Group 27

Software Quantification of Reading Patterns using Consumer-Grade Gaze Tracking Hardware

Progress Report – October 24th, 2014

Prepared By:
Paras Vora
Maeve Woeltje
David Young

Prepared For:
Dennis Barbour, M.D. Ph.D.
Jonathan Silva, Ph.D.
Wandi Zhu, Ph.D. Student

Washington University in St. Louis Department of Biomedical Engineering
BME 401: Biomedical Engineering Design

1.0 Need and Project Scope

1.1 Project Need

There is a current need for a consumer grade system to provide a user with information about how they read. Reading analyses using eye-tracking hardware have been conducted in clinical settings for research purposes. However, there is no intuitive, affordable software that provides feedback on a subject’s reading thoroughness to a teacher, parent, student, employer, or other interested individual. Thoroughness feedback, in this sense, is an analysis of reading patterns including notes on changes in speed, identification of apparent distraction, and a marking of segments where a reader skimmed, skipped, and re-read text. A software suite designed to measure performance through an analysis of reading patterns has the potential to improve the education process for interested consumers. Teachers would be able to identify whether or not children are actually reading their assignments. Employers could tell if employees thoroughly read a memo, set of instructions or contract. Furthermore, the thoroughness analysis could allow the interested party to identify portions of a document that readers are struggling with. Teachers, parents, and students would be able to track the improvement of a child’s reading performance. With a consumer size user population, gaze tracking data could be collected in greater quantities than clinical trials, to be used for big data analysis. An affordable gaze tracking solution designed specifically for reading would address the need for personal diagnostics to break into the consumer reading market.

1.2 Project Scope

This project aims to create an affordable consumer-friendly software toolset that will work in conjunction with existing gaze tracking hardware to quantify and evaluate reading patterns as well as store user session histories. Using experimental metrics, it will provide instant feedback to the user outlining their reading performance. Reading performance will be defined by a user’s reading speed, a breakdown of document sections skipped or skimmed, possible indications of distraction or boredom, and a list of sections or vocabulary words where the user somehow struggled (as indicated by prolonged fixation and redundant reading). This feedback will be presented in an intuitive package with the potential for a user to compare results to previous sessions or global averages. A user's progress history will be stored for recall at a later point in time, and potential use in big data analysis. This project will not include the design of any hardware, as there already exist eye-tracking devices on the market. The project will not serve to replace any existing clinical-grade diagnostics.

2.0 Design Schedule and Team Responsibilities

The specific schedule and team responsibilities are located in the Appendix A in the form of a color-coded Gantt chart. The color-coding describes each individual's task.

3.0 Specific Design Requirements

Useful data analysis and output representations	Visual representation (Heatmaps), Speed, Skimming and Skipping, Rereading and re-referencing, Extended Focus, Points of Distraction
Required Prior Training	None, User Intuitive step by step instructions, approachable interface and menu system, child friendly
Calibration	User intuititve step by step graphical calibration procedure 12 points of fixation during the calibration procedure Time required for attempted calibration < 30seconds
Software Adaptability	Compatible with text input provided in form of .txt file Support for landscape monitor orientations up to 23" in diameter Support for resolutions up to 2560x1440p Support for Full Screen Mode Only
Safety	Display backgrounds that reduce eye fatigue and strain Dark text (black) on light backgrounds (white)
Storage Requirements	Average session data size <5MB Program size <500MB
Completion Date	December 9th 2014
Ease of use	Intuitive GUI and menu system Average GUI task completion rate > 80% System Usability Scale > 68

4.0 Design Alternatives Analysis

Every level of software development involves choices between alternate implementations. Because programing languages can accomplish tasks through a seemingly infinite variety of methods, the development decision tree's high-level branches concerning software function, data-structure selection or algorithm design, extend down to the infinite sub-branches of syntactical implementation. By postponing the low level details to the implementation phase, it is easier to partition a portion of the decision tree to accomplish during design. Outlining this finite and manageable set of decisions during the design phase will be instrumental during implementation, acting as a blueprint.

