The potential of declarative programming languages to support user interface programming: the case of Elm

Simon Buist
The University of Bath

April 15, 2014

Abstract

Your abstract should appear here. An abstract is a short paragraph describing the aims of the project, what was achieved and what contributions it has made.

It consists of two paragraphs.

Contents

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

Signed:

The potential of declarative programming languages to support user interface programming: the case of Elm

Submitted by: Simon Buist

COPYRIGHT

Declaration

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

Signed:

Acknowledgements

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

1 Introduction

`Introduce the topic, Summarise the report`

2 Literature Survey

2.1 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.

2.2 What does it mean to be 'easy to use?'

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

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

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

The catalyst to this whole dissertation was a talk about the concept of a highly reactive development environment – where changes in the code result in instantaneous updates to the runtime, 'on-the-fly'. This was presented in Bret Victor's "Inventing on Principle" (Victor, 2012). In his presentation Bret makes several assertions about the 'traditional' style of coding, one statement of which is that "most of the developer's time is spent looking at the code, blindly without an immediate connection to the thing they're making". He argues that "so much of creation is discovery, and you can't discover anything if you can't see what you're doing" – alluding to his earlier statement that the compile-run-debug cycle is much like this.

Evan Czaplicki, in his thesis of which Elm is the product (Czaplicki, 2012), makes similar claims – "[Elm] makes it *quick and easy* to create and combine text, images, and video into rich multimedia displays." While the evaluation of Elm's usability is not the focus of the thesis, rather, it is to establish a context for Functional Reactive Programming and describe the implementation details, he makes other usability claims without evidence – "[non-declarative frameworks for graphical user interfaces] mire programmers in the many small, nonessential details of handling user input and modifying the display.",

"FRP makes GUI programming much more manageable", and in a section entitled *The Benefits of Functional GUIs*, "In Elm, divisions between data code, display code, and user interaction code arise fairly naturally, helping programmers write robust GUI code". If these claims are true, there is all the more evidence that Elm should be a language of choice for GUI programmers, but experiments must be done to determine this.

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

2.3 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, 2013).

3 Requirements

If you are doing a primarily software development project, this is the chapter in which you review the requirements decisions and critique the requirements process.

4 Design

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

4.1 Pilot study

In reflection, the task I chose was too difficult to capture the cognitive load incurred by the language itself for a given task, due to the difficulty of the task itself creating noise. I could improve this by simplifying the task, in a way that is 'language agnostic', i.e. that is not idiomatic of Elm or JavaScript (the two languages that I am comparing). Something like the following will never be that easy in JavaScript:

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

Group meeting with Leon at East Building, 11:15 Friday 4th October 2013

N.B. READ UP ON AND REMIND YOURSELF OF HCI STUFF (Year 2) AND SOFTWARE ENGINEERING STUFF (Year 1)

4.1.1 Reading material

In email repsonse to request for FYP meeting, **Leon writes:**

Please do a bit of reading around beforehand. Go to the ACM Digital Library and search on 'user interface programming'.

1. ACM Conference on Human Factors in Computing Systems
2. ACM CSCW: Conference on Computer Supported Cooperative Work
3. ACM UIST: Symposium on User Interface Software and Technology

In moodle project page, **Leon writes:**

Your project must be related to contemporary developments in Human-Computer Interaction, and preferably to the part of the HCI world that focuses on interactive systems for collaboration

1. ???
2. ???

Also In moodle project page, **Leon also writes:**

*It normally starts with some user-centred research (observations, interviews, pilot experiment) to ground the problem, carried out concurrently with literature research.

The research problem is normally boiled down to something that can be addressed through the production of alternative versions of an interactive system.

This is closely followed by initial design work and the production of a rough but working prototype leading up to Christmas.

After the January exams, my students typically re-scope their research problem, based in the outcome of their initial work, and solidify their implementation ready for a full evaluation in March and April.*

Thus, my answers to the questions Leon posed should follow this structure in terms of what I want to get out of it. I can use the above structure to identify **concerns** of potential challenges in each step/combination of steps/step-transitions (e.g. step dependencies, resource procurement)

Also in moodle product page, **Leon also writes:**

Students should prepare for their projects by refreshing their memories about Interaction from CM20216 activities. You should read about HCI in general, and support for collaboration in particular. Look at any or all of the following book chapters:

- Sharp, Rogers and Preece (2007) Interaction Design. hapter 4: Designing to Support Communication and Collaboration.
- Dix, Finlay, Abowd and Beale (2004) Human-Computer Interaction. hapter 14: Communication and Collaboration Models.
- Shneiderman and Plaisant (2005) Designing the User Interface. hapter 10: Collaboration.

4.1.2 Leon asked us to answer these questions and bring a notebook:

4.1.2.1 Q1. What I hope to get out of my FYP as an experience?

I hope to gain a deep and meaningful understanding of the programmer as a user, as an individual and the context of that individual – e.g. in a software team inside a department, inside a management structure... inside a company.

I hope to use this understanding to determine processes/work-flows that programmers experience in the endeavour of User Interface Design, both from the individual perspective and as a team.

Within these work flows, I wish to identify, in detail, metrics to gauge productivity, in order to measure this in experiments, perhaps doing A/B testing with Elm and some other, perhaps procedural language. This is an example of an objective measure.

I would also like to gather self-reported, more "fuzzy" feedback on user's perception of their productivity – pain points, advantages, etc. they experience in using Language X to product a UI compared to Language Y (Declarative languages like Elm, etc)

I wish to verify, empirically, the comparisons and claims made on the What is FRP? page of the elm-lang.org website, and those claimed it's research paper (detailing the implementation of Elm, **benefits**, etc.)

In email again, **Leon writes:**

*The Elm site makes **comparative statements.** That is encouraging because it sets up opportunities for you to test some of the claims they make, and to ask new questions about Elm that its proponents may not have considered.*

These are:

1. "most current frameworks for graphical user interfaces are not declarative. They mire programmers in the many small, nonessential details of handling user input and manually modifying the display."
2. "with FRP, many of the irrelevant details are left to the compiler, freeing the programmer to think about things that matter."
3. "Coding these examples in a traditional GUI framework such as HTML/CSS/JavaScript . would require significantly more work and headache."
4. "Not only is that painful to code, but it also requires broad and deep knowledge of inconsequential things."
5. "FRP makes tasks considerably easier by taking care of the messy .how. of events, display, and updates."

4.1.2.2 Q2. Where my Project Idea came from (what inspired me)?

- The pain of coding and writing GUIs in PyQt4 while at my last job at Altran
- The joys of coding in Haskell
- The pain of writing GUIs in Haskell
- The joys of coding and writing GUIs in Elm!

