implicit association that cannot be properly represented in the diagram.

In the following sections the different documents classes are detailed and explained.

4.3.1 Document Hash

The documents hash is not really a document, it is the api used to retrieve documents.

It was named an "Hash", because the way to retrieve a document is by its key, very much like a value is retrieved in a regular hash.

The key itself has some structure to it.

In the following code it is shown how a document can be accessed.

```
1  sharejs.open('experiment.1.run.2a', 'json', function(error,
       experimentRun) {
2      ...
3  });
```

'experiment.1.run.2a' is the retrieved document's key. The example document is an experiment run with id '2a' from the experiment with id '1'.

With regards to the key formatting, a key can be of the following types:

```
1  experiments
2  experiment.ID
3  experiment.ID.run.ID
```

'ID' itself is a type, it's a string of numbers and letters, which must not include the character '.'.

4.3.2 Experiments Index

The experiments index it's one JSON document only, a singleton. This document functions as an index for all available experiment ids.

4.3.2.1 Experiment Meta

The index also has a preview of data on the experiment's it is indexing.

Such preview information can useful for group operations.

For example, if a common group operations needs to be performed on a subset of all of the experiments, the value that discriminates said subset can be placed in the Experiment Meta, rather than on the experiment document.[1] This way, every

[1] An hypothetical group operation could be deleting all the experiments created in the past month.

single experiment document does not need to be retrieved in order to determine the set.

4.3.3 Experiment

An experiment document contains all the information necessary to run an experiment.

To retrieve an experiment document its id is necessary. The IDs can be found on the Experiment Index. The key to access the document can then be constructed by appending the ID to "experiment.". eg: "experiment.12"

4.3.3.1 Task

Tasks are stored in a list of experiment's document. Their order matters, as it defines the sequence on which the tasks are displayed to the subject.

4.3.3.1.1 Checkpoint The concept of a checkpoint is to have an action performed after a certain event happens. There can be only one active checkpoint at a time. An active checkpoint waits for its trigger event to fire to then perform its action. After an action is performed, the next checkpoint is activated.

For now checkpoint are used only for two things: to divide the tracked data in an experiment and to trigger the next task.

The condition that triggers checkpoints are only URLs for now.

When the URL of the test application matches the URL condition, the checkpoint is triggered and its task executed.

The URL condition is a regular expression, which can match more than one URL [1].

When a checkpoint is triggered, the next available one is activated.

The last checkpoint of the task triggers a task change, or, in case there are no more task, finishes the experiment.

[1] This can be usefull for when a checkpoint can be triggered by different URLs. For example, in a use case where the user's name is present on the URL condition, it wouldn't be possible to list all of the possible urls, since the list of possible user's name is immense, but with a regular expression, it is indeed possible to define it.

4.3.3.1.2 Task Description A task has a task description, which has the information necessary to inform the test subject of what it is that he needs to perform.

4.3.3.2 Runs Index

Like the experiments index, the runs index has all of the ids of the runs of the experiment in question.

4.3.3.2.1 Runs Meta Like the Experiment Meta, the purpose of this document type is to make group operations less computational intensive.

4.3.4 Experiment Run

All of the information tracked when an experiment is being undergone by a subject is stored in this document.

4.3.4.1 Task Run

A task run is where the tracked information of a task is stored. Task runs are stored in a list, this list has the same order as the list of task found in the corresponding experiment document. This order must be preserved, otherwise it is not possible to correspond the task run to its task, which is essential.

4.3.4.1.1 Checkpoint Run The actual tracked information is stored in the Checkpoint Run documents. A checkpoint is the most basic unit of a task.

Notes can be stored per checkpoint, providing a granularity that can be defined by the tester himself, since it is he, who defines the checkpoints.

Track The relevant information about a tracked event is stored in a track. For example, if a url changes in the application, the new url, together with the timestamp of the change is stored in a track. Tracks are stored in a list, the order represent the chronological order of when the event took place.

Chapter 5

Risk Analysis

In this section an analysis of the risks associated with this project are detailed, along with their mitigation strategies.

5.1 Open source tools

Using open source tools always carries the risk of facing poor support, now or in the future.

Most tools used in this project are actually quite mature and do offer very good support, both in the form of documentation and community support.

A good proxy for maturity of a tool is the number of companies that use it in production, it's a sign of trust. A good set of functional tests also provides confidence that the tool will not break behaviour as it is updated in time. All of these factors were taken into account when choosing the different open source tools.

However, some of the tools used in the development are not ready yet for production, they have not been tested well enough and are undergoing active development.

The biggest risk was to have a show stopping bug that would invalidate the entire architecture. In order to mitigate this risk a loosely coupled strategy was used to insert them in the architecture. Meaning, if they had to replaced, the amount of refactoring would be small, or at least very localised.

Also a proof of concept was always the first priority, in order to make sure that the architecture plan was valid and that it could move forward. The proof of concept was also timeboxed, in case it wasn't feasible a different approach would have to be tried.

5.2 JavaScript

The entire stack of the CUE is written in JavaScript. The the risk exists of future project contributors to not be familiar with the language.

But since the profile of these possible contributors is unknown, the only way to analyse this risk is to look at the popularity of the language.

The trend is and has been growing when it comes to JavaScript adoption. The open source community has embraced the language, with GitHub reporting it as the most popular language on their website.

The chances of JavaScript completely disappearing as a language are slim, seeing as it as part a core component of the Web standard, which also doesn't seem to be going away.

5.2.1 Cross Browser Inconsistencies

The Web standards are constantly evolving. They have moved from a medium focused on the static display of information, to a much more interactive one. The interactivity is supported by extensions to the JavaScript and CSS API, which this project makes a lot of use of.

However, due to the standard evolving nature, browsers have in many occasions inconsistent behaviour.

Many factors contribute to it, such as:

1. Standards adoption from the part of the vendors
2. User adoption rate on up to date browser versions
3. Standards implementation discrepancies

5.2.1.1 User adoption

This problem seem to be fading as most users run a fairly updated browser [1] . Also, features like automatic updates are starting to emerge[2], which ensure more effectively that the browser remains updated.

5.2.1.2 Implementation discrepancies

Even though the trend seems to be improving, inconsistency issues are still common and they have happened in the development of this project.

The most problematic happened on iOS6[3] and delayed the first real world test by a week.

Since it is not feasible to test for every single browser and their different platform implementations [4], the following process was used:

1. Daily builds tested on up-to-date Chrome (Macbook Pro)
2. Weekly builds tested on iPad and iPhone5, running the latest iOS version.

