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 28, 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

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

Declaration

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

Evan Czaplicki, in his senior thesis "Elm: Concurrent FRP for Functional GUIs", presents a groundbreaking unification of various styles of Functional Reactive Programming — Arrowized FRP, Concurrent FRP and others — resulting in his implementation of Elm, a Functional Reactive programming language in an aim to "simplify the complicated task of creating responsive and usable graphical user interfaces." (Czaplicki, 2012). However, there is not, as of yet any evidence to support this claim and others similar to this, and the thesis has inspired me to build further on his work.

In this dissertation, I am going to substantiate the following claims:

- Recording mouse-click data of users completing a programming task models cognitive load (Section 11). I have shown that there is statistical significance between the number of clicks in regions over languages Elm and Javascript.
- Users completing a task in Elm exhibit less thrash (*defined as:* number of clicks in task-regions over the duration of the task) than those that completed the same task in Javascript (Section 11.3.2).

2 Motivation

I am interested in the effect of Functional Reactive Programming [FRP] on User Interface programming.

I first grew an interest in the field of Functional Reactive Programming after seeing Bret Victor's "Inventing on Principle" (Victor, 2012). His talk claims that, in the traditional compile-run-debug cycle of coding, "most of the developer's time is spent looking at the code, blindly without an immediate connection to the thing they're making". He goes on to show a side-by-side illustration of a new style of development – on one side is the runtime preview, and on the other side is the code pertaining to said runtime. Changes in the code update the runtime, live. 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. I would like to investigate the claims Bret Victor makes, and indeed Elm, an instance of such a FRP, whose website also makes similar claims.

A counter-argument may be that this is much like giving a child a chainsaw. Is it too powerful? Does this tight feedback loop cut out a perhaps crucial pause for thought? Furthermore – is this appropriate for all types of programming? Is it at least appropriate for User Interface design? It has been shown that novices tend to “thrash” about, trying out many ideas that may or may not be a solution, whereas experts think much more about the problem at hand before proceeding with a solution (Lopez et al., 2012).

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.

As far as the scope of this project goes, I will be researching as much as is necessary in order to meet the aims of the project listed. Should I complete these aims, I may go on to do further user studies, or attempt to further analyse, compare and contrast User Interface Design and Declarative/Functional Reactive Programming languages against other methods, so as to make firmer statements about the benefits of Elm.

3 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 the paper by (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), Hunt & Thomas (1999)).

3.1 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.

4 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).

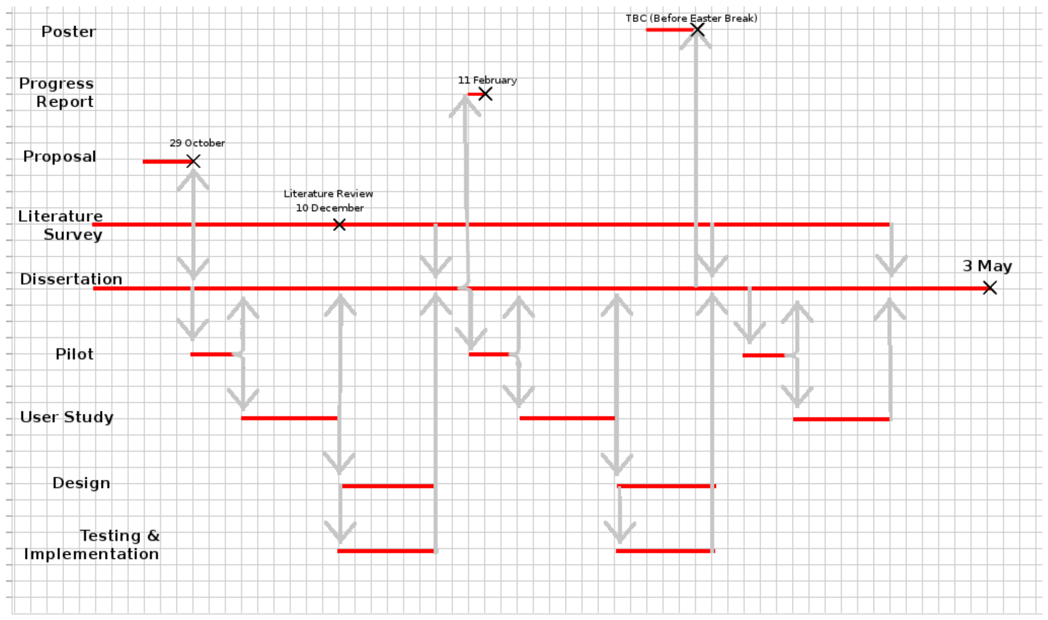


Figure 1: Gantt Chart

5 Literature Survey

5.1 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.

5.2 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. (an inspiring paper that provides foundational research into the behaviour of software developers while programming) 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).

5.3 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 millitary 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).

6 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.

6.1 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 correlation 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".

6.2 Pilot Studies

What might be surprising insights into declarative programming languages for User Interface Design in the case of Elm? I may explore Speak-aloud protocols where I 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.

An example dialog with the user may begin (Me speaking first):

- 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?
- ...

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

Motivating questions for these pilot studies are:

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

7 Pilot Study 1

Using a per-participant questionnaire (See 13.5.2), 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).

7.1 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.

7.1.1 Method

1. Consent form is signed (See 13.5.1)
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 13.5.3)
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.

7.1.2 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. essentialist NOT constructionist, 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.

Phase	Description of the process
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.

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

7.2 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.

7.3 Observations

7.3.1 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.

7.3.2 Model Adjustment 1

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

7.3.3 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?

7.3.4 Model Adjustment 2

- Scrap the Thematic Analysis methodology entirely. Instead, 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 – countrary 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.

7.4 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.

8 Requirements

8.1 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.

8.2 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, captuing 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

8.3 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

9 System Architecture

Before starting implementation, the design phase (more appropriately titled System Architecture, here) was an important step in the creation of the eventual IDE, as it set the direction in which the implementation began. In designing the extensions I wanted to make to the pre–existing Elm IDE, I first had to see if it was feasible. I devised a architectural diagram illustrating the control flow of clicking the *Compile* button in the editor (See Figure 2) to get an idea of where the extensions might fit. The stages (1,2,3,4), labelled in the diagram, are described in more detail in the list below.

1. On pressing the *Compile* button, `resources/misc/editor.js:compile()` submits the input form (the editor code pane), performing a `POST` of the source code in the editor code pane to the URL `/compile?input="<source code here>"` — the source code is passed as a raw string in the `input` parameter

2. The Server running the IDE receives this POST event and looks up the function mapping associated with the `/compile` route. In this case it is the `compile` function, which in turn applies the function `serveHtml` to result of `Generate.getHtmlPage getParam "input"` — more on this next.
3. Inside `Generate.hs`, `Elm.compile` is applied to the Elm source code string, which lexes, parses, and generates the Javascript source, returning it to the `getHtmlPage` function.
4. From then on the HTML page structure of the runtime is built back up again, resulting in the new runtime (in this case, the MovingBox program).

My extensions that I planned to make at the design stage are loosely illustrated in Figure 3.

It became apparent that I would need to make several modifications at each stage of the round–trip (1,2,3,4), and indeed heavily modify the code pane (left) in order to incorporate the same compile loop *within* the code pane in order to embed arbitrary Elm code. The embedded code would itself need to be created, and hooked into a database back–end to perform the JSON click data storage functionality

- For Stage 1., I determined that would probably need to send some extra information in the `POST` submission to `/compile`, perhaps using another parameter.
- For Stage 2., the `getHtmlPage` function in `Generate.hs` was predicted to need heavy modification, due to the fact that this is where the IDE builds the HTML page back up after compiling the Elm source. I proposed that I would need some way of differentiating between an *Embedded Elm* compile, and the *Elm IDE* (the pre–existing implementation) compile.
- I hoped that no modifications would be necessary at Stage 3., seeing as it is very difficult and time consuming to modify the compiler as, not only would several stages of the compilation process need modification, but the resulting generated Javascript and the Elm-runtime itself would

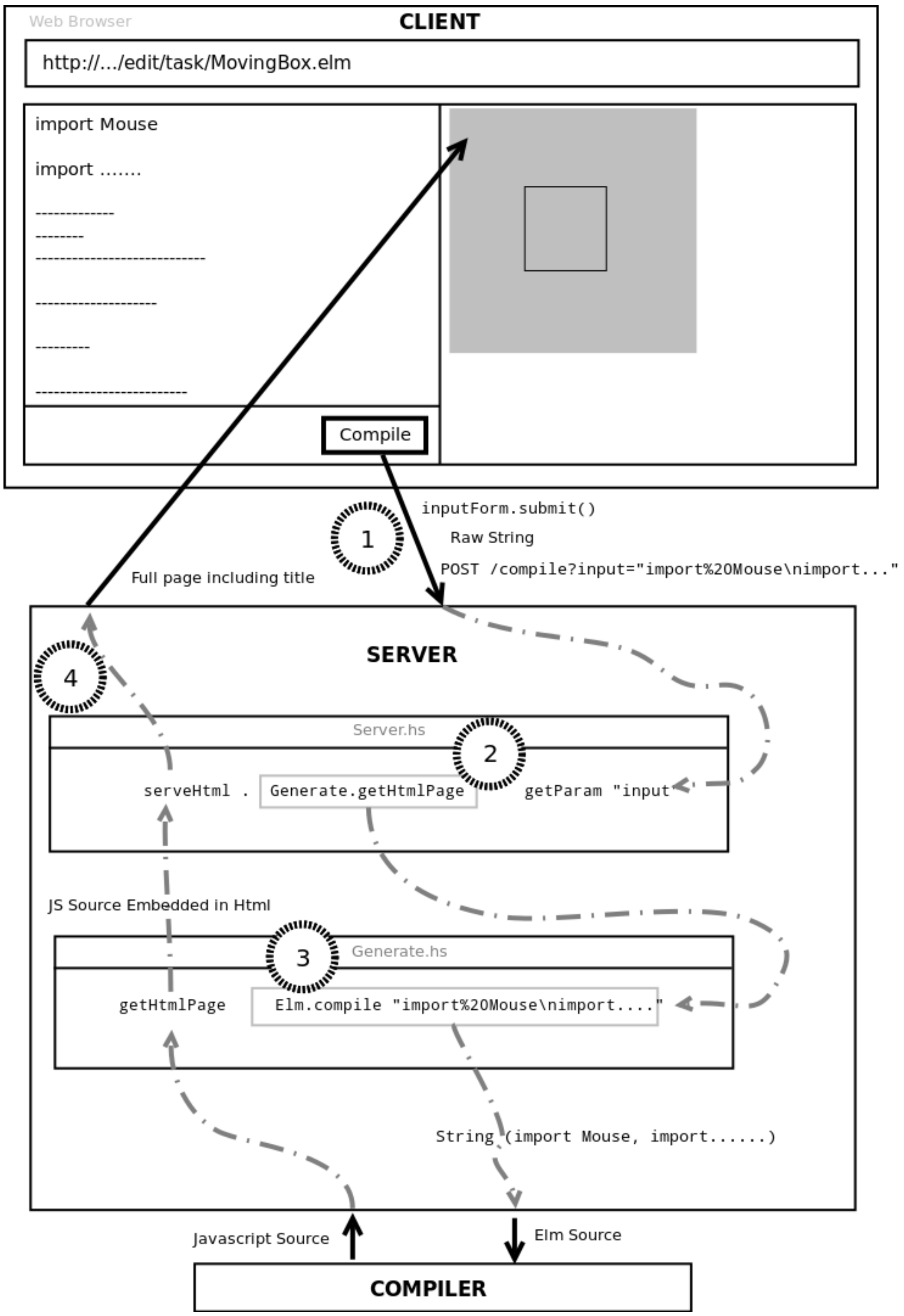
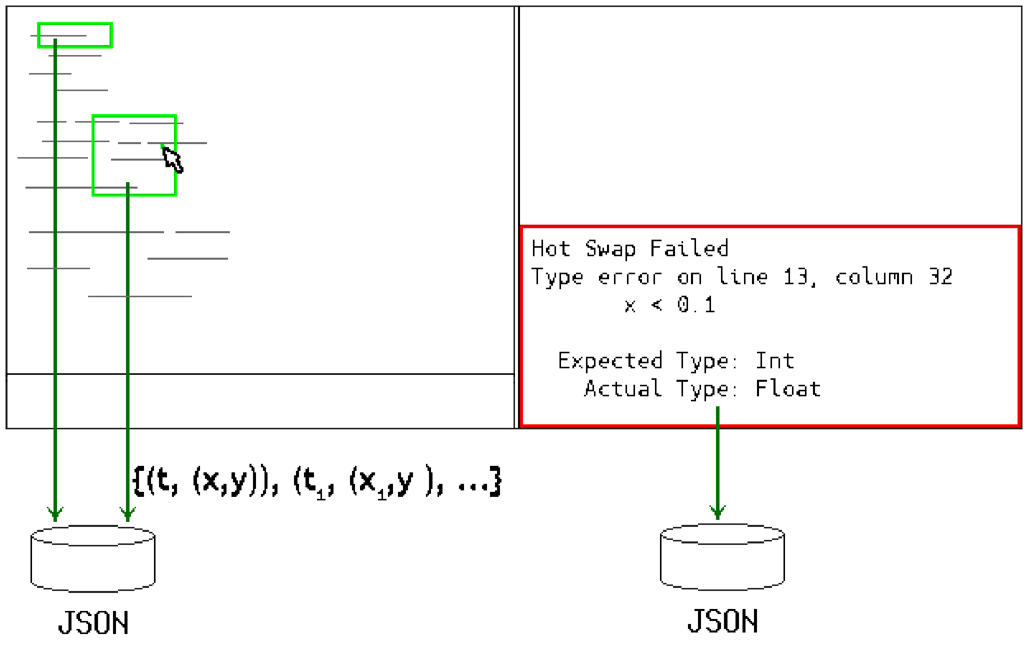

