

Extending an IDE to support input device
logging of programmers during the activity of
user–interface programming: Analysing
cognitive load

Simon Buist
The University of Bath

April 26, 2014

Abstract

This dissertation provides a browser-based Integrated Development Environment [IDE] that logs input device data for the purpose of performing user participation studies, whose implementation is described herein. The IDE is then used to conduct a study comparing the cognitive load experienced with two languages, Elm and JavaScript, during the completion of a basic task: to modify given source code in order to restrict the movement of a 2-D box from leaving the bounds of a frame. In order to do this comparison, the metric of **number of mouse clicks per code region** is selected, as an operationalisation of the concept of *thrashing* (Lopez et al., 2012), as being indicative of cognitive load during task completion. The study found that there is indeed statistical significance in the differences between clicks per code region, and presents suggestions for improvements and further work.

Contents

COPYRIGHT	vi
Declaration	vi
Acknowledgements	vii
1 Introduction	1
2 Project Plan	1
2.1 Required Resources	2
3 Ethical considerations	3
4 Literature Survey	4
4.1 Introduction to the problem area	4
4.2 What does it mean to be ‘easy to use?’	5
4.3 Running User Studies	6
5 Experimental methodology	8
5.1 Experiment process	9
5.2 Pilots	9
5.3 Actions for next meeting	10
6 Pilot Study 1	10
6.1 Hypotheses	11
6.1.1 Method	11
6.1.2 Using Thematic Analysis in Studies	11
6.2 Results	13
6.3 Observations	13

6.3.1	Observation 1	13
6.3.2	Model Adjustment 1	13
6.3.3	Observation 2	14
6.3.4	Model Adjustment 2	14
6.4	Discussion	14
7	Requirements	15
7.1	High-level goals	15
7.2	Functional Requirements	16
7.3	Non-Functional Requirements	17
8	Design	18
9	Implementation	18
9.1	Finding my feet	18
9.2	Issues encountered	19
9.3	Designing a task in JavaScript and Elm	22
10	Pilot Study 2	22
10.1	Observations	22
10.1.1	Hypotheses	23
10.2	Experiment	23
10.2.1	Method	23
10.3	Results	25
10.3.1	Analysis	27
10.3.2	T-statistic result between groups Elm and JS	27
10.4	Interpretation	29
10.5	Discussion	29

11 Further work	31
12 Conclusions	31
12.1 TIE THIS BACK TO THE ELM CLAIMS MADE IN LITER- ATURE SURVEY!	32
Bibliography	32
13 Appendices	38
Appendix A	38
13.1 Design Diagrams	38
Appendix B	38
13.2 Raw results output	38
13.2.1 firebase-mouseclick-data.json	38
13.2.2 error_log.json	39
Appendix C	40
13.3 Code	40
13.3.1 LICENSE	40
13.3.2 install_elm.sh	41
13.3.3 Server.hs	42
13.3.4 Editor.hs	50
13.3.5 Generate.hs	55
13.3.6 EmbedMe.elm	58
13.3.7 editor.js.diff	60
13.3.8 MovingBox.js	61
13.3.9 MovingBox.elm	63
13.3.10 ClicksPerCategory.py	64

13.3.11 get_clicks_per_category.py	66
13.3.12 DecodeMouseData.py	66
13.3.13 test_DecodeMouseData.py	68
13.3.14 ttest_scipy.py	70
Appendix D	71
13.4 Pilot Study 1	71
13.4.1 Consent Form	71
13.4.2 Pre-questionnaire	72
13.4.3 Post-Questionnaire	74
13.4.4 Participant 1	75
13.5 Pilot Study 2	80
13.5.1 SPSS Multiple Regression	80
13.5.2 Pre-questionnaire	84

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:

Extending an IDE to support input device logging of programmers during the activity of user–interface programming: Analysing cognitive load

Submitted by: Simon Buist

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath’s policy on intellectual property (see <http://www.bath.ac.uk/ordinances/22.pdf>). This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

Acknowledgements

In no particular order, I would like to thank my dissertation supervisor, Dr. Leon Watts, for guiding me along the way — suggesting relevant literature, helping me stay on track and for lifting my spirits. I thank my parents Gail and Joseph for being a constant inspiration in my life, in what it means to be determined, and for believing in me. I am grateful to Remco Meeuwissen for his experience and input in writing the JavaScript task. I thank the lecturers and staff at The University of Bath for their help and time in providing me with the foundation and support that made this dissertation possible. My regards go to all those that kindly gave of their time to participate in the pilot studies and user studies.

1 Introduction

Talk about project in general. . .

Determining the cognitive load induced during programming, and by extension, determining the competency of a programmer, is not an easy endeavour –

2 Project Plan

I will now explain my current plan for the project. Notice that I say current here – this may change throughout the course of the project: I may narrow in on a topic of interest, or branch out to investigate anomalous research findings.

I will be building the end product – the dissertation and software – via a process of iterations, much like an iterative Software Lifecycle. The Literature Survey is ongoing – throughout the whole project from beginning to end – feeding into all parts of the dissertation, and indeed this Proposal, as shown in the Gantt chart (Figure 1). The literature I choose is sometimes chosen to support points I wish to make, sometimes acting to guide my next area of research, reinforce findings, compare or contrast with other research, and probably many other things I have not yet thought of. Most importantly, I will be looking at who the paper/article etc. is cited by, preferring sources that are peer-reviewed.

As well as this literature research, I will also have an ongoing Product Literature Survey – looking at existing software out there that is related to my current area of interest.

Central to this idea of iteration is my desired method of performing user studies: I will first do what I have called a “Pilot” – a short and shallow trial User Study that focuses not on the research I’m concerned with, but instead the particular experimental design I would like to use in my actual User Study. By employing a Pilot I can hopefully get an idea of the nature of the experimental design – perhaps discovering any variables I had not previously considered that will require me to increase my sample size or simplify the experiment in order to mitigate their effect on the dependent variable I wish to test for. These are all problems discovered in Yates (2012) – including

basic teething problems in getting the experiment to flow smoothly. In an even less detailed aspect, the pilot may allow me to look at what is out there. It may help to not look for anything in particular initially, and see what happens.

At this stage, with the help of discussion with my Project Supervisor, I have some ideas about how to gather data in User Studies and these pilots could prove to be a useful testbed for such tools. I have a hypothesis that the novice developer “thrashing” Lopez et al. (2012) can be observed by shorter pauses between editing and experimentation, and I could measure this by way of measuring the mouse position relative to the IDE, clicks, and key-presses, using tools built-in to Elm and a bit of extension to stream this over the Internet to my storage facilities Czaplicki (2013b).

As you will see in the Gantt chart (Figure 1) I have included Testing & Implementation under the same heading as I will be doing Test Driven Development. My experience on Placement at PicoChip, my job as a Software Engineer at Altran and readings have helped me realise that this way of developing is time-saving and improves code quality by enforcing modularity in order to test it Martin (2008) and Hunt & Thomas (1999).

2.1 Required Resources

I will now talk about the resources I require for the completion of this dissertation, including the availability of these resources.

I will require users for my user study. These users must be proficient in at least one programming language (declarative programming languages are niche in and of themselves, never mind the discipline of programming, so some basic knowledge is required in order to see useful patterns in User Interface Design). Suitable candidates are First and Second Year Computer Science students from most Universities in the UK. Their availability is limited – Christmas holidays and coursework deadlines may mean that certain periods of the term are particularly busy for them. At Bath, suitable periods are therefore November, January to Mid February (inclusive), Mid-March to April (inclusive). It will be useful to procure free periods for other nearby Universities to hedge my bets, and to have a decent random assignment of users so I can make equivalent groups in my experiments.

The ACM Digital library, accessible via the Bath portal either from University or from home via Single-sign-on is a valuable resource for research papers, articles and references. The Cited-By feature will allow me to assert the popularity/ranking of each resource. Another valuable resource is the Psychology of Programming Interest Group, a “[group of] people from diverse communities to explore common interests in the psychological aspects of programming and in the computational aspects of psychology”, with peer reviewed papers on particularly relevant topics to my area of research.

I will require regular access to the Internet, Emacs with haskell-mode installed and Elm version 0.10 Czaplicki (2013a). I will also need git for software source control, and bitbucket.org for online, private backups of my work. I require LaTeX to type up my dissertation, and have chosen texlive on Ubuntu 12.04.3 as my development environment of choice. The full development environment is installed at the house I am staying in, in Bath, on my laptop. I am also able to replicate this environment to a satisfactory level at Bath University on any computer with access via Putty/SSH or similar to LCPU, as all the above software can be installed and run on my Bath University account.

I am using Chromium Version 28.0.1500.71 Ubuntu 12.04 (28.0.1500.71-0ubuntu1.12.04.1) to run the Elm IDE, which is an important dependency that may cause problems in getting Users in User Studies to run a functionally equivalent browser. Only recent editions of Chrome, Chromium, Firefox, Opera and Safari (not Internet Explorer) support Elm web programs.

3 Ethical considerations

In conducting User Studies, I will be interacting with people and collecting data from them, so I must be considerate and mindful of those I talk to and the information I handle.

An Ethical Checklist such as the one Bath University uses as it’s template Bath (2013) may assist my research such that I treat each participant with care and respect. I may learn from the discoveries made by others – in my reading, I came across a paper (also mentioned earlier) that highlighted concerns that participants under study had, and the paper detailed ways to mitigate these concerns so as to make the participant feel that are informed and safe Yates (2012).

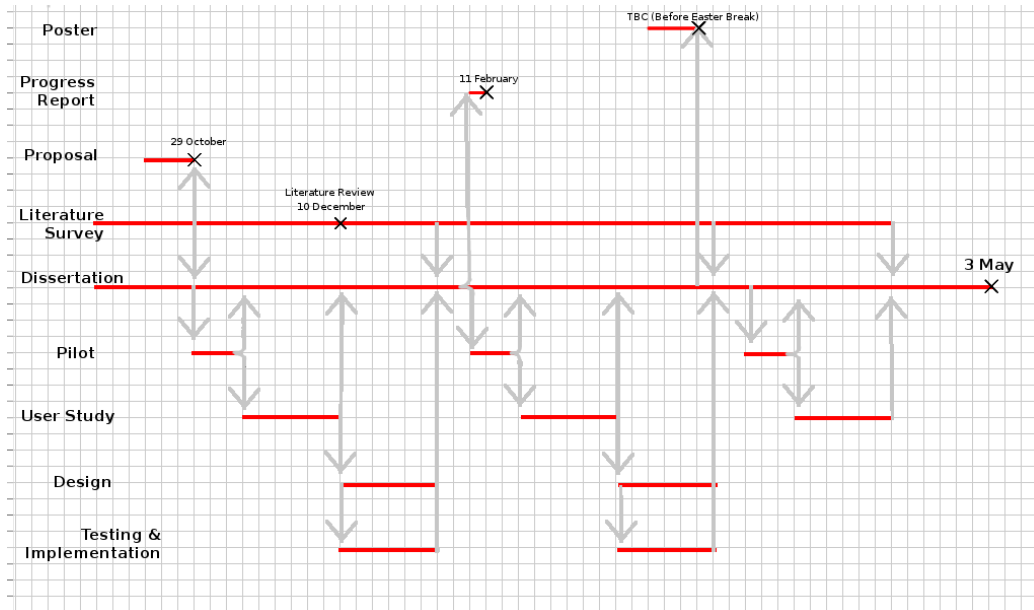


Figure 1: Gantt Chart

4 Literature Survey

4.1 Introduction to the problem area

The problem area of user-interface programming, and more generally, the activity of programming in a context such as a software engineering environment, encompasses certain realms of interest. Through my survey of literature, my research has touched upon the above-mentioned terms, and I have discovered some thought-provoking problems that exist in the field of programming. The concept of ‘Programming’ embodies other concepts – art-forms, engineering processes, science, language, and mathematics, among others. To me, programming is a creative endeavour unlike any other – in which the programmer wields materials of no substance – the code – by manipulating symbols on a screen, which represent states in the machine being used. There are so many programming languages, and all languages (all that are Turing-complete) reduce to the same language – that of a Turing Machine. So, *why do we have so many programming languages?*

Beware of the Turing tar-pit in which everything is possible but nothing of

interest is easy. (Perlis, 1982)

Different languages lend themselves to different ways of thinking about problems. They may place emphasis on one feature, for example list manipulation and hide others such as types. The language or programming environment may make explicit the effect of changes as they are encoded, as opposed to queuing up a block of changes and the programmer having to initiate an update manually.

I would like to draw your attention in particular to the terms **Abstraction**, **Cognitive offloading**, **Feedback**, **Loss of information?**/**Augmented reality?**, **Thrashing**, and **“Programming blind”**. These, at current, are my topics of interest, and my literature review has up to this point been inextricably and heavily influenced by this.

4.2 What does it mean to be ‘easy to use?’

In the process of surveying relevant (and sometimes irrelevant) literature to this dissertation, recurring conceptual patterns were observed – one particular instance of this is that several authors seem to lay victim to the trap of claiming their creation is “easy to use”, “better”, “simpler than x ” without providing any supportive evidence of this.

Perhaps these are incidents of ‘experimenter bias’ – where the evaluator is naturally predisposed to a positive appraisal of their own findings. One way to avoid this is to have one set of people perform the data capture and another set perform the data analysis. Nevertheless, these patterns emerge, and present numerous opportunities for experimentation and subsequent evidence supporting or contradicting these claims. Experiments may see if the same conclusions are reached as the above-mentioned authors, accounting for the ‘evaluator effect’ (Hertzum & Jacobsen, 2001).

Whether this particular route is taken for experimentation hinges on pilot studies that will be conducted concurrently to the Literature Survey, each inextricably shaping the other’s direction of investigation and inquiry.

The catalyst to this whole dissertation was a talk about the concept of a highly reactive development environment – where changes in the code result in instantaneous updates to the runtime, ‘on-the-fly’. This was presented in Bret Victor’s “Inventing on Principle” (Victor, 2012). In his presentation Bret

makes several assertions about the ‘traditional’ style of coding, one statement of which is that “most of the developer’s time is spent looking at the code, blindly without an immediate connection to the thing they’re making”. He argues that “so much of creation is discovery, and you can’t discover anything if you can’t see what you’re doing” – alluding to his earlier statement that the compile-run-debug cycle is much like this.

Evan Czaplicki, in his thesis of which Elm is the product (Czaplicki, 2012), makes similar claims – “[Elm] makes it *quick and easy* to create and combine text, images, and video into rich multimedia displays.” While the evaluation of Elm’s usability is not the focus of the thesis, rather, it is to establish a context for Functional Reactive Programming and describe the implementation details, he makes other usability claims without evidence – “[non-declarative frameworks for graphical user interfaces] mire programmers in the many small, nonessential details of handling user input and modifying the display.”, “FRP makes GUI programming much more manageable”, and in a section entitled *The Benefits of Functional GUIs*, “In Elm, divisions between data code, display code, and user interaction code arise fairly naturally, helping programmers write robust GUI code”. If these claims are true, there is all the more evidence that Elm should be a language of choice for GUI programmers, but experiments must be done to determine this.

And perhaps this rapid development cycle is not always suitable – in their 2012 paper, Lopez et al. show that novices tend to “thrash” about, trying out many ideas that may or may not be a solution, and executing “poorly directed, ineffective problem solving . . . failing to realise they are doing it in good time, and fail to break out of it”, whereas experts think much more about the problem at hand before proceeding with a solution (Lopez et al., 2012).

4.3 Running User Studies

Perhaps a further direction of investigation may be running an experiment to spot whether or not Elm’s auto-updating IDE lends to a lack of critical thinking – some operationalization may be *pauses reported as ‘thinking’ made during development* – where a pause is disambiguated as ‘thinking’ by the experimenter asking the participant why they did not perform any interaction with the computer for more than 10 seconds, and the participant reports

that they were planning/designing/other similar activity. Along this line of thinking, a paper studying the relationship between speech pauses and cognitive load (Khawaja et al., 2008) found through studying 48 mixed gender participants that there is statistically significant indicators of cognitive load through analysing pauses in speech. Perhaps this concept of pauses can be applied to the activity of programming. However, the planned method of disambiguating pauses via self-reporting (previously mentioned) would not be suitable according to these authors – “such measures can be either physically or psychologically intrusive and disrupt the normal flow of the interaction”, although a paper cited by (Khawaja et al., 2008) itself claims that “although self-ratings may appear questionable, it has been demonstrated that people are quite capable of giving a numerical indication of their perceived mental burden (Gopher & Braune, 1984)”. Indeed a pilot study by Fraser and Kölling (McKay & Kölling, 2012) structures the self-reporting by getting the users to evaluate an IDE as they use it using a set of subject-specific heuristics that they have designed. They showed that this customised set of heuristics helped guide the user more effectively than Nielsen’s heuristics in evaluating usability, so one could develop a custom set of heuristics for evaluating the usability of Elm.