4.1.2.3 Q3. What are my concerns?

1. Difficulty procuring programmers (users) – specifically those that meet my criteria of not having used a declarative programming language.
2. Difficulty procuring programmers working in a team
3. The complexity/scope of the project – is it enough for a FYP; is it too much?

4. Looking at the production of User Interfaces using a programming language, there are many variables – how will I devise an experiment to minimise this and isolate a variable so that I can make some causal/correlational conclusions?
5. Concern regarding the dependency of a subsequent part of the project on a previous step – this is inherent of all projects, though.

4.2 Experimental design

4.2.1 30/10/2013 @ 9:15 am in STV

4.2.1.1 Individual Meeting after Proposal hand-in Our discussion centered around the direction I wish to take following my Project Proposal.

4.2.1.1.1 AB Testing of the language 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).

4.2.1.1.2 Test just the paradigm He also suggested a potential experiment to test just the paradigm, eliminating the IDE from the experiment above. Perhaps for a Pilot study.

4.2.1.1.3 Experiment process

1. Study question (e.g. Is it easy?)
2. Measurement concept (e.g. "Easy")
3. Operationalisation – taking a measurement concept and mapping it to something concrete (e.g. if completing a pre-defined task the user must complete takes `< 5` steps, it is 'easy' – we can then compare instances of these studies given our definition of easy). This is much like mapping a design to an implementation, and there is a risk of losing information, or ending up with a mismatched concrete instance that does not represent the concept we wish to convey.

4. Do another operationalisation of our measurement concept – this allows us to get a different perspective of the same concept. (e.g. if total length of pauses during a 1 hour experiment is `< 10` minutes, it is 'easy'). We do this to get 'coverage' of the measurement concept. It is a form of cross validation. If we see an overlap in the correlational results after analysis, we can make a stronger assertion that e.g. "language A is easier than language B.". The idea I am describing here is methodological decision-making.
5. Predict what will be the likely results of our experiments on the operationalised measurements. This is "feed forward validation".
6. Do the experiement.
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 infact objectively true/false".

4.2.1.1.4 Pilots We also spoke about ideas for pilot studies – asking "What might be surprising insights into declarative programming languages for User Interface Design – the case of Elm?".

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

I might ask

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

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

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

4.2.1.2 Actions for next meeting Devise a Pilot study, answering these 3 questions:

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

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

4.2.2 Wed Mar 25 14:30 GMT 2014

(Several meetings undocumented)

TODO: Refer to notes in Diary for previous entries.

4.2.2.1 Progress since last meeting Discussed findings from analysis of pilot study

4.2.2.1.1 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.

4.2.2.1.2 Model Adjustment 1

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

4.2.2.1.3 Observation 2

- Sifting through 1-hour+ of video data capture for incidences of cognitive load is *HARD!*. Is there some programmatic way of narrowing the video data to points of interest?

4.2.2.1.4 Model Adjustment 2

- Track 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.
- Leon suggested a further refinement of this idea in order to further narrow 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). His refinement was to 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.

4.2.2.2 Further discussion We then discussed some questions that might lead my direction of study in the next steps of my research:

- 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.

4.2.2.3 We also discussed... The result of the dissertation will be a list of observed cognitive easing/loading that each language produces for users, much like an advantage/disadvantage comparison:

Elm	JavaScript
+ ...	+ ...
+ ...	- ...
- ...	- ...
- ...	+ ...
+ ...	_

4.2.2.4 Actions

1. 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.

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

4.2.3 Tue Apr 1 14:30:00 BST 2014

Discussed progress made and what hypotheses to form that may usefully model cognitive load.

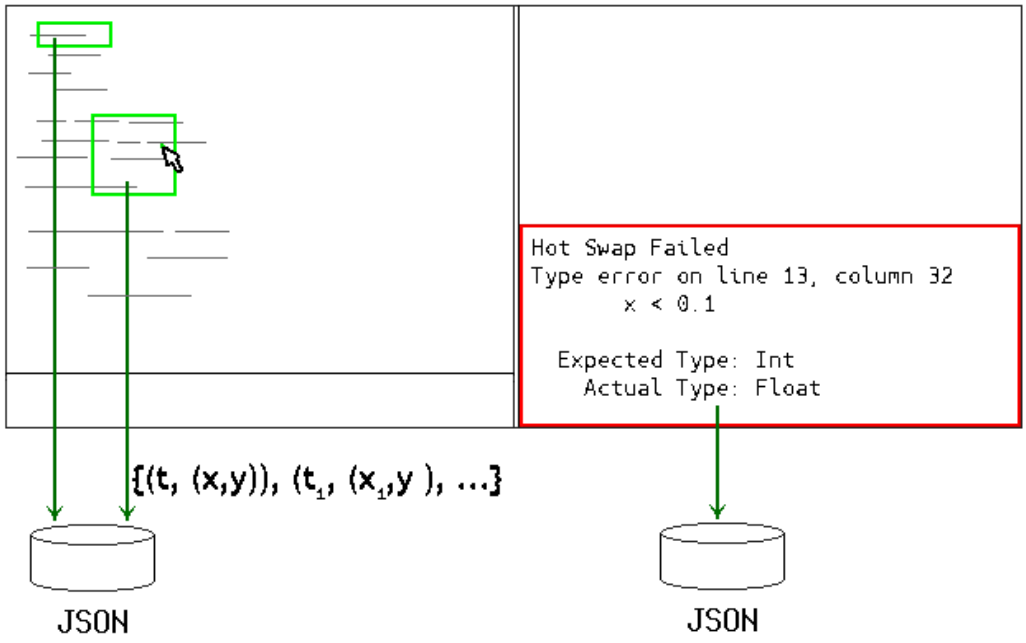

Figure 1: Extensions made to the Elm IDE

4.2.3.1 Progress since last meeting I have implemented full-screen mouse tracking that stores to a database a tuple:

```
(t, (x, y))
```

for every mouse move, producing a list in JSON (so it's more like `{{uniq-userid: {125125, (67, 321)}}, {uniq-userid: {125126, (67, 322)}} ...}`)

I am ready to demo this (See Action 1.)

The only issue worth tweaking is that user activity data is captured separately from the error output, so I will need to collate the data afterwards or find some way to feed it into the same data store.

4.2.3.2 Meeting Discussion 2 Hypotheses

1. Why the regions (*see green boxes in figure above*) I define in the code (to mouse-track e.g.) are meaningful
2. Frequency of semantically or syntactically incorrect errors made will differ as a function of the language under study

These need narrowing as they are too broad to test. Explode them into multiple, tighter hypotheses.

They are *valid* because they are *well-founded* – i.e. I have good reason to believe that # of errors made is an indication of cognitive load. I have good reason to believe that the selected regions will have more mouse activity (or whatever activity I suspect indicates higher cognitive load) as they are harder regions of code OR they pertain to achieving the set task.

