Accessible Pointing Approaches for Web Applications [Proposal]

Alexander Birch

October 2013

1 Problem

1.1 Problem Context

Some domain knowledge will be divulged in this section, prior to describing the problem under investigation. Namely, the case for accessible navigation alternatives' necessity, and also the relevance of the web in contributing to the size of the problem.

Need for accessibility Computer users are not uniform in their capabilities and preferences. For this reason, users differ in the input mechanisms they choose for interacting with software. Some users may prefer to perform tasks using a keyboard rather than using a mouse, for example. Commonly mouse and keyboard will be recruited together. However some users' input strategies are constrained by factors such as disability: occupational syndromic conditions such as Repetitive Strain Injury (RSI) can preclude the ability to use a mouse, keyboard or both [1]. RSI and related disabilities are growing in prevalence, representing by some estimates 22% of people [2]. This disability accounts for one third of workers' compensation costs in the US private industry [3].

When the user cannot or will not use the primary input scheme of the software, it becomes necessary to provide alternative input modes. For simple interfaces that use standard components, keyboard and mouse can be both supported implicitly without additional development effort. Conversely, complex interfaces or ones involving non-standard components require explicit treatment from the developer to ensure that all funcitonality can be accessed. In cases where accessibility is not a priority or concern of the software developer, users can be left unable to use said software.

Rise of web applications Even as early as 2001, software made the leap to the web [4]. Services such as email, banking and word processing are being produced as web applications, due to a few advantages:

- high cross-platform, cross-device support & penetration
- centralized maintenance and upgrade
- centralized storage of user data
- (consensual) monetization of user information, telemetry
- scalability (down and up)
- zero end-user installation
- redundancy

Email clients such as Outlook [5] benefit from this; emails can be accessed from any web-enabled device without any need for installation, all users can be kept on the latest version without effort on their part, and user data can be analyzed to improve content filters or provide relevant advertisement. At peak times of operation, more servers can be recruited, and vice-versa, such that the cost of operations can be finely controlled based on usage.

1.2 Problem Description

This transition away from native applications is leading to sacrifices in accessibility. Web applications run within a native application (the browser), so there is conflict when it comes to the following accessibility paradigms:

- menu actions
- keyboard shortcuts
- tab order
- citizenship amongst other applications

Even on the conceptual level, there are problems with using the web to serve applications: a web application tries to fit a full application interface inside a web browser's (already saturated) interface. It is a second-class citizen; a user cannot 'switch application' to it, only trawl for it amongst the tabs and windows of some web browser.

Additionally, web applications are often guilty of foregoing standard UI components in favour of a bespoke UI made in JavaScript or CSS. This can be used to deliver impressive effects like transitions, or serve complicated shapes of interactive content or buttons. But this freedom makes it all too easy to produce inaccessible elements, such as buttons that can't be pressed or focused using the keyboard.

1.3 Existing Approaches

There exist at present a few categories of solution to the problem of web accessibility:

- following web development guidelines to produce a standards-compliant website
- using web technologies to create a non-standard interaction behaviour
- making the web browser behave in a non-standard manner

1.3.1 Recommended web practice

Guidance is provided on how to develop accessible web applications, in the form of the World Wide Web Consortium (W3C)'s Web Content Accessibility Guidelines (WCAG) [6]. A description and analysis on its recommendations for keyboard access follows:

Pointing with access keys Access keys are one paradigm used for keyboard accessibility in the HTML5 specification [7]. Hotkeys can be assigned for the focusing of interface elements, used in conjuction with modifier keys. This system allows the user to jump instantly between distant or unrelated interface elements. However, its usefulness is predicated on the discoverability and availability of said key bindings.

It is hard to avoid conflicts with existing shortcuts used by the user's accessibility technologies, since there is no standardisation between browsers as to which modifier keys to use [8]. Recommendation exists that access keys be avoided altogether for this reason. Another problem with access keys is that they rely on explicit implementation by developers. They also require learning of a layout, and although attempts have been made at increasing standardisation [9], that standardisation is not prevalent.

Making interface tab-accessible It is difficult to develop a tab-accessible web application [10]. Since the only pointing expression assumed is the tab key, layout must be linearized. Not only is this hard to design and develop for (requiring explicit effort and markup changes), the end result is not that impressive; content that is far away in the tab order unavoidably requires repeated hammering of the tab key (making it far harder to reach with the keyboard than with a pointing device). Plus linearization doesn't suit web content, which often has navigation bars, side bars, feeds or some similar multi-dimensional layout.