From the Elm thesis (Czaplicki, 2012), the language syntax and rapid feedback seem simple enough that it is conceivable (or at the very least, possible and of experimental interest) to allow the user to customise the UI layout to their liking. Letting the user shape the UI in concert with a UI programmer is covered the study of the interface development environment “Mobi-D” in military and medical applications (Puerta, 1997), with success in those fields. It may be worth speculating how Elm would fit into the development cycle that Puerta’s paper outlines, as this may lend inspiration to potential user interface enhancements to the Elm IDE for A/B testing. It must be noted that there does not seem to be a re-emergence of Mobi-D since the paper was written, however.

My goal is to answer these questions. By way of conducting user studies, leveraging Elm with extensions to do A/B testing to illustrate it’s effectiveness (or ineffectiveness) at enhancing User Interface Design.

Central to this idea of iteration is my desired method of performing user studies: I will first do what I have called a “Pilot” – a short and shallow trial User Study that focuses not on the research I’m concerned with, but instead

the particular experimental design I would like to use in my actual User Study. By employing a Pilot I can hopefully get an idea of the nature of the experimental design – perhaps discovering any variables I had not previously considered that will require me to increase my sample size or simplify the experiment in order to mitigate their effect on the dependent variable I wish to test for. These are all problems discovered in (Yates, 2012) – including basic teething problems in getting the experiment to flow smoothly. In an even less detailed aspect, the pilot may allow me to look at what is out there. It may help to not look for anything in particular initially, and see what happens.

At this stage, with the help of discussion with my Project Supervisor, I have some ideas about how to gather data in User Studies and these pilots could prove to be a useful testbed for such tools. I have a hypothesis that the novice developer “thrashing” (Lopez et al., 2012) can be observed by shorter pauses between editing and experimentation, and I could measure this by way of measuring the mouse position relative to the IDE, clicks, and key-presses, using tools built-in to Elm and a bit of extension to stream this over the Internet to my storage facilities (Czaplicki, 2013b).

5 Experimental methodology

Here are some possible approaches I could take to analysing the paradigm of declarative versus imperative programming.

1. AB Testing of the languages with the same IDE?

The primary direction I mentioned (as echoed in my Proposal) was doing AB testing of Elm vs. another language (e.g. JavaScript) (i.e. the language is the dependent variable) using the same Concurrent FRP IDE (the independent variable).

2. Test just the paradigm?

Test just the paradigm, eliminating the IDE from the experiment above. Perhaps for a Pilot study.

5.1 Experiment process

1. Study question (e.g. Is it easy?)
2. Measurement concept (e.g. “Easy”)
3. Operationalisation – taking a measurement concept and mapping it to something concrete (e.g. if completing a pre-defined task the user must complete takes < 5 steps, it is ‘easy’ – we can then compare instances of these studies given our definition of easy). This is much like mapping a design to an implementation, and there is a risk of losing information, or ending up with a mismatched concrete instance that does not represent the concept we wish to convey.
4. Do another operationalisation of our measurement concept – this allows us to get a different perspective of the same concept. (e.g. if total length of pauses during a 1 hour experiment is < 10 minutes, it is ‘easy’). We do this to get ‘coverage’ of the measurement concept. It is a form of cross validation. If we see an overlap in the correlational results after analysis, we can make a stronger assertion that e.g. “language A is easier than language B.”. The idea I am describing here is methodological decision-making.
5. Predict what will be the likely results of our experiments on the operationalised measurements. This is “feed forward validation”.
6. Do the experiment.
7. Analyse the data. See if the data has patterns that correlate with the assertion I wish to make. I will be representing the raw data in some outcome measure – that is turning the raw data into a set of (or a single) value for comparison.
8. Does the data answer the study question I set out to ask? This is now “feed backwards validation”.
9. Write-up including the ‘nitty-gritty’ of the user study, and a statement like “Given our definition of easy, our multiple operationalisations of the concept of easy show that this is in fact objectively true/false”.

5.2 Pilots

What might be surprising insights into declarative programming languages for User Interface Design – the case of Elm?

Speak-aloud protocols where you prompt/facilitate the user to say what is on their mind when that e.g. pause for more than 10 seconds – a measurement I set out to look for during an experiment.

I might ask

- I notice you have paused for at least 10 seconds – why did you?
- I thought the code would do X, but it did Y.
- Why did you think it would do X?
- ...

I must ask the participant questions designed in a way that they are not leading.

Leon suggested I gather a rich data set, as it's difficult to take notes AND prompt the user during an experiment. SO difficult. Perhaps record video.

5.3 Actions for next meeting

Devise a Pilot study, answering these 3 questions:

1. What might I ask people to do?
2. How will I gather data?
3. How will I analyse the data?

Also see paper Leon will send me on “Thematic analysis & Psychology”

6 Pilot Study 1

Using a per-participant questionnaire (See 13.4.2), I captured video & audio data of 2 participants while they completed the task of extending a mario game to make mario fly. This initial pilot study was done to get a feel of what behaviours may be worth investigating further while a user completes

a programming task. I may then refine the methodology to enable said behaviours to be isolated more effectively, varying some dependent variable to see if it has any effect. I have one hypothesis based on my understanding of thrashing from the literature review, (Lopez et al., 2012).

6.1 Hypotheses

- H_1 . Novice users (*defined as*: those that list themselves as being new to functional programming in the pre-questionnaire) will press compile at least once every 2 minutes during the programming task.
- H_2 . Novice users will not pause (*defined as*: no mouse movement, no typing) for more than 2 minutes during the programming task.

6.1.1 Method

1. Consent form is signed (See 13.4.1)
2. Pre-questionnaire is given out.
3. User is informed that they may ask for help, and that they will be prompted if they pause (at the moment they break the pause), to ask why they paused. They are also informed that they can end at any time, and the goal is to make mario fly.
4. User is shown the result of completed task (mario flying).
5. Programming task is begun.
6. User completes or ends the task.
7. Post-questionnaire is given out (See 13.4.3)
8. De-briefing.
9. Study ends.

Using Thematic analysis (Braun & Clarke, 2006) to code the captured audio and video data, I will transcribe the programming activity, to see what themes arise.

6.1.2 Using Thematic Analysis in Studies

In doing Thematic Analysis [TA], as a researcher, one must make **explicit** the particular variant of TA that you intend to carry out, and the whole analysis

must be framed from that point on by that experimental methodology.

One must make the statement along the lines of: “Amongst the number of different branches that the paper talks about, X with Y themes and Z focus is the flavour of thematic analysis that I am going to use” – e.g. essentialism NOT constructionism, one aspect NOT whole, theoretical NOT inductive...

As this is a Pilot Study I will simply be using the 6-phase analysis to gather themes in the data (if there are any), as shown in Table 1.

Phase	Description of the process
1. Familiarising yourself with your data:	Transcribing data (if necessary), reading and re-reading the data, noting down initial ideas
2. Generating initial codes:	Coding interesting features of the data in a systematic fashion across the entire data set, collating data relevant to each code.
3. Searching for themes:	Collating codes into potential themes, gathering all data relevant to each potential theme.
4. Reviewing themes:	Checking in the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), generating a thematic ‘map’ of the analysis.
5. Defining and naming themes:	Ongoing analysis to refine the specifics of each theme, and the overall story the analysis tells; generating clear definitions and names for each theme.
6. Producing the report:	The final opportunity for analysis. Selection of vivid, compelling extract examples, final analysis of selected extracts, relating back of the analysis to the research question and literature, producing a scholarly report of the analysis.

Phase	Description of the process
-------	----------------------------

Table 1: Phases of Thematic Analysis, from (Braun & Clarke, 2006)

6.2 Results

The participant that listed themselves as being Experienced in Functional Programming did in fact exhibit more thrash than the one that listed themselves as being a Novice, contrary to H_1 and H_2 . They were observed clicking compile once every 1m3s on average, to the Novice’s 2m3s, and pausing for an average of 3m24s to the Novice’s 1m45s. I **reject** hypotheses H_1 and H_2 . It is important to see that this is an extremely small sample size, not nearly enough to achieve saturation, and so not much meaning can be derived from this result, but it is interesting.

6.3 Observations

6.3.1 Observation 1

- Prompting “*What are you thinking about?*” etc. seemed to place additional cognitive load on the user as they spent longer resuming than when not prompted. This caused noise in assessing the actual cognitive load incurred during the completion of the **task**. Were the signs of struggling/undergoing difficulty due to simply not understanding the language, or were they due to the difficulty of the task?
- In particular, the majority of instances where the users paused turned out to be confusion as to the semantics & syntax of the language.

6.3.2 Model Adjustment 1

- Add tooltips that appear as the user places the keyboard cursor to the right of a token in the language.

6.3.3 Observation 2

- More of a meta-observation about the methodology than about the experiment findings itself: Sifting through 1-hour+ of video data capture for incidences of cognitive load is tedious and not particularly fruitful or objective. Is there some programmatic way of narrowing the video data to points of interest?

6.3.4 Model Adjustment 2

- Extend the IDE to allow for tracking the user mouse and keyboard movements in a 3-tuple: (`Time t`, (`Mouse.x`, `Mouse.y`), `Keypress k`)
 - It doesn't have to be implemented this way. I could extend **Model Adjustment 1** to define blocks of code as tokens in themselves, and capture how long the cursor is static on that particular token.
 - A further refinement of this idea is to filter the data (in fact, just capturing mouse & keyboard movements will result in an explosion of the volume of data – contrary to what I intend to achieve): define regions of interest in the code pane, and *only when the mouse/key cursor is in the region, do I capture data*.
 - Use the `if cursor in region then log (Time t, (Mouse.x, Mouse.y), Keypress k)` functionality as a *lens* to focus on significant portions of video capture.

6.4 Discussion

Following the Pilot Study, I drafted some questions and thoughts that might lead my direction of study in the next steps of my research:

- How can I capture a more objective measure of a user's interaction with the IDE?
- What behaviours are indicative of a user experiencing cognitive load?

- Is the mouse/cursor position a proxy for someone’s attention as they carry out the task?
- Often when I’m coding I’ll leave the cursor where it is but think about other regions of code. I don’t necessarily move the keyboard/mouse cursor to the section of code I’m thinking about. Instead, I use it as a ‘bookmark’ to track what I’m currently implementing, and may scroll around to other parts.

At this point my goal of this dissertation is to obtain a list of observed cognitive easing/loading that each language produces for users, much like an advantage/disadvantage comparison:

Elm	JavaScript
+ ...	+ ...
+ ...	- ...
- ...	- ...
- ...	+ ...
+ ...	—

I will now discuss Requirements that need to be met in order to realise this goal.

7 Requirements

7.1 High-level goals

1. Augment the Elm IDE to support the logging of input device activities during a programming task
2. Identify metrics that accurately model cognitive load
3. Design a task in JavaScript to go inside this adjusted model (incorporating Model Adjustment 1 and 2).

This will require a degree of “*implementation juggling*” in order to find a balance of code-length/difficulty over the same task in Elm in such a way that is not creating noise in the thing being studied: Cognitive load.

Keep the reactivity constant, compare the differences in ease between JS and Elm.

4. If time available, run another Pilot study on this task + adjusted model

I will now identify what the requirements are for the project in order to achieve these High-level goals.

7.2 Functional Requirements

1. Write software to assist the capture of objective data to inform me of the user’s activities as they use the Elm IDE.
 1. The program must be able to capture data on-the-fly collecting mouse and keyboard activity.
Priority: High
 2. The program must be able to support the loading of both arbitrary Elm and arbitrary Javascript tasks.
Priority: High
 3. The program should only capture data relevant to the study at hand – for example if we are interested in a rectangular region defined by co-ordinates (top left, bottom right): (x:10, y:45), (x:50, y:90), the program must only capture data within that region.
Priority: Medium
 4. The user of the program must be able to define regions with which to filter the data set.
Priority: High
 5. The program should capture syntax and semantic errors made when the user attempts to compile erroneous code.
Priority: Medium

2. Link experiment to each user
 1. The experiment must be able to support remote data capture — i.e. support sending a URL of the IDE and task to participants, capturing their interaction remotely.
Priority: High
 2. The experiment must link the questionnaire to the task and to the compile error output so that one can be certain of a 1–1 correspondence with each source of data to each respondent, for collating afterwards.
Priority: High
3. Perform Pilot and User Studies
 1. I must perform Pilot and User Studies in an iterative fashion, each one learning and building upon discoveries made in prior ones, starting vague and getting more and more focused on a particular facet of User Interface Design and/or Declarative programming as an activity.
Priority: High
 2. I must use these studies to inform experimental and software design to disambiguate and filter data collected in the experiment, and to exercise hypotheses.
Priority: High

7.3 Non-Functional Requirements

1. Source code
 1. The software must be written clearly and simply.
Priority: High
 2. The software must have suitable, concise comments which explain the program intent, but only where the code alone is not enough.
Priority: High
2. Activity recording

1. The program activity recording feature must not slow down the user's use of the IDE more than 1ms difference than without it.
Priority: High
2. There must be software to visualise the usage data
Priority: High

8 Design

This is the chapter in which you review your design decisions at various levels and critique the design process.

9 Implementation

In this section I will describe how I implemented the desired requirements for the IDE, detailing the decisions made and issues encountered.

9.1 Finding my feet

Initially, the implementation involved a lot of hacking around in order to try and figure out how each component of the existing IDE fit together, trying to determine how to get constantly updating user input signals to stream via a `port` from Elm to JavaScript, so that I could store `Mouse.position` and `Keyboard.keysDown` in a DB backend for Elm that for embedding into the code pane. I found an example by Dénes Harmath that was close to what I wanted to achieve and forked the repository. It lives at <http://github.com/spanners/elm-lib>

It is nice and lightweight, and I applied these ideas to mouse input. See the palindrome Elm example at <http://elm-lang.org/edit/examples/Intermediate/TextReverse.elm> for relevant input signals as the inspiration for click logging that I used.

However, there are two **disadvantages with this approach**:

1. elm-lib uses an old Elm Foreign Function Interface [FFI] – instead of using `ports` it uses `foreign import jsevent`.

I looked at how Evan Czaplicki does this in his `elm-js-and-html` example, forked his repository at <http://github.com/spanners/elm-js-and-html>, and managed to see how he ported it from 0.10 to 0.12 which gave me some idea about how to do the same for `elm-lib`.

2. The second disadvantage is that it uses Firebase (<http://firebase.io>) as a backend for data storage.

This is a proprietary application. I would much rather use something Open Source that I can host myself for two reasons: Firstly for data protection reasons. This shouldn't be a problem due to the fact that it is mouse (x,y) co-ordinates and timestamps. Secondly because it has a hard-limit on the amount of data you can store on the FREE plan: 100MB. Again, this shouldn't be a problem so long as I filter the input intelligently.

9.2 Issues encountered

My initial implementation of click logging used `Elm.embed` and offset a `<div>` tag as the place to load the arbitrary Elm code. This meant that all co-ordinates were offset by the height of the code `<div>`, and I would have to compute the offset after gathering the data. I tried to use `Elm.worker(Main.Elm, div, {})` and make the `<div>` element encompass the entire CodeMirror window, but the `Elm.worker` function turns out to be only for computational code, not for interacting with the outside world (Input/Output).

I managed to get Evan's stamps example (<http://github.com/evancz/elm-html-and-js>) example working with Firebase (on my own repository, here: <http://github.com/spanners/elm-js-and-html>). Now I can successfully store user mouse events persistently in a JSON file.

This required dealing with **Disadvantage 1**. which I did successfully, using the new `port FFI`.

I convert Elm Records into JSON Strings to be stored in Firebase like so:

```
1 firebaseRequest requestType requestData =
2   Http.request requestType
3       "https://username.firebaseio-com/dissertation.json"
4       requestData
```

```

5           []
6
7   serialize r = r |> JEXP.fromRecord
8                 |> Json.fromJSObject
9                 |> Json.toJSString " "
10                |> JS.toString
11
12   toRequestData (x,y) = {x = x, y = y} |> serialize
13
14   toRequest event = case event of
15     (x,y) -> firebaseRequest "post" (event |> toRequestData)
16
17   requests = clicks ~> toRequest
18
19   sendRequests = Http.send requests

```

With the Elm-runtime 0.12 port I was working on, I got the bug `elm: bad ADT got to port generation code` and had no luck finding why this was occurring, so posted on the *elm-discuss* mailing list asking for help (https://groups.google.com/forum/#!searchin/elm-discuss/ADT/elm-discuss/aIUk_MiW3yo/FZ0oSx-a1wYJ) on March 30th 2014.