4.2.3.3 Actions

1. Refine Mouse logging
    1. **DONE** Make it so that I can run arbitrary Elm code in the editor via a fileOpen operation
    2. **DONE** Make an Elm file that logs mouse movements ready to be loaded into Editor.hs
    3. **DONE** Load it into the editor and test it uploads to Firebase
    4. **DONE** Modify Generate.hs

    ```
    case (Elm.compile elmSrc) of
        Left jsSrc -> ...
        Right _ -> error "blah"
    ```

    So that when we get an error, we timestamp and append it to a log file so this can later be collated with the Firebase to determine when errors were made

    I'll need to insert a layer between `compile :: Snap()` and `serveHtml :: MonadSnap m => H.Html -> m ()` that performs the logging. It will have type signature `TypedHtml -> H.Html`

    See the functions `compile` and `serveHtml` in Server.hs

5. Make it so I can define regions in the mouse tracking – i.e. ONLY within a defined region is the mouse movement tracked e.g. `if mouse(x,y) in some2by2Square then Just mouse(x,y) else Nothing`

See https://github.com/spanners/laska/blob/master/Signals.elm

2. Demo to Hilary Johnson

    1. Install on VPS (See install_elm.sh)
    2. Run these:

```
git clone https://github.com/spanners/elm-lang.org
cd elm-lang.org
cabal install --bindir=.
```

3. **DONE** Design a task in JS and Elm
4. Define regions to select for logging activity. Why? Because:

    - Complex logic in code, OR
    - Relevant to task
    - Captures Thrash (keep on going over the same thing, e.g.). Errors made also captures thrash!

5. **DONE** Determine what to do with mouse (for example) data.

4.2.4 Tuesday Apr 8 14:30:00 BST 2014

What makes code difficult to understand and work with?

- Bit twiddling?
- Declaring and defining simultaneously?
- Compound if/then/else statements?

[Programming is] manipulating symbols blindly ~ Bret Victor

Do a 2x2 study, defining regions in the code monitoring mouse clicks. Regions can either be simple/hard in complexity (exhibiting/not-exhibiting one of the

above 'difficult' properties). Or code can be task-oriented or not, that is *the code does/does not need to be changed to achieve the completed task set for the user*:

Elm	-
Simple/Task	Hard/Task
Simple/Not-Task	Hard/Not-Task

JavaScript	-
Simple/Task	Hard/Task
Simple/Not-Task	Hard/Not-Task

4.2.4.1 2x2 study between-subjects

4.2.4.2 Study method Look at total and/or mean time in each of these areas for comparison.

My 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

4.2.4.3 Actions

1. **DONE** Reorder divs so embedded div is on top of editor div.

   This turned out (I am fairly certain) to be due to codemirror.js binding mouse clicks. It was solved by using Elm's `Mouse.isDown`. Using `Mouse.isDown` has the added benefit of tracking mouse selects and drags, because it logs `(x,y)` when the mouse is down and `(x,y)` again when it is up.

2. **DONE** Create a task that features *Hard/Simple x Task/Not-task* (See table above)

3. ~~Implement *Region filtering* functionality so mouse activity is only logged when the clicks occur within defined region(s)~~

   I have instead defined bounding boxes that pertain to the regions I want to track as a mouse-data filter – that is, I capture all click data for the whole frame, and then filter it by comparing x,y co-ordinates with my bounding boxes. If it's in the box, keep it, otherwise discard.

4. **DONE** Integrate JS task into IDE

5. **DONE** Perform pilot study

6. **WIP** Visualise mouse data

5 Implementation and Testing

This is the chapter in which you review the implementation and testing decisions and issues, and critique these processes. Code can be output inline using `some code`. For example, this code is inline: `public static int example = 0;` (I have used the character | as a delimiter, but any non-reserved character not in the code text can be used.) Code snippets can be output using the environment with the code given in the environment. For example, consider listing 5.1, below. Listing 5.1: Example code

Code listings are produced using the package "Listings". This has many useful options, so have a look at the package documentation for further ideas.

6 Results

6.0.5 Tue Apr 15 15:50:38 BST 2014

Operationalisation of thrash (the concept), i.e. cementing the concept by a metric that models cognitive load (does it? we don't know – further work after the analysis of this may determine if it is a plausible indicator of cognitive load)

Leon suggested an improvement over this experimental method is to take people who are new, and train them up either in JS or Elm, and then run the same task. That way, their level of ability is comparable. (New as in never having used JS or Elm)

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.

Talk about "phases" in a programmer's activities during task-completion:

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

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 (?)

6.0.5.1 Mention the ways in which the study is flawed:

1. Self-reported expertise
2. Self-reported task completion
3. No way to be sure which error log pertains to which compile
4. Unique participant ID per Surveymonkey
5. Surveymonkey has taken my data hostage
6. window dimensions?!

7. Syntax reference 404
8. I did not capture their code solution, so relied on trust

6.0.5.2 Results

1. Describe data collected
2. How it was analysed (I aggregated regions and looked at number of clicks per region (Hard/Task, Hard/Not-Task, Simple/Task, Simple/Not-Task)*(Elm, JavaScript))
3. Presentation of data (summary means std dev.)
    1. x^2 frequency analyses
    2. `2x2x2` making 8 cells. My expected is an even distribution of clicks in each category, i.e. if I have 80 clicks in total across all groups, I expect to find 10 in each cell if there is no correlation.

Time (min)	Clicks
38.717217	183
8.034583	130
7.878533	39
23.672500	25
29.754533	391
14.993517	78
48.960367	769
6.354050	71
7.878533	39
29.698267	501
40.302217	803
12.319317	65
17.106933	79
12.958300	119

This is the chapter in which you review the outcomes, and critique the outcomes process.

You may include user evaluation here too.

7 Conclusions

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

Bibliography

Czaplicki, E. (2012) 'Elm: Concurrent FRP for Functional GUIs',

Czaplicki, E. (2013) '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.

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.

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.

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.