Testing on more devices would have been unfeasible for a team of only one member.

5.2.1.3 Standards adoption

Standard adoption varies quite a bit between browsers, however a good mitigation strategy employed for this issue was to the website caniuse.com [5] .

This website allows to check the support, in percentage, for every web standard API. Every time an API was considered, this website was used, and if the support was poor, a fallback solution would be chosen.[6]

[1] http://www.w3counter.com/globalstats.php

[2] Chrome's default behaviour is to update itself without requesting confirmation from the user.

[3] The problem was specific to the ShareJS communication mechanism, that based itself on a common AJAX behaviour, the long pooling.

[4] Browsers from the same vendor can have inconstencies as well. Safari and iOS's Safari Mobile behave differently

[5] http://caniuse.com/

[6] Not even this strategy was always enough. The previously mentioned iOS6 bug could not have been mitigated by using this strategy, since AJAX long polling is not enforced by standard.

5.3 Future scalability

Even though scalability was not a priority in the development, it was still a concern when looking into the future of the project.

5.3.1 ShareJS

The biggest risk in terms of scalability in the architecture is ShareJS. The library is under going heavy initial development and hasn't been tested properly in production.

However, it is gaining a good momentum. The developer, Joseph Gentle, has now joined the DerbyJs team [1] [2].

This merge won't affect ShareJS negatively, it will remain to be a separate library, focused only on real-time data synchronisation. The library will fit under Derby and, hypothetically, other similar frameworks.

The merge will however create a positive architectural change, with minimal impact on on the existing API. The new goals are precisely to make ShareJs more scalable.

Even if, in the future, ShareJS is found out to not scale to an expected quality, the module can be easily substituted by another similar library. That was, in fact, the motivation behind the development of the Share+KO library. By using this mediator, all the communication between the Knockout code and the ShareJS code is done through Share+KO.

This way both are loosely coupled, meaning all Knockout depended code can still be used even if ShareJS is replaced or if its API changes drastically in the future, the reverse is also true, however, Knockout seems to carry little risk on that regard.

However, the fallback solution found for that issue, WebSockets, is enforced by standard, ant the website was able to provide a figure fot it.

[1] https://groups.google.com/forum/#!msg/sharejs/Uvw86KlOr-g/EUcH9uei8WMJ

[2] DebbyJs is a full-stack framework for real-time collaborative apps. The team has also announced 11million dollars of funding recently. http://techcrunch.com/2012/07/27/move-over-meteor-derby-is-the-other-high-speed-node-js-framework-in-town/

5.3.1.1 Alternative

In case there is a need for replacing ShareJS completely, CouchDB seems to be a good alternative. It is also document oriented and the documents are JSON formatted as well. It does not support real-time operations however, so if that is to remain a feature, a communicator model similar to ShareJS will have to be implemented.

Chapter 6

Testing

The CUE was tested all through out the development using a MacBook Pro and the Chrome browser.

The User Interface of the application was also tested and reviewed by the Cambridge supervisor, Ian Hosking, weekly and modified accordingly.

However, some tests were done more formally and they will be discussed in the following sections.

6.1 Test Case: Smartphone Prototype

The CUE is in essence an application to the test other applications with regards to usability. Therefore it was necessary to have a good test case for it. The test case chosen was a prototype the author had developed in the first semester.

The test case application was an horizontal prototype of a SmartPhone's User Interface. It was all written in HTML, CSS and JavaScript, running solely on the browser. The web server acted as a file server and nothing else, no back office was developed.

This prototype was also a good opportunity for the author to familiarise himself with existing open source tools and evaluate then with regards to flexibility and maturity.

The requirements for the prototype were ever changing so the development had to be very flexible.

The requirements also kept growing, so a strategy to manage the complexity and allow for a good extension of the system also had to be employed.

The process for changing or adding a requirement usually followed the following pattern:

1. Quick implementation of the requirement, so that the stakeholders could quickly give their feedback on the design ideas.

2. When there was down time, due to the intermittent nature of the client's communication, the time was used to refactor the system and make it more prone to extensibility.

The smart phone followed in part the design found in many existing devices today: a launcher populated with different apps.

Many of these apps have similar building blocks: scrolling lists, navigation controllers, options menus, grid views, etc. All of these building blocks should be shared amongst different applications, eliminating redundancy by making the code more reusable across the system.

This approach was used on the prototype and ended up creating a nice set of controllers and views, all of which can now be used to make the development of future requirements on the prototype much faster.

Another advantage, is the fact that changing a requirement in the building block level as an effect on the all applications that are using it, effectively making the system respond to change more efficiently.

6.2 Non-Functional Requirements

In this section it is detailed how different non-functional requirements were tested.

6.2.1 Usability: Birmingham Experiments

In order to test the CUE with subjects other than the author and the supervisor, the team went to two social centres for the elderly.

The experiments were divided in two weeks. Initially the weeks were planned to be used to test the CUE together with its main test case: the smartphone prototype.

Due to a bug caused by a last minute update to the Operating System of the test equipment, the first week of the experiment was changed to a pilot for the smartphone prototype only.

In retrospect, this was an advantage, since if the experiment had not been phased, and the CUE and the smartphone were tested at the same time, it would have been more difficult to isolate the data collected for both.

This way, in the second experiment, the author was able to better discriminate the problems specific to the SmartPhone, making it also easier to identify problems exclusive to the CUE.

The pilot also gave the opportunity to understand the environment on which the experiments were to take place. This lead to a revised protocol, which in turn led to CUE's requirements changes, including the addition of 2.2.1.3.

In the following section, the experiment on the CUE, using the smartphone prototype as the test case, is detailed.

Since the CUE itself is a platform to create experiments for other applications, inevitably testing the CUE leads to testing two applications at the same time. However, the research questions and the conclusions of the experiment for the test case are outside of the scope of this report. The methodology and findings of the smartphone prototype are only mentioned when they overlapped with the ones of the CUE.

6.2.1.1 Aim

There were different question which were trying to be answered. The focus of the experiment was on the Usability on the subject executing experiment tasks through the CUE. The usability of the tester's use case were not the focus os this test.

The questions to be answered were:

1. **Task Interference** How much does the CUE interfere with the execution of the task?

2. **Intructions Presentation** Are the instructions presented clearly?

3. **Clear Separation** Is the separation of the CUE and the test application clear to subject?

4. **User Interface Responsive** Is the user interface responding properly to the user intentions?