For a while I only knew how to get visualisation working in Elm-runtime-0.10, which meant that I lost a lot of the benefits of the latest version (currently 0.12). I had to use regular expressions to swap where I'm doing a POST to submit mouseclick data, with GET to visualise the mouse data, depending on whether I want to visualise or capture mouse data. In essence, I would enter a special "View participant mouse data" mode into the IDE e.g. specify a url path in addition to the experiment and the participant ID, and then it loaded elm-runtime-0.10.js instead and did GET instead of POST on the same data

2 days after I posted to elm-discuss, Evan Czaplicki himself, semi-ported my half-working version of elm-lib (<http://github.com/spanners/elm-lib>) `StampTogether/Main.elm` to Elm-runtime 0.12 which is very helpful – I can now modify this to suit my needs for the Firebase upload of click data

20 days after my post, Dénes Harmath (the original creator of elm-lib) published a fully-working port for Elm-runtime 0.12, but by this time I had found a workaround. I eventually used his version as it was much cleaner.

I fixed the quirk of having to click an offset from the code – it turns out that CodeMirror.js binds to `mouse.click` and it was stealing the click from Elm's `Mouse.click`. Using `Mouse.isDown` instead solves this. I also fixed the quirk of having to compute the offset – using `Elm.fullscreen` was the eventual solution to encompassing the whole editor `<div>` with click logging.

From that point on, much of the remaining modifications involved fitting JavaScript as a supported language, into the existing IDE `Editor.hs`, `Server.hs` and `Generate.hs` code (See 13.3.4, 13.3.3 and 13.3.5 respectively). It is, in my opinion, the least elegant part of the implementation. In order to toggle whether we are using the JavaScript interpreter or the Elm compiler, I pass around a `type Lang = Elm | Javascript`. An example from `Editor.js` (See 13.3.4) is given below:

```
1
2 ide :: Lang -> String -> String -> FilePath -> String -> Html
3 ide lang cols participant fileName code =
4   case lang of
5     Javascript -> buildIde ("JS Editor: ", "/_compile")
6     Elm -> buildIde ("Elm Editor: ", "/compile")
7   where buildIde (editStr, compileStr) =
8     ideBuilder lang
9       cols
10      participant
11      (editStr ++ FP.takeBaseName fileName)
12      fileName
13      (compileStr ++ "?input=" ++ urlEncode code)
```

I have a case block `case lang of Javascript -> ... Elm -> ...` which passes a different title and compile URL route to `buildIde` depending on the type of `lang`

If further languages need support, these case blocks (there are 2 in the codebase) would grow linearly, which is awkward and violates the software engineering principle of *Don't Repeat Yourself*.

Refinements in later iterations of the codebase included:

1. Allowing Elm code to be read from a file into ``Editor.hs``

2. Modifying the stamps example to be bare
3. Modifying the stamps example to define regions of code to be logged for input
4. Visualising the mouse data straight from Firebase in the Elm 0.12 version
5. Writing python scripts to interpret the captured mouse

Which were all relatively straightforward by then.

9.3 Designing a task in JavaScript and Elm

In order to minimise the effect of the length of Source Lines Of Code [SLOC] on task difficulty, and to incorporate the 2 by 2 regions — (Hard, Easy) x (Task-relevant, Task-irrelevant) — I had to be very careful about how the Elm and Javascript tasks were presented. Pixi.js (<http://www.pixijs.com/>) allows for a relatively similar (in API difficulty and function call SLOC), but still imperative paradigm for manipulating objects on a canvas, so this was the library of choice for the JS version of the task. The Elm version of the moving box task pre-existed on Elm-lang.org, and needed some minor adjustments to approach similarity with the Javascript version. This is the “*task juggling*” I predicted would be necessary in order to reach an acceptable equilibrium between the Elm and Javascript tasks.

10 Pilot Study 2

Following from the outcomes of Pilot Study 1, including the modifications made to the experimental model, and the feature-augmented Elm IDE, I would like to conduct another Pilot Study to test the features and determine whether it accurately models thrashing/cognitive load. This section describes the Observations made both from Pilot Study 1, Hypotheses I form due to these Observations, The experimental Method, Results, Analysis and Discussion.

10.1 Observations

Observations and participant feedback from Pilot Study 1 (See 13.4.4) suggest that the task I chose, and the way in which I carried out the experiment,

was too taxing to capture the cognitive load incurred by the language itself for a given task, due to the difficulty of the task itself creating noise, and the experimental methodology incurring cognitive load – my prompting and questioning causing pauses. I could improve this by simplifying the task, in a way that is ‘language agnostic’, i.e. that is not idiomatic of Elm or JavaScript (the two languages that I am comparing). Something like the following will never be that easy in JavaScript:

```
1 main = lift asText Mouse.position
```

10.1.1 Hypotheses

- H_0 . There will be a uniform distribution of the total amount of clicks per region (Null hypothesis), for an α of 0.05
- H_1 . There will not be a uniform distribution of the total amount of clicks per region (Alternative hypothesis), for an α of 0.05

10.2 Experiment

10.2.1 Method

Using the IDE I have augmented to gather click data, and a pre-questionnaire to determine level of expertise (See Section 13.5.2), I will two independent pilots of users completing the same task in either Elm or JavaScript.

The task (See code listings 13.3.9 and 13.3.8 for the Elm and JS versions, respectively) is designed in such a way as to approximate the property of being ‘language agnostic’ – the versions of the task are reasonably similar in length, have the same comments, and variables are named similarly. The task is to make the moving box clamp to the grey window’s edges when moved with arrow keys, preventing it from disappearing. It must be clamped in such a way that, upon attempting to move the box *out of the grey window*, it stops half-way. A youtube video (See <https://www.youtube.com/watch?v=cUgK42N7kt8>) is to be given to the participants so that they can see what the completed task looks like.

The experiment will be conducted remotely. I will send the participants a link to the pre-questionnaire and either the Elm or JS task, and the youtube video link. The pre-questionnaire data will be stored on Survey Monkey, the click data will be stored on Firebase in JSON via my EmbedMe.elm (See 13.3.6) mouse click tracking program, and I will log the time at which a click occurs and it's (x,y) coordinates under a participant number unique to each participant.

This will be a $2 \times 2 \times 2$ study, using geometrically defined regions (See Figures 6 and 7), also known as bounding boxes, in the code, monitoring the count of mouse clicks per region as an indicator of thrashing/cognitive load. Regions can either be easy/hard in complexity (exhibiting/not-exhibiting one of the above 'difficult' properties). Or code can be task-relevant or task-irrelevant, that is *the code does/does not need to be changed to achieve the completed task set for the user*:

Elm	
Easy/Relevant	Hard/Task
Easy/Irrelevant	Hard/Irrelevant
JavaScript	
Easy/Relevant	Hard/Relevant
Easy/Irrelevant	Hard/Irrelevant

Table 3: $2 \times 2 \times 2$ study between-subjects

I will look at total and/or mean time in each of these areas for comparison. The study will be **between-subjects** instead of within-subjects. That is, I will study *different users* for different languages. If a user has completed the task in Elm, I can not have them complete the task in JavaScript, and vice-versa.

I will necessarily make a compromise here:

Between-subjects:

- I lose the ability to keep programmer competence as constant, thus it is a confounding variable

- I gain the ability to ignore learned-experience in completing the task – the participant is different every time so will not have done this task before, thus this is not a confounding variable.

Within-subjects is the converse of the above methodological properties

On the resulting raw data, I will perform a multiple regression — on the 2 Languages (Elm, JavaScript) \times 2 region Difficulties (Hard, Simple) \times 2 region Relevances (Relevant, Not relevant) — to determine if the number of mouse clicks per region differs across regions.

10.3 Results

The raw click data (See 13.2.1), was processed with Python scripts (See 13.3.10, 13.3.11, 13.3.12) to produce the following tables:

Time (min)	Clicks
38.717	183
8.034	130
7.878	39
23.672	25
29.754	391
14.993	78
48.960	769
6.354	71
7.878	39
29.698	501
40.302	803
12.319	65
17.106	79
12.958	119

Time (min)	Clicks
------------	--------

Table 4: Session time and clicks per session for Elm task

Category	Expected %	Expected	Observed
relevant \times hard \times Elm	12.5 %	106.37	76
relevant \times hard \times JS	12.5 %	106.37	33
relevant \times easy \times Elm	12.5 %	106.37	487
relevant \times easy \times JS	12.5 %	106.37	12
irrelevant \times hard \times Elm	12.5 %	106.37	105
irrelevant \times hard \times JS	12.5 %	106.37	69
irrelevant \times easy \times Elm	12.5 %	106.37	66
irrelevant \times easy \times JS	12.5 %	106.37	3
TOTAL	100%	851	851

Table 5: $2 \times 2 \times 2$ comparison of clicks per category — Expected and Observed

Table 5, in the Expected column, shows a normal distribution of clicks per category – if our null hypothesis H_0 holds, the observed outcome be 5% either side of this.

See Figure 2 for the visualisation of participant 18 completing the JavaScript version of the task. The augmented IDE can be used to visualise the click data for any participant number. Uncomment *line 32* of *EmbedMe.elm* (See code listing 13.3.6) and reload the task editor with the desired participant ID as an input parameter p to the URL, e.g. to visualise participant 18 click data:

<http://0.0.0.0:8000/edit/task/MovingBox.elm?p=18>

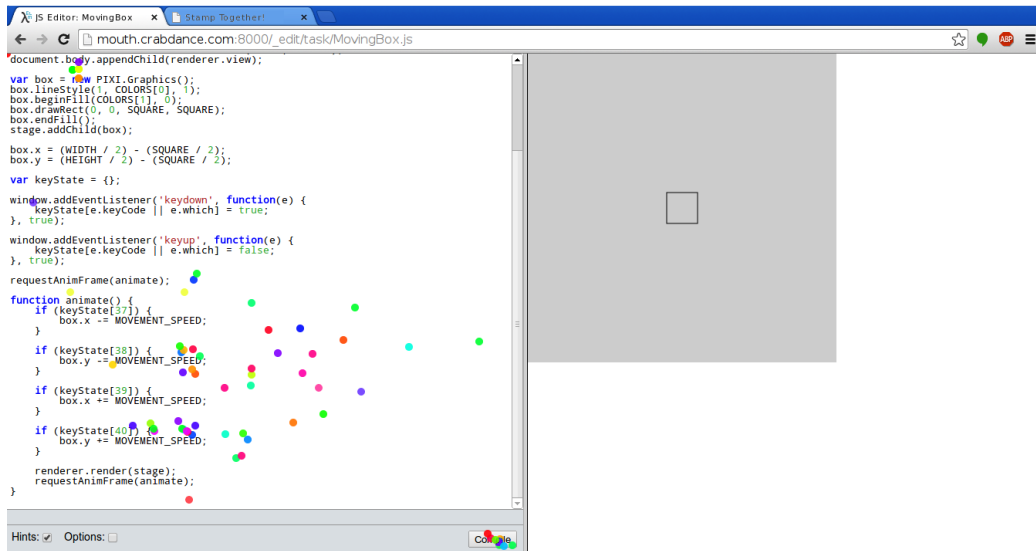


Figure 2: Participant 18, JS task (Overlaid with mouse clicks)

10.3.1 Analysis

I will now talk about how I analysed the raw data that was captured.

I performed the multiple regression on the categories defined using the statistics tool SPSS. See Section 13.5.1 in Appendix for SPSS multiple regression output. (**N.B** languages (Lan) and the relevance (Rel) and difficulty (Diff) are set to variables in the output: Elm := 1.00, JS := 2.00, and later in the *K-Way and Higher Order Effects* tables, Rel := 1, Diff := 2, Lan := 3). The following Chi-square (χ^2) results were obtained.

χ^2	1633.879
degrees of freedom	7
ρ -value	0
Yates' χ^2	1626.741
Yates' ρ -value	0

Table 6: χ^2 calculation of clicks per quadrant

10.3.2 T-statistic result between groups Elm and JS

Using SciPy's `stats.ttest` function (See 13.3.14), and assuming no variance between groups, and the following clicks per language in each category,

- Elm: `sum([76, 487, 105, 66]) = 734` clicks
- JS: `sum([33, 12, 69, 3]) = 117` clicks



Figure 3: Participant 15, Elm task (Illustrating the potential offset)

10.4 Interpretation

Looking at the Pearson’s lookup table of χ^2 value *vs.* ρ -value (probability) we can see that, for an α of 5%, that is a ρ -value of 0.05, and 7 degrees of freedom, a χ^2 -value of 12.59 is enough to show significance, and we obtained a χ^2 -value of 1633.955 (approx.). This means table 6 shows that there is statistical significance, and there less than 0.05 probability this significance arose out of chance (in fact, it is so small that SPSS rounded to 0). We can therefore **reject the null hypothesis** H_0 , and **accept the alternative hypothesis** H_1 , assuming a rigorous study – more on this in the Discussion section that follows.

10.5 Discussion

During the few (2 Elm tasks) Pilot Study 2 experiments that I did observe (not included in the results), I observed what could be interpreted as “phases” in a programmer activity during task-completion:

1. Familiarisation – Where is the bit I need to change?
2. Narrowing in on the task once discovered – Oh I need to change X, but how?
3. Solved task
4. Playing (?)

(Not necessarily distinct and in sequence — more often interleaved)

Since I conducted Pilot Study 2 remotely by sending the participant a link to the pre-questionnaire survey (See 13.5.2) and a link to either the Elm or JS task (See 13.3.9 and 13.3.8, respectively) I can not assert with any certainty that this was occurring during the experiments in Pilot Study 2.

The study, although found that there is a significant difference in the number of clicks in regions between languages (as an operationalisation of cognitive load experienced in completing the same task), unfortunately had a number of flaws which confound it’s rigor, and therefore no meaning can be derived from these results.

In summary, the flaws I have identified are as follows:

1. Much smaller respondent count for JS than Elm.
2. Small sample size – this forced me to use a T-test rather than a Z-test, meaning I can not be as sure of the results.
3. Self-reported expertise – (Hansson, 2001) found that “these judgements (self-reports) are well performed and accurate enough to be incorporated as a valuable tool”, but “the findings suggest that taking into account the individual’s perception of how important the specific competence is for performing a particular job (relative competence) might be a way to handle problems with the variation in the importance of different competencies.”, and there is no job-critical scenario here in this user participation study. We rely on the user themselves trustfully reporting their expertise in the languages (See pre-questionnaire 13.5.2) without the incentive of a salary or the risk of job loss.
4. Self-reported task completion – We rely on the user themselves reporting that they have completed the task according to the specification
5. No way to be sure which error log pertains to which compile – A shortcoming that was discovered *after* Pilot Study 2 is that the error log that captures when the user tries to compile syntactically/semantically correct code *does not log the participant ID the error pertains to*
6. No unique participant ID per SurveyMonkey – We can not be sure that, although the timestamps are similar, a person filling out the survey at time t may not be the same person starting the task at time t .
7. Window dimensions not captured – How can we be sure that a click in location (x, y) *on the user’s screen and resolution* is the same (x, y) that I capture in the database? Participant 15 (See Figure 3) very likely had a much shorter window height than I have used here. I suspect this is the case because of the cluster of mouse clicks in the same range of the x axis as the Compile button, but much further up in the y axis, but I have no way to be sure.
8. I did not capture window resizing – Same problem as above
9. Mouse scrolling not captured – Same problem as above. Although the task code was purposefully designed to fit in as small a screen space as possible, if the user has a smaller screen than the text’s dimensions, they may scroll, therefore offsetting the captured clicks.
10. Syntax reference 404 links – It was discovered *after* Pilot Study 2 that the Syntax reference links returned **Server Error 404**, due to me accidentally failing to include the documentation in the site where the

task was hosted. Captured click data suggests that people did attempt to follow the Syntax reference links, and the server access error logs support this indication with multiple 404s to the Syntax Reference.

11 Further work

In addition to fixing these flaws/foibles, an improvement over this experimental method is to take people who are new – as in: never having used Elm or JS, and train them up either in JS or Elm, and then run the same task. That way, their level of ability is much more comparable.

My current method creates quite a bit of noise in the data, because I rely on self-reported level of expertise in JS/Functional languages. I don't know how to modify the data to account for this. I could group the analyses into categories – i.e those who reported being experts at JS, those who reported never having used it, those who reported being experts in at least one FP language, and those who reported being new, and make cross comparisons with groups of equal levels of ability.

Furthermore, can I be sure that the operationalisation of thrash (the concept) I have chosen, i.e. cementing the concept by a metric to model cognitive load, is a positive indicator?

12 Conclusions

This is the chapter in which you review the major achievements in the light of your original objectives, critique the process, critique your own learning and identify possible future work.