Most of these crucial decisions involve selecting a path to implement a section of the software architecture. Such early decisions play an important role in avoiding future roadblocks, optimizing the speed of the program, and ensuring different sections of the software will interact efficiently. Often there are intriguing alternative directions for each individual section, but selecting an option that cooperates with the grander architecture is key.

The "big picture" view of the software architecture is shown in Appendix B. It is not an exact map of class structure, but serves as a general flow chart/blue print for the design process. From this outline five important large-scale decisions must be made before implementing the software. The five sections include the methods of processing user input, the G.U.I, storing data and forming a database, data analysis and output representations, and lastly text layout and collision detection.

4.1 Processing User Input

The EyeTribe tracker (hardware) connects to a running server on the user's computer. EyeTribe's provided SDK and drivers for the tracker handle the communication between the hardware and server. The method of connecting the program to the server and retrieving data in real time is something that must be determined.

The first option is to write a processor from scratch that will connect to the server, retrieve data in the form of JSON packets and parse the packets for the gaze coordinates and timestamp. While the relatively universal JSON parser might be easy to implement, the server is EyeTribe's own creation and the difficulty of communicating with it is unknown and poorly documented.

Luckily, EyeTribe also provides a Java TET (The Eye Tribe) API for communicating with the server. Having the client and server designed by the same entity helps to ensure compatibility. Additionally, the API contains a wealth of extra functionality beyond simply retrieving data, including calibrating the device. By using the existing TET API, the development team can focus their limited resources on the software functionality without creating a complicated interface from scratch.

Because of the overwhelming benefits and lack of any substantial drawbacks, the TET API will be used to communicate with the server and receive gaze data.

4.2 Graphical User Interface

Many requirements should be considered in choosing a GUI. Due to the fact that the solution should work on multiple platforms, the choices are limited to SWT, Swing, and JavaFX.

SWT was created for Eclipse, one of the premier Java-focused Integrated Developer Environments (IDEs) today, and also coincidentally the same IDE used by the development team. The included Java library is based on native OS components, which allows for a more professional experience on each platform. Additionally, there exists support for displaying and analyzing styled text, which is important to ensure project success. On non-Windows operating systems, SWT is sometimes unstable due to bugs not resolved in the latest version of Java. There is no customizability of the look and feel due to the native bindings present, making user-friendly interfaces more difficult to create.

Swing is the oldest and most supported GUI framework for Java applications. This framework is the most familiar to the team, as members have experience through academic coursework. This allows the team to utilize previous work and experience. Additionally, the framework follows the Model-View-Controller (MVC) methodology, making it easy to add new features to existing components. The object tree approach allows increased flexibility when creating an interface. Unfortunately, Swing is hardware accelerated, so the interface is slow on older devices. There are bugs that will not be resolved in newer versions of Java since Swing is no longer supported. Luckily, JavaFX, a replacement for Swing, promises to become the de facto standard in building Java applications.

JavaFX, created by Oracle, replaces Swing and AWT as the default GUI framework for Java. It is bundled in the latest version of Java, and named JavaFX 8. With this new version comes a very useful feature for the project – rich text support. This will allow for easy checking of entering and exiting words being read. JavaFX user interfaces will match the look and feel of each platform, but also have a native HTML/CSS/JavaScript rendering engine, making it easy to customize for an aesthetically pleasing design and responsive interface. JavaFX also uses an object tree framework, and the data picker widget will make output visualization very simple. Additionally, there is a visual editor, allowing for very rapid prototyping. However, as with each option discussed, there are downsides. Since the team is unfamiliar with this framework, there would be a steep learning curve. Also, JavaFX is a relatively new framework, and although full documentation exists, developer community support is lacking.

The team will use JavaFX 8 as the GUI framework going forward. This is the newest version of JavaFX, and is included in the latest version of Java shipped on millions of devices today. JavaFX provides the familiarity with object tree approaches, a native web rendering engine, and a visual editor, called Scene Builder, allowing for quick and iterative designs. Oracle backs this framework as the replacement for Swing and AWT, signifying long term support. Additionally, JavaFX can render both Swing and SWT within its framework.