5. **Versatility** One of the purpose of the CUE is to be able to set remote experiments, but can it also function well in local scenarios?

6.2.1.2 Sample

The experiments happened in three difference social centres in Birmingham. The age demographics of the subjects was in the age range of 60 to 100 years old. Almost no subjects had experience with technology in general and few had ever used a phone.

6.2.1.3 Hypothesises

The running hypothesis was that some users would find it hard to dissociate the CUE from the test application and would end up trying to interact with the CUE's UI rather than the test application.

6.2.1.4 Methodology

The methodology together with his protocol was elaborated by Ian Hosking. The first run of the experiment was performed by Ian himself, while the author observed. The following runs were performed by the author, following Ian's approach.

Protocol The experiments were run on a two person basis, with the tester (the supervisor of the experiment) and the subject sitting side by side on the same table.

In the beginning of the experiment, the subject was instructed by the tester of how the experiment was going to proceed.

It was briefly explained that they were going to play with a smartphone prototype and that the instruction on what to do could be found at the top of the screen.

Due to the nature of the subjects, the protocol also included subjects who might have vision problems. For those who had difficult seeing, the subjects were explained that they could request for the tester to read them the unreadable text.

Due to a lack of a time, it was stipulated that the tester could intervene in case the subject found himself stuck. This would allow to keep the momentum of the experiment going and potentially find more relevant data.

Methods The methods used to collect data were mixed. Qualitatively notes were taken by the tester has he observed the experiment. In the end of the experiment follow up questions were also asked.

Quantitatively, the execution time of the tasks, together with the navigation path taken by the user was also being tracked by the CUE.

Test Devices Due to the lack of internet connectivity. Both the CUE server and the Smartphone prototype server were hosted locally on the author MacBook Pro, which acted as a private local router.

An iPhone 5, running iOS6, was used by the subjects to perform the experiments. This device was chosen in part due to its tall height screen. The height was able to accommodate both the task description and the test application.

6.2.1.5 Execution

The final sample size for the experiment was of usable 8 data points. Some data points were ignored due to some subjects lack of cognitive skills.

In general experiments ran without much problems, however some connectivity issues, between the Macbook and the iPhone, did happen on occasion, but luckily they did not disrupt running experiments midway.

6.2.1.6 Conclusion

In general, subjects tended to blame themselves for the issues they faced with the User Interface. Therefore discussions between the tester and the subject were common through out the experiments, in order to try and understand the design problems more deeply.

Altough, it was underestimated how chatty the subjects can get. Even though the conversations gave a lot of insight to the problems, the time of execution of the tasks, which was being tracked by the CUE, ended up being inconclusive in part due to these conversations[1]. This, however, gave birth to the requirement 2.2.5.2.

The great bulk of data came from the qualitative methods. The observations made on site together with the follow up questions & answers lead to a great understanding of the problems the users faced, not just with the smartphone prototype, but with the usability of the CUE as well.

The answers to the research questions are briefly summarised in the following paragraphs:

Intructions Presentation The on-screen CUE instructions were overall understandable by most subjects. Doubts came only with the semantics of the task descriptions themselves, which are not a problem of the CUE, but of the tester who defined the experiment together with its tasks.

User Interface A recurring problem was found both in the CUE as well as in the smartphone prototype. The users in this experiment tended to press really hard on the screen in order try and press a button.

Since the touch screen is oleophobic, as the subject pressed harder, their fingers tended to slip and move away from their first point of touch.

This was a problem because the way a button press is registered by the browser follows the following model:

1. User touches button

[1] The small sample size was a factor as well.

2. If user moves his finger away from the button target area and then lifts his finger, the button press is cancelled.

3. If the user lifts his finger while still inside of the button target area, then a button press is registered.

What happened to most users was the second case. They pressed so hard that by the time they lifted their fingers, it was already outside of the target area.

A possible solution is to register a button press as soon as the user touches the button; or make the button target area bigger than its visible area[1].

Clear Separation As hypothesised some users pressed the CUE's description of the task in hope some action would ensue.

One way to fix this problem is to have a screen pop up every time the task description is pressed, reexplaining how the experiment is supposed to be performed.

However, such solution needs to be tested, since it might lead to even more confusion.

Task Interference The smooth and relatively slow transitions of the CUE's experiment runner created a user flow that didn't seem to interfere with the execution of the task.

Versatility The CUE was found to be versatile in local experiments like this one. The qualitative data was so insightful that a new requirement, 2.2.5, which allows to observe running experiments and take digital notes on them, was added out of necessity.

6.2.2 Applications Compatability: WebProxy

The Web Proxy is an integral part of the architecture, without it only web applications with the same domain as the CUE will work.

[1] Apple's iOS applications follow both approaches in differente cases

The main way to measure the quality of the requirement 2.3.1, is to measure the degree of compatibility of the Web Proxy to different existing web apps.

For this initial phase of the development, only the main test case, the smartphone prototype, was required to work with the Web Proxy.

However, during the development of this component the developer did informally test the proxy with other web sites. The testing was explorative by nature. The methodology was simple:

1. Turn off browser extensions (Adblock, NoScript).
2. Open a website through the proxy on one window, open the same website on another window, and put them side by side.
3. Execute the same operation on both windows and then check if the result are the same.

Amongst the tested web sites there were Google, Google Maps, Imdb and Amazon, which all worked well apart from operations that required the client to store cookies (hold state).

Naturally, more web sites need to be tested with the proxy, but this set gave a some what good understanding of how functional the web proxy is at the moment.

6.2.2.1 Automated testing

Due to the very systematic employed testing of the proxy, an automated testing routine could be used to implemented it.

Tools like PhatomJS [1] [2] could be used to interact with both the original and the proxied version of a website and then inspect their HTML document, looking for relevant differences.

6.2.3 Scalability

Scalability was not able to be tested.

[1] http://phantomjs.org/

[2] PhatomJS is a headless scriptable WebKit JavaScript API. In a context such as Node.js, PhatomJS can be used to programatically interact with a browser and its HTML documents.

However, there are some components that should be tested, amongst them are:

1. **Web Proxy** Its apparent bottleneck is the rewriting of the HTML document to proxy. It can be optimised both by caching the proxied document and by making the rewriting a non-blocking operation.

2. **ShareJS** It has been mentioned before 5.3.1 that ShareJS might have issues scaling, so testing on this end should be performed.

6.3 Functional Requirements

The functional requirements were tested by the developer himself, since their evaluation was simple and binary by nature.