8 Appendix A

8.1 Design Diagrams

9 Appendix B

9.1 Raw results output

9.1.1 firebase-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
        }
      }
    }
  }
```

9.1.2 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"
   },
}
```

10 Appendix C

10.1 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

10.1.1 LICENSE

```
Copyright (c) 2012-2013 Evan Czaplicki

All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:

    * Redistributions of source code must retain the above copyright
      notice, this list of conditions and the following disclaimer.

    * Redistributions in binary form must reproduce the above
      copyright notice, this list of conditions and the following
      disclaimer in the documentation and/or other materials provided
      with the distribution.

    * Neither the name of Evan Czaplicki nor the names of other
      contributors may be used to endorse or promote products derived
      from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
```

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

10.1.2 install_elm.sh

```
#!/bin/bash

sudo apt-get install libgl1-mesa-dev libglc-dev freeglut3-dev libedit-dev libglw1-
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 8
sudo make install
cd
wget http://lambda.haskell.org/platform/download/2013.2.0.0/haskell-platform-2013.
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
```

10.1.3 Server.hs

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

import Data.Monoid (mempty)
import qualified Data.ByteString as BS
import qualified Data.ByteString.Char8 as BSC
import qualified Elm.Internal.Utils as Elm
import Control.Applicative
import Control.Monad.Error

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

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

import qualified Elm.Internal.Paths as Elm
import qualified Generate
import qualified Editor

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

flags :: Flags
flags = Flags
```

```
37    { port = 8000 &= help "set the port of the server"
38    }
39
40  -- | Set up the server.
41  main :: IO ()
42  main = do
43    setNumCapabilities =<< getNumProcessors
44    putStrLn "Initializing Server"
45    getRuntimeAndDocs
46    setupLogging
47    precompile
48    cargs <- cmdArgs flags
49    httpServe (setPort (port cargs) defaultConfig) $
50        ifTop (serveElm "public/Empty.elm")
51        <|> route [ ("try", serveHtml Editor.empty)
52                  , ("edit", edit)
53                  , ("_edit", jsEdit)
54                  , ("code", code)
55                  , ("_code", jsCode)
56                  , ("compile", compile)
57                  , ("_compile", jsCompile)
58                  , ("hotswap", hotswap)
59                  ]
60        <|> error404
61
62  error404 :: Snap ()
63  error404 =
64      do modifyResponse $ setResponseStatus 404 "Not found"
65         serveElm "public/build/Error404.elm"
66
67  serveElm :: FilePath -> Snap ()
68  serveElm = serveFileAs "text/html; charset=UTF-8"
69
70  logAndServeJS :: MonadSnap m => H.Html -> m ()
71  logAndServeJS = serveHtml
72
73  logAndServeHtml :: MonadSnap m => (H.Html, Maybe String) -> m ()
74  logAndServeHtml (html, Nothing)  = serveHtml html
```

```
75  logAndServeHtml (html, Just err) =
76      do timeStamp <- liftIO $ readProcess "date" ["--rfc-3339=ns"] ""
77         liftIO $ appendFile "error_log.json" $ "{\"" ++ (init timeStamp)
78                                                ++ "\","
79                                                ++ (show (lines err))
80                                                ++ "},"
81         setContentType "text/html" <$> getResponse
82         writeLBS (BlazeBS.renderHtml html)
83
84
85  embedJS :: MonadSnap m => H.Html -> String -> m ()
86  embedJS js participant =
87      do
88         elmSrc <- liftIO $ readFile "EmbedMeJS.elm"
89         setContentType "text/html" <$> getResponse
90         writeLBS (BlazeBS.renderHtml (embedMe elmSrc js participant))
91
92  embedHtml :: MonadSnap m => H.Html -> String -> m ()
93  embedHtml html participant =
94      do elmSrc <- liftIO $ readFile "EmbedMeElm.elm"
95         setContentType "text/html" <$> getResponse
96         writeLBS (BlazeBS.renderHtml (embedMe elmSrc html participant))
97
98  serveHtml :: MonadSnap m => H.Html -> m ()
99  serveHtml html =
100     do setContentType "text/html" <$> getResponse
101        writeLBS (BlazeBS.renderHtml html)
102
103 hotswap :: Snap ()
104 hotswap = maybe error404 serve =<< getParam "input"
105     where
106       serve code =
107           do setContentType "application/javascript" <$> getResponse
108              writeBS . BSC.pack . Generate.js $ BSC.unpack code
109
110 jsCompile :: Snap ()
111 jsCompile = maybe error404 serve =<< getParam "input"
112     where
```

```
113         serve = logAndServeJS . Generate.logAndJS "Compiled JS" . BSC.unpack
114
115 compile :: Snap ()
116 compile = maybe error404 serve =<< getParam "input"
117     where
118       serve = logAndServeHtml . Generate.logAndHtml "Compiled Elm" . BSC.unpack
119
120 edit :: Snap ()
121 edit = do
122   participant <- BSC.unpack . maybe "" id <$> getParam "p"
123   cols <- BSC.unpack . maybe "50%,50%" id <$> getQueryParam "cols"
124   withFile (Editor.ide cols participant)
125
126 jsEdit :: Snap ()
127 jsEdit = do
128   participant <- BSC.unpack . maybe "" id <$> getParam "p"
129   cols <- BSC.unpack . maybe "50%,50%" id <$> getQueryParam "cols"
130   withFile (Editor.jsIde cols participant)
131
132 code :: Snap ()
133 code = do
134   participant <- BSC.unpack . maybe "" id <$> getParam "p"
135   embedWithFile Editor.editor participant
136
137 jsCode :: Snap ()
138 jsCode = do
139   participant <- BSC.unpack . maybe "" id <$> getParam "p"
140   jsEmbedWithFile Editor.jsEditor participant
141
142 embedee :: String -> String -> H.Html
143 embedee elmSrc participant =
144     H.span $ do
145       case Elm.compile elmSrc of
146         Right jsSrc -> do
147             embed $ H.preEscapedToMarkup (subRegex oldID jsSrc newID)
148         Left err ->
149             H.span ! A.style "font-family: monospace;" $
150             mapM_ addSpaces (lines err)
```

```
151         script "/fullScreenEmbedMe.js"
152     where addSpaces line = H.preEscapedToMarkup (Generate.addSpaces line) >> H.br
153           oldID = mkRegex "var user_id = \"1\";"
154           newID = ("var user_id = " ++ participant ++ "+'';" :: String)
155           jsAttr = H.script ! A.type_ "text/javascript"
156           script jsFile = jsAttr ! A.src jsFile $ mempty
157           embed jsCode = jsAttr $ jsCode
158
159 embedMe :: String -> H.Html -> String -> H.Html
160 embedMe elmSrc target participant = target >> (embedee elmSrc participant)
161
162 jsEmbedWithFile :: (FilePath -> String -> H.Html) -> String -> Snap ()
163 jsEmbedWithFile handler participant = do
164   path <- BSC.unpack . rqPathInfo <$> getRequest
165   let file = "public/" ++ path
166   exists <- liftIO (doesFileExist file)
167   if not exists then error404 else
168       do content <- liftIO $ readFile file
169          embedJS (handler path content) participant
170
171 embedWithFile :: (FilePath -> String -> H.Html) -> String -> Snap ()
172 embedWithFile handler participant = do
173   path <- BSC.unpack . rqPathInfo <$> getRequest
174   let file = "public/" ++ path
175   exists <- liftIO (doesFileExist file)
176   if not exists then error404 else
177       do content <- liftIO $ readFile file
178          embedHtml (handler path content) participant
179
180 withFile :: (FilePath -> String -> H.Html) -> Snap ()
181 withFile handler = do
182   path <- BSC.unpack . rqPathInfo <$> getRequest
183   let file = "public/" ++ path
184   exists <- liftIO (doesFileExist file)
185   if not exists then error404 else
186       do content <- liftIO $ readFile file
187          serveHtml $ handler path content
188
```