4.3 Data Storage

Because the application must store data that persists through multiple sessions, some form of database storage is necessary. The types of data that will be stored will consist of user information, a user history of gaze data from the eye-tracking device, as well as any computed analysis data during or after the session. Many feedback output representations presented to the final user will require analysis of gaze data. For example, a user may prefer a list of all words they struggled with, but also sorted by how much the user struggled. In scenarios such as this, the team requires a database that supports rapid querying and sorting. The team also prefers an engine that can be embedded into the application, so as to prevent extra installation procedures for the user. Based on these two requirements, the following three database engines stand out: HSQLDB, H2, and JavaDB.

As stated before, all of these database engines are ‘embedded’ meaning that they can be package with the application with minimal effort. HSQLDB is fully multithreaded, making it one of the fastest databases listed, almost 34 times as fast as MySQL, the most popular database in the world. The software is very lightweight, taking maybe 1-2MB of memory. HSQLDB is extremely stable, and supports SQL queries, allowing for easy retrieval of queries such as the example above. This database engine is not great for retrieving or manipulating data outside of the application in which it is embedded, and slows down proportionally to the size of data. HSQLDB does not support Hash or Full-Text Indexing, both of which would be useful considering the amount of text being used in our project.

H2 is another embeddable database, but is also made by the original developer of HSQLDB, rewritten from the ground up completely in Java. Out of the three database engines listed, it has the smallest footprint at 1MB. The databases and tables can be used in-memory, i.e. while the application is running, or disk-based, which will allow for persistent storage. Full text search is actually implemented in H2, which will become very useful when querying for words and paragraphs. There are a few downsides to the H2 database engine, in that it has the same pitfall as HSQLDB with increasing database size. Additionally, H2 is relatively new. Although documentation is abundant, developer community support is not up to par with other database engines.

JavaDB is open source, from Oracle, who as stated earlier maintains Java. This guarantees widespread support from Oracle itself as well as the community. JavaDB fully supports SQL programming APIs, which the team is familiar with. JavaDB can also allow clients to connect to databases over the internet, which could provide useful features such as benchmarking reading performance across users. JavaDB is the slowest engine compared to all others listed, and there is no hash or full text indexing available, making it hard to justify its use.

The team has chosen the H2 database engine for the project. This option provides the best performance, the smallest footprint, and useful features that will be supported longer than the other options.

4.4 Forms of Data Output

The data outputs were chosen based on the expected needs and interests of the consumer. While it is possible to create a wide range of outputs, the ones included in the software should fit within the project scope of creating a consumer accessible device. As a result, the outputs are reading pattern analysis measurements understandable to the average consumer, not literacy or comprehension metrics used by readings specialists. Furthermore, in order to enhance the academic functionality of the software, the outputs all are under some level of direct control by the user. The final data outputs chosen are: visual representation of gaze duration, speed, sections skimmed and skipped, rereading and re-referencing, extended focus, and points of distraction.

Some other possible output metrics include the total number of saccades, average saccade length, average fixation length, and fraction of non-forward saccades. However, these were ultimately discarded because they do not provide the user with practical information that could identify potential reading weaknesses. Rather, the chosen outputs utilize aspects of these metrics to provide the user with a more intelligible interpretation of their reading patterns.

4.4.1 Visual Representation

The simplest output for a user to interpret is a visual representation of their reading patterns, including their eye movement and how much time they spent looking at each section. The most useful forms of representation are heat map overlays, standard visual aids such as bar charts and video playback of gaze data. Of these three, a heat map is the most efficient and direct form of conveying gaze data. Any graphical representation, such as a bar chart, could not effectively show eye movement. Furthermore, it would be difficult to break up the data into sections to graph and even more difficult to interpret. A video would show eye movement effectively, but it would take a long time to play back and provide the user with information that can be interpreted at a glance using a heat map. The heat map could be enhanced with lines across the page showing eye movements, colored with a gradient that indicates how long the portion of the text beneath was gazed at. Through a quick glance, users could understand the basics of how they read a document.

4.4.2 Speed

The reading speed is a relatively simple metric indicating the amount of time that it took a user to read a particular document. Speed can be calculated using letters, words, fixations, or saccades per unit time. Using the eye tracking hardware, it is most reasonable to focus on fixations per unit time or saccades per unit time, as these readings will provide a more precise and unique measure. A measurement of saccadic movement includes every glance across the page, potentially leading to a falsely high speed. Fixations are slower, and represent the reading period during which the user actually absorbs data. A measurement of speed through fixation per unit time would provide the most accurate result.