The CUE was tested all through out the development using a MacBook Pro and the Chrome browser.

Two iOS6 devices, iPad2 and iPhone 5, were also used for testing, but only on a weekly basis.

Both integration and unit testing were performed by the developer himself. Opportunities like the Birmingham experience and the weekly meeting with the supervisor also allowed for explorative testing that lead indirectly to the testing of functional requirements.

6.3.1 Automated testing

Automated testing was not implemented for any of the requirements.

This was in part due to time constrains.

However, as the architecture grows automated testing should be employed in order to avoid regression bugs.

The author experiment with some JavaScript testing frameworks. Mocha [1] was found to be a good solution, together with PhatomJS. The latter is to used for more complicated integration tests and the former for more simple unit testing.

[1] http://visionmedia.github.io/mocha/

Chapter 7

Planning and Work Process

In this section the team and stakeholder involved in the development are discussed as well as the processes used.

7.1 The Team

The team was composed of the author, as the lead developer and architect; and the Cambridge supervisor, Ian Hosking, as the project manager.

7.1.1 Stakeholders

There were two stakeholders of the CUE and its main test case. Ian Hosking served as the bridge of communication between the them.

7.1.2 Company X

Company X, here not mentioned due to requested confidentially, was the main stakeholder for test case of the CUE: the smartphone prototype. They also had an interest in the CUE as a project, however their stake in it was small, but may grow in the future.

7.1.3 EDC: Cambridge Engineering Design Centre

The Cambridge Engineering Design Centre (EDC) undertakes research to create knowledge, understanding, methods and tools that will contribute to improving the design process.

The author is a payed Research Assistant with the Inclusive Design Group. The Inclusive Design Group is part of the EDC, but its research is scoped to improve the design process of day to day products so that they can include a larger set of the population.

7.1.3.1 Ian Hosking

The main stakeholder of the CUE is Ian Hosking. He was the project manager and the communication bridge between all the parties involved. His role in the EDC is that of Senior Research Associate.

He is involved in multiple projects and has had over 20 years of industry experience applying usability to commercial products.

7.2 Agile Processes

Due to the small size of the team processes were considerably informal, agile by nature.

Weekly meetings where held between the two, where both would propose and discuss potential requirements. The developer would then give his best estimate as to the complexity of the proposed feature and a priority would be assigned to it.

This meetings were also held to ensure that the development followed the initial plan.

Deliverables were always on the form of working software and milestones were set at different points in time.

There was, however, some difference in the development of the CUE platform and its main test case: the smartphone.

The CUE was much more methodic, with much more consideration taken on the design of the architecture. The Prototype, due to its ever changing nature,

was much more hands-on, with implementations cycles as short as one hour.

7.2.1 CUE Development Process

The first semester was focused on researching the state of the art for existing usability testing tools. A preliminary architecture was also envisioned, together with a proof of concept for it.

The second semester was dedicated to development of a solid and well though architecture, together with the development of a working version of said architecture.

An important milestone was the Birmingham experiments. Since it was there that the system underwent its first real world validation.

7.2.2 Smartphone Prototype

As mentioned before the prototype's requirement changed very often. Therefore, a different approach had to be used. Most new requirements were very quickly implemented.

However, if this approach was taken without any control, the system could have easily grown to an unmanageable state. To compensate for this, on down times the developer would refactor the system to eliminate redundancies and increase its modularity. Version control was beneficial here, to ensure that older working versions were still accessible.

7.2.3 Version Control: Git

Both the development of the CUE and its test case used Git as its version control.

There are two main reasons why the author chose GIT. They are discussed in the following sections.

7.2.3.1 Open Source

Git is a popular version control system for open source projects, aligning it well with the Open Source requirement. It is also the revision control system of

GitHub, the most popular open source online repository [1] , potentially facilitating future contributions to the project in the future, as specified in the requirement 2.3.2.1.

7.2.3.2 Decentralized

Git is distributed, unlike many other version controls, like for example SVN.

What this entails is that every development machine has a local working repository of the project. Commits can be made without a connection to a centralised repository. When repositories are synchronised, conflicts are resolved automatically when possible, otherwise the developer specifies the resolution.

This design solution allows for the developer to version their code even when offline; and also adds portability to the project, since every local repository can be used to start a new online repository.

The disadvantage to a more centralised approach is that Git is inherently more complicated to grasp at fist.

However, online support by the community is extensive and even the author of Git, Linus Torvalds, has published educational videos explaining the core concepts [2] .

7.2.4 Planning

The first semester was more explorative in nature, with the author evaluating different Usability tools in order to develop the requirements of the CUE to a satisfying level. Technical solutions were also explored and a rough draft of the Architecture was envisioned.

The first CUE deliverable was as a proof of concept. This proof of concept revealed too many problems with the underlying architecture, but also helped to clarify the future design.

In the first semester the development of the CUE was done in part time. Since the author was working full time, he was allocated to another simultaneous

[1] http://readwrite.com/2011/06/02/github-has-passed-sourceforge
[2] http://www.youtube.com/watch?feature=player_embedded&v=4XpnKHJAok8