```haskell
setupLogging :: IO ()
setupLogging =
    do createDirectoryIfMissing True "log"
       createIfMissing "log/access.log"
       createIfMissing "log/error.log"
    where
      createIfMissing path = do
        exists <- doesFileExist path
        when (not exists) $ BS.writeFile path ""

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

```

```
227 adjustHtmlFile :: FilePath -> IO ()
228 adjustHtmlFile file =
229   do src <- BSC.readFile file
230      let (before, after) = BSC.breakSubstring "<title>" src
231      BSC.writeFile (FP.replaceExtension file "elm") $
232         BSC.concat [before, style, after]
233      removeFile file
234
235 style :: BSC.ByteString
236 style =
237     "<style type=\"text/css\">\n\
238     \  a:link {text-decoration: none; color: rgb(15,102,230);}\n\
239     \  a:visited {text-decoration: none}\n\
240     \  a:active {text-decoration: none}\n\
241     \  a:hover {text-decoration: underline; color: rgb(234,21,122);}\n\
242     \  body { font-family: \"Lucida Grande\",\"Trebuchet MS\",\
243     \  \"Bitstream Vera Sans\",Verdana,Helvetica,sans-serif !important; }\n\
244     \  p, li { font-size: 14px !important;\n\
245     \          line-height: 1.5em !important; }\n\
246     \</style>"
```

10.1.4 Editor.hs

```
1  {-# LANGUAGE OverloadedStrings #-}
2  module Editor (editor,jsEditor,jsIde,ide,empty) where
3
4  import Data.Monoid (mempty)
5  import Text.Blaze.Html
6  import qualified Text.Blaze.Html5 as H
7  import qualified Text.Blaze.Html5.Attributes as A
8  import Network.HTTP.Base (urlEncode)
9  import qualified System.FilePath as FP
10
11 import qualified Elm.Internal.Utils as Elm
12 import Data.Maybe (fromMaybe)
13
14 import Generate (addSpaces)
```

```haskell

-- | Display an editor and the compiled result side-by-side.
jsIde :: String -> String -> FilePath -> String -> Html
jsIde cols participant fileName code =
    jsIdeBuilder cols
                 participant
                 ("JS Editor: " ++ FP.takeBaseName fileName)
                 fileName
                 ("/_compile?input=" ++ urlEncode code)

-- | Display an editor and the compiled result side-by-side.
ide :: String -> String -> FilePath -> String -> Html
ide cols participant fileName code =
    ideBuilder cols
               participant
               ("Elm Editor: " ++ FP.takeBaseName fileName)
               fileName
               ("/compile?input=" ++ urlEncode code)

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

jsIdeBuilder :: String -> String -> String -> String -> String -> Html
jsIdeBuilder cols participant title input output =
    H.docTypeHtml $ do
      H.head . H.title . toHtml $ title
      preEscapedToMarkup $
         concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
          , "  <frame name=\"input\" src=\"/_code/", input, "?p=",
            participant, "\" />\n"
          , "  <frame name=\"output\" src=\"", output, "\" />\n"
          , "</frameset>" ]


ideBuilder :: String -> String -> String -> String -> String -> Html
ideBuilder cols participant title input output =
```

```
53      H.docTypeHtml $ do
54        H.head . H.title . toHtml $ title
55        preEscapedToMarkup $
56            concat [ "<frameset cols=\"" ++ cols ++ "\">\n"
57                   , "  <frame name=\"input\" src=\"/code/", input, "?p=",
58                     participant, "\" />\n"
59                   , "  <frame name=\"output\" src=\"", output, "\" />\n"
60                   , "</frameset>" ]
61
62  -- | list of themes to use with CodeMirror
63  themes :: [String]
64  themes = [ "ambiance", "blackboard", "cobalt", "eclipse"
65           , "elegant", "erlang-dark", "lesser-dark", "monokai", "neat", "night"
66           , "rubyblue", "solarized", "twilight", "vibrant-ink", "xq-dark" ]
67
68  jsFiles :: [AttributeValue]
69  jsFiles = [ "/codemirror-3.x/lib/codemirror.js"
70            , "/codemirror-3.x/mode/elm/elm.js"
71            , "/misc/showdown.js"
72            , "/misc/editor.js?0.11" ]
73
74  jsFiles2 :: [AttributeValue]
75  jsFiles2 = [ "/codemirror-3.x/lib/codemirror.js"
76            , "/codemirror-3.x/mode/javascript/javascript.js"
77            , "/misc/showdown.js"
78            , "/misc/editor.js?0.11" ]
79
80  jsEditor :: FilePath -> String -> Html
81  jsEditor filePath code =
82      H.html $ do
83        H.head $ do
84          H.title . toHtml $ "JS Editor: " ++ FP.takeBaseName filePath
85          H.link ! A.rel "stylesheet"
86                 ! A.href "/codemirror-3.x/lib/codemirror.css"
87          mapM_ themeAttr themes
88          H.link ! A.rel "stylesheet" ! A.type_ "text/css"
89                                      ! A.href "/misc/editor.css"
90          mapM_ script jsFiles2
```

```
91          script "/elm-runtime.js?0.11"
92          script "http://cdn.firebase.com/v0/firebase.js"
93        H.body $ do
94          H.form ! A.id "inputForm"
95                 ! A.action "/_compile"
96                 ! A.method "post"
97                 ! A.target "output" $ do
98             H.div ! A.id "editor_box" $
99               H.textarea ! A.name "input" ! A.id "input" $ toHtml ('\n':code)
100            H.div ! A.id "options" $ do
101              bar "documentation" docs
102              bar "editor_options" editorOptions
103              bar "always_on" (buttons >> options)
104         embed "initEditor();"
105   where themeAttr theme = H.link ! A.rel "stylesheet"
106                                  ! A.href (toValue ("/codemirror-3.x/theme/"
107                                                     ++ theme
108                                                     ++ ".css" :: String))
109         jsAttr = H.script ! A.type_ "text/javascript"
110         script jsFile = jsAttr ! A.src jsFile $ mempty
111         embed jsCode = jsAttr $ jsCode
112
113 -- | Create an HTML document that allows you to edit and submit Elm code
114 --   for compilation.
115 editor :: FilePath -> String -> Html
116 editor filePath code =
117     H.html $ do
118       H.head $ do
119         H.title . toHtml $ "Elm Editor: " ++ FP.takeBaseName filePath
120         H.link ! A.rel "stylesheet"
121                ! A.href "/codemirror-3.x/lib/codemirror.css"
122         mapM_ themeAttr themes
123         H.link ! A.rel "stylesheet" ! A.type_ "text/css"
124                                     ! A.href "/misc/editor.css"
125         mapM_ script jsFiles
126         script "/elm-runtime.js?0.11"
127         script "http://cdn.firebase.com/v0/firebase.js"
128       H.body $ do
```