12.1 TIE THIS BACK TO THE ELM CLAIMS MADE IN LITERATURE SURVEY!

Bibliography

- Bath, U. (2013) ‘Research ethics framework checklist’, [online] Available from: <http://www.bath.ac.uk/research/pdf/ethics/EIRA1ethicsform.doc>.
- Braun, V. and Clarke, V. (2006) ‘Using thematic analysis in psychology’, *Qualitative Research in Psychology*, 3(2), pp. 77–101, [online] Available from: <http://www.tandfonline.com/doi/abs/10.1191/1478088706qp063oa>.
- Czaplicki, E. (2013a) ‘Elm 0.10’, [online] Available from: <http://elm-lang.org/blog/announce/0.10.elm>.
- Czaplicki, E. (2012) ‘Elm: Concurrent FRP for Functional GUIs’,
- Czaplicki, E. (2013b) ‘What is functional reactive programming?’, [online] Available from: <http://elm-lang.org/learn/What-is-FRP.elm> (Accessed 1 October 2013).
- Gopher, D. and Braune, R. (1984) ‘On the psychophysics of workload: Why bother with subjective measures?’, *Human Factors: The Journal of the Human Factors and Ergonomics Society*, SAGE Publications, 26(5), pp. 519–532.
- Hansson, B. (2001) ‘Competency models: are self-perceptions accurate enough?’, *Journal of European Industrial Training*, MCB UP Ltd, 25(9), pp. 428–441.
- Hertzum, M. and Jacobsen, N. E. (2001) ‘The evaluator effect: A chilling fact about usability evaluation methods’, *International Journal of Human-Computer Interaction*, Taylor & Francis, 13(4), pp. 421–443.
- Hunt, A. and Thomas, D. (1999) *The pragmatic programmer: from journeyman to master*, Boston, MA, USA, Addison-Wesley Longman Publishing Co., Inc.
- Khawaja, M. A., Ruiz, N. and Chen, F. (2008) ‘Think before you talk: An empirical study of relationship between speech pauses and cognitive load’, In *Proceedings of the 20th australasian conference on computer-human interaction: Designing for habitus and habitat*, OZCHI ’08, New York, NY,

USA, ACM, pp. 335–338, [online] Available from: <http://doi.acm.org/10.1145/1517744.1517814>.

Lopez, T., Petre, M. and Nuseibeh, B. (2012) ‘Thrashing, tolerating and compromising in software development’, In Jing, Y. (ed.), *Psychology of Programming Interest Group Annual Conference (PPIG-2012)*, London Metropolitan University, UK, London Metropolitan University, pp. 70–81.

Martin, R. C. (2008) *Clean code: A handbook of agile software craftsmanship*, 1st ed. Upper Saddle River, NJ, USA, Prentice Hall PTR.

McKay, F. and Kölling, M. (2012) ‘Evaluation of subject-specific heuristics for initial learning environments: A pilot study’, In *Proceedings of the 24th Psychology of Programming Interest Group Annual Conference 2012*, London Metropolitan University, pp. 128–138.

Perlis, A. J. (1982) ‘Epigrams on programming’, *SIGPLAN Notices*, 17(9), pp. 7–13.

Puerta, A. R. (1997) ‘A Model-Based Interface Development Environment’, *IEEE Softw.*, Los Alamitos, CA, USA, IEEE Computer Society Press, 14(4), pp. 40–47, [online] Available from: <http://dx.doi.org/10.1109/52.595902>.

Victor, B. (2012) ‘Inventing on principle’, In *Proceedings of the canadian university software engineering conference (CUSEC)*, [online] Available from: <http://vimeo.com/36579366> (Accessed 15 March 2014).

Yates, R. (2012) ‘Conducting field studies in software engineering: An experience report’, In Jing, Y. (ed.), *Psychology of Programming Interest Group Annual Conference (PPIG-2012)*, London Metropolitan University, UK, London Metropolitan University, pp. 82–85.