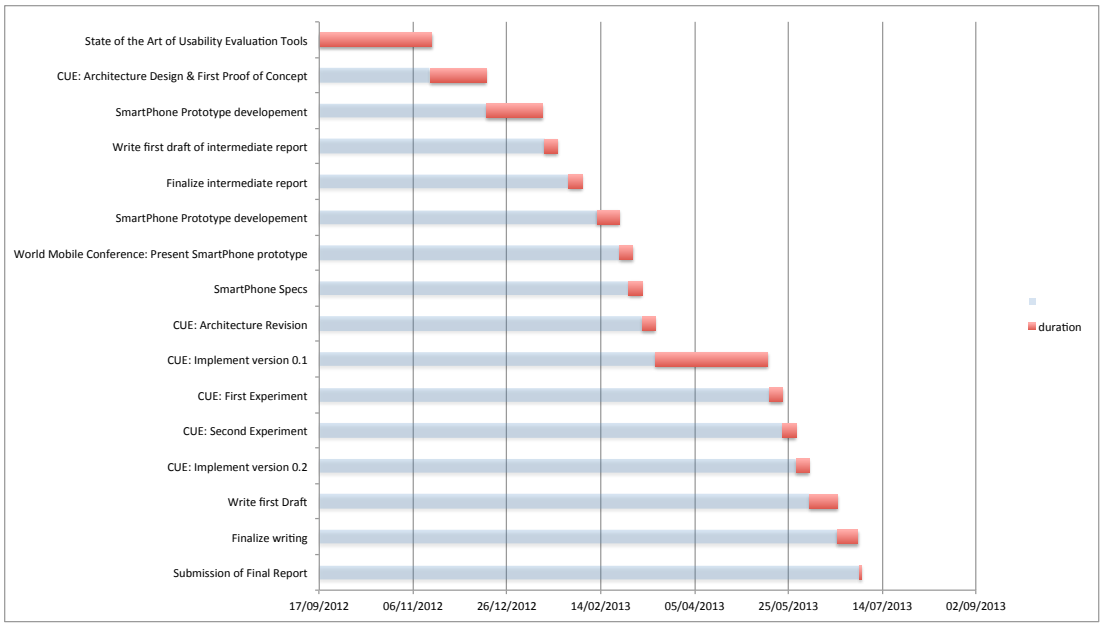

Figure 7.1: Gantt Diagram of the work plan of the CUE and its main test case: the Smartphone Prototype.

project: the development of a smartphone prototype. This prototype ended up being the main test case for the CUE, aligning the two projects together.

In the beginning of the second semester, due to the involvement of the author in the development of the smartphone prototype, he was invited to join the company X to the World Mobile Conference in Barcelona. There he presented the prototype to potential investors.

After the conference, the author joined the founder of the company X on a one week workshop on the smartphone.

Afterwards, the author was exclusively dedicated to development of the CUE. A detailed plan of the implementation was elaborated after the architecture designed was revised.

A milestone was set for the development, where the CUE was to be tested with real users. The focus of this release was to have the Experiment Runner & Recorder fully working.

After the experiment, a new requirement, 2.2.5, was added and a short cycle of development was performed in order to implement it. The requirement was

implemented and tested by both the author and supervisor.

Its relative quick implementation was a show case for the flexibility and extensibility of the system, since both the data structure was changed, in order to accommodate the requirement, and the functionality increased with the addition.

Chapter 8

Conclusion

In following section the author describes the value of his personal experience. And in the last sections future plans are mentioned.

8.1 Personal Experience

The learning experience through out the year has been tremendous. I was not expecting to grow my knowledge so deeply and yet, at the same time, so broadly.

I was taught how to design User Interfaces keeping in mind the prior experience and mental model of users.

I saw first hand how users can struggle even with the most basics of designs; and how some design oversights, deemed small by some, can damage the experience for so many groups of users, specially the elderly.

I experimented with multiple web tools, from small libraries to full fledge frameworks. I formed my own opinion on their strengths and weakness and I feel prepared to architecture systems that can be both flexible and extensible.

I used almost all available Web APIs and I feel empowered when it comes to implement an highly interactive web client, with both good levels of performance and fidelity.

Unfortunately, scaling systems in a production scenario was not part of the experience, but hopefully it will be in the future.

The writing of the report itself was also a learning experience. The process

helped me formalise and clarify the thinking employed through of the development of the project.

8.2 Future Plans

There still are some functional requirements which need implementation.

It would be interesting to see the community use the tool and suggest new requirements.

Automated testing should also be employed, in order to facilitate the contribution of external developers on an open source level.

Appendix A

State of the Art

A.1 Analysis of Usability Testing Tools

A.1.1 UserTesting.com [1]

This is a web based user testing application, which allows companies, or whoever is interested in usability, to create testing tasks for any website or mobile app. These tasks will be performed by regular users, who are associated with the UserTesting.com company. Their execution of the tasks is video recorded and when complete, the video is sent to the client along with other information that the test subject might find relevant to point out.

Technically, the actual execution of the testing experiments is not described in depth by the company. This is due to the fact that actual testing is not supposed to be of the concern of the client, the client's concern should be the testing results only. So it's possible that testing tools given to the test subjects might be rather crude; and that the test subjects themselves are taught how to use them.

A.1.1.1 Advantages

1. **Video and Audio Recording** A full recording of a test subject's session can be very rich, specially if they vocalise their thoughts, which seems to be part of the guidelines: [use12a, "To be a helpful user tester, you should

[1] www.usertesting.com

be good at thinking out loud"]

2. **Targeting Users** Being able to choose, demographically, the users to get feedback from, [use12b], is a huge advantage to products with segmented markets.

3. **Follow-up questions** It's possible to ask follow up questions to the user who provided the feedback, [use12b], allowing the designer to explore and understand the user problem at a much deeper level.

4. **No prototype integration needed** Since the tests are recorded with a video camera, any product can in theory be tested, it all depends on the test subject's abilities.

5. **Test Subjects Screening and Client feedback** Qualitative research is limited by the users selected. Therefore, the quality of the users is crucial for this type of research. The test subjects need to represent many groups and also need to be vocal and concise about their impressions as much as possible.

   In order to try and improve this facet, UserTesting.com states that they screen the test subjects and act on the client's feedback of said subjects.[use12c]

A.1.1.2 Disadvantages

1. **Video data extraction** Even though video recording is very information rich, it's also not easy to extract information from it in a programmatic manner.

2. **Experimentation Bias** Even though the company seems to employ good quality assurance methods to their body of test subjects, there is still a possibility that these test subjects become too trained for testing, making themselves less representative of real users due to being over exposed to too many testing scenarios.

3. **Prototyping toolkit** The service is completely decoupled from the prototyping, to the extent of not offering any tools or services from the end users

to design their designs in terms of usability. It's focus towards more mature products.

A.1.2 Concept Feedback [1]

In this web service the user submits his work, which can range from a fully fledged application to wireframes. Then the asset is analysed by a set of experts, who afterwards feedback their opinions to the client.

This web service is very simple technically. In fact, the only reason it's being analysed is simply to make sure all the options are considered.

A.1.2.1 Advantages

1. **Expert Feedback** Not forgetting the goal of this project, which is to feedback relevant usability information into the design process, getting feedback from a good set of experts might be quite helpful when designing a product.

2. **Review Content Type** Since information is processed by UI knowledgable individuals, almost any piece of content interpreted by them can be submitted for review([Fee12]), reducing integration issues to a bare minimum.

3. **Reviewers Reliability** Since only qualitative analysis is provided, the reliability of such data boils down to the reliability of the reviewers. Fortunately, Concept Feedback puts a face to every reviewer and makes sure there is a detailed bio to go with it.

A.1.2.2 Disadvantages

1. **Qualitative Analysis Only** Concept Feedback does qualitative analysis in a very efficient and transparent way, but since even experts can be wrong ([Wat11] [1]), so not providing any form of quantitive data needs to be pointed out as a disadvantage.