Web Browser
CLIENT
http://.../edit/task/MovingBox.elm
import Mouse
import ....
Compile
1
inputForm.submit()
Raw String
POST /compile?input="import%20Mouse\nimport..."
Full page including title
SERVER
4
Server.hs
2
serveHtml . Generate.getHtmlPage getParam "input"
JS Source Embedded in Html
3
Generate.hs
getHtmlPage Elm.compile "import%20Mouse\nimport...."
String (import Mouse, import......)
Javascript Source
Elm Source
COMPILER

Figure 2: Control flow of a compile in the Elm IDE



```
Hot Swap Failed
Type error on line 13, column 32
       x < 0.1

  Expected Type: Int
    Actual Type: Float
```

Figure 3: Extensions to be made to the IDE

need modifying. All I will be doing is compiling standard (although embedded) Elm code, so I suspected I could avoid this as long as I place my efforts on ensuring I write standard Elm.

I do note here that the design was not solely a sealed–off, Waterfall style phase in the development process. Rather, I gathered an intuition for where a feature might go, attempted to implement it, and possibly came back to rethink the architecture. The next section describes the implementation details, issues I came across in doing so, and a general commentary of the process.

10 Implementation

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

10.1 Establishing context

Initially, the implementation involved a lot of exploratory programming, 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.

10.2 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:

```
firebaseRequest requestType requestData =
  Http.request requestType
               "https://username.firebaseio-demo.com/dissertation.json"
               requestData
```

```
                []

serialize r = r |> JEXP.fromRecord
                |> Json.fromJSObject
                |> Json.toJSString " "
                |> JS.toString

toRequestData (x,y) = {x = x, y = y} |> serialize

toRequest event = case event of
  (x,y) -> firebaseRequest "post" (event |> toRequestData)

requests = clicks ~> toRequest

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 essense, 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 13.4.4, 13.4.3 and 13.4.5 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 13.4.4) is given below:

```haskell

ide :: Lang -> String -> String -> FilePath -> String -> Html
ide lang cols participant fileName code =
    case lang of
         Javascript -> buildIde ("JS Editor: ", "/_compile")
         Elm        -> buildIde ("Elm Editor: ", "/compile")
  where buildIde (editStr,compileStr) =
            ideBuilder lang
                       cols
                       participant
                       (editStr ++ FP.takeBaseName fileName)
                       fileName
                       (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.

10.3 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.

11 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.

11.1 Observations

Observations and participant feedback from Pilot Study 1 (See 13.5.4) 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:

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

11.1.1 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

11.2 Experiment

11.2.1 Method

Using the IDE I have augmented to gather click data, and a pre-questionnaire to determine level of expertise (See Section 13.6.2), I will split the sample of users completing the same task by language: into either Elm or JavaScript.

The task (See code listings 13.4.9 and 13.4.8 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 13.4.6) 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 2×2×2 study, using geometrically defined regions (See Figures 8 and 9), 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 some/all of the 'difficult' properties identified in the Design (See Section 9)). 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/Relevant
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.

11.3 Results

The raw click data (See 13.3.1), was processed with Python scripts (See 13.4.10, 13.4.11, 13.4.12) to produce the following tables:

Participant	Time (min)	Clicks
1	38.717	183
2	8.034	130
3	7.878	39
4	23.672	25
5	29.754	391
6	14.993	78
7	48.960	769
8	6.354	71
9	7.878	39
10	29.698	501
11	40.302	803

Participant	Time (min)	Clicks
12	12.319	65
13	17.106	79
14	12.958	119

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

Participant	Time (min)	Clicks
1	8.545	126
2	3.766	41
3	18.731	75
4	4.537	117

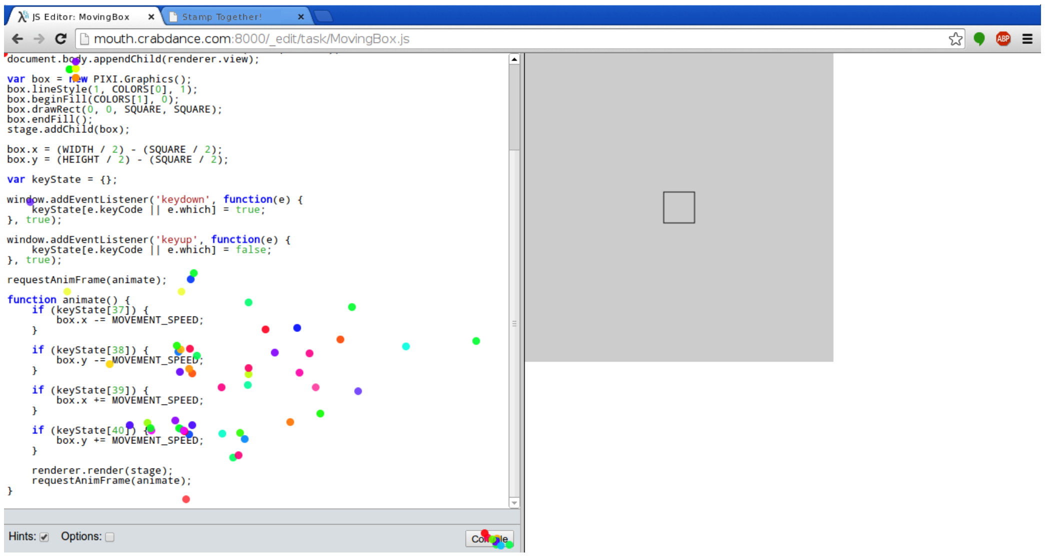
Table 5: Session time and clicks per session for JS task

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

Category	Expected %	Expected	Observed

Table 6: 2×2×2 comparison of clicks per category — Expected and Observed

Table 6, 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.

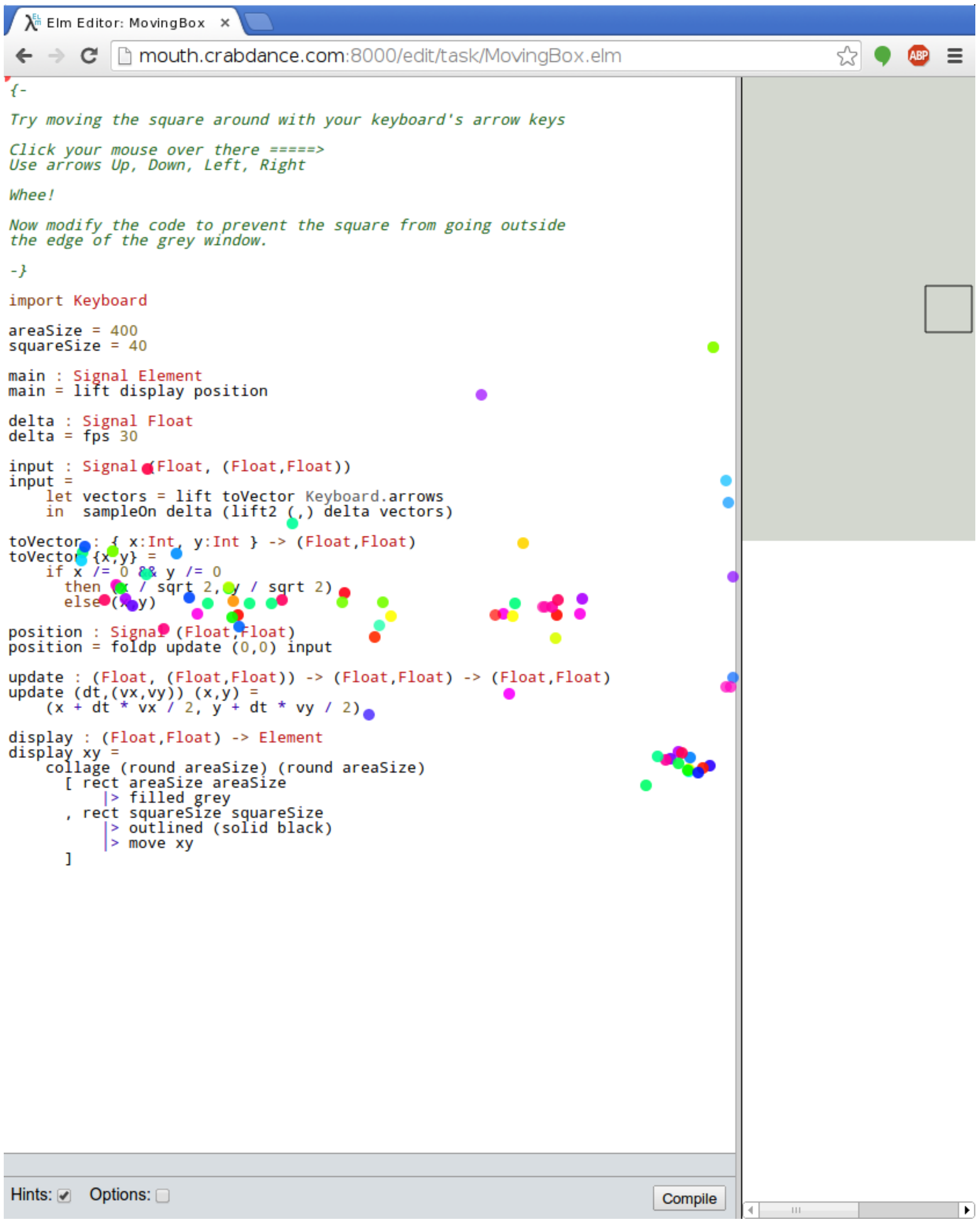


```
document.body.appendChild(renderer.view);

var box = new PIXI.Graphics();
box.lineStyle(1, COLORS[0], 1);
box.beginFill(COLORS[1], 0);
box.drawRect(0, 0, SQUARE, SQUARE);
box.endFill();
stage.addChild(box);

box.x = (WIDTH / 2) - (SQUARE / 2);
box.y = (HEIGHT / 2) - (SQUARE / 2);

var keyState = {};

window.addEventListener('keydown', function(e) {
    keyState[e.keyCode || e.which] = true;
}, true);

window.addEventListener('keyup', function(e) {
    keyState[e.keyCode || e.which] = false;
}, true);

requestAnimFrame(animate);

function animate() {
    if (keyState[37]) {
        box.x -= MOVEMENT_SPEED;
    }

    if (keyState[38]) {
        box.y -= MOVEMENT_SPEED;
    }

    if (keyState[39]) {
        box.x += MOVEMENT_SPEED;
    }

    if (keyState[40]) {
        box.y += MOVEMENT_SPEED;
    }

    renderer.render(stage);
    requestAnimFrame(animate);
}
```