Short-circuiting tab journey with skip-links An exception to the content linearization problem is skip links, which can short-circuit the tab journey around a page by allowing the user to choose alternative insertion points. These again require effort and understanding from the developer. It is hard to provide a suitable set of skip links, and again the best case is still a hamfisted pointing experience, where the user can't predict where they are going to or from.

Conclusion Even a website designed with accessibility standards in mind can be quite hard to navigate; web standards prescribe only a very limited amount of keyboard expression, so meeting these provides a very limited experience.

1.3.2 Using web technologies to augment accessibility

Whilst it is possible (to an extent) to create key bindings in web content (through access keys [section 1.3.1], or through JavaScript event libraries like MouseTrap [11]), for the most part web applications do not make use of this. Possible reasons for this are:

- no agreed standard exists for how websites should provide keyboard shortcuts
- web application shortcuts are likely to conflict with native application's shortcuts
- existing native application shortcuts are hard to tiptoe around, since any browser or extensions could be used
- some devices (for example smartphones) do not have the same expressive keyboard capability
- would only be available on those websites that chose to implement it
- websites not guaranteed to implement in a uniform way

1.3.3 Changing the web browser

Adding accessibility on top of web content can be achieved through a web browser extension. This can change the way content is treated so as to work better for the preferred input mode. In some implementations, the browser is extended through the use of web technologies, which can coexist with the content of the website. One such example is Type-Ahead-Find [12], which allows users to move keyboard focus to hyperlinks by typing the content of said hyperlink. Another example is Stylish [13], which allows users to redesign a website to suit their needs.

Web browser extensions solve some of the problems associated with using web technologies alone to augment accessibility: key bindings are uniform across the user's browsing experience, so a user can configure non-conflicting shortcuts. The solution is also easy to distribute via web browser extension repositories.

2 Proposed Solution

2.1 Approach

Half the problem is the limited amount of expression afforded to the keyboard; if tabbing is the only navigation method offered, then there is a severe constraint on what can be achieved. A means for utilising more keys is necessary. Unfortunately websites cannot provide this, as they cannot predict which key bindings will be free on the user's computer. Nor should they try: each website would have to be learned individually.

The burden of reserving key bindings should fall on the web browser, or an extension thereof. Here it can be controlled and configured, as well as uninstalled should it cause conflicts. A novel shortcut scheme would be used to avoid conflicts, such as sequences rather than chords, or invoking the extension explictly as a precursor to action.

The rest of the problem is that accessibility is presently found only on websites that design for it. It is easy to blame developers for making websites that only work with a mouse or touchscreen, but the real problem here isn't that the website is designed wrong, it is that the keyboard isn't powerful enough to cope with these pointing situations.

Again, a solution where the keyboard is augmented via the browser is desirable. Increasing keyboard control diminishes the number of problem situations that exist for it.

2.1.1 What is novel about the approach?

Previous attempts (such as suggested best practice, described in [section 1.3.1]) at introducing keyboard accessibility to web applications have been focused on redesigning the website. The novelty of our approach is that we redesign instead the manner of traversing the website. Pointing will be designed from the ground-up to expect and support complex non-linear layouts using non-standard components.

2.1.2 Why does it make sense?

The approach embraces the way the world is making web applications (complex, bespoke, mouse-optimized interfaces), and attacks the problem that necessitated non-standard interfaces: that web applications are second-class

citizens, described as 'content' in a pane of a native application. The pointing system will exist inside the context of the web application rather than the native application, and will acknowledge the web interface as the highest-level citizen. This makes for a clearer paradigm.

3 Methodology

3.1 Deliverables

The goal of this work is to produce an accessibility layer to sit between a web browser and web content, which would augment the accessibility of said content. The primary purpose of the accessibility layer is to achieve effective pointing via a sequence of keystrokes.

The system will aim to solve better the pointing situations to which tab navigation is less suited. Pointing contexts such as form navigation, at which tab navigation is satisfactorily effective, need not be a focus of the novel system. Ideally the novel system would co-exist with tab navigation, allowing the user to fall back on tab navigation in cases where it is suitably effective (or more effective).