The most meaningful unit of time could vary depending on metric. Therefore the unit of time could be a user selectable feature. However, there are multiple ways to calculate the overall time, listed below.

1. Time of opening document to time of closing document
2. Time from first gaze at first word, to last gaze on last word
3. Total time of all fixations and saccades

Of these three options, the simplest to work with is option one, real time. Option number two is technically more accurate, as it cuts out the time that the document is open on the desktop, though not being read. The third option takes it a step further, cutting out all of the time spent not looking at the screen. However, distraction and contemplation while not glancing at the screen are all normal parts of active reading. Option three, therefore, may in fact be too conservative, making option number two the most reliable.

4.4.3 Skimming and Skipping

Skimming and skipping parts of the text can indicate a variety of factors including boredom, confusion or distraction. Regardless of the reasons for skipping text, feedback on sections skipped could indicate how thoroughly the piece of text is being read. The computational analysis of skimming and skipping is closely related to the computation of speed. Because reading speed varies during the course of a session, the computation will be performed at the end of a user session using the readers own reading habits as points of comparison. There are multiple comparisons which could be used to assess when a user is skimming or skipping text. These are listed below:

1. Speed comparison
2. Fewer fixations per interest area(s)
3. Forward saccade length

While skimming and skipping are similar, their slightly different characteristics require they be considered separately when choosing the proper analysis. Skimming is defined as reading through text quickly in order to absorb a general concept rather than the details.

Skipping is completely passing a section without consideration. The first comparison could work on a very basic level. Sections read significantly more quickly than the rest of the document as a whole could be marked as skimmed sections, while those read faster than the maximum reading rate of 1600 words/minute would be marked as skipped. However, a metric based on speed alone is easily swayed based on the user's average speed reading a specific document. For example, the average reading speed will be low for a typically fast reader who slows down for highly technical paragraphs in a difficult document. If speed was the only metric used to interpret skimming, the paragraphs read at a normal rate may be marked as having been skimmed. Furthermore, setting the skipping parameter at the maximum reading rate would not take into account slower readers, such as children.

The second option measures the number of fixations within a specific interest area. An interest area consists of anywhere from one long word to multiple small words. It will be a dynamic parameter defined through the text layout system. As the reader moves through a document they should have a relatively fixed number of fixations within a specific interest area. During points of confusion, a reader will naturally linger within an interest area for longer periods of time. Conversely, if a reader is moving through a document quickly, they will have fewer fixations within specific interest areas and skip other interest areas entirely. Thus, skimming could be defined as having relatively few fixations within a cluster of interest areas. The number of consecutive interest areas without points of fixation could be a metric used to identify sections which were skipped while reading. This measurement goes beyond speed, to take into account the rapid and distant fixations which occur during skimming. However, this eye movement pattern would also be evident in a user glancing randomly around a page. To control for this, the distance between fixations which occur after forward saccades can be used to calculate the points of skimming. If no fixation occurs for a large number of fixation areas, such as an entire line of text, then the behavior can be classified as skipping. This option is the best option for measuring skipping and skimming.

The length of forward saccades is implicit in the calculation between fixations, and is a metric which may itself be used to measure how many interest areas are being overlooked between each fixation. As the reader fixates on fewer words and interest areas, inferring the information between fixations, the saccades between fixations will naturally lengthen. However, this method has its limits, especially in measuring skipping. Functional reading saccades have a maximum length, and the fixations between them may not be long enough for the reader to absorb information. Therefore, while saccade length appears to be a good metric, it overlooks the fixations which are the important points in reading.