```
129         H.form ! A.id "inputForm"
130                ! A.action "/compile"
131                ! A.method "post"
132                ! A.target "output" $ do
133            H.div ! A.id "editor_box" $
134              H.textarea ! A.name "input" ! A.id "input" $ toHtml ('\n':code)
135            H.div ! A.id "options" $ do
136              bar "documentation" docs
137              bar "editor_options" editorOptions
138              bar "always_on" (buttons >> options)
139         embed "initEditor();"
140   where themeAttr theme = H.link ! A.rel "stylesheet"
141                                  ! A.href (toValue ("/codemirror-3.x/theme/"
142                                                     ++ theme
143                                                     ++ ".css" :: String))
144         jsAttr = H.script ! A.type_ "text/javascript"
145         script jsFile = jsAttr ! A.src jsFile $ mempty
146         embed jsCode = jsAttr $ jsCode
147
148 bar :: AttributeValue -> Html -> Html
149 bar id' body = H.div ! A.id id' ! A.class_ "option" $ body
150
151 buttons :: Html
152 buttons = H.div ! A.class_ "valign_kids"
153                 ! A.style "float:right; padding-right: 6px;"
154                 $ compileButton
155       where
156         compileButton =
157             H.input
158                  ! A.type_ "button"
159                  ! A.id "compile_button"
160                  ! A.value "Compile"
161                  ! A.onclick "compile()"
162                  ! A.title "Ctrl-Enter: change program behavior \
163                            \but keep the state"
164
165 options :: Html
166 options = H.div ! A.class_ "valign_kids"
```

```
167                 ! A.style "float:left; padding-left:6px; padding-top:2px;"
168                 $ (docs' >> opts)
169     where
170       docs' =
171         H.span  ! A.title "Show documentation and types." $ "Hints:" >>
172             H.input ! A.type_ "checkbox"
173                     ! A.id "show_type_checkbox"
174                     ! A.onchange "showType(this.checked);"
175
176       opts =
177         H.span  ! A.title "Show editor options."
178                 ! A.style "padding-left: 12px;" $ "Options:" >>
179             H.input ! A.type_ "checkbox"
180                     ! A.id "options_checkbox"
181                     ! A.onchange "showOptions(this.checked);"
182
183 editorOptions :: Html
184 editorOptions = theme >> zoom >> lineNumbers
185     where
186       optionFor :: String -> Html
187       optionFor text =
188           H.option ! A.value (toValue text) $ toHtml text
189
190       theme =
191           H.select ! A.id "editor_theme"
192                    ! A.onchange "setTheme(this.value)"
193                    $ mapM_ optionFor themes
194
195       zoom =
196           H.select ! A.id "editor_zoom"
197                    ! A.onchange "setZoom(this.options[this.selectedIndex].\
198                                 \innerHTML)"
199                    $ mapM_ optionFor ["100%", "80%", "150%", "200%"]
200
201       lineNumbers = do
202         H.span ! A.style "padding-left: 16px;" $ "Line Numbers:"
203         H.input ! A.type_ "checkbox"
204                 ! A.id "editor_lines"
```

```
205                 ! A.onchange "showLines(this.checked);"
206
207 docs :: Html
208 docs = tipe >> desc
209     where
210       tipe = H.div ! A.class_ "type" $ message >> more
211
212       message = H.div !
213                   A.style "position:absolute; left:4px; right:36px;\
214                            \overflow:hidden; text-overflow:ellipsis;" $ ""
215
216       more = H.a ! A.id "toggle_link"
217                  ! A.style "display:none; float:right;"
218                  ! A.href "javascript:toggleVerbose();"
219                  ! A.title "Ctrl+H"
220                  $ ""
221
222       desc = H.div ! A.class_ "doc"
223                    ! A.style "display:none;"
224                    $ ""
225
```

10.1.5 Generate.hs

```
1  {-# LANGUAGE OverloadedStrings #-}
2  module Generate (logAndJS, logAndHtml, html, js, addSpaces) where
3
4  import Data.Monoid (mempty)
5  import Data.Maybe (fromMaybe)
6  import Text.Blaze (preEscapedToMarkup)
7  import Text.Blaze.Html5 ((!))
8  import qualified Text.Blaze.Html5 as H
9  import qualified Text.Blaze.Html5.Attributes as A
10
11 import qualified Elm.Internal.Utils as Elm
12 import Utils
13
```

```
14 logAndJS :: String -> String -> H.Html
15 logAndJS name src = getJSPage name src
16
17 logAndHtml :: String -> String -> (H.Html, Maybe String)
18 logAndHtml name src =
19     let elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
20     in
21       case Elm.compile src of
22           Right jsSrc -> do
23               (getHtmlPage name elmname jsSrc, Nothing)
24           Left err -> do
25               (getErrPage name err, Just err)
26
27 getJSPage :: String -> String -> H.Html
28 getJSPage name jsSrc =
29   H.docTypeHtml $ do
30       H.head $ do
31         H.meta ! A.charset "UTF-8"
32         H.title . H.toHtml $ name
33         H.link ! A.rel "stylesheet" ! A.type_ "text/css"
34                                     ! A.href "/misc/js.css"
35         script "/pixi.js"
36       H.body $ do
37         H.div ! A.style "width: 400px; height: 400px; position:\
38                         \ absolute; top: 0; left: 0; opacity: 0;" $ mempty
39         jsAttr $ preEscapedToMarkup jsSrc
40  where jsAttr = H.script ! A.type_ "text/javascript"
41        script jsFile = jsAttr ! A.src jsFile $ mempty
42        embed jsCode = jsAttr $ jsCode
43
44 getHtmlPage :: String -> String -> String -> H.Html
45 getHtmlPage name elmname jsSrc =
46   H.docTypeHtml $ do
47       H.head $ do
48         H.meta ! A.charset "UTF-8"
49         H.title . H.toHtml $ name
50         H.style ! A.type_ "text/css" $ preEscapedToMarkup
51          ("a:link {text-decoration: none; color: rgb(15,102,230);}\n\
```

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