Figure 4: Participant 18, JS task (Overlaid with mouse clicks)

See Figure 4 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 13.4.6) 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
```



```
{-

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

-}

import Keyboard

areaSize = 400
squareSize = 40

main : Signal Element
main = lift display position

delta : Signal Float
delta = fps 30

input : Signal (Float, (Float,Float))
input =
    let vectors = lift toVector Keyboard.arrows
    in  sampleOn delta (lift2 (,) delta vectors)

toVector : { x:Int, y:Int } -> (Float,Float)
toVector {x,y} =
    if x /= 0 && y /= 0
      then (x / sqrt 2, y / sqrt 2)
      else (x, y)

position : Signal (Float,Float)
position = foldp update (0,0) input

update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
update (dt,(vx,vy)) (x,y) =
    (x + dt * vx / 2, y + dt * vy / 2)

display : (Float,Float) -> Element
display xy =
    collage (round areaSize) (round areaSize)
      [ rect areaSize areaSize
          |> filled grey
      , rect squareSize squareSize
          |> outlined (solid black)
          |> move xy
      ]
```

Figure 5: Participant 15, Elm task (Illustrating the potential offset)

11.3.1 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 13.6.1 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 7: χ^2 calculation of clicks per quadrant

11.3.2 T–statistic result between groups Elm and JS

Using SciPy's `stats.ttest` function (See 13.4.14), 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

we can obtain a T–test value and ρ–value.

T–value:	1.50
ρ–value:	0.23

Table 8: T–value and ρ–value between languages

11.4 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 7 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.

11.5 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 with the solved task

(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 13.6.2) and a link to either the Elm or JS task (See 13.4.9 and 13.4.8, 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 13.6.2) 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 5) 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.

12 Conclusions

In this dissertation I have managed to achieve the product of an extended IDE that is capable performing user participation studies by logging input device data (Section 10). I have illustrated it's utility as a way of modelling cognitive load by way of conducting a user study (Section 11), whose findings have shown that there is statistical significance in the amount of user activity depending on the difficulty of the task, language being used, and the relevance of the code to the task at hand (Section 11.3.1).

In light of the original objectives, I have not been able to assert the claims Evan made in his senior thesis that Elm is indeed easy – in the Discussion of the second Pilot Study (Section 11.5) I list reasons that the interpretation I can make from the results is a lot weaker than I had anticipated, due to the flaws in the experimental methodology and the execution of the study. The Further Work Section (See 12.1) suggests ways in which one could pursue a more rigorous study using the existing tools I have created.

A caveat I would suggest to others doing a similar project: I could not find much in the way of background/related work that has done what I have done. Thus, I took a risk in pursuing this research, and I feel I have provided important groundwork in the study of cognitive load, as well as providing a set of IDE extensions to do so, remotely.

I have learned a lot about experimental methodologies – the importance of Pilot studies, how to perform Thematic Analysis (and where it is suitable). I have also learned about responding to user feedback and my own observations in these pilot studies to produce tighter hypotheses and implement experiments to test them.

12.1 Further work

In addition to fixing the flaws/foibles identified (Section 11.5), 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?

I would love to continue pursuing this research with a view to exploring these questions, and am grateful for all the work Bret Victor, Evan Czaplicki and Lopez et. al., that have provided the catalyst that started this dissertation.

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.

13 Appendices

Appendix A

13.1 User Manual

13.1.1 README.md

```
Welcome to the elm-lang.org README!
-----------------------------------


Installation
============

See INSTALL for installation instructions


How do I run user experiments?
==============================

After installing elm, elm-server, and building the executable, and
running ./run-elm-server, navigate to one of the following URLs to
see the user experiment interface in action:

* Elm task: http://0.0.0.0:8000/edit/task/MovingBox.elm
* JS task : http://0.0.0.0:8000/_edit/task/MovingBox.js

By default, clicks that occur in the code pane are saved to this
json file:
```

`https://sweltering-fire-9141.firebaseio.com/dissertation.json`

See the section 'How do I save to my own Firebase?' for customisation

If you write your own tasks, visiting `/edit/<path/to/task>` will populate the click databse elm directory with click data as you use it, for your task.

Furthermore, appending `?p=<participant-id>` allows you to annotate the data with the particular participant performing those clicks.

If you host the elm environment on a server on the Internet, you can send the link to your tasks to participants with a separate

Example:

`http://my.server.com:8000/edit/task/MyTask.elm?p=42`

Send that to participant 42 and the firebase database will save to

`https://sweltering-fire-9141-firebaseio.com/dissertation/elm/42.json`

How do I visualise the click data?
==================================

Navigate to the root directory where the elm-lang.org was installed

`$ cd elm-lang.org/`

then open `EmbedMe.elm` with your favourite text editor, and uncomment the line:

`-- main = lift2 scene Window.dimensions Mouse.position`

by removing the `-- ` (`--` is a comment in Elm)

How do I save to my own Firebase (or other Database)
==

Currently, EmbedElm only supports databases with a
RESTful API.

If you have a databse with a RESTful API, do as follows:

As in the section "How do I visualise the click data?", open
`EmbedMe.elm` and change
`https://sweltering-fire-9141-firebaseio.com/` to your own
personal database.

You must also modify the same URL in
`elm-lang.org/server/Server.hs`. Go to the function `embedee`
and where the variable `visualiser` is defined:

```
visualiser =
  concat [ "var firebaseData = new Firebase('"
         , "http://sweltering-fire-9141.firebaseio.com/"
 , "dissertation/
 , langStr
        ...
```

and change the URL to your own.

How do I add support for my own languages?
==

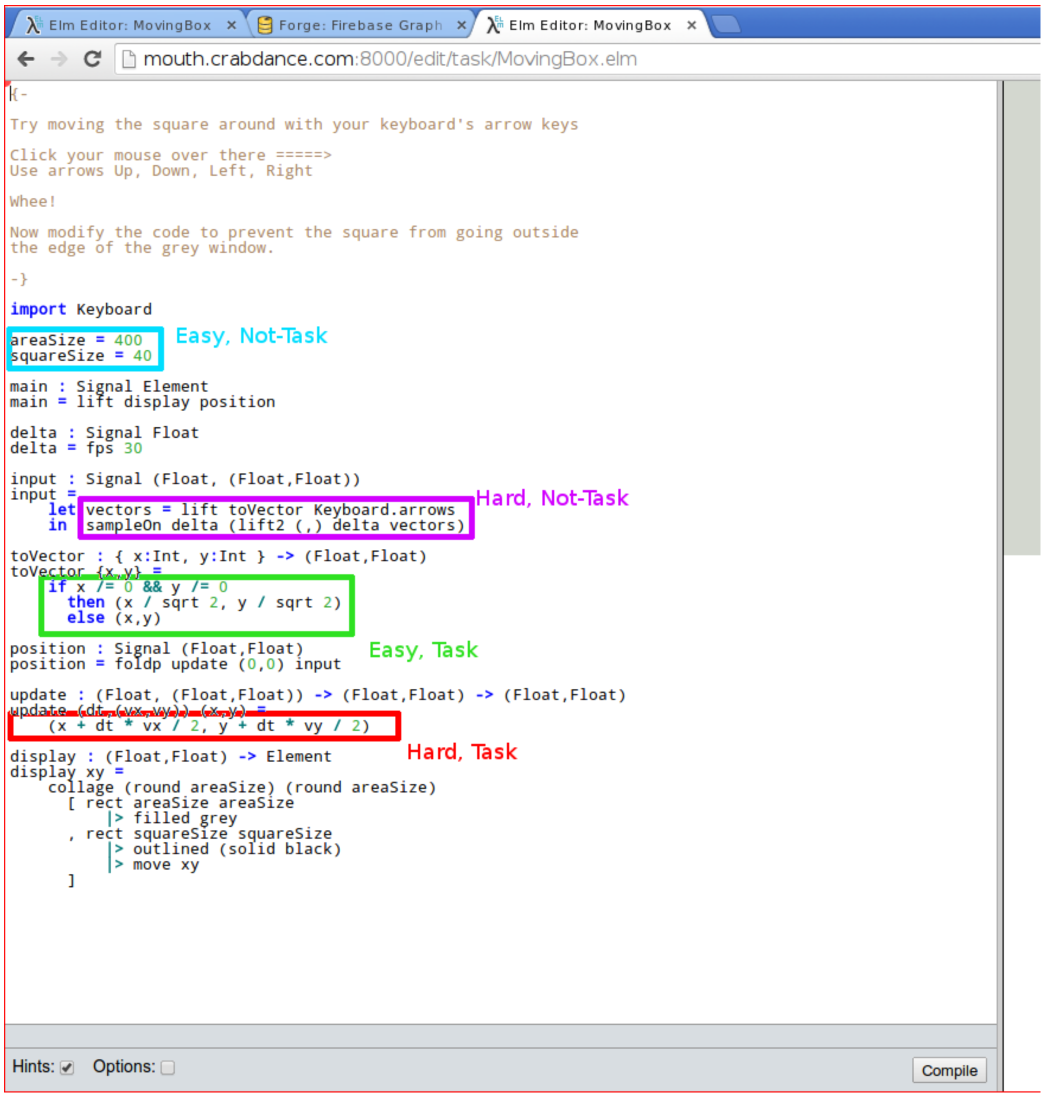
Currently, the IDE only supports Elm and Javascript, but if you are
feeling adventurous, and would like to add support for more, you will
need to modify at least the following code:

* The `data Lang` type constructor

```
* All functions with Lang in the type signature, e.g.

   `embedee :: Lang -> String -> String -> H.Html`

* The URL routing in `elm-lang.org/server/Server.hs`, e.g.

   ~~~~~~~~~~{.haskell}
   main = do
      ...
      <|> route [ ("try", serveHtml Editor.empty)
                , ("edit", edit Elm)
         , ("_edit", edit Javascript)
         , ("__edit", edit MyLanguage)
         , ("code", code Elm)
         , ("_code", code Javascript)
         , ("__code", code MyLanguage)
         , ("compile", compile Elm)
         , ("_compile", compile Javascript)
         , ("__compile", compile MyLanguage)
   ...
   ~~~~~~~~~~~~~~~~~~~~
```

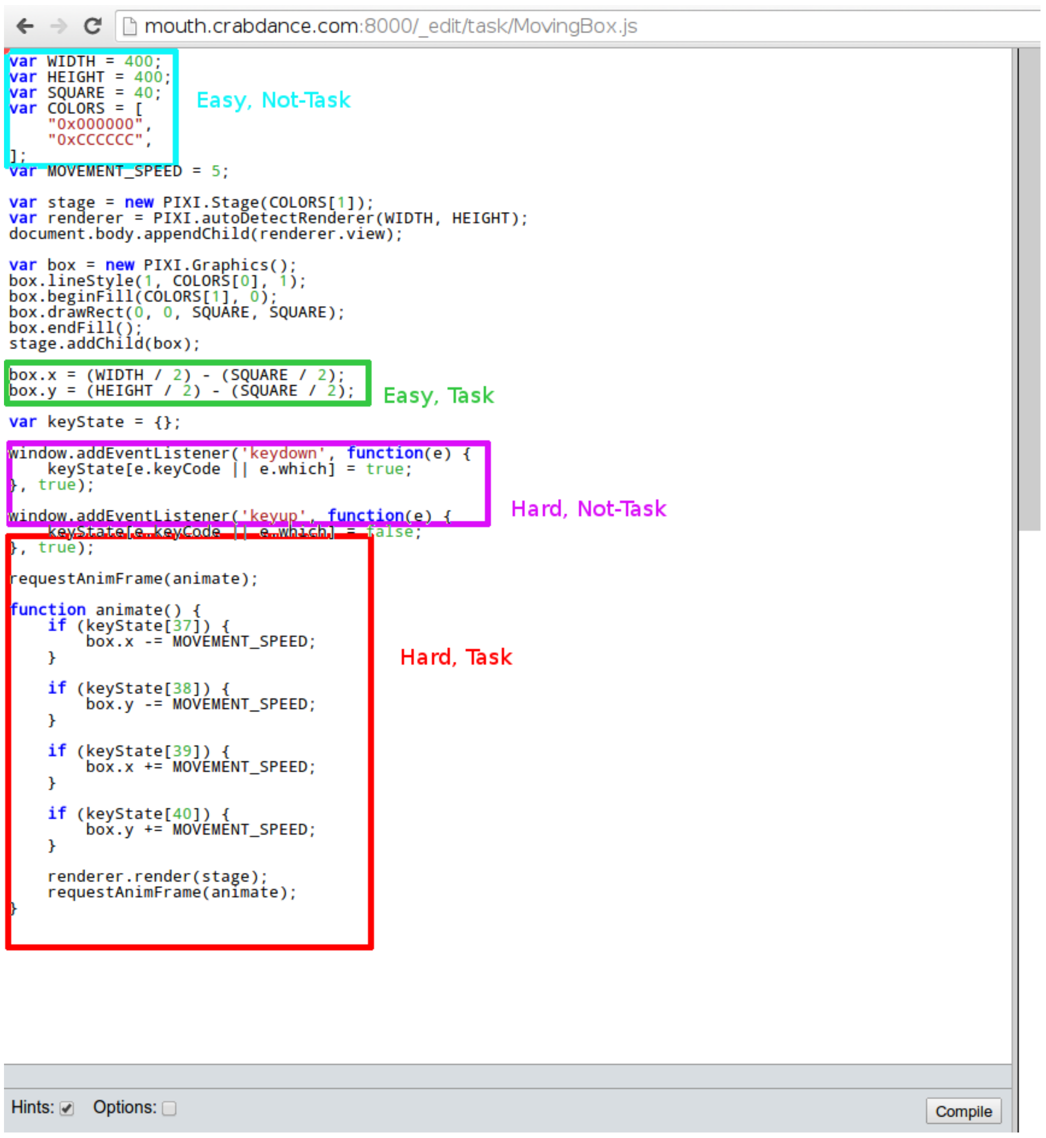
13.1.2 INSTALL