[1] http://www.conceptfeedback.com/

[1] In his book Watts argues to distrust experts opinions and gives examples of frequent flawed predictions by fields experts. Although the examples given are possibly not applicable to Usability, it's still important to consider the inherent subjectivity of qualitative feedback.

A.1.3 Optimal Workshop [2]

This website allows clients to submit prototype data in three forms, it even separates their forms into three differently branded products. They are the following:

Challmark [3] In this case the user submits a representative image of a state of the prototype, either a wireframe or an actual screen shot. Then, usability tasks are defined. These are very bare bone tasks, they are all of the form: "If you want X, where would you click?". These tasks become associated with the representative image. In order to determine if the test subject is successful or not, the client also has to define the hotspot in the image, where the user has to click in order to succeed.

OptimalSort [4] Grouping items is a very common task when designing for usability, it's also far from an easy problem to solve. It's not just a matter of semantics, it's also heavily influenced by users' prior experience and how navigation is actually undergone. Regardless of these considerations, the OptimalSort takes a more pragmatic approach: it asks client chosen users to sort the items themselves, in a very abstract way. The users are given a list of items, which then they can create named groups with; the items and groups can also be sorted and grouped around.

TreeJack [5] Navigating between menus or pages is a problem all users face when browsing applications. Therefore the optimal organisation of content is a recurring problem for most designers and one without a simple answer. It's for this usecase that TreeJack is useful. The client/designer only has to input a tree of information. Afterwards he can define a set of associated tasks, all with a defined end node. If the test subjects reaches the end node, the task is considered complete. Along the test other relevant informations are also collected, like time spent on a particular node and the path taken by the subject.

[2] http://www.optimalworkshop.com/
[3] http://www.optimalworkshop.com/optimalsort.html
[4] http://www.optimalworkshop.com/treejack.html
[5] http://www.optimalworkshop.com/chalkmark.html

Test subjects and contributors are sourced by link sharing. This approach separates the testing utility from the subjects sourcing, allowing the client to choose the service for that effect.

The service is just a regular website, it provides no integration with real working prototypes, simply because the "prototype" is actually rendered by the website itself, based on client submitted data.

A.1.3.1 Advantages

1. **Cross platform** Since the possible prototypes tested by the service are actually generated by the website itself, it makes them cross platform, at least for platforms that have a proper Web support.

2. **Data Breakdown and Navigation Tasks** It is possible to define classical navigation tasks, like "Find X", and from the data of such experiments a lot of processed data is made available. Data such as: users path breakdown [tre12] and heatmaps [cha12] .

A.1.3.2 Disadvantages

1. **No Working Prototype Integration** Since the service offers no integration with existing prototypes, the features that the client can test are limited to the three offered products. Therefore more complicated interactions cannot be tested with this service.

2. **Abstract Style** Since the prototypes are rendered by the website itself, they all are presented with the same style. So styling decisions, which more often that not end up influencing usability, cannot be tested, limiting again the testing scope of this service

A.1.4 Silverback [1]

Silverback is an application that allows to create synchronised composed video of the user's screen and webcam feed. But unlike UserTesing.com, Silverback, as

[1] http://silverbackapp.com/

a product, is actually a system application and does not provide any service to actually test the prototypes. It is the client's responsibility to set the experiments and use the application in the way he see fit.

The application can run on OSX systems and is simple to set up.

A.1.4.1 Advantages

1. **Decoupling** By decoupling the data collection from the actual usability testing, Silverback allows clients to source their own test subjects, therefore having more control of their own experiments.

A.1.4.2 Disadvantages

1. **Not cross platform** Since it's an application that runs only on Mac OSX systems, other platforms are not supported, leaving out mobile applications and other non Web and non OSX desktop applications.

2. **No Test Subject Sourcing** As with many good features, they can be a double edge sword. In this case, the previously mentioned Decoupling is good on one hand, but the absence of a service that can source the test subjects and guide them through the experiment might steer way clients who are looking for a one stop solution.

A.1.5 ClickHeat [2]

ClickHeat is an HTML mouse click logger. It displays a heat map, illustrating the regions of a page that are most often clicked on. This provides a hint to the designer about the most often clicked links, and therefore the ones to prioritise the most. It also says something about possible pitfalls, for example: regions of a page that might look like a button, but are not and therefore need style changes to further illustrate their function or lack of it.

The application is split into a server side data collector and a javascript script that actually does the collecting and sends it to the server.

[2] http://www.labsmedia.com/clickheat/156926.html

A.1.5.1 Advantages

1. **Open source and decoupled** ClickHeat is not exactly a user friendly application. It is more of a framework, a plugin that developers can implement on their servers. Therefore, it does allow for great versatility of usage.

A.1.5.2 Disadvantages

1. **Configuration heavy** The fact that the application requires both installation on the server and on the client side code, makes it quite integration heavy for a client who just wants to quickly test a web app.

2. **Lowed feature** It is true that the application is meant to be just one piece in a greater puzzle, but either way, logging just mouse clicks is not enough for a proper quantitative analysis. Other user inputs like: mouse moves, scrolls, touch events, should have also been included.

A.1.6 ClickTale [1]

ClickTale records every move, every input, from every user. With that data it then tries to extract relevant information and present it to the client. The Client will then be able to better understand possible problems with the design of his application; and at the same time get a better understanding of the audience.

It's also more of a business analytics tool, but it's focus on usability and number of related features is impressive.

The way the data is collected is through a small javascript file that is run on the user browser and that collects all the user generated events and then proceeds to send them to a server. Integration of the service and the client's application (or prototype) seems minimal, possibly only a line of markup code referencing the collector script has to be inserted on the documents sent to the end user.

A.1.6.1 Advantages

1. **Data readability and completeness** Since every user input is recorded in a computer readable format, the potential of extraction of quantitive infor-

[1] http://www.clicktale.com/

mation from the data collected is very significant.

2. **Mouse movement heat map** There is a strong correlation between eye movement and mouse movement, and seen as eye tracking data is actually quite relevant to understand how the design of the information presented is being perceived by the end user, it does make this feature quite appealing.

3. **Attention Heatmaps** From coalescing such data as user scrolling, mouse moves and mouse clicks (and in the future touch taps as well, [cli12b]), comes a feature that is actually of real of interest: attention heatmaps. heatmaps are a tool that try to illustrate the importance that a user gives to certain section of a product. The problem they usually have is that they are either based on clicks, moves or scrolls, never on an aggregation of the three. This is where Attention heatmaps differ. They try to coalesce different type of user inputs to create a more representative heat map of user attention.