```
90            \a:hover {text-decoration: underline;\
91            \ color: rgb(234,21,122);}" :: String)
92        H.body $ do
93          let js = H.script ! A.type_ "text/javascript"
94              elmname = "Elm." ++ fromMaybe "Main" (Elm.moduleName src)
95              runFullscreen =
96                    "var runningElmModule = Elm.fullscreen(" ++ elmname
97                                                            ++ ")"
98          js ! A.src (H.toValue ("/elm-runtime.js?0.11" :: String)) $ ""
99          case Elm.compile src of
100           Right jsSrc -> do
101               js $ preEscapedToMarkup jsSrc
102               js $ preEscapedToMarkup runFullscreen
103           Left err ->
104               H.span ! A.style "font-family: monospace;" $
105               mapM_ (\line -> preEscapedToMarkup (addSpaces line) >> H.br)
106                       (lines err)
107
108 addSpaces :: String -> String
109 addSpaces str =
110   case str of
111     ' ' : ' ' : rest -> "  " ++ addSpaces rest
112     c : rest -> c : addSpaces rest
113     [] -> []
114
115 js :: String -> String
116 js src = case Elm.compile src of
117            Right js -> "{ \"success\" : " ++ show js ++ " }"
118            Left err -> "{ \"error\" : " ++ show err ++ " }"
```

10.1.6 EmbedMeElm.elm

```
1  module EmbedMe where
2
3  import Mouse
4  import Window
5  import Keyboard
```

```
6  import JavaScript as JS
7  import JavaScript.Experimental as JEXP
8  import Http
9  import Json
10
11 (~>) = flip lift
12 infixl 4 ~>
13
14 clicks : Signal (Time, (Int,Int))
15 clicks = timestamp (sampleOn Mouse.isDown Mouse.position)
16
17 user_id = "1"
18
19 firebaseRequest requestType requestData =
20   Http.request requestType
21     ("https://sweltering-fire-9141.firebaseio.com/dissertation/elm/" ++ user_id
22                                                                      ++ ".json")
23     requestData []
24
25 serialize r = r |> JEXP.fromRecord
26                |> Json.fromJSObject
27                |> Json.toJSString " "
28                |> JS.toString
29
30 toRequestData (t, (x,y)) = {t = t, x = x, y = y} |> serialize
31
32 toRequest event = case event of
33   (t, (x,y)) -> firebaseRequest "post" (event |> toRequestData)
34
35 requests = clicks ~> toRequest
36
37 sendRequests = Http.send requests
```

10.1.7 EmbedMeJS.elm

```
1  module EmbedMe where
2
```

```
3  import Mouse
4  import Window
5  import Keyboard
6  import JavaScript as JS
7  import JavaScript.Experimental as JEXP
8  import Http
9  import Json
10
11 (~>) = flip lift
12 infixl 4 ~>
13
14 clicks : Signal (Time, (Int,Int))
15 clicks = timestamp (sampleOn Mouse.isDown Mouse.position)
16
17 user_id = "1"
18
19 firebaseRequest requestType requestData =
20   Http.request requestType
21     ("https://sweltering-fire-9141.firebaseio.com/dissertation/js/" ++ user_id
22                                                                     ++ ".json")
23     requestData []
24
25 serialize r = r |> JEXP.fromRecord
26                 |> Json.fromJSObject
27                 |> Json.toJSString " "
28                 |> JS.toString
29
30 toRequestData (t, (x,y)) = {t = t, x = x, y = y} |> serialize
31
32 toRequest event = case event of
33   (t, (x,y)) -> firebaseRequest "post" (event |> toRequestData)
34
35 requests = clicks ~> toRequest
36
37 sendRequests = Http.send requests
```

10.1.8 fullScreenEmbedMe.js

```
var firebaseData = new Firebase(
                'https://sweltering-fire-9141.firebaseio.com/dissertation');
var embedMe = Elm.fullscreen(Elm.EmbedMe, {});
```

10.1.9 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();
 }
```

10.1.10 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.

*/

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

// Setting up Pixi
// http://www.goodboydigital.com/pixijs/docs/
var stage = new PIXI.Stage(COLORS[1]);
var renderer = PIXI.autoDetectRenderer(WIDTH, HEIGHT);
```

```
29 document.body.appendChild(renderer.view);
30
31 // Creating the box
32 var box = new PIXI.Graphics();
33 box.lineStyle(1, COLORS[0], 1);
34 box.beginFill(COLORS[1], 0);
35 box.drawRect(0, 0, SQUARE, SQUARE);
36 box.endFill();
37 stage.addChild(box);
38
39 // Setting the position of the box to the middle of the screen
40 box.x = (WIDTH / 2) - (SQUARE / 2);
41 box.y = (HEIGHT / 2) - (SQUARE / 2);
42
43 // When a key is pressed set it as true in the keyState object
44 // Once the key is let go set it back to false
45 var keyState = {};
46
47 window.addEventListener('keydown', function(e) {
48     keyState[e.keyCode || e.which] = true;
49 }, true);
50
51 window.addEventListener('keyup', function(e) {
52     keyState[e.keyCode || e.which] = false;
53 }, true);
54
55 // Start the animate loop
56 requestAnimFrame(animate);
57
58 function animate() {
59     // Left arrow
60     if (keyState[37]) {
61         box.x -= MOVEMENT_SPEED;
62     }
63
64     // Up arrow
65     if (keyState[38]) {
66         box.y -= MOVEMENT_SPEED;
```

```
    }

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

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

    // Draw everything and keep the loop going
    renderer.render(stage);
    requestAnimFrame(animate);
}
```

10.1.11 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
```

```
19
20 main : Signal Element
21 main = lift display position
22
23 delta : Signal Float
24 delta = fps 30
25
26 input : Signal (Float, (Float,Float))
27 input =
28     let vectors = lift toVector Keyboard.arrows
29     in  sampleOn delta (lift2 (,) delta vectors)
30
31 toVector : { x:Int, y:Int } -> (Float,Float)
32 toVector {x,y} =
33     if x /= 0 && y /= 0
34       then (x / sqrt 2, y / sqrt 2)
35       else (x,y)
36
37 position : Signal (Float,Float)
38 position = foldp update (0,0) input
39
40 update : (Float, (Float,Float)) -> (Float,Float) -> (Float,Float)
41 update (dt,(vx,vy)) (x,y) =
42     (x + dt * vx / 2, y + dt * vy / 2)
43
44 display : (Float,Float) -> Element
45 display xy =
46     collage (round areaSize) (round areaSize)
47       [ rect areaSize areaSize
48           |> filled grey
49       , rect squareSize squareSize
50           |> outlined (solid black)
51           |> move xy
52       ]
```

11 Appendix D

11.1 Meeting minutes (sample)

11.1.1 Fri Oct 4 11:15 GMT 2013

Group meeting with supervisor Leon Watts

N.B. READ UP ON AND REMIND YOURSELF OF HCI STUFF (Year 2) AND SOFTWARE ENGINEERING STUFF (Year 1)

- Reading material

In email repsonse to request for FYP meeting, **Leon writes:**

```
Please do a bit of reading around beforehand. Go to the ACM Digital Library
and search on 'user interface programming'.
```

1. ACM Conference on Human Factors in Computing Systems
2. ACM CSCW: Conference on Computer Supported Cooperative Work
3. ACM UIST: Symposium on User Interface Software and Technology

In moodle project page, **Leon writes:**

```
Your project must be related to contemporary developments in Human-Computer
Interaction, and preferably to the part of the HCI world that focuses on
interactive systems for collaboration.
```

1. ???
2. ???