```
Installing elm-lang.org

$ cabal install --bindir=.
$ ./run-elm-server
```



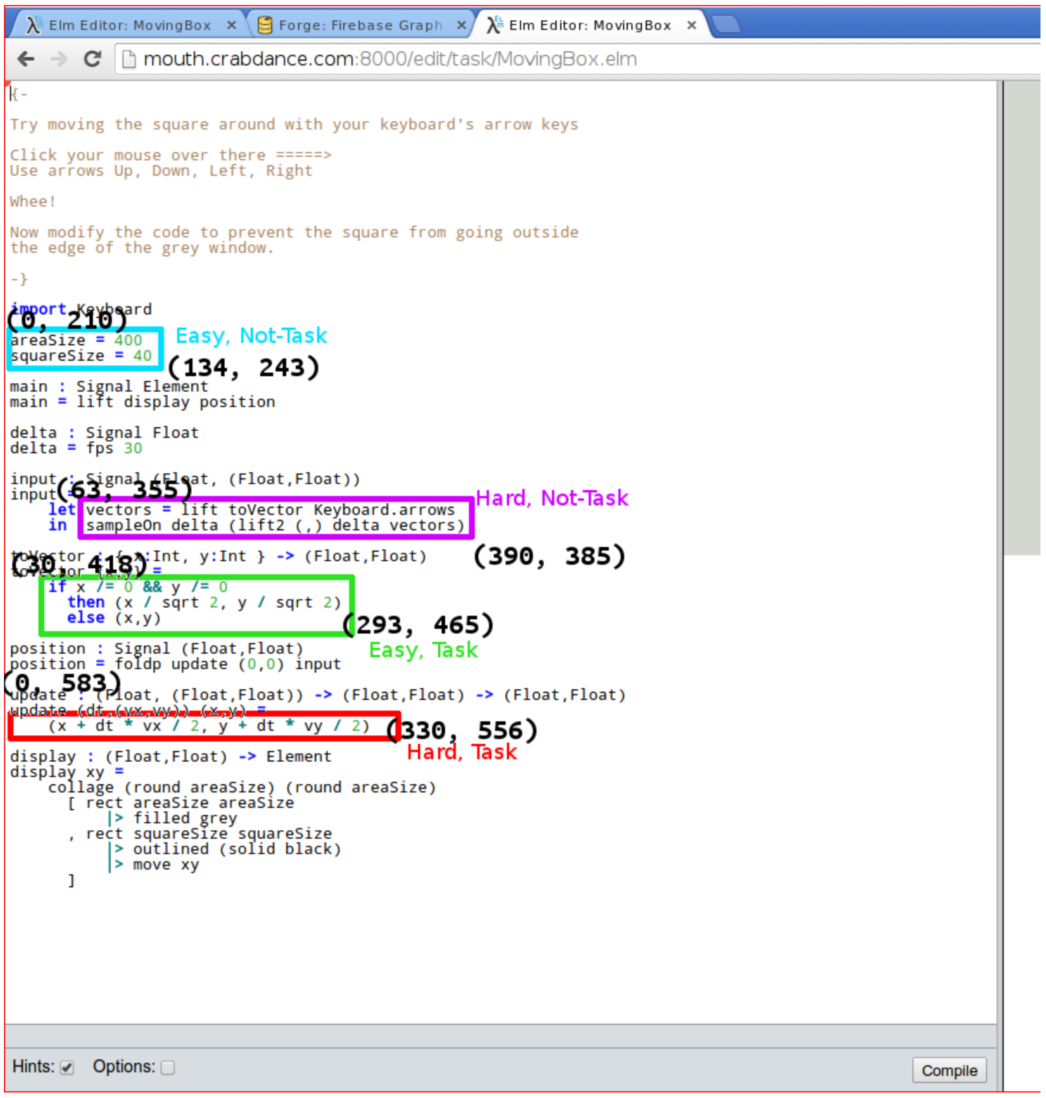
```
{-
Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

-}

import Keyboard

areaSize = 400
squareSize = 40

main : Signal Element
main = lift display position

delta : Signal Float
delta = fps 30

input : Signal (Float, (Float,Float))
input =
    let vectors = lift toVector Keyboard.arrows
    in  sampleOn delta (lift2 (,) delta vectors)

toVector : { x:Int, y:Int } -> (Float,Float)
toVector {x,y} =
    if x /= 0 && y /= 0
      then (x / sqrt 2, y / sqrt 2)
      else (x,y)

position : Signal (Float,Float)
position = foldp update (0,0) input

update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
update (dt,(vx,vy)) (x,y) =
    (x + dt * vx / 2, y + dt * vy / 2)

display : (Float,Float) -> Element
display xy =
    collage (round areaSize) (round areaSize)
      [ rect areaSize areaSize
          |> filled grey
      , rect squareSize squareSize
          |> outlined (solid black)
          |> move xy
      ]
```

Figure 6: Elm click regions



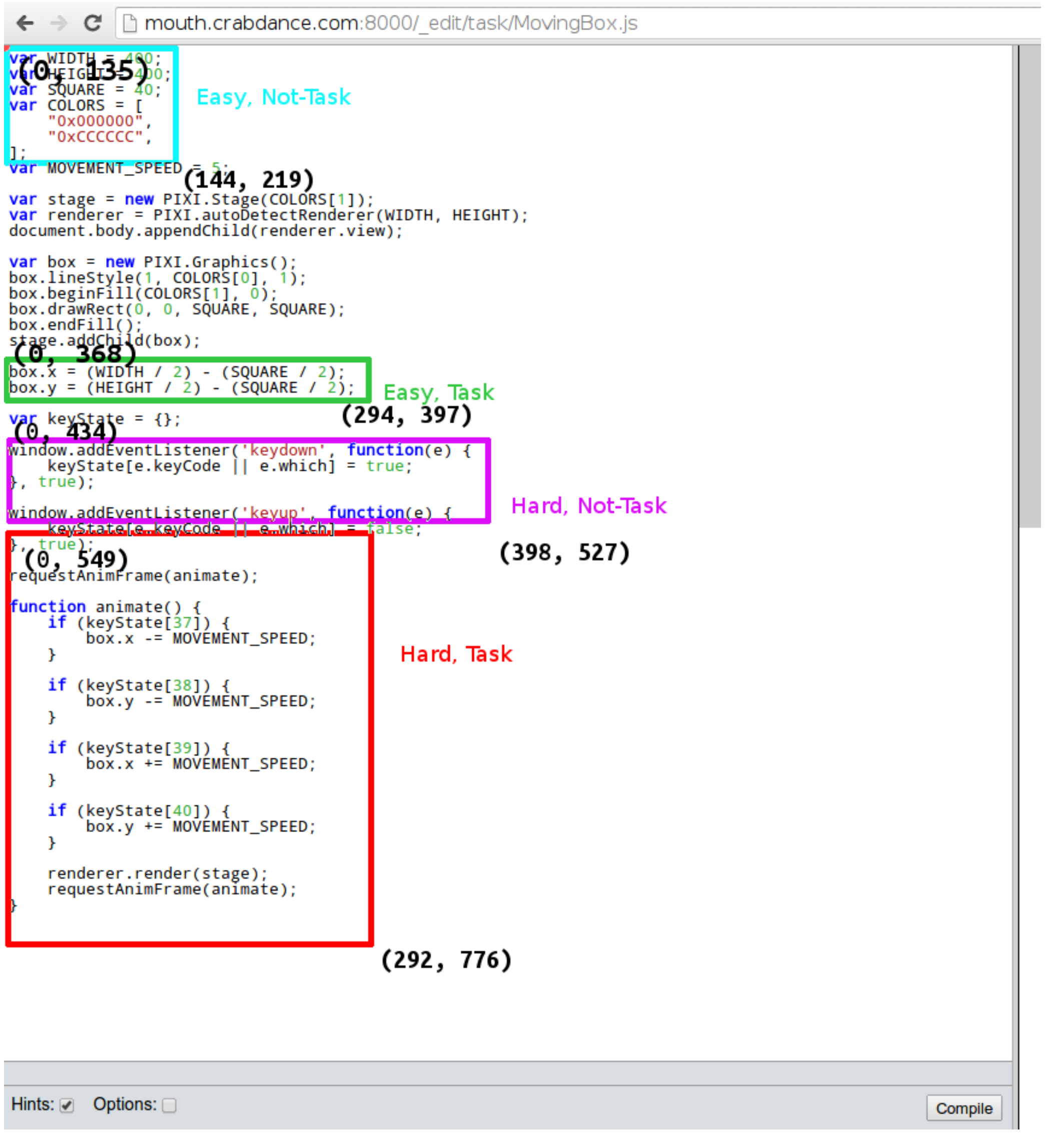
```
var WIDTH = 400;
var HEIGHT = 400;
var SQUARE = 40;
var COLORS = [
    "0x000000",
    "0xCCCCCC",
];
var MOVEMENT_SPEED = 5;

var stage = new PIXI.Stage(COLORS[1]);
var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
document.body.appendChild(renderer.view);

var box = new PIXI.Graphics();
box.lineStyle(1, COLORS[0], 1);
box.beginFill(COLORS[1], 0);
box.drawRect(0, 0, SQUARE, SQUARE);
box.endFill();
stage.addChild(box);

box.x = (WIDTH / 2) - (SQUARE / 2);
box.y = (HEIGHT / 2) - (SQUARE / 2);

var keyState = {};

window.addEventListener('keydown', function(e) {
    keyState[e.keyCode || e.which] = true;
}, true);

window.addEventListener('keyup', function(e) {
    keyState[e.keyCode || e.which] = false;
}, true);

requestAnimFrame(animate);

function animate() {
    if (keyState[37]) {
        box.x -= MOVEMENT_SPEED;
    }

    if (keyState[38]) {
        box.y -= MOVEMENT_SPEED;
    }

    if (keyState[39]) {
        box.x += MOVEMENT_SPEED;
    }

    if (keyState[40]) {
        box.y += MOVEMENT_SPEED;
    }

    renderer.render(stage);
    requestAnimFrame(animate);
}
```

Figure 7: JS click regions