3.1.1 Software solution

The choice of browser for implementation is immaterial; the same logic can be applied to any extensible browser. This work will concern itself with Google Chrome, but the choice is arbitrary. The accessibility layer will be implemented in the form of a web browser extension, as this meets the criteria of sitting between browser and content, persists between page navigations, and doesn't incur work on the website developer's end.

3.1.2 High-level Requirements

This section defines the requirements of the software solution.

Priorities are as follows:

1. High- primary goal of work
2. Med- secondary goal of work
3. Low- goal not essential for work, but appealing nonetheless

Non-functional requirements

- **High** Software is easy for users to obtain and install
- **Med** Software features are easy to invoke

- **Low** Software is a good citizen (conflicting key bindings should be reconfigurable or non-existent)
- **Low** Default settings of software should cause minimal conflict in key bindings versus web browser
- **Low** Default settings of software should cause minimal conflict in key bindings versus accessibility technologies
- **High** Software should be resilient to bad markup
- **High** Software should work on websites that are not designed for accessibility

Functional requirements

- **High** Improve lookup time (compared to tabbing) of any given interface element
- **High** Improve lookup time (compared to tabbing) of interface elements far away from currently focused element
- **Med** Detect related interface elements, such as repeating elements in a list
- **Med** Provide a means to travel between related elements
- **Low** Improve lookup time (compared to tabbing) of related interface elements
- **Low** Improve lookup time (compared to tabbing) of unrelated interface elements

It must be noted that this pointing system is being designed for a different problem domain than tabbing; tabbing behaviour seeks the next focusable element found in a depth-first trawl of the webpage markup (excepting 'tabindex' overrides, which can be used to rewrite the order or move elements in and out of the tab order). This aspect of tab behaviour makes it very well-suited to traversing forms, where ideally the next element for interaction will be the next one in the markup. Problematically, the websites mix paradigms and use tabbing for pointing, which it is less good at; outside of the context of a form, it is far less likely for the desired element to be adjacent (or even close) in markup.

Since there are a variety of ways in which a novel pointing approach could be achieved, a study will likely be necessary to decide which is the most effective to pursue. A small group of users will be asked to provide feedback on the various proposed versions of the system, to see which is preferred.

3.2 How novel approach will be evaluated

This pointing system is intended to improve upon the pointing system offered by tabbing. The planned improvements pertain to lookup time for interface elements: that is to say, the number of key presses required to travel to some interface element from the user's initial state.

At this stage, some distinct cases of lookup to consider are:

- (From initial state) Lookup time of arbitrary interface element
- (From some focused element) Lookup time of element distant in markup
- (From some focused element) Lookup time of spatially distant element
- (From some focused element) Lookup time of semantically related element; for example, the next element in a list
- (From some focused element) Lookup time of semantically unrelated element; for example, traveling from a navigation bar to a content item

The system will try to support as many of these navigation types as possible. Evaluation will only be pursued on those navigation cases for which engineering is satisfactorily completed.

The system aims to be an effective, efficient and fast method for pointing. The system aims to compete primarily with keyboard-based tab-navigation. It is also intended to navigate interfaces that are designed for mice, so it is worth comparing its performance to mouse navigation also. Touch navigation deserves a mention, as it is an emergent pointing-based navigation method similar to the mouse. We chose to leave comparisons with touch-based systems out of the scope of this work.

Use of Metrics Metrics are required to evaluate whether the new system delivers on its goals, and to draw comparisons with the performance of existing systems. Care has to be taken to ensure the use of valid comparisons. Keyboard-based navigation methods can be easily compared with each other, since they can both be distilled to a sequence of keypresses. It is, however, harder to compare keyboard navigation to mouse navigation, as mouse pointing is done as a single continuous action (movement of the mouse) rather than a discrete sequence.