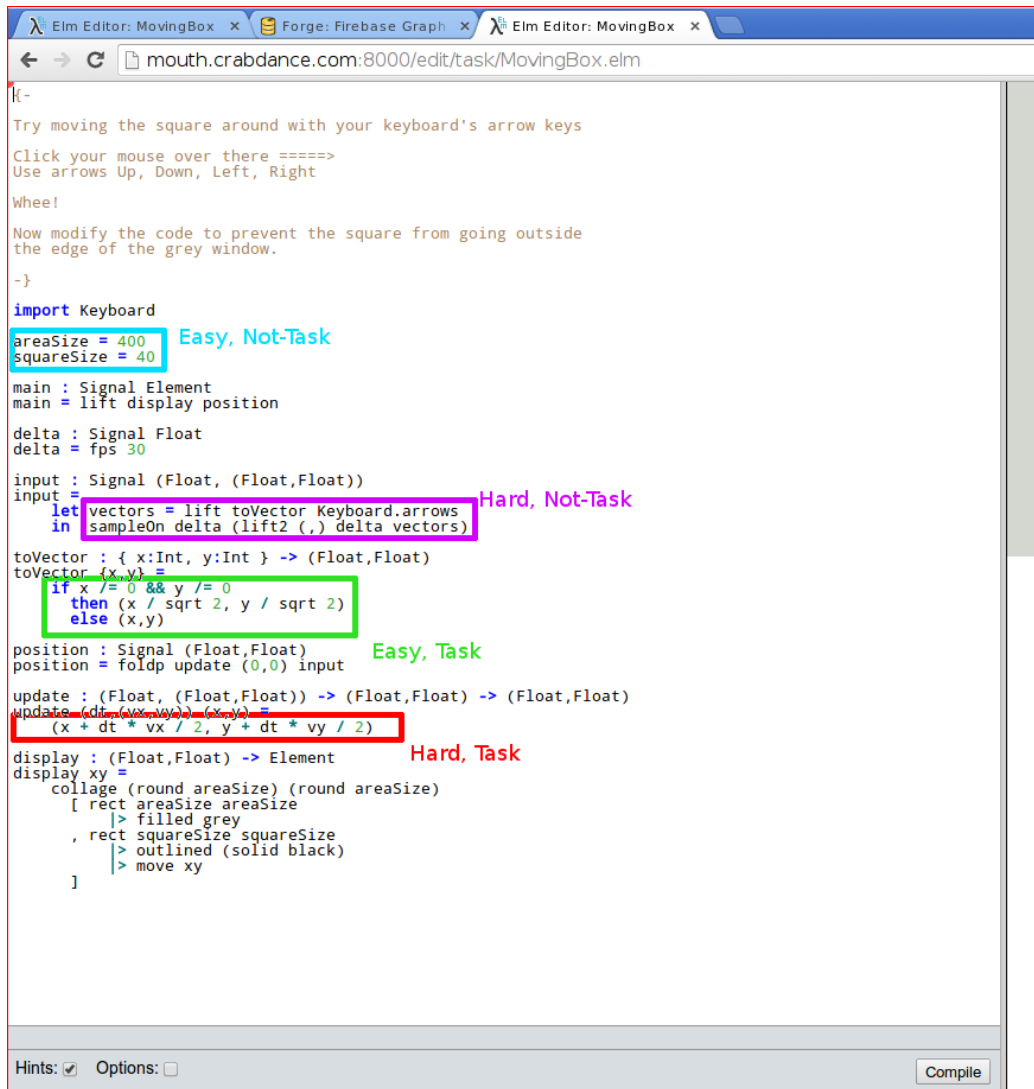


Figure 4: Elm click regions

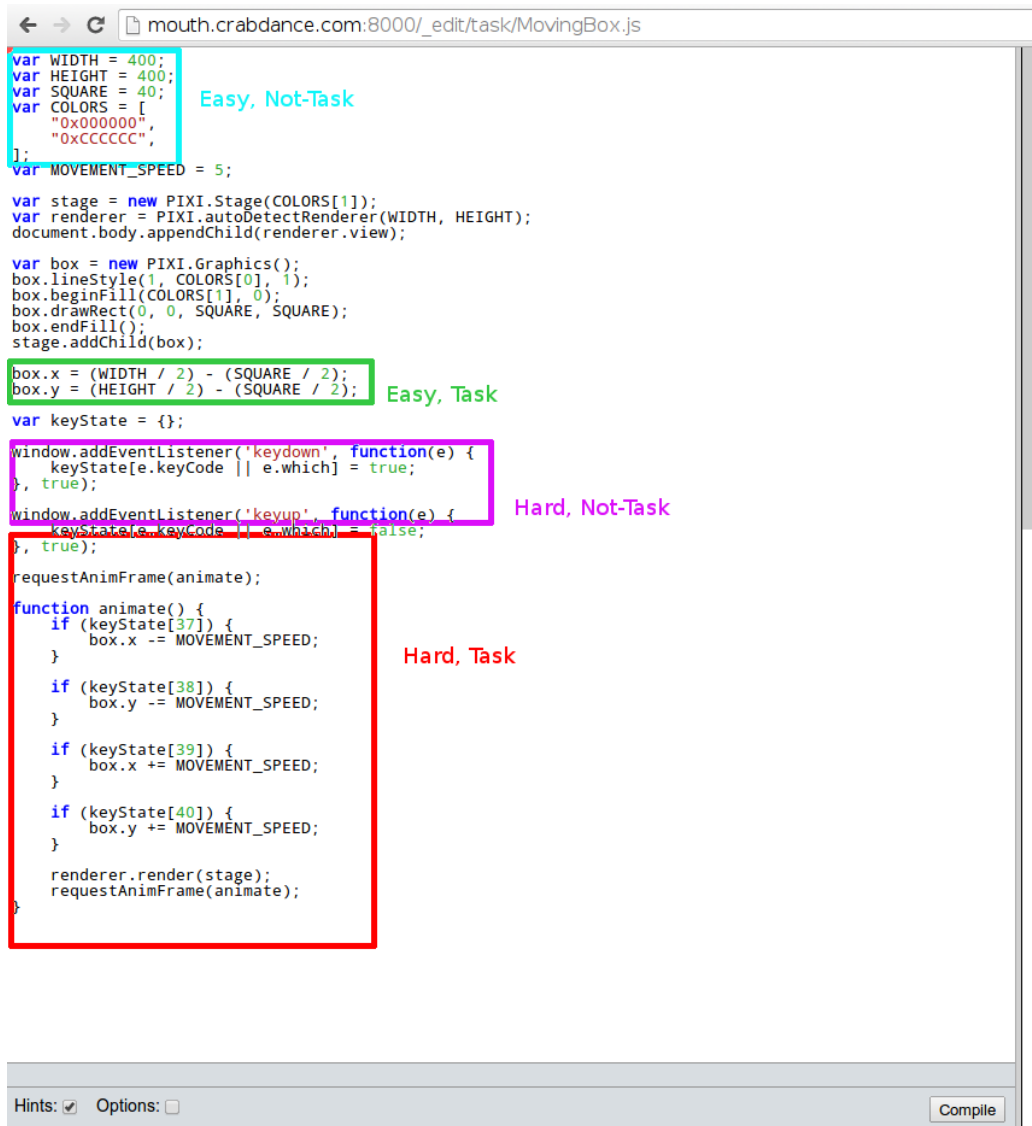


Figure 5: JS click regions

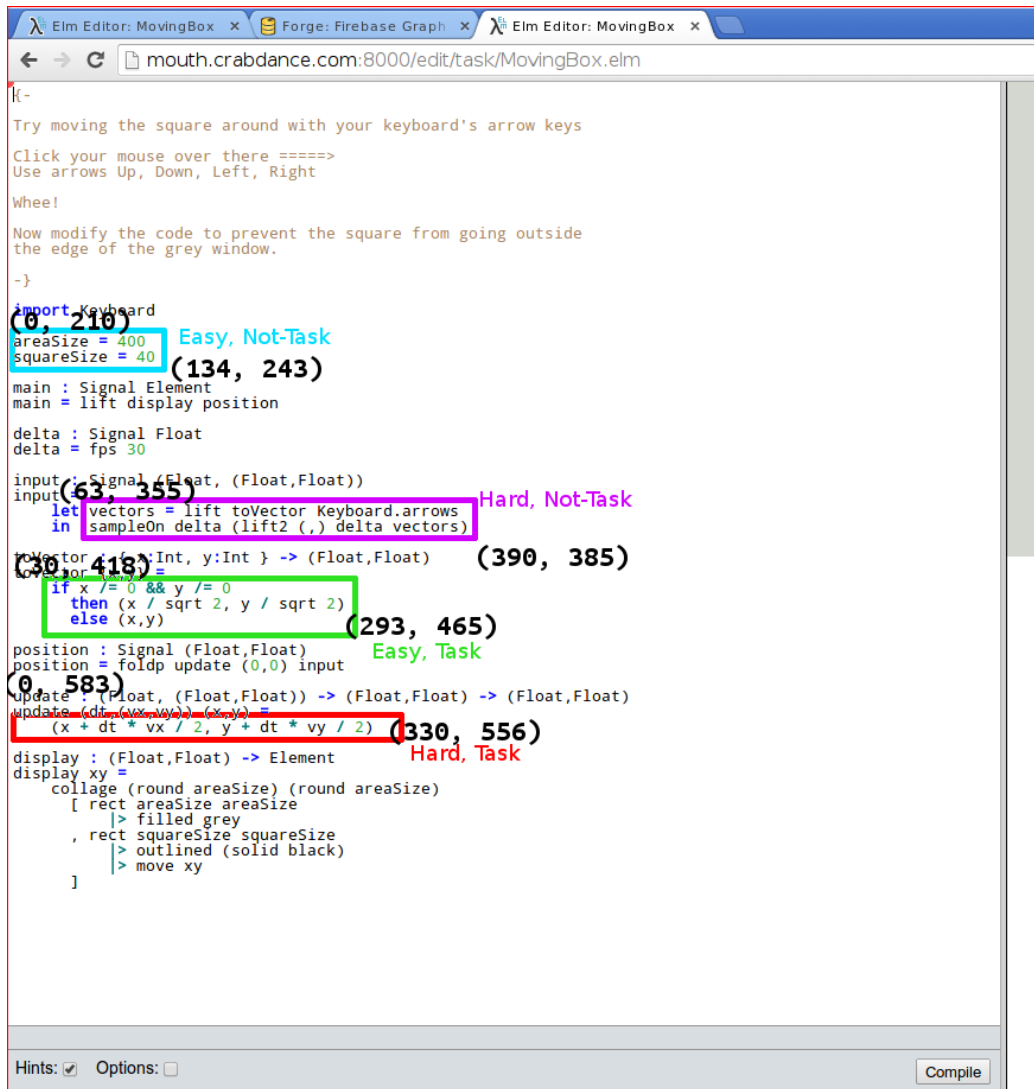


Figure 6: Elm labelled regions

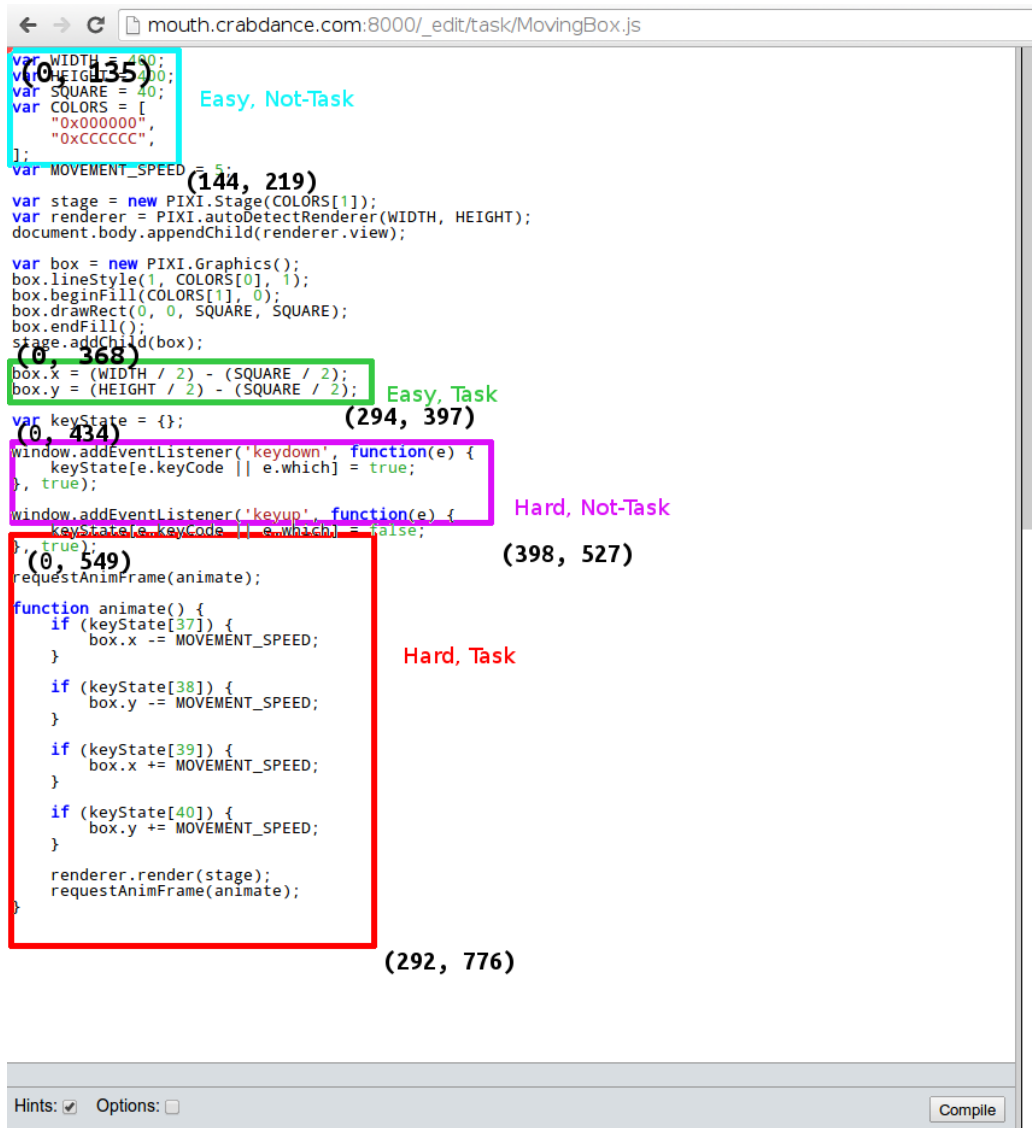


Figure 7: JS labelled click regions

13 Appendices

Appendix A

13.1 Design Diagrams

Appendix B

13.2 Raw results output

13.2.1 firebase-mouseclick-data.json

```
1 {
2   "js" : {
3     "12" : {
4       "-JKVwXgnfg4POT9MArjy" : {
5         "t" : 1397490853983,
6         "y" : 0,
7         "x" : 0
8       },
9       "-JKVwXl8I_bLZNqxpP36c" : {
10        "t" : 1397490854551,
11        "y" : 444,
12        "x" : 417
13      },
14      "-JKVwXj0LD-uRKDsT20m" : {
15        "t" : 1397490854503,
16        "y" : 444,
17        "x" : 417
18      }
19    }
20  },
21  "elm" : {
22    "33" : {
23      "-JKRgqJIRPH2EG-edZb5" : {
24        "t" : 1397419631953,
```

```

25         "y" : 249.59999084472656,
26         "x" : 48.79999923706055
27     },
28     "-JKRhRQi31pr9AP1p1Rr" : {
29         "t" : 1397419787709,
30         "y" : 294.3999938964844,
31         "x" : 152.8000030517578
32     },
33     "-JKRffOszNGfgwNdnO_X" : {
34         "t" : 1397419324585,
35         "y" : 608,
36         "x" : 346.3999938964844
37     }
38 }
39 }
40 }

```

13.2.2 error_log.json

```

1  {
2      "2014-04-11 21:14:32.141743994+01:00":{
3          "Parse error at (line 37, column 44):",
4          "unexpected 'a'",
5          "expecting \"{ - \", \" \" or end of input"
6      },
7      "2014-04-11 21:35:41.694436974+01:00":{
8          "Type error on line 27, column 16 to 77:",
9          "      (min (max x (-hHeight)) hHeight,",
10         "      (min (max y (-hWidth)),hWidth))",
11         "",
12         "Expected Type: Float",
13         "Actual Type: (Float -> Float, Float)"
14     },
15     "2014-04-12 00:19:14.945129550+01:00":{
16         "Parse error at (line 1, column 1):",
17         "unexpected \"<\"",
18         "expecting reserved word 'module', reserved word 'import'"

```



```

19         or at least one datatype or variable definition"
20     },
21     "2014-04-12 00:19:21.553633974+01:00":{
22         "Parse error at (line 1, column 1):",
23         "unexpected \"/\",",
24         "expecting reserved word 'module', reserved word 'import'
25         or at least one datatype or variable definition"
26     },
27     "2014-04-12 00:19:27.053901481+01:00":{
28         "Parse error at (line 1, column 1):",
29         "unexpected \"/\",",
30         "expecting reserved word 'module', reserved word 'import'
31         or at least one datatype or variable definition"
32     },
33 }

```

Appendix C

13.3 Code

- All code available here: <https://github.com/spanners/elm-lang.org>.
 - This is a modified version of Evan Czaplicki's elm-lang.org code, available here: <https://github.com/elm-lang/elm-lang.org>
- Elm task here: <http://mouth.crabdance.com:8000/edit/task/MovingBox.elm>
- Javascript task here: http://mouth.crabdance.com:8000/_edit/task/MovingBox.js

13.3.1 LICENSE

Copyright (c) 2012-2013 Evan Czaplicki

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of Evan Czaplicki nor the names of other contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

13.3.2 install_elm.sh

```
1  #!/bin/bash
2
3  sudo apt-get install libgl1-mesa-dev libglc-dev freeglut3-dev
4  sudo apt-get install libedit-dev libglw1-mesa libglw1-mesa-dev
5  sudo apt-get install ghc
6  wget http://www.haskell.org/ghc/dist/7.6.3/ghc-7.6.3-src.tar.bz2
7  tar xjf ghc-7.6.3-src.tar.bz2
8  cd ghc-7.6.3/mk
9  cp build.mk.sample build.mk
```

```

10 sed -i 's/^#BuildFlavour = quick/BuildFlavour = quick/' build.mk
11 cd ..
12 ./configure
13 make -j 8
14 sudo make install
15 cd
16 platform="2013.2.0.0/haskell-platform-2013.2.0.0.tar.gz"
17 wget "http://lambda.haskell.org/platform/download/"$platform
18 tar xzvf haskell-platform-2013.2.0.0.tar.gz
19 cd haskell-platform-2013.2.0.0
20 ./configure
21 make
22 sudo make install
23 cabal update
24 cabal install cabal-install
25 cabal install elm
26 cabal install elm-server
27 exit 0

```

13.3.3 Server.hs

```

1 {-# OPTIONS_GHC -W #-}
2 {-# LANGUAGE DeriveDataTypeable #-}
3 {-# LANGUAGE OverloadedStrings #-}
4 module Main where
5
6 import           Control.Applicative
7 import           Control.Monad.Error
8 import qualified Data.ByteString           as BS
9 import qualified Data.ByteString.Char8    as BSC
10 import qualified Data.HashMap.Strict      as Map
11 import           Data.Maybe               (fromMaybe)
12 import qualified Elm.Internal.Utills      as Elm
13
14 import qualified Text.Blaze.Html.Renderer.Utf8 as BlazeBS
15 import           Text.Blaze.Html5            ((!))
16 import qualified Text.Blaze.Html5            as H

```

```

17 import qualified Text.Blaze.Html5.Attributes as A
18 import           Text.Regex
19
20 import           GHC.Conc
21 import           Snap.Core
22 import           Snap.Http.Server
23 import           Snap.Util.FileServe
24 import           System.Console.CmdArgs
25 import           System.Directory
26 import           System.FilePath           as FP
27 import           System.Process
28
29 import qualified Editor
30 import qualified Elm.Internal.Paths         as Elm
31 import qualified Generate
32 import           Utils                      (Lang (..))
33
34 data Flags = Flags
35   { port :: Int
36   } deriving (Data, Typeable, Show, Eq)
37
38 flags :: Flags
39 flags = Flags
40   { port = 8000 &= help "set the port of the server"
41   }
42
43 -- / Set up the server.
44 main :: IO ()
45 main = do
46   setNumCapabilities =<< getNumProcessors
47   putStrLn "Initializing Server"
48   getRuntimeAndDocs
49   setupLogging
50   precompile
51   cargs <- cmdArgs flags
52   httpServe (setPort (port cargs) defaultConfig) $
53     ifTop (serveElm "public/Empty.elm")
54     <|> route [ ("try", serveHtml Editor.empty)

```

```

55         , ("edit", edit Elm)
56         , ("_edit", edit Javascript)
57         , ("code", code Elm)
58         , ("_code", code Javascript)
59         , ("compile", compile Elm)
60         , ("_compile", compile Javascript)
61         , ("hotswap", hotswap)
62     ]
63     <|> serveDirectoryWith directoryConfig "public/build"
64     <|> serveDirectoryWith simpleDirectoryConfig "resources"
65     <|> error404
66
67 error404 :: Snap ()
68 error404 =
69     do modifyResponse $ setResponseStatus 404 "Not found"
70     serveElm "public/build/Error404.elm"
71
72 serveElm :: FilePath -> Snap ()
73 serveElm = serveFileAs "text/html; charset=UTF-8"
74
75 logAndServeJS :: MonadSnap m => H.Html -> m ()
76 logAndServeJS = serveHtml
77
78 logAndServeHtml :: MonadSnap m => (H.Html, Maybe String) -> m ()
79 logAndServeHtml (html, Nothing) = serveHtml html
80 logAndServeHtml (html, Just err) =
81     do timeStamps <- liftIO $ readProcess "date" ["--rfc-3339=ns"] ""
82     liftIO $ appendFile "error_log.json" $ "{\n" ++ init timeStamps
83     ++ "\n",
84     ++ show (lines err)
85     ++ "\n",
86     setContentType "text/html" <$> getResponse
87     writeLBS (BlazeBS.renderHtml html)
88
89 embedHtml :: MonadSnap m => H.Html -> Lang -> String -> m ()
90 embedHtml html lang participant =
91     do elmSrc <- liftIO $ readFile "EmbedMe.elm"
92     setContentType "text/html" <$> getResponse

```

```

93         writeLBS (BlazeBS.renderHtml (embedMe lang elmSrc html participant))
94
95     serveHtml :: MonadSnap m => H.Html -> m ()
96     serveHtml html =
97         do setContentType "text/html" <$> getResponse
98           writeLBS (BlazeBS.renderHtml html)
99
100    hotswap :: Snap ()
101    hotswap = maybe error404 serve =<< getParam "input"
102        where
103            serve code =
104                do setContentType "application/javascript" <$> getResponse
105                  writeBS . BSC.pack . Generate.js $ BSC.unpack code
106
107    compile :: Lang -> Snap ()
108    compile lang = maybe error404 serve =<< getParam "input"
109        where
110            serve = case lang of
111                Elm -> logAndServeHtml
112                      . Generate.logAndHtml "Compiled Elm"
113                      . BSC.unpack
114                Javascript -> logAndServeJS
115                              . Generate.logAndJS "Compiled JS"
116                              . BSC.unpack
117
118    edit :: Lang -> Snap ()
119    edit lang = do
120        participant <- BSC.unpack . fromMaybe "" <$> getParam "p"
121        cols <- BSC.unpack . fromMaybe "50%,50%" <$> getQueryParam "cols"
122        withFile (Editor.ide lang cols participant)
123
124    code :: Lang -> Snap ()
125    code lang = do
126        participant <- BSC.unpack . fromMaybe "" <$> getParam "p"
127        embedWithFile Editor.editor lang participant
128
129    embedee :: Lang -> String -> String -> H.Html
130    embedee lang elmSrc participant =

```

```

131 H.span $ do
132   case Elm.compile elmSrc of
133     Right jsSrc ->
134       jsAttr $ H.preEscapedToMarkup (subRegex oldID jsSrc newID)
135     Left err ->
136       H.span ! A.style "font-family: monospace;" $
137         mapM_ (\line ->
138             H.preEscapedToMarkup
139               (Generate.addSpaces line)
140             >> H.br)
141           (lines err)
142   jsAttr $ H.preEscapedToMarkup $ visualiser
143   where langStr = (case lang of
144                     Elm -> "elm"
145                     Javascript -> "js")
146   visualiser =
147     concat [ "var firebaseData = new Firebase('"
148             , "http://sweltering-fire-9141.firebaseio.com/"
149             , "dissertation/"
150             , langStr
151             , "/"
152             , participant
153             , "');"
154             , "var elm = Elm.fullscreen(Elm.EmbedMe, {"
155             , "  stamped: {"
156             , "    t: 0,"
157             , "    x: 0,"
158             , "    y: 0"
159             , "  }"
160             , "});"
161             , "firebaseData.on('child_added',"
162             , "function(snapshot) {"
163             , "  elm.ports.stamped.send(snapshot.val());"
164             , "});" ]
165   oldID = mkRegex "var user_id = \"1\";"
166   newID = "var user_id = \"" ++ langStr
167           ++ "/"
168           ++ participant

```

```

169                                     ++ "\";"
170         jsAttr = H.script ! A.type_ "text/javascript"
171
172     embedMe :: Lang -> String -> H.Html -> String -> H.Html
173     embedMe lang elmSrc target participant = target >> embedee
174                                     lang
175                                     elmSrc
176                                     participant
177
178     embedWithFile :: (Lang -> FilePath -> String -> H.Html) -> Lang
179                                     -> String
180                                     -> Snap ()
181
182     embedWithFile handler lang participant = do
183         path <- BSC.unpack . rqPathInfo <$> getRequest
184         let file = "public/" ++ path
185         exists <- liftIO (doesFileExist file)
186         if not exists then error404 else
187             do content <- liftIO $ readFile file
188                embedHtml (handler lang path content) lang participant
189
190     withFile :: (FilePath -> String -> H.Html) -> Snap ()
191     withFile handler = do
192         path <- BSC.unpack . rqPathInfo <$> getRequest
193         let file = "public/" ++ path
194         exists <- liftIO (doesFileExist file)
195         if not exists then error404 else
196             do content <- liftIO $ readFile file
197                serveHtml $ handler path content
198
199     directoryConfig :: MonadSnap m => DirectoryConfig m
200     directoryConfig =
201         fancyDirectoryConfig
202         { indexGenerator = defaultIndexGenerator
203           , mimeTypeTypes = Map.insert ".elm" "text/html" defaultMimeTypes
204         }
205
206

```



```

207 indexStyle :: BS.ByteString
208 indexStyle =
209     "body { margin:0; font-family:sans-serif; \
210     \         background:rgb(245,245,245);\
211     \         font-family: calibri, verdana, helvetica, arial; }\
212     \div.header { padding: 40px 50px; font-size: 24px; }\
213     \div.content { padding: 0 40px }\
214     \div.footer { display:none; }\
215     \table { width:100%; border-collapse:collapse; }\
216     \td { padding: 6px 10px; }\
217     \tr:nth-child(odd) { background:rgb(216,221,225); }\
218     \td { font-family:monospace }\
219     \th { background:rgb(90,99,120); color:white; text-align:left;\
220     \     padding:10px; font-weight:normal; }"
221
222 setupLogging :: IO ()
223 setupLogging =
224     do createDirectoryIfMissing True "log"
225        createIfMissing "log/access.log"
226        createIfMissing "log/error.log"
227     where
228         createIfMissing path = do
229             exists <- doesFileExist path
230             unless exists $ BS.writeFile path ""
231
232     -- | Compile all of the Elm files in public/,
233     -- | placing results in public/build/
234 precompile :: IO ()
235 precompile =
236     do setCurrentDirectory "public"
237        files <- getFiles True ".elm" "."
238        forM_ files $ \file ->
239            rawSystem "elm" [ "--make"
240                            , "--runtime=/elm-runtime.js"
241                            , file ]
242        htmls <- getFiles False ".html" "build"
243        mapM_ adjustHtmlFile htmls
244        setCurrentDirectory ".."