```
{-

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

-}

import Keyboard

areaSize = 400
squareSize = 40

main : Signal Element
main = lift display position

delta : Signal Float
delta = fps 30

input : Signal (Float, (Float,Float))
input =
    let vectors = lift toVector Keyboard.arrows
    in  sampleOn delta (lift2 (,) delta vectors)

toVector : { x:Int, y:Int } -> (Float,Float)
toVector {x,y} =
    if x /= 0 && y /= 0
      then (x / sqrt 2, y / sqrt 2)
      else (x,y)

position : Signal (Float,Float)
position = foldp update (0,0) input

update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
update (dt,(vx,vy)) (x,y) =
    (x + dt * vx / 2, y + dt * vy / 2)

display : (Float,Float) -> Element
display xy =
    collage (round areaSize) (round areaSize)
      [ rect areaSize areaSize
          |> filled grey
      , rect squareSize squareSize
          |> outlined (solid black)
          |> move xy
      ]
```

Figure 8: Elm labelled regions



```
var WIDTH = 400;
var HEIGHT = 400;
var SQUARE = 40;
var COLORS = [
    "0x000000",
    "0xCCCCCC",
];
var MOVEMENT_SPEED = 5;

var stage = new PIXI.Stage(COLORS[1]);
var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
document.body.appendChild(renderer.view);

var box = new PIXI.Graphics();
box.lineStyle(1, COLORS[0], 1);
box.beginFill(COLORS[1], 0);
box.drawRect(0, 0, SQUARE, SQUARE);
box.endFill();
stage.addChild(box);

box.x = (WIDTH / 2) - (SQUARE / 2);
box.y = (HEIGHT / 2) - (SQUARE / 2);

var keyState = {};

window.addEventListener('keydown', function(e) {
    keyState[e.keyCode || e.which] = true;
}, true);

window.addEventListener('keyup', function(e) {
    keyState[e.keyCode || e.which] = false;
}, true);

requestAnimFrame(animate);

function animate() {
    if (keyState[37]) {
        box.x -= MOVEMENT_SPEED;
    }

    if (keyState[38]) {
        box.y -= MOVEMENT_SPEED;
    }

    if (keyState[39]) {
        box.x += MOVEMENT_SPEED;
    }

    if (keyState[40]) {
        box.y += MOVEMENT_SPEED;
    }

    renderer.render(stage);
    requestAnimFrame(animate);
}
```

Figure 9: JS labelled click regions

13.2 Design Diagrams

Appendix B

13.3 Raw results output

13.3.1 example-of-mouseclick-data.json

```
{
  "js" : {
    "12" : {
      "-JKVwXgnfg4POT9MArjy" : {
        "t" : 1397490853983,
        "y" : 0,
        "x" : 0
      },
      "-JKVwXl8I_bLZNqxP36c" : {
        "t" : 1397490854551,
        "y" : 444,
        "x" : 417
      },
      "-JKVwXj0LD-uRKDsT2Om" : {
        "t" : 1397490854503,
        "y" : 444,
        "x" : 417
      }
    }
  },
  "elm" : {
    "33" : {
      "-JKRgqJIRPH2EG-edZb5" : {
        "t" : 1397419631953,
        "y" : 249.59999084472656,
        "x" : 48.79999923706055
      },
      "-JKRhRQi31pr9AP1p1Rr" : {
        "t" : 1397419787709,
        "y" : 294.3999938964844,
        "x" : 152.8000030517578
      },
      "-JKRffOszNGfgwNdnO_X" : {
        "t" : 1397419324585,
        "y" : 608,
        "x" : 346.3999938964844
      }
    }
  }
}
```

13.3.2 error_log.json

```
{
   "2014-04-11 21:14:32.141743994+01:00":{
      "Parse error at (line 37, column 44):",
      "unexpected 'a'",
      "expecting \"{-\", \" \" or end of input"
   },
   "2014-04-11 21:35:41.694436974+01:00":{
      "Type error on line 27, column 16 to 77:",
      "         (min (max x (-hHeight)) hHeight,",
      "          (min (max y (-hWidth)),hWidth))",
      "",
      "   Expected Type: Float",
      "     Actual Type: (Float -> Float, Float)"
   },
   "2014-04-12 00:19:14.945129550+01:00":{
      "Parse error at (line 1, column 1):",
      "unexpected \"<\"",
      "expecting reserved word 'module', reserved word 'import'
               or at least one datatype or variable definition"
   },
   "2014-04-12 00:19:21.553633974+01:00":{
      "Parse error at (line 1, column 1):",
      "unexpected \"/\"",
        "expecting reserved word 'module', reserved word 'import'
                or at least one datatype or variable definition"
    },
    "2014-04-12 00:19:27.053901481+01:00":{
        "Parse error at (line 1, column 1):",
        "unexpected \"/\"",
        "expecting reserved word 'module', reserved word 'import'
                or at least one datatype or variable definition"
    },
}
```

Appendix C

13.4 Code

- All code available here: https://github.com/spanners/elm-lang.org.
    - This is a modifed 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

13.4.1 LICENSE

```
    * 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.
```

13.4.2 install_elm.sh

```bash
#!/bin/bash

# This script installs:
#
# 1. GHC 7.6.3 (The Glasgow Haskell Compiler)
# 2. The Haskell Platform 2013.2.0.0
# 3. The Elm language
# 4. The Elm server

# Configuration
#
NUM_CPUS=8 # Eight cpus


```

```
# After this script has finished successfully,
#  you can install elm-lang.org

echo "This will take a while.. go make some tea.. and dinner"
sudo apt-get install libgl1-mesa-dev libglc-dev freeglut3-dev
sudo apt-get install libedit-dev libglw1-mesa libglw1-mesa-dev
sudo apt-get install ghc
wget http://www.haskell.org/ghc/dist/7.6.3/ghc-7.6.3-src.tar.bz2
tar xjf ghc-7.6.3-src.tar.bz2
cd ghc-7.6.3/mk
cp build.mk.sample build.mk
sed -i 's/^#BuildFlavour = quick/BuildFlavour = quick/' build.mk
cd ..
./configure
make -j $NUM_PROCS
sudo make install
cd
platform="2013.2.0.0/haskell-platform-2013.2.0.0.tar.gz"
wget "http://lambda.haskell.org/platform/download/"$platform
tar xzvf haskell-platform-2013.2.0.0.tar.gz
cd haskell-platform-2013.2.0.0
./configure
make
sudo make install
cabal update
cabal install cabal-install
cabal install elm
cabal install elm-server
exit 0
```

13.4.3 Server.hs

```
{-# OPTIONS_GHC -W #-}
{-# LANGUAGE DeriveDataTypeable #-}
{-# LANGUAGE OverloadedStrings  #-}
module Main where

```

```haskell
import           Control.Applicative
import           Control.Monad.Error
import qualified Data.ByteString                as BS
import qualified Data.ByteString.Char8          as BSC
import qualified Data.HashMap.Strict            as Map
import           Data.Maybe                     (fromMaybe)
import qualified Elm.Internal.Utils             as Elm

import qualified Text.Blaze.Html.Renderer.Utf8 as BlazeBS
import           Text.Blaze.Html5               ((!))
import qualified Text.Blaze.Html5               as H
import qualified Text.Blaze.Html5.Attributes    as A
import           Text.Regex

import           GHC.Conc
import           Snap.Core
import           Snap.Http.Server
import           Snap.Util.FileServe
import           System.Console.CmdArgs
import           System.Directory
import           System.FilePath                as FP
import           System.Process

import qualified Editor
import qualified Elm.Internal.Paths             as Elm
import qualified Generate
import           Utils                          (Lang (..))

data Flags = Flags
  { port :: Int
  } deriving (Data,Typeable,Show,Eq)

flags :: Flags
flags = Flags
  { port = 8000 &= help "set the port of the server"
  }

-- | Set up the server.
```

```haskell
main :: IO ()
main = do
  setNumCapabilities =<< getNumProcessors
  putStrLn "Initializing Server"
  getRuntimeAndDocs
  setupLogging
  precompile
  cargs <- cmdArgs flags
  httpServe (setPort (port cargs) defaultConfig) $
      ifTop (serveElm "public/Empty.elm")
      <|> route [ ("try", serveHtml Editor.empty)
                , ("edit", edit Elm)
                , ("_edit", edit Javascript)
                , ("code", code Elm)
                , ("_code", code Javascript)
                , ("compile", compile Elm)
                , ("_compile", compile Javascript)
                , ("hotswap", hotswap)
                ]
      <|> serveDirectoryWith directoryConfig "public/build"
      <|> serveDirectoryWith simpleDirectoryConfig "resources"
      <|> error404

error404 :: Snap ()
error404 =
    do modifyResponse $ setResponseStatus 404 "Not found"
       serveElm "public/build/Error404.elm"

serveElm :: FilePath -> Snap ()
serveElm = serveFileAs "text/html; charset=UTF-8"

logAndServeJS :: MonadSnap m => H.Html -> m ()
logAndServeJS = serveHtml

logAndServeHtml :: MonadSnap m => (H.Html, Maybe String) -> m ()
logAndServeHtml (html, Nothing)  = serveHtml html
logAndServeHtml (html, Just err) =
    do timeStamp <- liftIO $ readProcess "date" ["--rfc-3339=ns"] ""
```

```
      liftIO $ appendFile "error_log.json" $ "{\"" ++ init timeStamp
                                                   ++ "\","
                                                   ++ show (lines err)
                                                   ++ "},"
      setContentType "text/html" <$> getResponse
      writeLBS (BlazeBS.renderHtml html)

embedHtml :: MonadSnap m => H.Html -> Lang -> String -> m ()
embedHtml html lang participant =
    do elmSrc <- liftIO $ readFile "EmbedMe.elm"
       setContentType "text/html" <$> getResponse
       writeLBS (BlazeBS.renderHtml (embedMe lang elmSrc html participant))

serveHtml :: MonadSnap m => H.Html -> m ()
serveHtml html =
    do setContentType "text/html" <$> getResponse
       writeLBS (BlazeBS.renderHtml html)

hotswap :: Snap ()
hotswap = maybe error404 serve =<< getParam "input"
    where
      serve code =
          do setContentType "application/javascript" <$> getResponse
             writeBS . BSC.pack . Generate.js $ BSC.unpack code

compile :: Lang -> Snap ()
compile lang = maybe error404 serve =<< getParam "input"
    where
      serve = case lang of
                Elm -> logAndServeHtml
                        . Generate.logAndHtml "Compiled Elm"
                        . BSC.unpack
                Javascript -> logAndServeJS
                               . Generate.logAndJS "Compiled JS"
                               . BSC.unpack

edit :: Lang -> Snap ()
edit lang = do
```

```
  participant <- BSC.unpack . fromMaybe "" <$> getParam "p"
  cols <- BSC.unpack . fromMaybe "50%,50%" <$> getQueryParam "cols"
  withFile (Editor.ide lang cols participant)

code :: Lang -> Snap ()
code lang = do
  participant <- BSC.unpack . fromMaybe "" <$> getParam "p"
  embedWithFile Editor.editor lang participant

embedee :: Lang -> String -> String -> H.Html
embedee lang elmSrc participant =
    H.span $ do
      case Elm.compile elmSrc of
        Right jsSrc ->
            jsAttr $ H.preEscapedToMarkup (subRegex oldID jsSrc newID)
        Left err ->
            H.span ! A.style "font-family: monospace;" $
            mapM_ (\line ->
                      H.preEscapedToMarkup
                      (Generate.addSpaces line)
                      >> H.br)
                  (lines err)
      jsAttr $ H.preEscapedToMarkup $ visualiser
      where langStr = (case lang of
                            Elm -> "elm"
                            Javascript -> "js")
            visualiser =
              concat [ "var firebaseData = new Firebase('"
                      , "http://sweltering-fire-9141.firebaseio.com/"
                      , "dissertation/"
                      , langStr
                      , "/"
                      , participant
                      , "');"
                      , "var elm = Elm.fullscreen(Elm.EmbedMe, {"
                          , "stamped: {"
                              , "      t: 0,"
                              , "      x: 0,"
```