Efficiency Efficiency will be considered on a high level to be defined as:

$$\frac{\text{work input by user}}{\text{value gained by user}}$$

That is to say, how much effort is required to complete a task. Since the value to the user of a given navigation task is highly subjective, value shall

be an arbitrary constant, 1. It must be warned that efficiency comparisons between differing classes of task will be invalid, since their relative 'value gained by user' is only really equal within their task class. In the case of keyboard, 'work input by user' can be measured in terms of how many inputs are required to complete a task. This also correlates to physical exertion demanded of the user, as it is a counter for how many inputs they need to make. The metric should not be used to draw comparisons against mouse efficiency (as explained in [section 3.2]). Since no convenient analogue for 'number of actions' exists for mouse pointing (only one action is required), efficiency could instead be measured in terms of how much effort is required in steering the mouse. Fitts' Law [14] can be used to model pointing times. Since the user is working for the whole duration, time in this case is a measure of effort. The expectation set by Fitts' Law is that a necessary minimum amount of time will be spent in acceleration and deceleration to the target (the deceleration time being longer for smaller targets), and that longer journies take more time (because time is either invested covering distance at low speed, or increasing said speed and subsequently incurring a larger penalty in the deceleration stage). Care needs to be taken to minimize experimental error when measuring time, as the timer needs to be started and stopped on the right cues. Assuming a constant reaction time, this noise worsens (proportional to the signal) for shorter times. Thus steering experiments should be made long enough to minimize the error as a proportion, and/or repeated and averaged to reduce random error.

Speed Speed of the pointing mechanism will be defined as "time taken to complete task". This can be measured trivially, using a stopwatch (or more accurately by timers provided by the web technologies with which the user interacts). This is a metric that can be used to compare effectiveness of the novel system against tab-navigation and against mouse navigation. It can also be used to contrast the effectiveness of the three systems in each class of navigation. As explained in [section 3.2], precautions need to be taken when designing and measuring experiments that hinge on time.

Effectiveness Effectiveness is the capability of completing a navigation task. In some cases it will be objectively impossible to complete a task (for example if the controls lack events for keyboard interaction, and keyboard interaction is the mode of navigation). In other cases it will be subjectively difficult; the user might not know how to use the system, or the system might work for the task but require an unreasonable amount of effort. In difficult or impossible trials, evaluating all the factors of the system's performance becomes more complicated (as time can tend to infinite, where the user aborts the task). User feedback is useful to articulate what the problems are in such situations. Participants can explain in writing whether (and why)

the system is realistically useful for the anticipated pointing situations.

Learnability As users become experienced in using the system, their performance will converge on that of system experts. Their efficiency could improve (as they figure out how to minimize input work, or make navigation decisions that maximize output value), and/or their speed could improve (as a result of fewer actions or faster performance of said actions). This transition can be measured with subsequent measurements of speed or efficiency (on the same participant). Learnability of the novel system can be contrasted with learnability of the existing systems. The metric can also be used to ensure that users achieve some baseline level of competence with each system before measurements are taken (such that comparisons are fair). A feel can be garnered for how 'intuitive' the system is, by observing users' first-time performance. However with the exception of the novel system, it might be hard to find first-time users of the other pointing systems, as they are mature and widely available.

Obstacle courses for the systems to compete on will be real-world examples chosen to embody well the navigation class at hand. Experiments can be workflow-based, and filled with enough tasks to give suitable confidence in the measurements.

3.3 User participation

Users will be needed for testing the performance factors of the system defined in [section 3.2]. Random selection of users will meet the needs of the anticipated experiments, except for where comparisons to expert performance is required. Users will declare their competence in factors that might be significant to test results. Identification of participants will not be required at any point.

3.4 User confidentiality

This work responds to its institution's 13-point Ethics Checklist as follows:

1. **Have you prepared a briefing script for volunteers?** Briefing scripts will be provided to participants prior to any experiments. The purpose of the activities will be explained, as will be the destiny of the data produced by them.

2. **Will the participants be using any non-standard hardware?** As one goal of this work is to produce a highly-available system, hardware interaction is likely to be limited to keyboard & mouse work. Experiments involving gestures are not out of the bounds of possibility either, but no risks are anticipated with such interactions.

3. **Is there any intentional deception of the participants?** Deception will not be necessary for these experiments.

4. **How will participants voluntarily give consent?** Consent forms will be given to participants immediately preceding experiment participation.

5. **Will the participants be exposed to any risks greater than those encountered in their normal work life?** No; these experiments involve only safe human-computer interaction.

6. **Are you offering any incentive to the participants?** No incentive is expected to be offered to participants. Payment would not be used to increase the participant's risk of harm.

7. **Are any of your participants under the age of 16?** No.

8. **Do any of your participants have an impairment that will limit their understanding or communication?** No explicit selection will be made for such participants. The sampling will be random.