4. **Extensive Integration** Since ClickTale is an always recording service, it is always collecting data from all the website visitors, the integration with the web application is significant. Thankfully, the service provides an extensive number of plugins to the most popular web platforms ([cli12a]).

A.1.6.2 Disadvantages

1. No path break down
   Although the data collected is quite complete, there is some relevant derivative data that is not being extracted, for example: the average path of users. The last example can be very helpful to see if users are getting lost or wasting too much time on irrelevant parts of the website.

2. No experiments set up
   The service just collects all of the end user's inputs, it does not allow the client to set up experiments. This makes it more complicated to reduce variables and isolate information, which is in may cases vital to really understand the problem.

3. No client API
   Even though the data collect is quite complete and has a lot of potential, there does not seem to be open APIs that allow the client to extract even more information from the dataset, limiting one of the best features of the service.

A.1.7 Google Analytics [1]

Google Analytics focus is more on marketing than on usability. One of the key features is telling the client from which page your visitors arrived from, this gives the client a perspective of how affective or not his marketing really is. Though it is possibly to track certain user generated events, the service was not designed with that in mind, making that kind set up expensive.

Google analytics runs a script on the browser, which gathers user data and then sends it to the google analytics server for processing. It is there that the client can read relevant information about how it users are using and arriving to his website.

A.1.7.1 Advantages

1. **High level marketing oversight**
   Information like user landing page, page of where the user came from and user trends can help the client to understand, on a high level, of how its users are behaving and possibly adapt his strategies from design to marking to better fit his needs.

2. **Quick Integration** Setting up is as easy as adding a marking line of code to the website's documents.

A.1.7.2 Disadvantages

1. **Little insight on possible cause of problems** Features like heat maps are not present on Google Analytics; though the service can point out possible

[1] http://www.google.co.uk/analytics/

problems, by warning about negative costumer trends, it does little to provide information that can lead the client/designer to where the problem in fact lies.

A.1.8 Ethnio

Ethnio solves only one problem of usability testing, finding experiment participants.

Using social networks and a good set of incentives, Ethnio is able to gather a good amount of participants that might not be users of the client's website. However, this is not the only way, Ethnio, by having some level of integration with the client's website, gives the client the tools to recruit a user from the pool of users who are actually browsing his website.

A.1.8.1 Advantages

1. **Integration with usability testing services** Not all services that collect and set usability experiments provide the client a way to recruits participants for said experiments. Ethnio provides this service and also allows integration with the services that do not provide it, like the already discussed OptimalWorkshop and UserTesting.com

A.1.8.2 Disadvantages

1. **Feature light** Like with other highly specialised services, the fact that is light on features might steer way users, seeing that costs from all the tools used to perform the usability testing might end up higher compared to other one stop solution services.

A.1.9 Feng-GUI

Feng-GUI does not exactly collect user data, it generates it; by using an algorithm that supposedly mimics real human user behaviour, the service can then generate heat maps for wireframes or screencast submitted by the clients.

There is no real integration with any working application/prototype, the only thing that can be submitted by the clients is representative pictures.

A.1.9.1 Advantages

1. **Low Cost** Since no humans are actually used by the service any time, incentives for the participants, which are often financial, do not need to be user, keeping costs then down.

A.1.9.2 Disadvantages

1. **No apparent validation of the approach** Regardless of how effective the algorithm provided by Feng-GUI might be, it does not seem to be backed up by any scientific research, at least it is not mentioned in their marketing. Not knowing the degree of correlation between the algorithm and real humans leaves the client in a very uncertain position, with regards to the value of the provided data.

A.1.10 Five Second Test

The Five Second Test allows clients to upload representative images of their application/prototype, which will then be presented to a pool of user for five seconds only. After their brief interaction with the application, users will then write their feedback on it. The reason for the 5 seconds limit is that most users usually just spend 5 seconds on a website before deciding whether they want to stay or leave it.

Technically, the website is very unimpressive. All it does is allow the client to upload an image or share a URL. There is no data processing of any kind, since the feedback is all human generated.

A.1.10.1 Advantages

1. **Mimicing real life scenario** Whether the service is actually able to reduce experimentation bias, by limiting the user's observation to 5 seconds, is

debatable, but it does bring an interesting idea to the table, that might be of interest to explore.

A.1.10.2 Disadvantages

1. No user segmentation
   Relevant user data like age, gender, are not presented to the client, therefore leaving user segmentation out of the list of features.

2. No rating mechanism for subject feedback
   Like with all qualitative analysis, the strength lies in the trust one has of the of the data provider, in this case the test subjects that are voicing their opinions on the client's product. Therefore a rating system on the quality of a given feedback, and by association the creator of such feedback, should be in place to make it easier for the client to trust the different opinions presented to him.

A.1.11 loop11

loop11 allows clients to set up experiments on website by defining tasks. Those tasks can then be performed by users and their interaction logged to its fullest extent. Data is breakdown on path analysis, time and how the user walked the page graph of the web site in order to try and reach his destination; heat maps tells a story per page, of how the user interacted specifically with the content of a web page.

Technically, loop11 seems to be the most interesting of all applications. It is completely decoupled from the application to be tested, making it unnecessary to have testing code running either on the application's client or server side code. After some brief inspection, the web technology used for containing the test subject application seems to be the iFrame. The iFrame is used to wrap the test subject application, whereas the container document actually belongs to the loop11. Loop11's container listens and log the user's interaction by sitting in between the user and the actual subject application's content. The recoding of the user inputs must be performed by a javascript script, that does not run on

the actual application, but rather on loop11's testing application.

A.1.11.1 Advantages

1. **Loose coupling** loop11's architecture is the one until now that provides the least amount of configuration; by giving zero effort on integration, the client can even test competitor's website. This also allows for the integration with prototypes created by HTML developing tools of the client's choice.

2. **Setting up tasks for working applications** Even though some services do allow for the creation of experiment tasks, very few permit to create tasks for real working web applications, let alone on competitor's websites. With this feature it is even possible to test certain user interactions without even developing them, that is if they are already developed some where on an existing website.

A.1.11.2 Disadvantages

1. **No raw data export** Even though data results can in fact be exported, the raw data cannot. This limits the client, who could use that raw data to extract even more information from the experiments.