```
                                    , "      y: 0"
                                , "  }"
                            , "});"
                            , "firebaseData.on('child_added',"
                            , "function(snapshot) {"
                                , "elm.ports.stamped.send(snapshot.val());"
                            , "});" ]
             oldID = mkRegex "var user_id = \"1\";"
             newID = "var user_id = \"" ++ langStr
                                        ++ "/"
                                        ++ participant
                                        ++ "\";"
             jsAttr = H.script ! A.type_ "text/javascript"

embedMe :: Lang -> String -> H.Html -> String -> H.Html
embedMe lang elmSrc target participant = target >> embedee
                                                     lang
                                                     elmSrc
                                                     participant

embedWithFile :: (Lang -> FilePath -> String -> H.Html) -> Lang
                                                         -> String
                                                         -> Snap ()
embedWithFile handler lang participant = do
  path <- BSC.unpack . rqPathInfo <$> getRequest
  let file = "public/" ++ path
  exists <- liftIO (doesFileExist file)
  if not exists then error404 else
      do content <- liftIO $ readFile file
         embedHtml (handler lang path content) lang participant

withFile :: (FilePath -> String -> H.Html) -> Snap ()
withFile handler = do
  path <- BSC.unpack . rqPathInfo <$> getRequest
  let file = "public/" ++ path
  exists <- liftIO (doesFileExist file)
  if not exists then error404 else
      do content <- liftIO $ readFile file
```

```haskell
         serveHtml $ handler path content

directoryConfig :: MonadSnap m => DirectoryConfig m
directoryConfig =
    fancyDirectoryConfig
    { indexGenerator = defaultIndexGenerator
                         defaultMimeTypes
                         indexStyle
    , mimeTypes = Map.insert ".elm" "text/html" defaultMimeTypes
    }

indexStyle :: BS.ByteString
indexStyle =
    "body { margin:0; font-family:sans-serif; \
    \       background:rgb(245,245,245);\
    \       font-family: calibri, verdana, helvetica, arial; }\
    \div.header { padding: 40px 50px; font-size: 24px; }\
    \div.content { padding: 0 40px }\
    \div.footer { display:none; }\
    \table { width:100%; border-collapse:collapse; }\
    \td { padding: 6px 10px; }\
    \tr:nth-child(odd) { background:rgb(216,221,225); }\
    \td { font-family:monospace }\
    \th { background:rgb(90,99,120); color:white; text-align:left;\
    \     padding:10px; font-weight:normal; }"

setupLogging :: IO ()
setupLogging =
    do createDirectoryIfMissing True "log"
       createIfMissing "log/access.log"
       createIfMissing "log/error.log"
    where
      createIfMissing path = do
        exists <- doesFileExist path
        unless exists $ BS.writeFile path ""

-- | Compile all of the Elm files in public/,
--   placing results in public/build/
```

```haskell
precompile :: IO ()
precompile =
  do setCurrentDirectory "public"
     files <- getFiles True ".elm" "."
     forM_ files $ \file ->
                    rawSystem "elm" [ "--make"
                                    , "--runtime=/elm-runtime.js"
                                    , file ]
     htmls <- getFiles False ".html" "build"
     mapM_ adjustHtmlFile htmls
     setCurrentDirectory ".."
  where
    getFiles :: Bool -> String -> FilePath -> IO [FilePath]
    getFiles skip ext directory =
        if skip && "build" `elem` map FP.dropTrailingPathSeparator
                                      (FP.splitPath directory)
          then return [] else
          (do contents <- map (directory </>) `fmap`
                                               getDirectoryContents
                                               directory
              let files = filter ((ext==) . FP.takeExtension) contents
                  directories  = filter (not . FP.hasExtension) contents
              filess <- mapM (getFiles skip ext) directories
              return (files ++ concat filess))

getRuntimeAndDocs :: IO ()
getRuntimeAndDocs = do
  writeFile "resources/elm-runtime.js" =<< readFile Elm.runtime
  writeFile "resources/docs.json" =<< readFile Elm.docs

adjustHtmlFile :: FilePath -> IO ()
adjustHtmlFile file =
  do src <- BSC.readFile file
     let (before, after) = BSC.breakSubstring "<title>" src
     BSC.writeFile (FP.replaceExtension file "elm") $
        BSC.concat [before, style, after]
     removeFile file

```

```haskell
style :: BSC.ByteString
style =
    "<style type=\"text/css\">\n\
    \  a:link {text-decoration: none; color: rgb(15,102,230);}\n\
    \  a:visited {text-decoration: none}\n\
    \  a:active {text-decoration: none}\n\
    \  a:hover {text-decoration: underline; color: rgb(234,21,122);}\n\
    \  body { font-family: \"Lucida Grande\",\
    \ \"Trebuchet MS\",\"Bitstream Vera Sans\",Verdana,Helvetica,\
    \ sans-serif !important; }\n\
    \  p, li { font-size: 14px !important;\n\
    \          line-height: 1.5em !important; }\n\
    \</style>"
```

13.4.4 Editor.hs

```haskell
{-# LANGUAGE OverloadedStrings #-}
module Editor (editor,ide,empty) where

import           Data.Monoid                 (mempty)
import           Network.HTTP.Base           (urlEncode)
import qualified System.FilePath             as FP
import           Text.Blaze.Html
import qualified Text.Blaze.Html5            as H
import qualified Text.Blaze.Html5.Attributes as A

import           Data.Maybe                  (fromMaybe)
import qualified Elm.Internal.Utils          as Elm

import           Generate                    (addSpaces)
import           Utils


-- | Display an editor and the compiled result side-by-side.
ide :: Lang -> String -> String -> FilePath -> String -> Html
ide lang cols participant fileName code =
    case lang of
```

```haskell
         Javascript -> buildIde ("JS Editor: ", "/_compile")
         Elm        -> buildIde ("Elm Editor: ", "/compile")
  where buildIde (editStr,compileStr) =
            ideBuilder lang
                       cols
                       participant
                       (editStr ++ FP.takeBaseName fileName)
                       fileName
                       (compileStr ++"?input=" ++ urlEncode code)


-- | Display an editor and the compiled result side-by-side.
empty :: Html
empty = ideBuilder Elm "50%,50%" "1" "Try Elm" "Empty.elm" "/Try.elm"

ideBuilder :: Lang -> String
                   -> String
                   -> String
                   -> String
                   -> String
                   -> Html
ideBuilder lang cols participant title input output =
    case lang of
         Javascript -> makeIde "_code/"
         Elm        -> makeIde "code/"
  where
        makeIde codeStr =
          H.docTypeHtml $ do
            H.head . H.title . toHtml $ title
            preEscapedToMarkup $
               concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
                      , "  <frame name=\"input\" src=\"/", codeStr,
                        input, "?p=",
                        participant, "\" />\n"
                      , "  <frame name=\"output\" src=\"", output, "\" />\n"
                      , "</frameset>" ]


```

```haskell
-- | list of themes to use with CodeMirror
themes :: [String]
themes = [ "ambiance", "blackboard", "cobalt", "eclipse"
         , "elegant", "erlang-dark", "lesser-dark", "monokai"
         , "neat", "night", "rubyblue", "solarized", "twilight"
         , "vibrant-ink", "xq-dark" ]

jsFiles :: AttributeValue -> [AttributeValue]
jsFiles syntaxFile =
          [ "/codemirror-3.x/lib/codemirror.js"
          , syntaxFile
          , "/misc/showdown.js"
          , "/misc/editor.js?0.11" ]


-- | Create an HTML document that allows you to edit and submit Elm code
--   for compilation.
editor :: Lang -> FilePath -> String -> Html
editor lang filePath code =
    case lang of
         Javascript -> buildEditor ( "JS Editor: "
                                   , "/codemirror-3.x\
                                   \/mode/javascript/javascript.js"
                                   , "/_compile")
         Elm -> buildEditor ( "Elm Editor: "
                            , "/codemirror-3.x/mode/elm/elm.js"
                            , "/compile")
  where buildEditor (editStr, syntaxFile, compileStr) =
          H.html $ do
            H.head $ do
              H.title . toHtml $ editStr ++ FP.takeBaseName filePath
              H.link ! A.rel "stylesheet"
                     ! A.href "/codemirror-3.x/lib/codemirror.css"
              mapM_ themeAttr themes
              H.link ! A.rel "stylesheet" ! A.type_ "text/css"
                                          ! A.href "/misc/editor.css"
              mapM_ script $ jsFiles syntaxFile
              script "/elm-runtime.js?0.11"
              script "http://cdn.firebase.com/v0/firebase.js"
            H.body $ do
              H.form ! A.id "inputForm"
                     ! A.action compileStr
                     ! A.method "post"
                     ! A.target "output" $ do
                 H.div ! A.id "editor_box" $
                   H.textarea ! A.name "input"
                              ! A.id "input" $ toHtml ('\n':code)
                 H.div ! A.id "options" $ do
                   bar "documentation" docs
                   bar "editor_options" editorOptions
                   bar "always_on" (buttons >> options)
              jsAttr "initEditor();"
        themeAttr theme = H.link ! A.rel "stylesheet"
                                 ! A.href (toValue
                                            ("/codemirror-3.x/theme/"
                                                ++ theme
                                                ++ ".css" :: String))
        jsAttr = H.script ! A.type_ "text/javascript"
        script jsFile = jsAttr ! A.src jsFile $ mempty

bar :: AttributeValue -> Html -> Html
bar id' body = H.div ! A.id id' ! A.class_ "option" $ body

buttons :: Html
buttons = H.div ! A.class_ "valign_kids"
                ! A.style "float:right; padding-right: 6px;"
                $ compileButton
      where
        compileButton =
            H.input
                 ! A.type_ "button"
                 ! A.id "compile_button"
                 ! A.value "Compile"
                 ! A.onclick "compile()"
                 ! A.title "Ctrl-Enter: change program behavior \
                             \but keep the state"
```

```haskell

options :: Html
options = H.div ! A.class_ "valign_kids"
                ! A.style "float:left; padding-left:6px; padding-top:2px;"
                $ (docs' >> opts)
    where
      docs' =
        H.span  ! A.title "Show documentation and types." $ "Hints:" >>
            H.input ! A.type_ "checkbox"
                    ! A.id "show_type_checkbox"
                    ! A.onchange "showType(this.checked);"

      opts =
        H.span  ! A.title "Show editor options."
                ! A.style "padding-left: 12px;" $ "Options:" >>
            H.input ! A.type_ "checkbox"
                    ! A.id "options_checkbox"
                    ! A.onchange "showOptions(this.checked);"

editorOptions :: Html
editorOptions = theme >> zoom >> lineNumbers
    where
      optionFor :: String -> Html
      optionFor text =
          H.option ! A.value (toValue text) $ toHtml text

      theme =
          H.select ! A.id "editor_theme"
                   ! A.onchange "setTheme(this.value)"
                   $ mapM_ optionFor themes

      zoom =
          H.select ! A.id "editor_zoom"
                   ! A.onchange
                       "setZoom(this.options[this.selectedIndex].\
                       \innerHTML)"
                   $ mapM_ optionFor ["100%", "80%", "150%", "200%"]

      lineNumbers = do
        H.span ! A.style "padding-left: 16px;" $ "Line Numbers:"
        H.input ! A.type_ "checkbox"
                ! A.id "editor_lines"
                ! A.onchange "showLines(this.checked);"

docs :: Html
docs = tipe >> desc
    where
      tipe = H.div ! A.class_ "type" $ message >> more

      message = H.div !
                  A.style "position:absolute; left:4px; right:36px;\
                            \overflow:hidden; text-overflow:ellipsis;" $ ""

      more = H.a ! A.id "toggle_link"
                 ! A.style "display:none; float:right;"
                 ! A.href "javascript:toggleVerbose();"
                 ! A.title "Ctrl+H"
                 $ ""

      desc = H.div ! A.class_ "doc"
                   ! A.style "display:none;"
                   $ ""

```

13.4.5 Generate.hs