4.4.4 Rereading and Re-referencing

Rereading parts of text and re-referencing points earlier in passages are common aspects of reading because they indicate noteworthy passages. Rereading can be defined as going back to an earlier point in the passage, and continuing to read from that point whereas re-referencing is a shorter occurrence where the reader looks to a few prior interest areas and continues to read from the farthest point read. Rereading and re-referencing can be identified through regressions, or backward saccades. However, regressions occur naturally in reading, and not every occurrence should count as a point of re-referencing. These can be distinguished in the following ways:

1. Number of regressions
2. Length of regressions
3. Saccadic behavior after regressions

The number of regressions could indicate multiple things. Regressions which move increasingly far away from the original gaze point most likely indicate that the reader is looking for a previous passage. Likewise, a longer regression is more significant than a short one. However, neither of these metrics carry explicit information without being combined with the saccadic behavior after the regressions. For example, if the reader continues to saccade backwards, they are most likely looking for something. If they regress to a clear point and then continue to saccade forward for a significant number of interest areas then the behavior could reasonably be classified as rereading. However, if, after a few long regressions, they clearly saccade forward for a few interest areas it could be classified as re-referencing.

4.4.5 Extended Focus

Most readers reach a point in a text where they become confused. Usually, they consider the word or section longer than they do other text which they find simple. The methods of identifying these points of extended focus are as follows:

1. Length of fixations
2. Number of fixations
3. Shorter forward saccades
4. Backwards saccades within a region of interest areas

Readers confused on a word or a point will tend to focus on that section for a longer period of time. This will display itself through the eye tracking data as longer fixations, which can be measured. For long or detailed interest areas the reader may attempt to increase information intake by increasing the number of fixations within the area. The increased number may also be characterized through shorter saccades connecting the fixations as the reader begins to focus on smaller pieces of text at a time. However, multiple fixations need not be a result of multiple shorter saccades, but the movement back and forth between particular points in a text. These backward saccades indicate re-referencing points in a specific section and would result in more fixations within a section of interest.

There are multiple metrics which are characteristic of an extended period of focus on a specific area. However, none of these metrics provide the full analysis to interpret extended focus. As a result, a combination of these metrics would best analyze the data. Given that shorter saccades indicate more fixations, but more fixations do not necessarily indicate shorter saccades the saccade length analysis is redundant and can be omitted. Instead, identification of points where the reader is focusing extensively can be accomplished thoroughly through observation of the length of fixations, the number of fixations, and the number of regressions within a series of interest areas.

4.4.6 Points of Distraction

Just as it is common to fixate on a specific area while reading, it is also common to become distracted. Highlighting sections of distraction can be useful to identify points of a text where students lose focus, thus identifying points in a curriculum that may need to be restructured. The two clear alternatives to identify points of distraction are glancing off screen for some period of time and random saccadic movements.

Periods of distraction are considered points where the user is neither reading nor considering the text. The metrics above do not account for users considering text without reading it. However, glancing off screen is a more accurate indication of user distraction. Typically, when one is reading on a screen, they do not get distracted by random points on the screen – which is what option two would measure. This program will be run in full screen mode for the duration of its use. If they begin to ponder the text for a length of time, they are encouraged to pause the program. This manual function would help cut down on the innate error within this metric.

4.5 Methods of Text Layout and Collision Detection

The end software product will involve quantifying patterns between gaze location and text location. In order to draw such mappings, it is necessary to know both gaze and text location. While the gaze location is handled by the eye-tracking hardware and the input processing of the software, the text location must be handled by a different subsystem. This task will require a solid implementation of some form of text layout.

Text layout can be defined as the process of properly shaping and positioning text for display and extended functionality (such as hit detection or highlighting). The general process involves shaping text, ordering text, and positioning text. Shaping text involves creating glyphs (visual representations of a character or character set) and ligatures (two or more merged glyphs that take a different shape when placed side by side). Ordering text is the process of converting the logical text ordering to a visual ordering. Logical ordering is the order in which words are read while a visual ordering is the order in which words are displayed. Generally this is only an important step for alternate or bi-directional text such as Hebrew or Arabic. Measuring and positioning text is the process of determining the spacing or width of characters and ligatures required to properly position text. In a mono-spaced font where all character's glyphs share the same width, measuring is easy, but for non mono-spaced fonts each glyph or ligature must be measured to ensure proper text position.