9. **Are you in a position of authority or influence over any of your participants?** No.

10. **Will the participants be informed that they could withdraw at any time?** Yes; this will be explained in the introductory script.

11. **Will the participants be informed of your contact details?** Yes; the contact details of the investigator and supervisor as part of the debriefing.

12. **Will participants be de-briefed?** Yes; an introductory script will be provided.

13. **Will the data collected from the participants be stored in an anonymous form?** Yes; data will be stored anonymously and securely.

NAME: Alex Bird

SUPERVISOR (IF APPLICABLE): _____

SECOND READER (IF APPLICABLE): _____

PROJECT TITLE: _____

DATE: _____

4 Potential impact

If successful, this work will make web applications more accessible to users who can't (or won't) use the mouse as their pointing device. The software layer would create accessibility in websites which might previously have been difficult to use, or wholly unusable. For businesses serving web applications, that would otherwise be liable for excluding users (or would have to dedicate resources to remaking their websites), this would be a huge step forward.

It is hoped that the concept of 'pointing via key sequences' would be easily transferable to gesture sequences (in 2D/3D space), or spoken word sequences. This would allow the concept to have impact outside of the realm of accessibility, contributing to usability also – for example, in living room or mobile applications, where keyboard & mouse are not available for web browsing.

5 Roadmap

The system will go through the software lifecycle:

- detailed specification
- design
- development & testing

Then, the system will undergo study. Two studies are planned:
Study 1: Study of System Implementation Options
This study is to decide, should there be more than one option for how to make the system, which one is most well-received by users.
Study 2: Study of System Against Other Systems
This study is to evaluate the novel system and contrast it with the other navigation systems.

Each study will comprise:

- experimental design
- materials preparation for experiment
- experiment run
- results write-up

The roadmap for the project is described in fig. [1]. Should time pressures become apparent in the design/development phase, Study 1 could be omitted, with the navigation mode for the novel sytem then being chosen without user consultation.

6 Project Agreement

Student Signature: Alex Bird _____

Supervisor Signature: _____

7 Terms

access keys a means for users to travel between distant or unrelated interface components using keyboard shortcuts. 3

key binding mapping of keys, or combinations thereof, to actions. 3–5, 7

RSI Repetitive Strain Injury. 1

W3C World Wide Web Consortium. 3

WCAG Web Content Accessibility Guidelines. 3

web browser extension (in the context of the Google Chrome web browser): extensions are small software programs that can modify and enhance the functionality of the Chrome browser. You write them using web technologies such as HTML, JavaScript, and CSS.. 4–6

References

[1] Shari Trewin and Helen Pain. "Keyboard and mouse errors due to motor disabilities". In: *International Journal of Human-Computer Studies* 50.2 (1999), pp. 109–144.

[2] Eliana M Lacerda et al. "Prevalence and associations of symptoms of upper extremities, repetitive strain injuries (RSI) and'RSI-like condition'. A cross sectional study of bank workers in Northeast Brazil". In: *BMC Public Health* 5.1 (2005), p. 107.

[3] Ann E Barr and Mary F Barbe. "Pathophysiological tissue changes associated with repetitive movement: a review of the evidence". In: *Physical therapy* 82.2 (2002), pp. 173–187.

[4] Athula Ginige and San Murugesan. "Web engineering: An introduction". In: *Multimedia, IEEE* 8.1 (2001), pp. 14–18.

[5] Microsoft. *Outlook Web Client*. URL: http://www.outlook.com.

[6] W3C. *Web Content Accessibility Guidelines (WCAG) 2.0*. 2008. URL: http://www.w3.org/TR/WCAG20/.

[7] WHATWG. *HTML5 Specification - User Interaction*. URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/editing.html#the-accesskey-attribute.

[8] Web Accessibility in Mind. *Keyboard Accessibility*. URL: http://webaim.org/techniques/keyboard/accesskey.

[9] CabinetOffice. *Web page navigation*. 2002. URL: http://webarchive.nationalarchives.gov.uk/20100807034701/http://archive.cabinetoffice.gov.uk/e-government/resources/handbook/html/6-6.asp.

[10] Willian Massami Watanabe, Renata PM Fortes, and Ana Luiza Dias. "Using acceptance tests to validate accessibility requirements in RIA". In: *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*. ACM. 2012, p. 15.