```haskell
{-# LANGUAGE OverloadedStrings #-}
module Generate (logAndJS, logAndHtml, html, js, addSpaces) where

import           Data.Maybe                  (fromMaybe)
import           Data.Monoid                 (mempty)
import           Text.Blaze                  (preEscapedToMarkup)
import           Text.Blaze.Html5            ((!))
import qualified Text.Blaze.Html5            as H
import qualified Text.Blaze.Html5.Attributes as A

import qualified Elm.Internal.Utils          as Elm
import           Utils

logAndJS :: String -> String -> H.Html
logAndJS name src = getJSPage name src

logAndHtml :: String -> String -> (H.Html, Maybe String)
logAndHtml name src =
    let elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
    in
      case Elm.compile src of
          Right jsSrc -> do
              (getHtmlPage name elmname jsSrc, Nothing)
          Left err -> do
              (getErrPage name err, Just err)

getJSPage :: String -> String -> H.Html
getJSPage name jsSrc =
  H.docTypeHtml $ do
      H.head $ do
        H.meta ! A.charset "UTF-8"
        H.title . H.toHtml $ name
        H.link ! A.rel "stylesheet" ! A.type_ "text/css"
                                    ! A.href "/misc/js.css"
        script "/pixi.js"
      H.body $ do
        H.div ! A.style "width: 400px; height: 400px; position:\
                        \ absolute; top: 0; left: 0; opacity: 0;" $ mempty
        jsAttr $ preEscapedToMarkup jsSrc
 where jsAttr = H.script ! A.type_ "text/javascript"
       script jsFile = jsAttr ! A.src jsFile $ mempty
       embed jsCode = jsAttr $ jsCode

getHtmlPage :: String -> String -> String -> H.Html
getHtmlPage name elmname jsSrc =
  H.docTypeHtml $ do
      H.head $ do
```

```haskell
        H.meta ! A.charset "UTF-8"
        H.title . H.toHtml $ name
        H.style ! A.type_ "text/css" $ preEscapedToMarkup
         ("a:link {text-decoration: none; color: rgb(15,102,230);}\n\
          \a:visited {text-decoration: none}\n\
          \a:active {text-decoration: none}\n\
          \a:hover {text-decoration: underline; \
          \color: rgb(234,21,122);}" :: String)
      H.body $ do
        let js = H.script ! A.type_ "text/javascript"
            runFullscreen =
                "var runningElmModule = Elm.fullscreen(" ++ elmname
                                                         ++ ")"
        js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
        js $ preEscapedToMarkup jsSrc
        js $ preEscapedToMarkup runFullscreen

getErrPage :: String -> String -> H.Html
getErrPage name err =
  H.docTypeHtml $ do
      H.head $ do
        H.meta ! A.charset "UTF-8"
        H.title . H.toHtml $ name
      H.body $
        H.span ! A.style "font-family: monospace;" $
        mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
                (lines err)



-- | Using a page title and full source of an Elm program, compile down
--   to a valid HTML document.
html :: String -> String -> H.Html
html name src =
  H.docTypeHtml $ do
      H.head $ do
        H.meta ! A.charset "UTF-8"
        H.title . H.toHtml $ name
        H.style ! A.type_ "text/css" $ preEscapedToMarkup
         ("a:link {text-decoration: none; color: rgb(15,102,230);}\n\
          \a:visited {text-decoration: none}\n\
          \a:active {text-decoration: none}\n\
          \a:hover {text-decoration: underline;\
          \ color: rgb(234,21,122);}" :: String)
      H.body $ do
        let js = H.script ! A.type_ "text/javascript"
            elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
            runFullscreen =
                  "var runningElmModule = Elm.fullscreen(" ++ elmname
                                                           ++ ")"
        js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
        case Elm.compile src of
          Right jsSrc -> do
              js $ preEscapedToMarkup jsSrc
              js $ preEscapedToMarkup runFullscreen
          Left err ->
              H.span ! A.style "font-family: monospace;" $
              mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
                      (lines err)

addSpaces :: String -> String
addSpaces str =
  case str of
    ' ' : ' ' : rest -> " &nbsp;" ++ addSpaces rest
    c : rest -> c : addSpaces rest
    [] -> []

js :: String -> String
js src = case Elm.compile src of
           Right js -> "{ \"success\" : " ++ show js ++ " }"
           Left err -> "{ \"error\" : " ++ show err ++ " }"
```

13.4.6 EmbedMe.elm

```
module EmbedMe where

import Graphics.Input as Input
import JavaScript as JS
import JavaScript.Experimental as JEXP
import Http
import Json
import Mouse
import Dict
import Window


(~>) = flip lift
infixl 4 ~>

type Stamp = { t: Float, x: Float, y: Float }

-- Incoming

port stamped : Signal { t: Float, x: Float, y: Float }

stamps : Signal [Stamp]
stamps = foldp (::) [] stamped

scene (w,h) locs =
  let drawCircle {t, x, y} =
          circle 5 |> filled (hsva y 1 1 0.7)
                   |> move (x - toFloat w / 2, toFloat h / 2 - y)
  in  collage w h (map drawCircle locs)

-- Uncomment the following line in order to visualise clicks
--main = lift2 scene Window.dimensions stamps

-- Outgoing

-- Do not change user_id = "1"
-- It gets replaced with the actual user_id when compiled to JS
user_id = "1"

firebaseRequest requestType requestData =
  Http.request requestType
               ("https://sweltering-fire-9141.firebaseio.com/"
                  ++ "dissertation/"
                  ++ user_id
                  ++ ".json")
               requestData
               []

serialize r = r |> JEXP.fromRecord
                |> Json.fromJSObject
                |> Json.toJSString " "
                |> JS.toString

toRequestData (t, (x, y)) = {t = t, x = x, y = y } |> serialize

clicks = timestamp (sampleOn Mouse.isDown Mouse.position)

toRequest click = case click of
  (0, (0, 0)) -> firebaseRequest "get" ""
  _           -> firebaseRequest "post" (click |> toRequestData)

requests = clicks ~> toRequest

sendRequests = Http.send requests
```

13.4.7 editor.js.diff

```
diff --git a/resources/misc/editor.js b/resources/misc/editor.js
index d2bebc8..302663e 100644
--- a/resources/misc/editor.js
+++ b/resources/misc/editor.js
@@ -293,7 +293,7 @@ function showOptions(show) {
 function showType(show) {
     cookie('showtype', show);
     document.getElementById('show_type_checkbox').checked = show;
-    var newMode = (show ? { mode: Mode.TYPES, verbose: false }
+    var newMode = (show ? { mode: Mode.TYPES, verbose: true}
                         : { mode: Mode.NONE });
     if (mode.mode === Mode.OPTIONS) {
         mode.hidden = newMode;
@@ -305,8 +305,8 @@ function showType(show) {

 function toggleVerbose() {
     if (!mode.verbose) showType(true);
-    document.getElementById('toggle_link').innerHTML = mode.verbose ?
-      'more' : 'less';
-    mode.verbose = !mode.verbose;
+    document.getElementById('toggle_link').innerHTML = mode.verbose ? '' : '';
+    mode.verbose = true;
     updateDocumentation();
 }

@@ -318,8 +318,8 @@ function showVerbose() {
 function hideStuff() {
     if (mode.hidden) mode = mode.hidden;
     document.getElementById('options_checkbox').checked = false;
-    mode.verbose = false;
-    document.getElementById('toggle_link').innerHTML = 'more';
+    mode.verbose = true;
+    document.getElementById('toggle_link').innerHTML = '';
     updateDocumentation();
 }
```

13.4.8 MovingBox.js

```
/*

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
```

```
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

*/

var WIDTH = 400;
var HEIGHT = 400;
var SQUARE = 40;
var COLORS = [
    "0x000000",
    "0xCCCCCC",
];
var MOVEMENT_SPEED = 5;

var stage = new PIXI.Stage(COLORS[1]);
var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
document.body.appendChild(renderer.view);

var box = new PIXI.Graphics();
box.lineStyle(1, COLORS[0], 1);
box.beginFill(COLORS[1], 0);
box.drawRect(0, 0, SQUARE, SQUARE);
box.endFill();
stage.addChild(box);

box.x = (WIDTH / 2) - (SQUARE / 2);
box.y = (HEIGHT / 2) - (SQUARE / 2);

var keyState = {};

window.addEventListener('keydown', function(e) {
    keyState[e.keyCode || e.which] = true;
}, true);

```

```
window.addEventListener('keyup', function(e) {
    keyState[e.keyCode || e.which] = false;
}, true);

requestAnimFrame(animate);

function animate() {
    if (keyState[37]) {
        box.x -= MOVEMENT_SPEED;
    }

    if (keyState[38]) {
        box.y -= MOVEMENT_SPEED;
    }

    if (keyState[39]) {
        box.x += MOVEMENT_SPEED;
    }

    if (keyState[40]) {
        box.y += MOVEMENT_SPEED;
    }

    renderer.render(stage);
    requestAnimFrame(animate);
}
```

13.4.9 MovingBox.elm

```
{-

Try moving the square around with your keyboard's arrow keys

Click your mouse over there =====>
Use arrows Up, Down, Left, Right

Whee!

Now modify the code to prevent the square from going outside
the edge of the grey window.

-}

import Keyboard

areaSize = 400
squareSize = 40

main : Signal Element
main = lift display position

delta : Signal Float
delta = fps 30

input : Signal (Float, (Float,Float))
input =
    let vectors = lift toVector Keyboard.arrows
    in  sampleOn delta (lift2 (,) delta vectors)

toVector : { x:Int, y:Int } -> (Float,Float)
toVector {x,y} =
    if x /= 0 && y /= 0
      then (x / sqrt 2, y / sqrt 2)
      else (x,y)

position : Signal (Float,Float)
position = foldp update (0,0) input

update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
update (dt,(vx,vy)) (x,y) =
    (x + dt * vx / 2, y + dt * vy / 2)

display : (Float,Float) -> Element
display xy =
    collage (round areaSize) (round areaSize)
```

```
      [ rect areaSize areaSize
          |> filled grey
      , rect squareSize squareSize
          |> outlined (solid black)
          |> move xy
      ]
```

13.4.10 ClicksPerCategory.py

```python
import unittest
import json

class ClicksPerCategory(object):
    def get_clicks_in_region(self, jsonStr, x_min, y_min,
            x_max, y_max):
        region = {}
        mouse_data = json.loads(jsonStr)
        for click in mouse_data:
            coords = mouse_data[click]
            if ((x_min <= coords["x"] <= x_max) and
                    (y_min <= coords["y"] <= y_max)):
                region[click] = coords
        return region


class FooTests(unittest.TestCase):

    def setUp(self):
       self.mr = ClicksPerCategory()

    def test_get_clicks_in_region(self):
        jsonStr = '{"uniq_id1": {"t": 1, "x":1, "y":2},\
                   "uniq_id2": {"t":2, "x":3, "y":4}}'
        x_min = 0
        x_max = 4
        y_min = 3
        y_max = 4
        expected = {u"uniq_id2": {u"t":2, u"x":3, u"y":4}}
        actual = self.mr.get_clicks_in_region(jsonStr, x_min, y_min,
                x_max, y_max)
        self.assertEquals(actual, expected)

def main():
    unittest.main()

if __name__ == '__main__':
    main()

```

13.4.11 get_clicks_per_category.py

```python
import ClicksPerCategory
import sys
import json

cpc = ClicksPerCategory.ClicksPerCategory()

def get_all_clicks():
    _all = dict()
    contents = ""
    fname = ""
    region1 = (0,538, 330,556)
    for _file in sys.argv[1:]:
        with open(_file) as f:
            fname, contents = f.name, f.read()
        _all[fname] = cpc.get_clicks_in_region(contents, *region1)
    return _all

_all = get_all_clicks()
total = 0
for fname in _all:
    length = len(_all[fname])
    print fname, length
    total += length
```

```python
print "TOTAL:", total
```

13.4.12 DecodeMouseData.py

```python
#!/usr/bin/python2.7

import json
import sys

class DecodeMouseData(object):

    def decode(self, jsonString):
        return json.loads(jsonString)

    def getNumberOfClicks(self, jsonString):
        return len(self.decode(jsonString))

    def getSessionDuration(self, jsonString):
        decoded = self.decode(jsonString)

        finish = max([x["t"] for x in decoded.values()])
        start = min([x["t"] for x in decoded.values()])