One option for implementing text mapping is a rudimentary system coded from scratch for this project. Representing text with only mono-spaced font would permit the creation of a grid for text display where each unit grid space is the width of a single mono-spaced font character. Then, opening an input stream of text, drawing each character at the next available location and noting its position would make mapping relatively simple. Received coordinates from gaze data could be easily compared to text coordinates because the grids would be exact scaled multiples of each other. The benefits of this approach would be easy implementation, simple comparisons, and few formatting edge cases. Downsides include extra work associated with coding from scratch. There is the concern that mono-spaced font is not as common in real reading applications, therefore not an ideal graphical text display for this program. The grid would not be as universal and could not handle text formatting. Lastly it wouldn't come with predefined methods of hit detection etc.

The second option for text mapping is the existing Java API, simply called "TextLayout." The set of classes included with the API handle shaping, ordering, and measuring/positioning text. There already is a significant amount of useful documentation from Oracle on the functions of the API. "Text Layout" includes built-in functionality for hit testing on a single character as well as side mapping (for entrance and exit of a word) that is used for caret location. This could be useful in defining the gaze to discern between forward and regressive reading patterns.

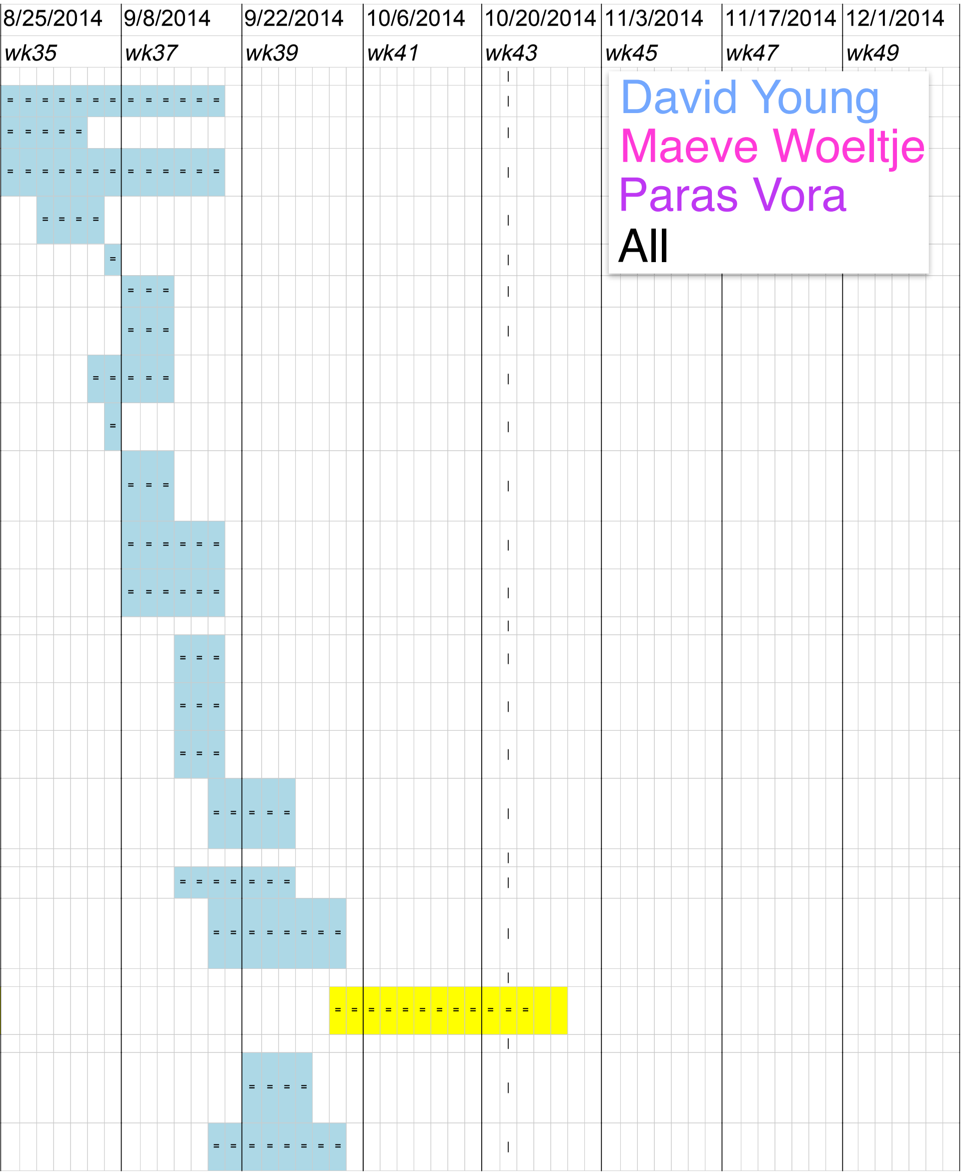
The process of mapping text in the final software will be accomplished with significant contribution from the “TextLayout” API. The existing set of classes provide a solid foundation of useful functionality including formatting rich text, automatic hit detection and highlighting as well as indication of directional cursor entry or exit from characters. Coding this functionality from scratch would be a large undertaking and there would likely be unforeseeable conflicts with graphics2D which is the partial backbone of Java’s display frameworks for both text and graphical user interfaces.