```

```

245     where
246         getFiles :: Bool -> String -> FilePath -> IO [FilePath]
247         getFiles skip ext directory =
248             if skip && "build" `elem` map FP.dropTrailingPathSeparator
249                 (FP.splitPath directory)
250             then return [] else
251                 (do contents <- map (directory </>) `fmap`
252                     getDirectoryContents
253                     directory
254                     let files = filter ((ext==) . FP.takeExtension) contents
255                         directories = filter (not . FP.hasExtension) contents
256                         filess <- mapM (getFiles skip ext) directories
257                         return (files ++ concat filess))
258
259     getRuntimeAndDocs :: IO ()
260     getRuntimeAndDocs = do
261         writeFile "resources/elm-runtime.js" =<< readFile Elm.runtime
262         writeFile "resources/docs.json" =<< readFile Elm.docs
263
264     adjustHtmlFile :: FilePath -> IO ()
265     adjustHtmlFile file =
266         do src <- BSC.readFile file
267         let (before, after) = BSC.breakSubstring "<title>" src
268         BSC.writeFile (FP.replaceExtension file "elm") $
269             BSC.concat [before, style, after]
270         removeFile file
271
272     style :: BSC.ByteString
273     style =
274         "<style type=\"text/css\">\n\
275         \ a:link {text-decoration: none; color: rgb(15,102,230);} \n\
276         \ a:visited {text-decoration: none} \n\
277         \ a:active {text-decoration: none} \n\
278         \ a:hover {text-decoration: underline; color: rgb(234,21,122);} \n\
279         \ body { font-family: \"Lucida Grande\", \
280         \ \"Trebuchet MS\", \"Bitstream Vera Sans\", Verdana, Helvetica, \
281         \ sans-serif !important; } \n\
282         \ p, li { font-size: 14px !important; \n\

```

```

283 \           line-height: 1.5em !important; }\n\
284 \</style>"

```

13.3.4 Editor.hs

```

1  {-# LANGUAGE OverloadedStrings #-}
2  module Editor (editor, ide, empty) where
3
4  import           Data.Monoid              (mempty)
5  import           Network.HTTP.Base       (urlEncode)
6  import qualified System.FilePath         as FP
7  import           Text.Blaze.Html
8  import qualified Text.Blaze.Html5       as H
9  import qualified Text.Blaze.Html5.Attributes as A
10
11 import           Data.Maybe               (fromMaybe)
12 import qualified Elm.Internal.Utls        as Elm
13
14 import           Generate                 (addSpaces)
15 import           Utls
16
17
18 -- | Display an editor and the compiled result side-by-side.
19 ide :: Lang -> String -> String -> FilePath -> String -> Html
20 ide lang cols participant fileName code =
21     case lang of
22         Javascript -> buildIde ("JS Editor: ", "/_compile")
23         Elm        -> buildIde ("Elm Editor: ", "/compile")
24     where buildIde (editStr, compileStr) =
25         ideBuilder lang
26             cols
27             participant
28             (editStr ++ FP.takeBaseName fileName)
29             fileName
30             (compileStr ++ "?input=" ++ urlEncode code)
31
32

```

```

33  -- / Display an editor and the compiled result side-by-side.
34  empty :: Html
35  empty = ideBuilder Elm "50%,50%" "1" "Try Elm" "Empty.elm" "/Try.elm"
36
37  ideBuilder :: Lang -> String
38              -> String
39              -> String
40              -> String
41              -> String
42              -> Html
43  ideBuilder lang cols participant title input output =
44      case lang of
45          Javascript -> makeIde "_code/"
46          Elm        -> makeIde "code/"
47      where
48          makeIde codeStr =
49              H.docTypeHtml $ do
50                  H.head . H.title . toHtml $ title
51                  preEscapedToMarkup $
52                      concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
53                          , "  <frame name=\"input\" src=\"/\" , codeStr,
54                          input, "?p=",
55                          participant, "\" />\n"
56                          , "  <frame name=\"output\" src=\"/\" , output, "\" />\n"
57                          , "</frameset>" ]
58
59
60  -- / list of themes to use with CodeMirror
61  themes :: [String]
62  themes = [ "ambiance", "blackboard", "cobalt", "eclipse"
63            , "elegant", "erlang-dark", "lesser-dark", "monokai"
64            , "neat", "night", "rubyblue", "solarized", "twilight"
65            , "vibrant-ink", "xq-dark" ]
66
67  jsFiles :: AttributeValue -> [AttributeValue]
68  jsFiles syntaxFile =
69      [ "/codemirror-3.x/lib/codemirror.js"
70        , syntaxFile

```

```

71         , "/misc/showdown.js"
72         , "/misc/editor.js?0.11" ]
73
74
75 -- / Create an HTML document that allows you to edit and submit Elm code
76 -- for compilation.
77 editor :: Lang -> FilePath -> String -> Html
78 editor lang filePath code =
79     case lang of
80         Javascript -> buildEditor ( "JS Editor: "
81                                   , "/codemirror-3.x\
82                                   \mode/javascript/javascript.js"
83                                   , "/_compile")
84         Elm -> buildEditor ( "Elm Editor: "
85                             , "/codemirror-3.x/mode/elm/elm.js"
86                             , "/compile")
87     where buildEditor (editStr, syntaxFile, compileStr) =
88         H.html $ do
89             H.head $ do
90                 H.title . toHtml $ editStr ++ FP.takeBaseName filePath
91                 H.link ! A.rel "stylesheet"
92                     ! A.href "/codemirror-3.x/lib/codemirror.css"
93                 mapM_ themeAttr themes
94                 H.link ! A.rel "stylesheet" ! A.type_ "text/css"
95                     ! A.href "/misc/editor.css"
96                 mapM_ script $ jsFiles syntaxFile
97                 script "/elm-runtime.js?0.11"
98                 script "http://cdn.firebase.com/v0/firebase.js"
99             H.body $ do
100                 H.form ! A.id "inputForm"
101                     ! A.action compileStr
102                     ! A.method "post"
103                     ! A.target "output" $ do
104                     H.div ! A.id "editor_box" $
105                         H.textarea ! A.name "input"
106                             ! A.id "input" $ toHtml ('\n':code)
107                     H.div ! A.id "options" $ do
108                         bar "documentation" docs

```

```

1109         bar "editor_options" editorOptions
1110         bar "always_on" (buttons >> options)
1111         jsAttr "initEditor();"
1112     themeAttr theme = H.link ! A.rel "stylesheet"
1113                             ! A.href (toValue
1114                                     ("/codemirror-3.x/theme/"
1115                                     ++ theme
1116                                     ++ ".css" :: String))
1117     jsAttr = H.script ! A.type_ "text/javascript"
1118     script jsFile = jsAttr ! A.src jsFile $ mempty
1119
1120 bar :: AttributeValue -> Html -> Html
1121 bar id' body = H.div ! A.id id' ! A.class_ "option" $ body
1122
1123 buttons :: Html
1124 buttons = H.div ! A.class_ "valign_kids"
1125             ! A.style "float:right; padding-right: 6px;"
1126             $ compileButton
1127
1128     where
1129         compileButton =
1130             H.input
1131                 ! A.type_ "button"
1132                 ! A.id "compile_button"
1133                 ! A.value "Compile"
1134                 ! A.onclick "compile()"
1135                 ! A.title "Ctrl-Enter: change program behavior \
1136                             \but keep the state"
1137
1138 options :: Html
1139 options = H.div ! A.class_ "valign_kids"
1140             ! A.style "float:left; padding-left:6px; padding-top:2px;"
1141             $ (docs' >> opts)
1142
1143     where
1144         docs' =
1145             H.span ! A.title "Show documentation and types." $ "Hints:" >>
1146                 H.input ! A.type_ "checkbox"
1147                         ! A.id "show_type_checkbox"
1148                         ! A.onchange "showType(this.checked);"

```

```

147
148     opts =
149         H.span ! A.title "Show editor options."
150             ! A.style "padding-left: 12px;" $ "Options:" >>
151             H.input ! A.type_ "checkbox"
152                 ! A.id "options_checkbox"
153                 ! A.onChange "showOptions(this.checked);"
154
155 editorOptions :: Html
156 editorOptions = theme >> zoom >> lineNumbers
157     where
158         optionFor :: String -> Html
159         optionFor text =
160             H.option ! A.value (toValue text) $ toHtml text
161
162     theme =
163         H.select ! A.id "editor_theme"
164             ! A.onChange "setTheme(this.value)"
165             $ mapM_ optionFor themes
166
167     zoom =
168         H.select ! A.id "editor_zoom"
169             ! A.onChange
170                 "setZoom(this.options[this.selectedIndex].\
171                 \innerHTML)"
172             $ mapM_ optionFor ["100%", "80%", "150%", "200%"]
173
174     lineNumbers = do
175         H.span ! A.style "padding-left: 16px;" $ "Line Numbers:"
176         H.input ! A.type_ "checkbox"
177             ! A.id "editor_lines"
178             ! A.onChange "showLines(this.checked);"
179
180 docs :: Html
181 docs = tipe >> desc
182     where
183         tipe = H.div ! A.class_ "type" $ message >> more
184

```

```

185     message = H.div !
186         A.style "position:absolute; left:4px; right:36px;\
187             \overflow:hidden; text-overflow:ellipsis;" $ ""
188
189     more = H.a ! A.id "toggle_link"
190         ! A.style "display:none; float:right;"
191         ! A.href "javascript:toggleVerbose();"
192         ! A.title "Ctrl+H"
193         $ ""
194
195     desc = H.div ! A.class_ "doc"
196         ! A.style "display:none;"
197         $ ""
198

```

13.3.5 Generate.hs

```

1 {-# LANGUAGE OverloadedStrings #-}
2 module Generate (logAndJS, logAndHtml, html, js, addSpaces) where
3
4 import           Data.Maybe           (fromMaybe)
5 import           Data.Monoid           (mempty)
6 import           Text.Blaze           (preEscapedToMarkup)
7 import           Text.Blaze.Html5     ((!))
8 import qualified Text.Blaze.Html5     as H
9 import qualified Text.Blaze.Html5.Attributes as A
10
11 import qualified Elm.Internal.Uutils    as Elm
12 import           Uutils
13
14 logAndJS :: String -> String -> H.Html
15 logAndJS name src = getJSPage name src
16
17 logAndHtml :: String -> String -> (H.Html, Maybe String)
18 logAndHtml name src =
19     let elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
20     in

```



```

21     case Elm.compile src of
22         Right jsSrc -> do
23             (getHtmlPage name elmname jsSrc, Nothing)
24         Left err -> do
25             (getErrPage name err, Just err)
26
27 getJSPage :: String -> String -> H.Html
28 getJSPage name jsSrc =
29     H.docTypeHtml $ do
30         H.head $ do
31             H.meta ! A.charset "UTF-8"
32             H.title . H.toHtml $ name
33             H.link ! A.rel "stylesheet" ! A.type_ "text/css"
34                                     ! A.href "/misc/js.css"
35             script "/pixi.js"
36         H.body $ do
37             H.div ! A.style "width: 400px; height: 400px; position:\
38                 \ absolute; top: 0; left: 0; opacity: 0;" $ mempty
39             jsAttr $ preEscapedToMarkup jsSrc
40     where jsAttr = H.script ! A.type_ "text/javascript"
41           script jsFile = jsAttr ! A.src jsFile $ mempty
42           embed jsCode = jsAttr $ jsCode
43
44 getHtmlPage :: String -> String -> String -> H.Html
45 getHtmlPage name elmname jsSrc =
46     H.docTypeHtml $ do
47         H.head $ do
48             H.meta ! A.charset "UTF-8"
49             H.title . H.toHtml $ name
50             H.style ! A.type_ "text/css" $ preEscapedToMarkup
51                 ("a:link {text-decoration: none; color: rgb(15,102,230);} \n\
52                 \a:visited {text-decoration: none} \n\
53                 \a:active {text-decoration: none} \n\
54                 \a:hover {text-decoration: underline; \
55                 \color: rgb(234,21,122);} " :: String)
56         H.body $ do
57             let js = H.script ! A.type_ "text/javascript"
58                 runFullscreen =

```

```

59         "var runningElmModule = Elm.fullscreen(" ++ elmname
60                                         ++ ")"
61     js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
62     js $ preEscapedToMarkup jsSrc
63     js $ preEscapedToMarkup runFullscreen
64
65 getErrPage :: String -> String -> H.Html
66 getErrPage name err =
67     H.docTypeHtml $ do
68         H.head $ do
69             H.meta ! A.charset "UTF-8"
70             H.title . H.toHtml $ name
71         H.body $
72             H.span ! A.style "font-family: monospace;" $
73             mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
74                 (lines err)
75
76
77
78 -- / Using a page title and full source of an Elm program, compile down
79 -- to a valid HTML document.
80 html :: String -> String -> H.Html
81 html name src =
82     H.docTypeHtml $ do
83         H.head $ do
84             H.meta ! A.charset "UTF-8"
85             H.title . H.toHtml $ name
86             H.style ! A.type_ "text/css" $ preEscapedToMarkup
87                 ("a:link {text-decoration: none; color: rgb(15,102,230);}\\n\\
88                 \\a:visited {text-decoration: none}\\n\\
89                 \\a:active {text-decoration: none}\\n\\
90                 \\a:hover {text-decoration: underline;\\
91                 \\ color: rgb(234,21,122);} " :: String)
92         H.body $ do
93             let js = H.script ! A.type_ "text/javascript"
94                 elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
95                 runFullscreen =
96                     "var runningElmModule = Elm.fullscreen(" ++ elmname

```

```

97                                     ++ ") "
98     js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
99     case Elm.compile src of
100       Right jsSrc -> do
101         js $ preEscapedToMarkup jsSrc
102         js $ preEscapedToMarkup runFullscreen
103       Left err ->
104         H.span ! A.style "font-family: monospace;" $
105         mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
106             (lines err)
107
108 addSpaces :: String -> String
109 addSpaces str =
110   case str of
111     ' ' : ' ' : rest -> " &nbsp;" ++ addSpaces rest
112     c : rest -> c : addSpaces rest
113     [] -> []
114
115 js :: String -> String
116 js src = case Elm.compile src of
117   Right js -> "{ \"success\" : " ++ show js ++ " }"
118   Left err -> "{ \"error\" : " ++ show err ++ " }"

```

13.3.6 EmbedMe.elm

```

1 module EmbedMe where
2
3 import Graphics.Input as Input
4 import JavaScript as JS
5 import JavaScript.Experimental as JEXP
6 import Http
7 import Json
8 import Mouse
9 import Dict
10 import Window
11
12

```

```

13  (~>) = flip lift
14  infixl 4 ~>
15
16  type Stamp = { t: Float, x: Float, y: Float }
17
18  -- Incoming
19
20  port stamped : Signal { t: Float, x: Float, y: Float }
21
22  stamps : Signal [Stamp]
23  stamps = foldp (:) [] stamped
24
25  scene (w,h) locs =
26    let drawCircle {t, x, y} =
27      circle 5 |> filled (hsva y 1 1 0.7)
28      |> move (x - toFloat w / 2, toFloat h / 2 - y)
29    in collage w h (map drawCircle locs)
30
31  -- Uncomment the following line in order to visualise clicks
32  --main = lift2 scene Window.dimensions stamps
33
34  -- Outgoing
35
36  -- Do not change user_id = "1"
37  -- It gets replaced with the actual user_id when compiled to JS
38  user_id = "1"
39
40  firebaseRequest requestType requestData =
41    Http.request requestType
42      ("https://sweltering-fire-9141.firebaseio.com/"
43      ++ "dissertation/"
44      ++ user_id
45      ++ ".json")
46      requestData
47      []
48
49  serialize r = r |> JEXP.fromRecord
50              |> Json.fromJSObject

```

```

51         |> Json.toJSString " "
52         |> JS.toString
53
54 toRequestData (t, (x, y)) = {t = t, x = x, y = y } |> serialize
55
56 clicks = timestamp (sampleOn Mouse.isDown Mouse.position)
57
58 toRequest click = case click of
59   (0, (0, 0)) -> firebaseRequest "get" ""
60   -          -> firebaseRequest "post" (click |> toRequestData)
61
62 requests = clicks ~> toRequest
63
64 sendRequests = Http.send requests

```

13.3.7 editor.js.diff

```

1  diff --git a/resources/misc/editor.js b/resources/misc/editor.js
2  index d2bebc8..302663e 100644
3  --- a/resources/misc/editor.js
4  +++ b/resources/misc/editor.js
5  @@ -293,7 +293,7 @@ function showOptions(show) {
6   function showType(show) {
7     cookie('showtype', show);
8     document.getElementById('show_type_checkbox').checked = show;
9   -   var newMode = (show ? { mode: Mode.TYPES, verbose: false }
10   +   var newMode = (show ? { mode: Mode.TYPES, verbose: true}
11     : { mode: Mode.NONE });
12     if (mode.mode === Mode.OPTIONS) {
13       mode.hidden = newMode;
14   @@ -305,8 +305,8 @@ function showType(show) {
15
16   function toggleVerbose() {
17     if (!mode.verbose) showType(true);
18   -   document.getElementById('toggle_link').innerHTML = mode.verbose ?
19   -     'more' : 'less';
20   -   mode.verbose = !mode.verbose;

```

```

21 +     document.getElementById('toggle_link').innerHTML = mode.verbose ? ' ' : '';
22 +     mode.verbose = true;
23     updateDocumentation();
24 }
25
26 @@ -318,8 +318,8 @@ function showVerbose() {
27     function hideStuff() {
28         if (mode.hidden) mode = mode.hidden;
29         document.getElementById('options_checkbox').checked = false;
30 -     mode.verbose = false;
31 -     document.getElementById('toggle_link').innerHTML = 'more';
32 +     mode.verbose = true;
33 +     document.getElementById('toggle_link').innerHTML = ' ';
34     updateDocumentation();
35 }

```

13.3.8 MovingBox.js

```

1  /*
2
3  Try moving the square around with your keyboard's arrow keys
4
5  Click your mouse over there =====>
6  Use arrows Up, Down, Left, Right
7
8  Whee!
9
10 Now modify the code to prevent the square from going outside
11 the edge of the grey window.
12
13 */
14
15 var WIDTH = 400;
16 var HEIGHT = 400;
17 var SQUARE = 40;
18 var COLORS = [
19     "0x000000",