[11] Craig Campbell. *Mousetrap: A simple library for handling keyboard shortcuts in Javascript*. 2012. URL: http://craig.is/killing/mice.

[12] tokland. *Type-ahead-find browser extension*. URL: https://chrome.google.com/webstore/detail/type-ahead-find/cpecbmjeidppdiampimghndkikcmoadk?hl=en.

[13] userstyles.org. *Stylish browser extension*. URL: https://chrome.google.com/webstore/detail/stylish/fjnbnpbmkenffdnngjfgmeleoegfcffe?hl=en.

[14] Paul M Fitts. "The information capacity of the human motor system in controlling the amplitude of movement." In: *Journal of experimental psychology* 47.6 (1954), p. 381.

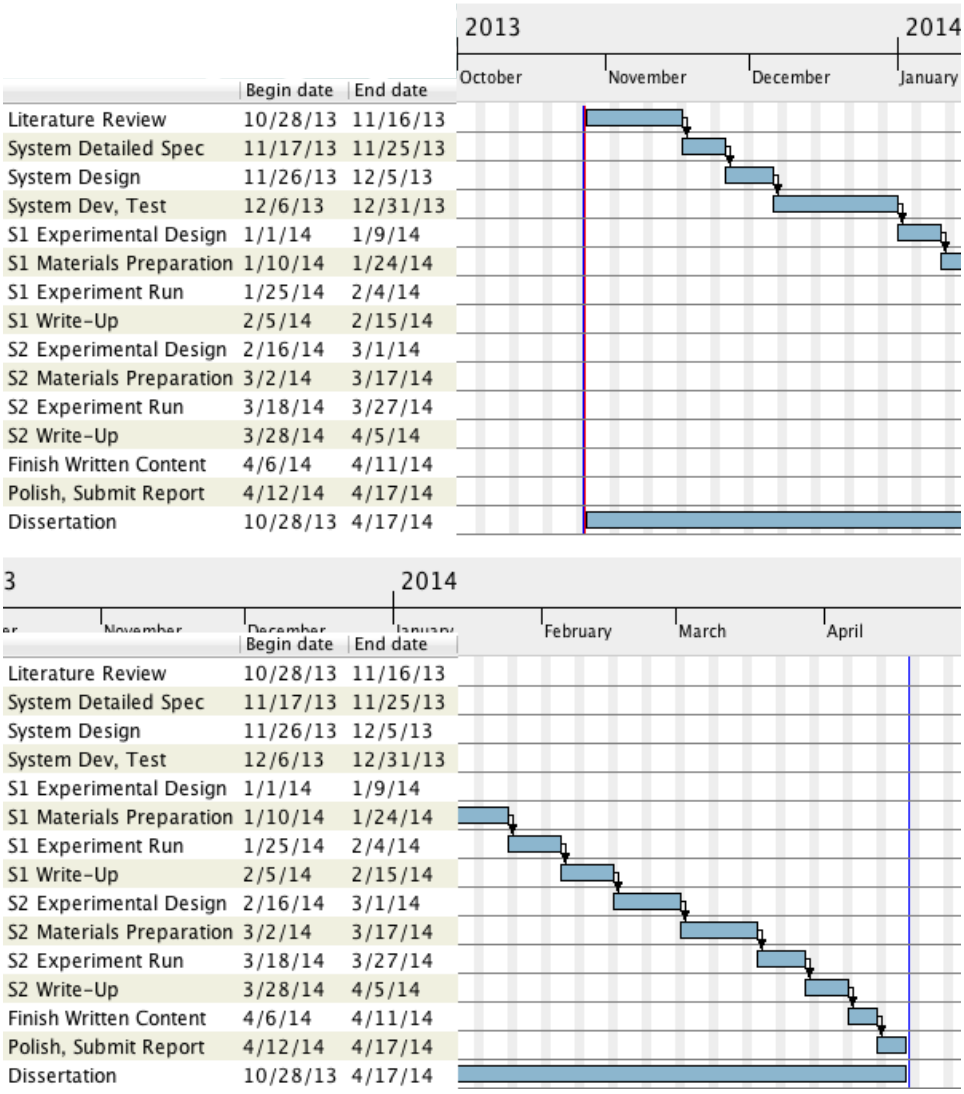

Figure 1: Roadmap of Work