5.0 Specific Design Details

Specific design details for each section of the software architecture were outlined in the alternatives sections, however a summarized version follows. A user’s gaze will be tracked using an EyeTribe tracker, which will update a server on the user’s computer. The software will communicate with the server to receive updated gaze data in real time using the TET API and SDK to handle all server client communication as well as calibrate the device. The program’s graphical user interface will be written in JavaFX and text will be mapped and displayed using Java’s <TextLayout> API to handle all ordering, measuring and positioning of text. The H2 database engine will store all of the user gaze history with the use of SQL queries to inject, retrieve, link and sort stored data. A session’s data will be analyzed with the following metric and method combinations. Visual representation of gaze duration will be shown with a heat map. Speed will be calculated using fixations per unit time. Skimming and skipping will be determined by noting significant decrease in fixations in a specific interest area. Rereading and re-referencing will be determined through a combination of analyses of regression length and saccadic behavior after regression. Extended focus will be determined by a combination of fixation length, number of fixations and number of regressions. Lastly, points of distraction will be noted by gaze location drifting off screen.

Appendix A

Task description	Start date	Finish date	Progress
Preliminary Tasks	8/25/2014	9/19/2014	100%
Identify Problem	8/25/2014	9/2/2014	100%
Confer with Dr. Barbour about Project Direction	8/25/2014	9/19/2014	100%
Perform Background Research	8/30/2014	9/5/2014	100%
Define Project Scope	9/6/2014	9/7/2014	100%
Perform Patent Search	9/9/2014	9/12/2014	100%
Search for Existing Solutions	9/9/2014	9/12/2014	100%
Reach Design Specifications	9/4/2014	9/12/2014	100%
Organize Team Responsibilities	9/6/2014	9/7/2014	100%
Carry out Preliminary Analyses and Calculations	9/8/2014	9/12/2014	100%
Prepare Preliminary Report	9/8/2014	9/19/2014	100%
Prepare Preliminary Presentation	9/8/2014	9/19/2014	100%
Acquire Eye Tracking Hardware	9/14/2014	9/19/2014	100%
Connect and Calibrate Device	9/14/2014	9/19/2014	100%
Test Device Functionality	9/14/2014	9/19/2014	100%
Familiarize and get Acquainted With Java API	9/19/2014	9/26/2014	100%
Consult CS Faculty	9/14/2014	9/26/2014	100%
Outline Basic Architecture for Software on Paper	9/19/2014	10/3/2014	100%
Confer with Dr. Barbour About Project Direction	10/3/2014	10/28/2014	80%
Code Test for Device Communication and Data Acquisition	9/23/2014	9/28/2014	100%
Determine Method of Data Storage	9/19/2014	10/3/2014	100%