        # compute number of minutes in this number of milliseconds
        return (finish - start) / 60000.0

    def getFilesTimeClickDict(self, files):
        timeAndClicks = dict()
        filename = str()
        for jsonFile in files:
            with open(jsonFile) as f:
                jsonString = f.read()
                filename = f.name
                print filename
            timeAndClicks[filename] = (self.getSessionDuration(jsonString),
                    self.getNumberOfClicks(jsonString))
        return timeAndClicks

    def getDictPrettyPrint(self, timeAndClicks):
        output = ""
        tablelines = "-"*11 + " " + "-"*10
        output += tablelines + "\n"
        output += "Time (min)  Clicks\n"
        output += tablelines + "\n"
        length = len(timeAndClicks)
        for i,filename in enumerate(timeAndClicks):
            output += ("%10f %10d" % (timeAndClicks[filename][0],
                        timeAndClicks[filename][1])) + "\n"
        output += tablelines
        return output


if __name__ == "__main__":
    dmd = DecodeMouseData()
    print dmd.getDictPrettyPrint(dmd.getFilesTimeClickDict(sys.argv[1:]))
    timeAndClicks = dmd.getFilesTimeClickDict(sys.argv[1:])
    total = 0
    for filename in timeAndClicks:
        total += timeAndClicks[filename][1]
    print "TOTAL: ", total
```

13.4.13 test_DecodeMouseData.py

```
import unittest
import DecodeMouseData as d


class FooTests(unittest.TestCase):

    def setUp(self):
        self.dmd = d.DecodeMouseData()

    def testDecode(self):
        expected = {'1':2, '3':4}
        actual = self.dmd.decode('{"1":2, "3":4}')

        self.assertEquals(actual, expected)

    def testMouseDecode(self):
        expected = {"-JKMBewWrFje3lHT8spD" :
                {"t" : 1397327310399, "y" : 646, "x" : 629}}
        actual = self.dmd.decode(
                '{"-JKMBewWrFje3lHT8spD" : ' +
                '{"t" : 1397327310399, "y" : 646, "x" : 629}}')

        self.assertEquals(actual, expected)

    def testNumClicks(self):
        expected = 1
        actual = self.dmd.getNumberOfClicks(
                '{"-JKMBewWrFje3lHT8spD" : ' +
                '{"t" : 1397327310399, "y" : 646, "x" : 629}}')

        self.assertEquals(actual, expected)

    def testLotsClicks(self):
        expected = 2

        actual = self.dmd.getNumberOfClicks("""{
  "-JKMBewWrFje3lHT8spD" : {
    "t" : 1397327310399,
    "y" : 646,
    "x" : 629
  },
  "-JKMBewawNo6G_Zdfnkk" : {
    "t" : 1397327310465,
    "y" : 646,
    "x" : 629
  }
}""")

        self.assertEquals(actual, expected)

    def testComputeSessionTime(self):
        expected = 0.0011

        actual = self.dmd.getSessionDuration("""{
  "-JKMBewWrFje3lHT8spD" : {
    "t" : 1397327310399,
    "y" : 646,
    "x" : 629
  },
  "-JKMBewawNo6G_Zdfnkk" : {
    "t" : 1397327310465,
    "y" : 646,
    "x" : 629
  }
}""")

        self.assertEquals(actual, expected)

def main():
    unittest.main()

if __name__ == '__main__':
    main()

```

13.4.14 ttest-scipy.py

```
from scipy import stats
import numpy as np

elm = [76,487,105,66]
js = [33,12,69,3]

outcome = stats.ttest_ind(elm,js, equal_var=False)

print("t-value: %4.2f, p-value: %4.2f" % (outcome[0], outcome[1]))
```

```

observed = elm + js

expected = sum(observed)/(len(observed)*1.0)

chi_squared = sum([((x - expected)**2)/expected for x in observed])

print("chi-squared : %4.2f" % chi_squared)
```

Appendix D

13.5 Pilot Study 1

13.5.1 Consent Form

13.5.1.1 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

13.5.2 Pre-questionnaire

13.5.2.1 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:
   - ....

13.5.2.2 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?
   - ....

13.5.2.3 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

13.5.3 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.

13.5.4 Participant 1

13.5.4.1 Consent Form

13.5.4.1.1 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

13.5.4.2 Pre-questionnaire

13.5.4.2.1 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**

13.5.4.3 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**

13.5.4.4 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

13.5.4.5 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.

13.6 Pilot Study 2

13.6.1 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			
Rel	Diff	Lan	Count[a]	%	Count	%	Residuals	Std. Residuals
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

13.6.2 Pre-questionnaire

Pre-questionnaire

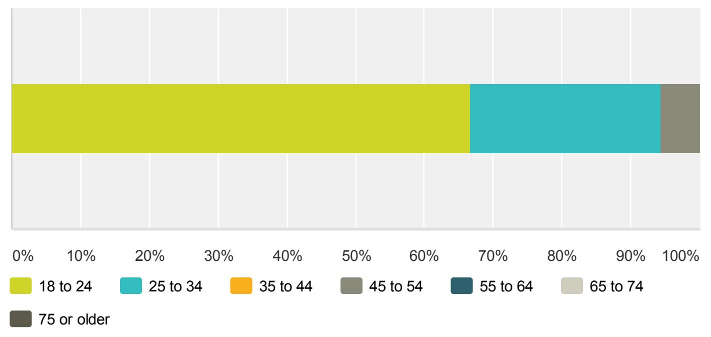
RESPONDENTS: 18 of 18

PAGE 2: Demographics

Q1

What is your age?

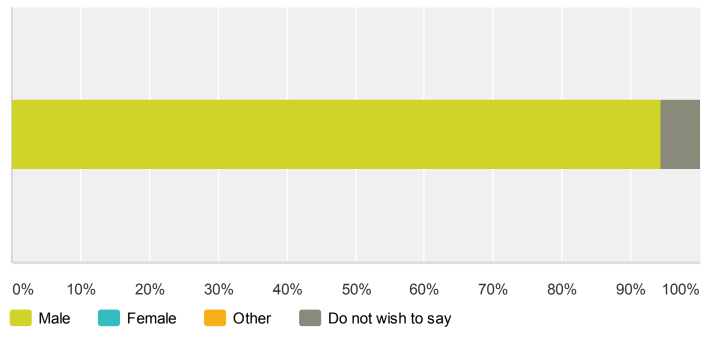
Answered: 18 Skipped: 0



Q2

Sex

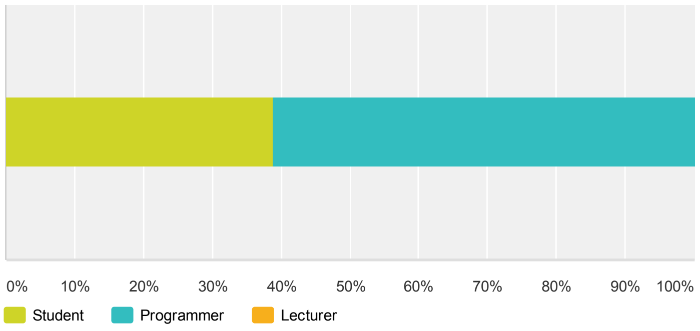
Answered: 18 Skipped: 0



Q3

Which of the following best describes your current occupation?

Answered: 18 Skipped: 0

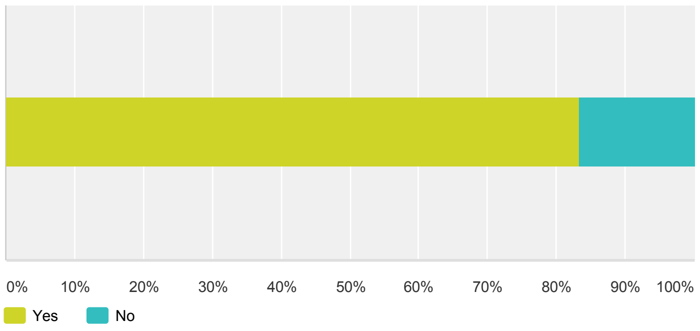


PAGE 3: Functional Programming

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

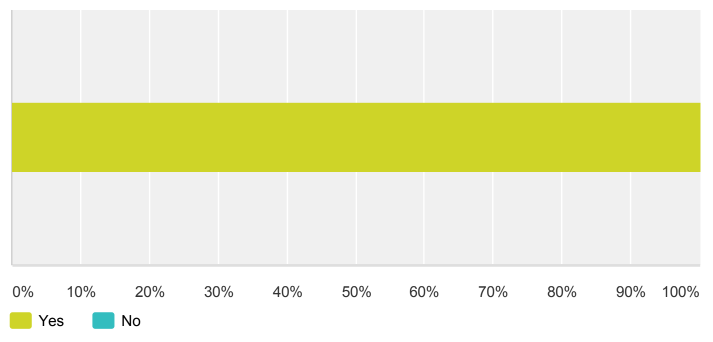
OCaml
4/12/2014 6:41 PM

PAGE 4: Design & Software

Q6

Have you designed software before?

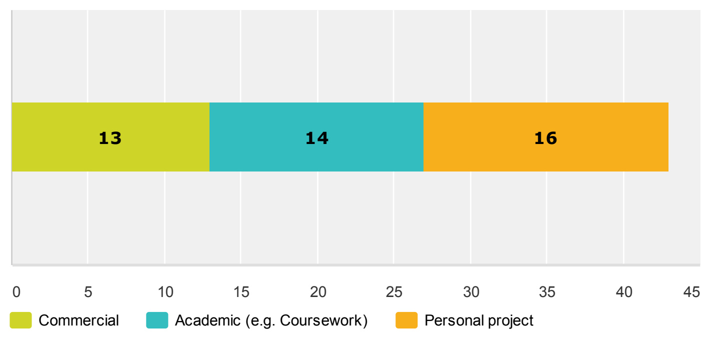
Answered: 18 Skipped: 0



Q7

If Yes, For what purposes have you designed software?

Answered: 18   Skipped: 0



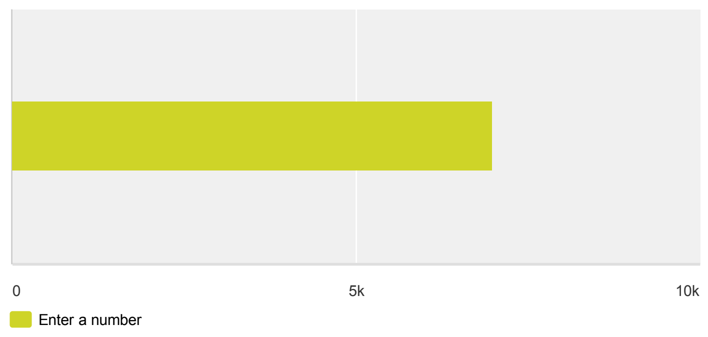
Answer Choices	Responses	
Commercial	72.22%	13
Academic (e.g. Coursework)	77.78%	14
Personal project	88.89%	16
Total Respondents: 18		

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

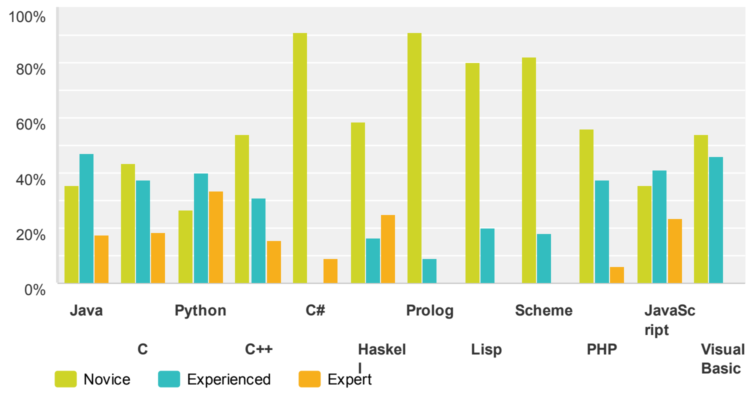
Total Respondents: 18

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, Experianced
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