```

```

20     "0xCCCCCC",
21 ];
22 var MOVEMENT_SPEED = 5;
23
24 var stage = new PIXI.Stage(COLORS[1]);
25 var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
26 document.body.appendChild(renderer.view);
27
28 var box = new PIXI.Graphics();
29 box.lineStyle(1, COLORS[0], 1);
30 box.beginFill(COLORS[1], 0);
31 box.drawRect(0, 0, SQUARE, SQUARE);
32 box.endFill();
33 stage.addChild(box);
34
35 box.x = (WIDTH / 2) - (SQUARE / 2);
36 box.y = (HEIGHT / 2) - (SQUARE / 2);
37
38 var keyState = {};
39
40 window.addEventListener('keydown', function(e) {
41     keyState[e.keyCode || e.which] = true;
42 }, true);
43
44 window.addEventListener('keyup', function(e) {
45     keyState[e.keyCode || e.which] = false;
46 }, true);
47
48 requestAnimationFrame(animate);
49
50 function animate() {
51     if (keyState[37]) {
52         box.x -= MOVEMENT_SPEED;
53     }
54
55     if (keyState[38]) {
56         box.y -= MOVEMENT_SPEED;
57     }

```

```

58
59     if (keyState[39]) {
60         box.x += MOVEMENT_SPEED;
61     }
62
63     if (keyState[40]) {
64         box.y += MOVEMENT_SPEED;
65     }
66
67     renderer.render(stage);
68     requestAnimationFrame(animate);
69 }

```

13.3.9 MovingBox.elm

```

1  {-
2
3  Try moving the square around with your keyboard's arrow keys
4
5  Click your mouse over there ==>
6  Use arrows Up, Down, Left, Right
7
8  Whee!
9
10 Now modify the code to prevent the square from going outside
11 the edge of the grey window.
12
13 -}
14
15 import Keyboard
16
17 areaSize = 400
18 squareSize = 40
19
20 main : Signal Element
21 main = lift display position
22

```



```

23 delta : Signal Float
24 delta = fps 30
25
26 input : Signal (Float, (Float,Float))
27 input =
28     let vectors = lift toVector Keyboard.arrows
29     in sampleOn delta (lift2 (,) delta vectors)
30
31 toVector : { x:Int, y:Int } -> (Float,Float)
32 toVector {x,y} =
33     if x /= 0 && y /= 0
34     then (x / sqrt 2, y / sqrt 2)
35     else (x,y)
36
37 position : Signal (Float,Float)
38 position = foldp update (0,0) input
39
40 update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
41 update (dt,(vx,vy)) (x,y) =
42     (x + dt * vx / 2, y + dt * vy / 2)
43
44 display : (Float,Float) -> Element
45 display xy =
46     collage (round areaSize) (round areaSize)
47     [ rect areaSize areaSize
48       |> filled grey
49     , rect squareSize squareSize
50       |> outlined (solid black)
51       |> move xy
52     ]

```

13.3.10 ClicksPerCategory.py

```

1 import unittest
2 import json
3
4 class ClicksPerCategory(object):

```

```

5     def get_clicks_in_region(self, jsonStr, x_min, y_min,
6         x_max, y_max):
7         region = {}
8         mouse_data = json.loads(jsonStr)
9         for click in mouse_data:
10            coords = mouse_data[click]
11            if ((x_min <= coords["x"] <= x_max) and
12                (y_min <= coords["y"] <= y_max)):
13                region[click] = coords
14        return region
15
16
17    class FooTests(unittest.TestCase):
18
19        def setUp(self):
20            self.mr = ClicksPerCategory()
21
22        def test_get_clicks_in_region(self):
23            jsonStr = '{"uniq_id1": {"t": 1, "x":1, "y":2},\
24                "uniq_id2": {"t":2, "x":3, "y":4}}'
25            x_min = 0
26            x_max = 4
27            y_min = 3
28            y_max = 4
29            expected = {"uniq_id2": {"t":2, "x":3, "y":4}}
30            actual = self.mr.get_clicks_in_region(jsonStr, x_min, y_min,
31                x_max, y_max)
32            self.assertEqual(actual, expected)
33
34        def main():
35            unittest.main()
36
37    if __name__ == '__main__':
38        main()
39

```

13.3.11 get_clicks_per_category.py

```
1 import ClicksPerCategory
2 import sys
3 import json
4
5 cpc = ClicksPerCategory.ClicksPerCategory()
6
7 def get_all_clicks():
8     _all = dict()
9     contents = ""
10    fname = ""
11    region1 = (0,538, 330,556)
12    for _file in sys.argv[1:]:
13        with open(_file) as f:
14            fname, contents = f.name, f.read()
15            _all[fname] = cpc.get_clicks_in_region(contents, *region1)
16    return _all
17
18 _all = get_all_clicks()
19 total = 0
20 for fname in _all:
21     length = len(_all[fname])
22     print fname, length
23     total += length
24 print "TOTAL:", total
```

13.3.12 DecodeMouseData.py

```
1 #!/usr/bin/python2.7
2
3 import json
4 import sys
5
6 class DecodeMouseData(object):
7
8     def decode(self, jsonString):
```

```

9         return json.loads(jsonString)
10
11     def getNumberOfClicks(self, jsonString):
12         return len(self.decode(jsonString))
13
14     def getSessionDuration(self, jsonString):
15         decoded = self.decode(jsonString)
16
17         finish = max([x["t"] for x in decoded.values()])
18         start = min([x["t"] for x in decoded.values()])
19
20         # compute number of minutes in this number of milliseconds
21         return (finish - start) / 60000.0
22
23     def getFilesTimeClickDict(self, files):
24         timeAndClicks = dict()
25         filename = str()
26         for jsonFile in files:
27             with open(jsonFile) as f:
28                 jsonString = f.read()
29                 filename = f.name
30                 print filename
31                 timeAndClicks[filename] = (self.getSessionDuration(jsonString),
32                                             self.getNumberOfClicks(jsonString))
33         return timeAndClicks
34
35     def getDictPrettyPrint(self, timeAndClicks):
36         output = ""
37         tablelines = "-"*11 + " " + "-"*10
38         output += tablelines + "\n"
39         output += "Time (min) Clicks\n"
40         output += tablelines + "\n"
41         length = len(timeAndClicks)
42         for i,filename in enumerate(timeAndClicks):
43             output += ("%10f %10d" % (timeAndClicks[filename][0],
44                                     timeAndClicks[filename][1])) + "\n"
45         output += tablelines
46         return output

```

```

47
48
49 if __name__ == "__main__":
50     dmd = DecodeMouseData()
51     print dmd.getDictPrettyPrint(dmd.getFilesTimeClickDict(sys.argv[1:]))
52     timeAndClicks = dmd.getFilesTimeClickDict(sys.argv[1:])
53     total = 0
54     for filename in timeAndClicks:
55         total += timeAndClicks[filename][1]
56     print "TOTAL: ", total

```

13.3.13 test__DecodeMouseData.py

```

1 import unittest
2 import DecodeMouseData as d
3
4
5 class FooTests(unittest.TestCase):
6
7     def setUp(self):
8         self.dmd = d.DecodeMouseData()
9
10    def testDecode(self):
11        expected = {'1':2, '3':4}
12        actual = self.dmd.decode('{"1":2, "3":4}')
13
14        self.assertEqual(actual, expected)
15
16    def testMouseDecode(self):
17        expected = {"-JKMBewWrFje3lHT8spD" :
18                    {"t" : 1397327310399, "y" : 646, "x" : 629}}
19        actual = self.dmd.decode(
20            '{"-JKMBewWrFje3lHT8spD" : ' +
21            '{"t" : 1397327310399, "y" : 646, "x" : 629}}')
22
23        self.assertEqual(actual, expected)
24

```

```

25     def testNumClicks(self):
26         expected = 1
27         actual = self.dmd.getNumberOfClicks(
28             '{"-JKMBewWrFje3lHT8spD" : ' +
29             '{"t" : 1397327310399, "y" : 646, "x" : 629}}}')
30
31         self.assertEqual(actual, expected)
32
33     def testLotsClicks(self):
34         expected = 2
35
36         actual = self.dmd.getNumberOfClicks("""{
37     "-JKMBewWrFje3lHT8spD" : {
38         "t" : 1397327310399,
39         "y" : 646,
40         "x" : 629
41     },
42     "-JKMBewawNo6G_Zdfnkk" : {
43         "t" : 1397327310465,
44         "y" : 646,
45         "x" : 629
46     }
47 }""")
48
49         self.assertEqual(actual, expected)
50
51     def testComputeSessionTime(self):
52         expected = 0.0011
53
54         actual = self.dmd.getSessionDuration("""{
55     "-JKMBewWrFje3lHT8spD" : {
56         "t" : 1397327310399,
57         "y" : 646,
58         "x" : 629
59     },
60     "-JKMBewawNo6G_Zdfnkk" : {
61         "t" : 1397327310465,
62         "y" : 646,

```

```

63         "x" : 629
64     }
65 }""")
66
67     self.assertEqual(actual, expected)
68
69 def main():
70     unittest.main()
71
72 if __name__ == '__main__':
73     main()
74

```

13.3.14 ttest_scipy.py

```

1 from scipy import stats
2 import numpy as np
3
4 elm = [76,487,105,66]
5 js = [33,12,69,3]
6
7 outcome = stats.ttest_ind(elm,js, equal_var=False)
8
9 print("t-value: %4.2f, p-value: %4.2f" % (outcome[0], outcome[1]))
10
11 observed = elm + js
12
13 expected = sum(observed)/(len(observed)*1.0)
14
15 chi_squared = sum([(x - expected)**2)/expected for x in observed])
16
17 print("chi-squared : %4.2f" % chi_squared)

```

Appendix D

13.4 Pilot Study 1

13.4.1 Consent Form

13.4.1.1 Study Overview This study aims to assess how Functional Reactive Programming Languages are used. To do this, we will be asking you to modify a Mario game to get him to fly. The session will take no more than 1 hour.

During the session, you will be introduced to Elm, a functional reactive programming language, as well as being shown what we want you to create. We'll also present you with a questionnaire to see what experience you've had with Functional programming (or similar concepts) before. Finally we'll give you another questionnaire to ask how you think the session went, and the level of workload in the task.

The session will be recorded on video and then the audio from the session will be transcribed anonymously in order to find any problems that you had during the session. During this process, the data will be stored securely.

Important Information

All data collected during this study will be recorded such that your individual results are anonymous and cannot be traced back to you. Your results will not be passed to any third party and are not being collected for commercial reasons. Participation in this study does not involve physical or mental risks outside of those encountered in everyday life. All procedures and information can be taken at face value and no deception is involved. You have the right to withdraw from the study at any time and to have any data about you destroyed. If you do decide to withdraw, please inform the experimenter.

By signing this form you acknowledge that you have read the information given above and understand the terms and conditions of this study.

Name	Age	Sex	Occupation
.....

Signed

-

Date

-

Experimenter: Simon Buist, Dept. of Computer Science. EMAIL ADDRESS

13.4.2 Pre-questionnaire

13.4.2.1 Functional Programming languages

1. Have you ever used a Functional programming language before? Examples are: Scheme, Lisp, Haskell, ML, SPARK. Please circle one answer)
 - Yes
 - No
2. If so, please list the Functional programming languages you've used before:
 -

13.4.2.2 Design & Software For the purposes of this questionnaire, we consider a piece of software to be an application for which you have received/conceived of a specification, and coded a solution that meets this solution.

1. Have you designed software before?
 - Yes
 - No
2. On what platforms have you designed software?

- Desktop
- Mobile
- Tablet
- Web

3. For what purposes have you designed software?

- Commercial
- Academic (e.g. Coursework)
- Personal project
- Other:

4. Roughly how many pieces of software have you designed?

- ...

5. What programming languages do you know?

-

13.4.2.3 General Demographics

1. How old are you?

- ...

2. What is your sex? (Please circle one answer)

- Male
- Female

3. What is the highest degree or level of education you have completed?
If currently enrolled please indicate the highest you have attained previously. (Please circle one answer)

- None
- GCSEs or equivalent
- A/AS levels or equivalent
- BSc/BA or equivalent
- MSc/MA or equivalent
- PhD or equivalent

4. In what field was your highest qualification?
 -
5. What is your current employment status? (Please circle one answer)
 - Unemployed
 - Self-employed
 - Employed
 - Student
 - Retired
 - Unable to work

13.4.3 Post-Questionnaire

1. Please detail any comments on the result that you achieved

.....

2. Please detail any comments on how you achieved it

.....

3. Please detail any other comments

.....

If you want to have the study as a whole explained to you, please do so now. However we ask that you refrain from discussing this with potential future participants.

13.4.4 Participant 1

13.4.4.1 Consent Form

13.4.4.1.1 Study Overview This study aims to assess how Functional Reactive Programming Languages are used. To do this, we will be asking you to modify a Mario game to get him to fly. The session will take no more than 1 hour.

During the session, you will be introduced to Elm, a functional reactive programming language, as well as being shown what we want you to create. We'll also present you with a questionnaire to see what experience you've had with Functional programming (or similar concepts) before. Finally we'll give you another questionnaire to ask how you think the session went, and the level of workload in the task.

The session will be recorded on video and then the audio from the session will be transcribed anonymously in order to find any problems that you had during the session. During this process, the data will be stored securely.

Important Information

All data collected during this study will be recorded such that your individual results are anonymous and cannot be traced back to you. Your results will not be passed to any third party and are not being collected for commercial reasons. Participation in this study does not involve physical or mental risks outside of those encountered in everyday life. All procedures and information can be taken at face value and no deception is involved. You have the right to withdraw from the study at any time and to have any data about you destroyed. If you do decide to withdraw, please inform the experimenter.

By signing this form you acknowledge that you have read the information given above and understand the terms and conditions of this study.

Name	Age	Sex	Occupation
Participant #1	23	Male	Student

Signed

- YES

Date

- **10/12/2013**

Experimenter: Simon Buist, Dept. of Computer Science. EMAIL ADDRESS

13.4.4.2 Pre-questionnaire

13.4.4.2.1 Functional Programming languages

1. Have you ever used a Functional programming language before? Examples are: Scheme, Lisp, Haskell, ML, SPARK. Please circle one answer)
 - **Yes**
 - No
2. If so, please list the Functional programming languages you've used before:
 - **Lisp**

13.4.4.3 Design & Software For the purposes of this questionnaire, we consider a piece of software to be an application for which you have received/conceived of a specification, and coded a solution that meets this solution.

1. Have you designed software before?
 - **Yes**
 - No
2. On what platforms have you designed software?
 - **Desktop**
 - Mobile
 - Tablet
 - Web
3. For what purposes have you designed software?
 - Commercial
 - **Academic (e.g. Coursework)**
 - Personal project
 - Other:

4. Roughly how many pieces of software have you designed?

- **5**

5. What programming languages do you know?

- **Java, prolog, php, lisp**

13.4.4.4 General Demographics

1. How old are you?
 - 23
2. What is your sex? (Please circle one answer)
 - **Male**
 - Female
3. What is the highest degree or level of education you have completed?
If currently enrolled please indicate the highest you have attained previously. (Please circle one answer)
 - None
 - GCSEs or equivalent
 - **A/AS levels or equivalent**
 - BSc/BA or equivalent
 - MSc/MA or equivalent
 - PhD or equivalent
4. In what field was your highest qualification?
 - General studies
5. What is your current employment status? (Please circle one answer)
 - Unemployed
 - Self-employed
 - Employed
 - **Student**
 - Retired
 - Unable to work

13.4.4.5 Post-Questionnaire

1. Please detail any comments on the result that you achieved
 - **I am happy with my result.**
2. Please detail any comments on how you achieved it
 - **My first intention was to get myself familiarised with the language and its syntax and understand the basics of what the code was doing. After that I experimented with a few changes and succeeded in finishing the task.**
3. Please detail any other comments
 - **I think that the goal of the task could have been made more clear from the beginning, specifically what is meant by having the character fly and what should happen when you press the jumping button. Also, had to ask for the movement characters as they were not explained in the task description.**

If you want to have the study as a whole explained to you, please do so now. However we ask that you refrain from discussing this with potential future participants.

13.5 Pilot Study 2

13.5.1 SPSS Multiple Regression

N.B languages (Lan) and the relevance (Rel) and difficulty (Diff) are set to variables in the output: Elm := 1.00, JS := 2.00, and later in the *K-Way* and *Higher Order Effects* tables, Rel := 1, Diff := 2, Lan := 3

```

HILOGLINEAR Rel(1 2) Diff(1 2) Lan(1 2)
/CRITERIA ITERATION(20) DELTA(.5)
/PRINT=FREQ RESID ESTIM
/DESIGN.