Task description	Start date	Finish date	Progress	8/25/2014	9/8/2014	9/22/2014	10/6/2014	10/20/2014	11/3/2014	11/17/2014	12/1/2014
				wk35	wk37	wk39	wk41	wk43	wk45	wk47	wk49
Test R/W Capabilities on Large Sets of Gaze Data	9/28/2014	10/3/2014	🟥 50%			🟥					
Experiment with Different Forms of Text Mapping	10/3/2014	10/17/2014	🟥 70%			🟥	🟥				
Experiment with Java TextLayout API	10/3/2014	10/17/2014	🟥 80%			🟥	🟥				
Experiment with 2D Graphics & Text in Swing API	10/3/2014	10/17/2014	100%			🟦	🟦				
Experiment with 2D Graphics & Text in SWT	10/3/2014	10/17/2014	100%			🟦	🟦				
Experiment with Hit Detection Methods	10/3/2014	10/17/2014	🟥 70%			🟥	🟥				
Architect and Code Basic GUI for Software	10/10/2014	10/24/2014	🟨 50%				🟨	🟨			
Create Basic Menu System	10/10/2014	10/24/2014	🟨 25%				🟨	🟨			
Create Basic Test for Text Mapping	10/17/2014	10/24/2014	🟨 50%				🟨	🟨			
Create Basic Test for Hit Confirmation	10/17/2014	10/24/2014	🟨 50%				🟨	🟨			
Prepare Progress Oral Report	10/20/2014	10/27/2014	100%					🟦			
Prepare Progress Written Report	10/11/2014	10/29/2014	100%				🟦	🟦			
Define All Data Output Forms of Feedback	9/19/2014	10/3/2014	100%		🟦	🟦					
Outline Format for Output Data	9/19/2014	10/3/2014	100%		🟦	🟦					
Design Methods for Creating Feedback	10/24/2014	10/31/2014	20%					🟦			
Code Methods for Creating Feedback	10/24/2014	10/31/2014	0%					🟦			
Test Feedback Generation	11/1/2014	11/8/2014	0%					🟦	🟦		
Create or Select Text Samples to Test	9/19/2014	11/15/2014	🟨 30%		🟨	🟨	🟨	🟨	🟨		
Test Device with Simulated Usage Scenarios	11/9/2014	11/23/2014	0%						🟦	🟦	

Task description	Start date	Finish date	Progress	8/25/2014	9/8/2014	9/22/2014	10/6/2014	10/20/2014	11/3/2014	11/17/2014	12/1/2014
				wk35	*wk37*	*wk39*	*wk41*	*wk43*	*wk45*	*wk47*	*wk49*
Re-Evaluate Aspects of Execution	11/9/2014	11/23/2014	0%						▨	▨	
Prepare Final Oral Report	11/23/2014	12/3/2014	0%							▨	▨
Prepare Final Written Report	11/12/2014	12/3/2014	0%						▨	▨	▨
Prepare Poster Presentation	11/23/2014	12/9/2014	0%							▨	▨
Create Demo Mode For Presentation	11/30/2014	12/4/2014	0%							▨	▨
Ensure Code is Adaptable	12/4/2014	12/10/2014	0%								▨
Consider Extra Functionality	12/4/2014	12/10/2014	0%								▨

Appendix B

- Software Engine (Control the flow of all other parts of the software)
- G.U.I. (graphically presenting anything and everything to the user)
    - G.U.I. Framework
    - Menu System
    - Specific Page Layouts
- Input (receiving input of any kind from the user)
    - Gaze Input Processor
    - Other User Input Processor
    - Sample Text Input Processor
- Output (analyzing data and generating forms of output represented to the user)
    - Analyzing Gaze Data
    - Generating Useful Feedback
- Database and Storage (Organizing and holding any information to be stored)
    - User Gaze History
    - User Information
- Text Layout, Mapping and Hit Detection Control (Mapping text to coordinates)

References

Cusimano, Corey. "Eye-Tracking While Reading." *Kertz Lab*. Brown University, 08 June 2012. Web. 18 Sept. 2014. <https://wiki.brown.edu/confluence/display/kertzlab/Eye-Tracking+While+Reading>.

MeasuringU. “Measuring Usability With The System Usability Scale (SUS)”. *MeasuringU.com.* October 2014. Web.

Oracle. “Class TextLayout”. *Oracle.com*. October 2014. Web.

H2. “Performance”. *H2database.com*. October 2014. Web.

HSQLDB. “Features Summary”. *Hsqldb.org.* October 2014. Web.

JavaFX. “JavaFX Overview”. *Oracle.com*. October 2014. Web.