2. **Task completion** Checking whether a task was completed successfully in an automatic fashion is a complicated problem, since a task might be completed successfully in more ways than one. However, by trying the demo provided by loop11, one can not be sure of whether they are able to do this for the most difficult cases.

A.1.12 GazeHawk

GazeHawk provides heat maps of a website or image. Unlike other websites who rely on mouse and eye movement correlation to bring significance to mouse move heatmaps, GazeHawk actually attempts to track the user's eyes using a regular webcam. They also have their own set of test subjects to whom they send the content to be tested.

It seems the eye tracking is done by a OS level application that is run by the subject as he executes a given test. There is no mention on the website of possibilities to license such app for general purposes, only if it is to set up test studies with the client's own test subjects; possibly such process is run under the supervision of GazeHawk to ensure protection of their IP.

A.1.12.1 Advantages

1. **No integration needed** Integrating the eye tracking solution with the client's applications is not a problem to the client, that is if GazeHawk's test subjects are used, since the integration cost is offloaded to them.

2. **Raw data availability** It is possible for the client to gain access to the raw eye point data, allowing him to extract even more relevant information than just heat maps.

3. **Outsourcing Test Subjects** Test subjects, other than GazeHawk's, can be used on the experiments, but I suspect that it will cost more for the service, since it will involve some level of application integration.

A.1.12.2 Disadvantages

1. **Not possible to specify user demographics** Even though it is one of the few services to provide an analysis of the user's taken path, it does lack in data extraction features, if one was to compare it to ClickTale for example.

2. **Little processed data** Though the raw data allows for great extensibility, the service could provide more than just heat maps as processed data.

A.1.13 Summary

Conclusion the different services seem to divide themselves into three different categories:
The different services seem to divide themselves in to two categories of analysis: quantitative and qualitative. Both of these groups seem to have very little overlap, meaning there is an absence of a service that provides both at a good level.

In terms of qualitative analysis, most services seem to rely on experts or random users reviews. The reviews are either done through text or with video and audio recordings of the subjects experimenting with the services.

In terms of quantitative analysis ClickTale seems to be the most complete of services. It's not the only service that fully logs all the user inputs, but it is the one with the most robust analysis of said data. GazeHawk is an interesting service due to being the only one to really collect eye tracking data. Other services rely on the correlation between mouse and eye movements to bring similar data to the client, but for touch devices, for example, that correlation is meaningless.

Very few applications actually have a subject sourcing service. Either it's not relevant to their product or they outsource that to the client.

In figure A.1.12.2 a comparative analysis of the features found in the different products is presented. Interestingly enough, only optimal Workshop tries to address the matter of fast prototyping in order to increase usability in the design process. Although their approach to prototyping is rather limited and abstract, it is indeed fast and does some what illustrates the potential of the idea, as well as its novelty value with regards to usability testing.

	Unsupervised Tasks	Visitors logging	Logging of user interactions	Playback (tester or visitor)	Data processing (quantitative)	Testers sourcing	Extensible (raw data export; open source)	Qualitative information	Integration (supported formats)	Prototyping Tools
UserTesting.com	General task definition	No support	No support	audio/video recording	No support	target users by demographics; no experts included	No support	Real-time user's impressions; Follow-up questions;	applications; no wireframes;	No support
Concept Feedback	No support	No support	No support	No support	No support	experts database	No support	Experts review	wireframes; applications;	No support
Optimal Workshop	Menu Navigation and Click the-right-element tasks;	No support		no support	path breakdown; heatmaps; destination tables	clients can/have to source the testers	No support	users can group menu items;	menu structures; navigation screen shots;	limited prototype construction
Silverback	No support	No support		audio/video recording	No support	No support	No support	Real-time user's impressions;	any format supported by tester and Mac OSX	No support
ClickHeat	No support	No support	mouse clicks	No support	heatmaps	No support	open source	No support	web applications;	No support
ClickTale	No support	all visitors only visitors with javascript enabled	mouse inputs; touch inputs (beta);	playback any user visit	attention heatmaps; form analytics; Conversion Funnels;	No support	No support	No support	web applications	No support
Google Analytics	No support		cumbersome to set up;	No support	marketing metrics;	No support	No support	No support	web applications;	No support
Ethnio	No support	No support	No support	No support	No support	provides tools for testers sourcing	No support	No support	web applications	No support
Feng-GUI	No support	No support	generates fake data	No support	heatmaps	No support	No support	No support	screen shots only;	No support
Five Second Test	No support	No support	No support	No support	No support	random pool of users;	No support	users' impressions;	screen shots only;	No support
loop11	General task definition	No support	mouse inputs; eye tracking; no actual inputs;	support	heatmaps; path analysis;	clients can/have to source the testers	No support	No support	web applications;	No support
GazeHawk	No support	No support		No support	eye tracking heatmaps	pool of users; exterior testers can be used;	No support	users' impressions;	web applications; wire frames;	No support

Figure A.1: Comparision of Usability Testing Applications

*

References

[cha12] Chalkmark results, 2012. Available from: http://www.optimalworkshop.com/chalkmark-results.

[cli12a] Click tale integrations and plugins, 2012. Available from: http://www.clicktale.com/support/integrations_and_plugins.

[cli12b] Click tale mobile beta, 2012. Available from: http://research.clicktale.com/ClickTale-Mobile-Beta.html.

[Fee12] Concept Feedback. concept feedback faq, 2012. Available from: http://www.conceptfeedback.com/faq/.

[IIB09] IIBA. *A Guide to the Business Analysis Body of Knowledge.* International Institute of Business Analysis, 2009.

[Nie93] Jakob Nielsen. *Usability Engineering.* Morgan Kaufmann Publishers Inc., 1993. Available from: http://www.nngroup.com/books/usability-engineering/.

[tre12] Treejack results, 2012. Available from: http://www.optimalworkshop.com/treejack-results.

[use12a] usertesting.com: become a testerr information, 2012. Available from: http://www.usertesting.com/be-a-user-tester.

[use12b] usertesting.com features, 2012. Available from: http://www.usertesting.com/features.

[use12c] usertesting.com who are the testers, 2012. Available from: http://www.usertesting.com/who-are-the-user-testers.

[W3C02] W3C. Usability - iso 9241 definition, 2002. Available from: http://www.w3.org/2002/Talks/0104-usabilityprocess/slide3-0.html.

[Wat11] Duncan J. Watts. *Everything Is Obvious: *Once You Know the Answer*. Crown Business, March 2011. Available from: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0385531680.