Also In moodle project page, **Leon also writes:**

It normally starts with some user-centred research (observations, interviews, pilot experiment) to ground the problem, carried out concurrently with literature research. The research problem is normally boiled down to something that can be addressed through the production of alternative versions of an interactive system.

This is closely followed by initial design work and the production of a rough but working prototype leading up to Christmas.

After the January exams, my students typically re-scope their research problem, based in the outcome of their initial work, and solidify their implementation ready for a full evaluation in March and April.

Thus, my answers to the questions Leon posed should follow this structure in terms of what I want to get out of it. I can use the above structure to identify **concerns** of potential challenges in each step/combination of steps/step-transitions (e.g. step dependencies, resource procurement)

Also in moodle product page, **Leon also writes:**

Students should prepare for their projects by refreshing their memories about Interaction from CM20216 activities. You should read about HCI in general, and support for collaboration in particular. Look at any or all of the following book chapters:

- Sharp, Rogers and Preece (2007) Interaction Design. hapter 4: Designing to Support Communication and Collaboration.
- Dix, Finlay, Abowd and Beale (2004) Human-Computer Interaction. hapter 14: Communication and Collaboration Models.
- Shneiderman and Plaisant (2005) Designing the User Interface. hapter 10: Collaboration.

11.1.1.1 Leon asked us to answer these questions and bring a notebook:

- Q1. What I hope to get out of my FYP as an experience?

I hope to gain a deep and meaningful understanding of the programmer as a user, as an individual and the context of that individual – e.g. in a software team inside a department, inside a management structure... inside a company.

I hope to use this understanding to determine processes/work-flows that programmers experience in the endeavour of User Interface Design, both from the individual perspective and as a team.

Within these work flows, I wish to identify, in detail, metrics to gauge productivity, in order to measure this in experiments, perhaps doing A/B testing with Elm and some other, perhaps procedural language. This is an example of an objective measure.

I would also like to gather self-reported, more "fuzzy" feedback on user's perception of their productivity – pain points, advantages, etc. they experience in using Language X to produce a UI compared to Language Y (Declarative languages like Elm, etc)

I wish to verify, empirically, the comparisons and claims made on the What is FRP? page of the elm-lang.org website, and those claimed it's research paper (detailing the implementation of Elm, **benefits**, etc.)

In email again, **Leon writes:**

```
The Elm site makes **comparative statements.** That is encouraging because
it sets up opportunities for you to test some of the claims they make, and
to ask new questions about Elm that its proponents may not have considered.
```

These are:

1. "most current frameworks for graphical user interfaces are not declarative. They mire programmers in the many small, nonessential details of handling user input and manually modifying the display."
2. "with FRP, many of the irrelevant details are left to the compiler, freeing the programmer to think about things that matter."
3. "Coding these examples in a traditional GUI framework such as HTML/CSS/JavaScript . would require significantly more work and headache."

4. “Not only is that painful to code, but it also requires broad and deep knowledge of inconsequential things.”
5. “FRP makes tasks considerably easier by taking care of the messy .how. of events, display, and updates.”

- Q2. Where my Project Idea came from (what inspired me)?
- The pain of coding and writing GUIs in PyQt4 while at my last job at Altran
- The joys of coding in Haskell
- The pain of writing GUIs in Haskell
- The joys of coding and writing GUIs in Elm!
- Q3. What are my concerns?

1. Difficulty procuring programmers (users) – specifically those that meet my criteria of not having used a declarative programming language.
2. Difficulty procuring programmers working in a team
3. The complexity/scope of the project – is it enough for a FYP; is it too much?
4. Looking at the production of User Interfaces using a programming language, there are many variables – how will I devise an experiment to minimise this and isolate a variable so that I can make some causal/correlational conclusions?
5. Concern regarding the dependency of a subsequent part of the project on a previous step – this is inherent of all projects, though.

11.1.2 Tue Oct 30 09:15 GMT 2013

Individual Meeting after Proposal hand-in

Our discussion centered around the direction I wish to take following my Project Proposal.

11.1.2.1 AB Testing of the language 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).

11.1.2.2 Test just the paradigm

He also suggested a potential experiment to test just the paradigm, eliminating the IDE from the experiment above. Perhaps for a Pilot study.

11.1.2.3 Experiment process

1. Study question (e.g. Is it easy?)
2. Measurement concept (e.g. "Easy")
3. Operationalisation – taking a measurement concept and mapping it to something concrete (e.g. if completing a pre-defined task the user must complete takes < 5 steps, it is 'easy' – we can then compare instances of these studies given our definition of easy). This is much like mapping a design to an implementation, and there is a risk of losing information, or ending up with a mismatched concrete instance that does not represent the concept we wish to convey.
4. Do another operationalisation of our measurement concept – this allows us to get a different perspective of the same concept. (e.g. if total length of pauses during a 1 hour experiment is < 10 minutes, it is 'easy'). We do this to get 'coverage' of the measurement concept. It is a form of cross validation. If we see an overlap in the correlational results after analysis, we can make a stronger assertion that e.g. "language A is easier than language B.". The idea I am describing here is methodological decision-making.
5. Predict what will be the likely results of our experiments on the operationalised measurements. This is "feed forward validation".
6. Do the experiement.
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 infact objectively true/false".

11.1.2.4 Pilots We also spoke about ideas for pilot studies – asking "What might be surprising insights into declarative programming languages for User Interface Design – the case of Elm?".

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

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

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

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

11.1.2.5 Actions for next meeting Devise a Pilot study, answering these 3 questions:

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

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

11.1.3 Wed Mar 25 14:30 GMT 2014

(Several meetings undocumented)

TODO: Refer to notes in Diary for previous entries.

11.1.3.1 Progress since last meeting

Discussed findings from analysis of pilot study

11.1.3.2 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.

11.1.3.3 Model Adjustment 1

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

11.1.3.4 Observation 2

- Sifting through 1-hour+ of video data capture for incidences of cognitive load is *HARD!*. Is there some programmatic way of narrowing the video data to points of interest?

11.1.3.5 Model Adjustment 2

- Track 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.

- Leon suggested a further refinement of this idea in order to further narrow 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). His refinement was to 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.

11.1.3.6 Further discussion We then discussed some questions that might lead my direction of study in the next steps of my research:

- 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.

11.1.3.7 Actions

1. 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.

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