```

Hierarchical Loglinear Analysis

Notes

Output Created		24-APR-2014 13:22:57
Comments		
Input	Active Dataset	DataSet0
	Filter	<none>
	Weight	Freq
	Split File	<none>
	N of Rows in Working Data File	8
Missing Value Handling	Definition of Missing	User-defined missing values are treated as missing.
	Cases Used	Statistics are based on all cases with valid data for all variables in the model.
Syntax		HILOGLINEAR Rel(1 2) Diff(1 2) Lan(1 2) /CRITERIA ITERATION (20) DELTA(.5) /PRINT=FREQ RESID ESTIM /DESIGN.
Resources	Processor Time	00:00:00.02
	Elapsed Time	00:00:00.01

[DataSet0]

Warnings

For Design 1, .500 has been added to all observed cells for this saturated model. This value may be changed by using the CRITERIA = DELTA subcommand.

Data Information

		N
Cases	Valid	8
	Out of Range ^a	0
	Missing	0
	Weighted Valid	851
Categories	Rel	2
	Diff	2
	Lan	2

a. Cases rejected because of out of range factor values.

Design 1

Convergence Information

Generating Class	Rel*Diff*Lan
Number of Iterations	1
Max. Difference between Observed and Fitted Marginals	.000
Convergence Criterion	.487

Cell Counts and Residuals

			Observed		Expected		Residuals	Std. Residuals
Rel	Diff	Lan	Count ^a	%	Count	%		
1.00	1.00	1.00	76.500	9.0%	76.500	9.0%	.000	.000
		2.00	33.500	3.9%	33.500	3.9%	.000	.000
	2.00	1.00	487.500	57.3%	487.500	57.3%	.000	.000
		2.00	12.500	1.5%	12.500	1.5%	.000	.000
2.00	1.00	1.00	105.500	12.4%	105.500	12.4%	.000	.000
		2.00	69.500	8.2%	69.500	8.2%	.000	.000
	2.00	1.00	66.500	7.8%	66.500	7.8%	.000	.000
		2.00	3.500	0.4%	3.500	0.4%	.000	.000

a. For saturated models, .500 has been added to all observed cells.

Goodness-of-Fit Tests

	Chi-Square	df	Sig.
Likelihood Ratio	.000	0	.
Pearson	.000	0	.

K-Way and Higher-Order Effects

	K	df	Likelihood Ratio		Pearson	
			Chi-Square	Sig.	Chi-Square	Sig.
K-way and Higher Order Effects ^a	1	7	1154.129	.000	1633.879	.000
	2	4	396.766	.000	519.719	.000
	3	1	.076	.783	.078	.780
K-way Effects ^b	1	3	757.363	.000	1114.160	.000
	2	3	396.690	.000	519.640	.000
	3	1	.076	.783	.078	.780

K-Way and Higher-Order Effects

	K	Number of Iterations
K-way and Higher Order Effects ^a	1	0
	2	2
	3	6
K-way Effects ^b	1	0
	2	0
	3	0

a. Tests that k-way and higher order effects are zero.

b. Tests that k-way effects are zero.

Parameter Estimates

Effect	Parameter	Estimate	Std. Error	Z	Sig.	95% Confidence Interval	
						Lower Bound	Upper Bound
Rel*Diff*Lan	1	-.039	.084	-.463	.643	-.203	.125
Rel*Diff	1	-.540	.084	-6.437	.000	-.704	-.375
Rel*Lan	1	.141	.084	1.682	.093	-.023	.305
Diff*Lan	1	-.671	.084	-8.001	.000	-.835	-.506
Rel	1	.277	.084	3.302	.001	.112	.441
Diff	1	.323	.084	3.854	.000	.159	.487
Lan	1	.981	.084	11.709	.000	.817	1.146

13.5.2 Pre-questionnaire

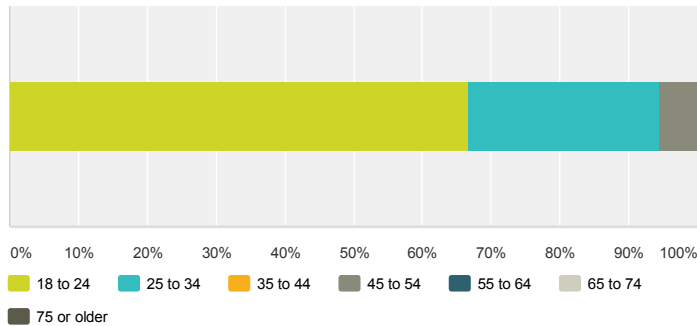
RESPONDENTS: 18 of 18

PAGE 2: Demographics

Q1

What is your age?

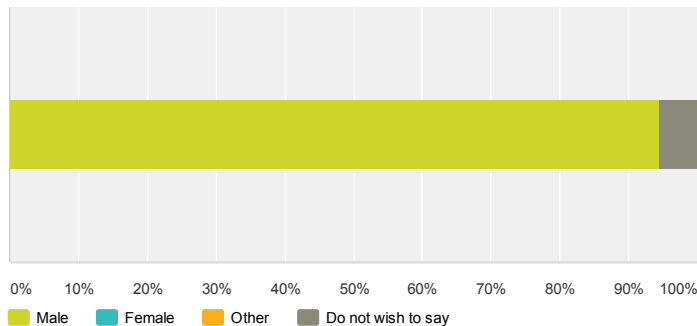
Answered: 18 Skipped: 0



Q2

Sex

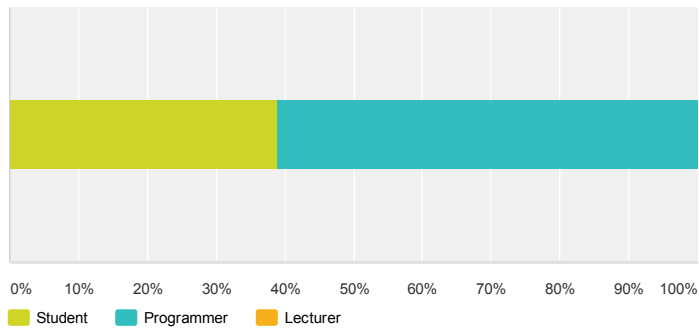
Answered: 18 Skipped: 0



Q3

Which of the following best describes your current occupation?

Answered: 18 Skipped: 0

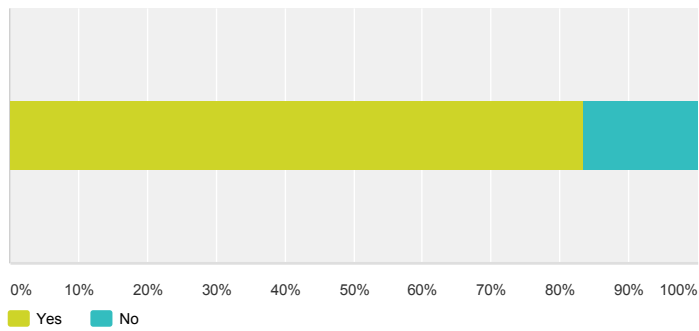


PAGE 3: Functional Programming

Q4

Have you ever used a Functional programming language before? Examples are: Scheme, Lisp, Haskell, ML, SPARK

Answered: 18 Skipped: 0



Q5

If so, please list the Functional Programming languages you've used before

Answered: 15 Skipped: 3

Answer Choices	Responses
1. Responses	100.00% 15

Showing 15 responses

Lisp
4/19/2014 12:11 AM
Lisp/scheme
4/13/2014 3:21 PM
Lisp
4/13/2014 12:35 PM
Haskell
4/13/2014 10:35 AM
Lisp
4/13/2014 10:12 AM
Lisp
4/13/2014 10:06 AM
Lisp
4/13/2014 12:19 AM
Lisp
4/12/2014 9:11 PM
Scheme
4/12/2014 8:33 PM
Haskell (relatively little time spent with it)
4/12/2014 7:26 PM
Common Lisp
4/12/2014 7:24 PM
Haskell
4/12/2014 7:24 PM
Haskell
4/12/2014 7:21 PM
Haskell
4/12/2014 7:16 PM
Haskell
4/12/2014 6:41 PM

— 2.	Responses	66.67%	10
------	-----------	--------	----

Showing 10 responses

Haskell
4/19/2014 12:11 AM
Elm
4/13/2014 10:35 AM
Python
4/13/2014 10:12 AM
Haskell
4/12/2014 9:11 PM
Lisp
4/12/2014 8:33 PM
Scheme
4/12/2014 7:24 PM
racket
4/12/2014 7:24 PM
OCaml
4/12/2014 7:21 PM
Erlang
4/12/2014 7:16 PM
Agda
4/12/2014 6:41 PM

– 3.	Responses	46.67%	7
------	-----------	--------	---

Showing 7 responses

Idris
4/13/2014 10:35 AM
Scala
4/12/2014 9:11 PM
Haskell
4/12/2014 7:24 PM
sml
4/12/2014 7:24 PM
Common Lisp
4/12/2014 7:21 PM
OCaml
4/12/2014 7:16 PM
Scheme
4/12/2014 6:41 PM

– 4.	Responses	40.00%	6
------	-----------	--------	---

Showing 6 responses

Common Lisp
4/13/2014 10:35 AM
Clojure
4/12/2014 9:11 PM
Erlang
4/12/2014 7:24 PM
Perl
4/12/2014 7:21 PM
Javascript
4/12/2014 7:16 PM
Common Lisp
4/12/2014 6:41 PM

– 5.	Responses	26.67%	4
------	-----------	--------	---

Showing 4 responses

Scheme
4/13/2014 10:35 AM

Elm
4/12/2014 9:11 PM

Lambda Calculus
4/12/2014 7:21 PM

Idris
4/12/2014 6:41 PM

6.	Responses	20.00%	3
----	-----------	--------	---

Showing 3 responses

OCaml
4/13/2014 10:35 AM

C++ templates
4/12/2014 7:21 PM

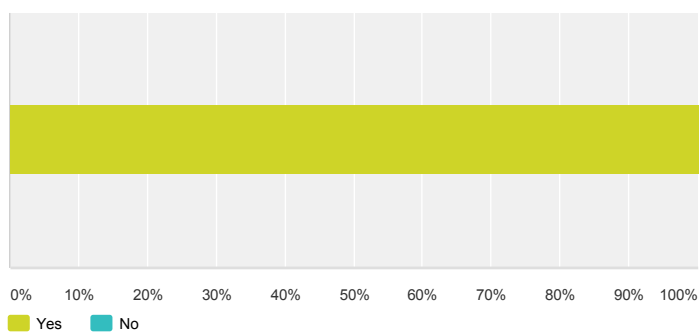
OCaml
4/12/2014 6:41 PM

PAGE 4: Design & Software

Q6

Have you designed software before?

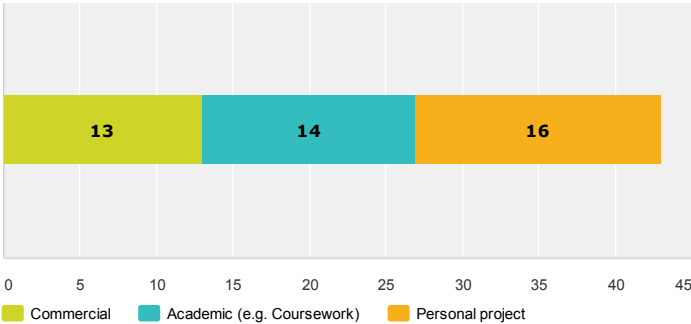
Answered: 18 Skipped: 0



Q7

If Yes, For what purposes have you designed software?

Answered: 18 Skipped: 0



Answer Choices	Responses
Commercial	72.22% 13
Academic (e.g. Coursework)	77.78% 14
Personal project	88.89% 16

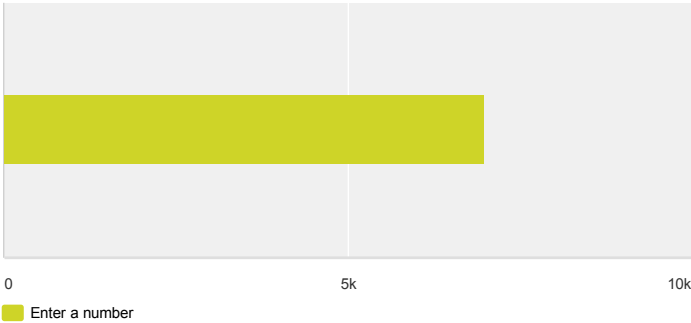
Total Respondents: 18

[Comments \(1\)](#)

Q8

Roughly how many pieces of software have you designed?

Answered: 18 Skipped: 0



Answer Choices	Average Number	Total Number	Responses
Enter a number Responses	6,984	125,720	18

Showing 18 responses

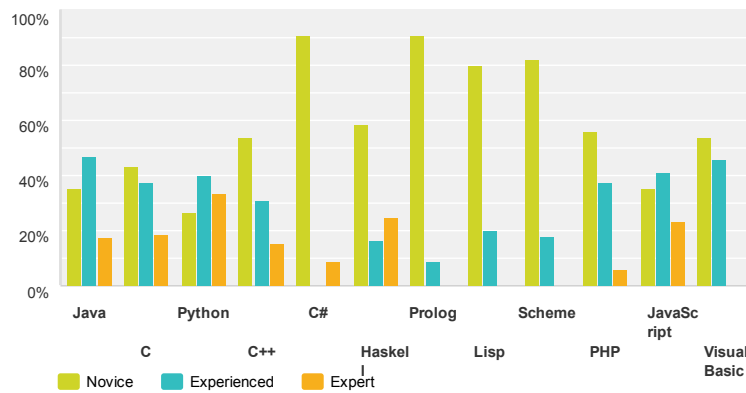
4	
4/19/2014 12:11 AM	
8	
4/18/2014 12:02 PM	
15	
4/16/2014 6:47 PM	
123456	
4/13/2014 4:00 PM	
10	
4/13/2014 12:35 PM	
10	
4/13/2014 10:36 AM	
5	
4/13/2014 10:13 AM	
10	
4/13/2014 10:06 AM	
10	
4/13/2014 12:19 AM	
1000	
4/12/2014 9:12 PM	
50	
4/12/2014 8:33 PM	
5	
4/12/2014 8:30 PM	
7	
4/12/2014 7:26 PM	
10	
4/12/2014 7:26 PM	
100	
4/12/2014 7:25 PM	
1000	
4/12/2014 7:23 PM	
10	
4/12/2014 7:17 PM	
10	
4/12/2014 6:42 PM	
Total Respondents: 18	

PAGE 5: Programming Proficiency

Q9

What programming languages do you know, and what is your proficiency in them? Novice: 0 to 6 months using the language Experienced: 1 to 2 years using the language at work or University Expert: 2+ years using the language for several projects, both as a hobby and at work

Answered: 17 Skipped: 1



	Novice	Experienced	Expert	Total	Average Rating
Java	35.29% 6	47.06% 8	17.65% 3	17	1.82
C	43.75% 7	37.50% 6	18.75% 3	16	1.75
Python	26.67% 4	40.00% 6	33.33% 5	15	2.07
C++	53.85% 7	30.77% 4	15.38% 2	13	1.62
C#	90.91% 10	0.00% 0	9.09% 1	11	1.18
Haskell	58.33% 7	16.67% 2	25.00% 3	12	1.67
Prolog	90.91% 10	9.09% 1	0.00% 0	11	1.09
Lisp	80.00% 12	20.00% 3	0.00% 0	15	1.20
Scheme	81.82% 9	18.18% 2	0.00% 0	11	1.18
PHP	56.25% 9	37.50% 6	6.25% 1	16	1.50
JavaScript	35.29% 6	41.18% 7	23.53% 4	17	1.88
Visual Basic	53.85% 7	46.15% 6	0.00% 0	13	1.46

[Comments \(7\)](#)

Showing 7 responses

Elm,Novice; Idris,Novice; F#,Novice; OCaml,Novice; LOLCode,Novice; Scala,Novice; Dart,Novice; BASH,Expert; HTML,Expert; CSS,Expert; LaTeX,Expert; XML,Expert; Ebuild,Expert
4/13/2014 10:42 AM

Ada, Experienced
4/13/2014 12:20 AM

Scala, Experienced; Clojure, Expert; Groovy, Experienced
4/12/2014 9:13 PM

Pascal, Expert
4/12/2014 8:34 PM

Perl,Expert; OCaml,Novice; bash,Expert; make,Expert; brainfuck,Novice; Delphi,Experienced; whitespace,Novice; Shakespeare Programming Language (SPL),Novice; INTERCAL,Novice; SQL,Expert; vim,Experienced; Windows batch (cmd.exe),Novice
4/12/2014 7:34 PM

Erlang,Novice; Brainfuck,Novice; Befunge,Experienced; Lua,Experienced; Bash,Novice
4/12/2014 7:31 PM

Ruby,Expert
4/12/2014 7:26 PM