Group 27

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

Final Report – December 1, 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 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

3.0 Design Analysis

3.1 Run Time and Hardware Considerations

The team’s primary goal during software development was to push a working prototype as quickly as possible. Initial proof of concept code is often prototyped with naïve, redundant or otherwise inefficient algorithms. Because prototype testing was carried out on tiny datasets (generally reading sessions less than 2 minutes), these naïve algorithms can yield working results. However, during development, special care was taken to make decisions that would benefit performance down the line and avoid bottlenecking the software’s flow path when the size of the session data increases.

The software uses an SQL database to handle information storage. This permits rapid injection of data as well as search and retrieval (expected asymptotic complexity on the order of logn and worst case performance on the order of n). The team considered buffering data injections to a temporary data structure while reading, injecting them all together at the end of a reading session. However, the individual injections didn’t prove to be at all taxing during run time, so it was left alone. Additionally, the H2 SQL database has an internal batch that it will populate if queries begin to build up, handling a bit of buffering logic behind the scenes.

When sorting data by fixation count during the comprehensive analyses, the team avoided nested code loops and redundant searches by storing data in multi-dimensional arrays and sorting them by specific columns. This was accomplished with nested array lists (where each element was an array list itself) and individually defined comparators. Using this data structure permitted sorting interest areas by fixation count and then trivially retrieving the character index or line index from the already sorted structure without unnecessary additional comparisons. This drastically reduced the run time when generating analyses.

As a general rule of thumb, whenever a specific calculated value was required more than once, the team opted to store it as a global variable so that different methods would not have to repeat the calculation. For example, the size of data sets and the ID of the current user were only calculated once, preventing redundant processing later. Additionally, if multiple statistics or metrics were defined by common data, an effort was made to accomplish both in the same run through a data set. In many places it was possible to avoid iterating through a dataset more than once, thus functionally reducing the processing time by 50% or more.

Hardware limitations are few. Although the software has not yet been ported out to an executable, the package is completely contained and doing so would be trivial. Such an executable could be run on any device that can run Java, and supports the Eye Tribe SDK. This includes most systems running Mac OSX or Windows 7. As of now there is no Linux support for the SDK, however the software is written to be versatile if Linux support is ever added. As for CPU requirements, the software was developed and tested primarily on a workstation desktop with an overclocked Intel Core i7 3770K running at 4.9GHz. However the team tested functionality on older laptops whose CPUs were running at 2.7GHz without any issues. The program is primarily serial, and little internal software benefit comes from a processor with a higher core count. However a more capable computer has more resources to partition between different programs, permitting the software to run smoothly while other programs are in use simultaneously. Despite the program never showing memory usage above 1GiB, testing showed that on older machines it was beneficial to run the program alone without other processes competing for system resources.

A functional analysis of the run time of the final software demonstrates working results. There may be inefficiencies yet to improve, but a user is able to use the software functionally in real time. A user can login, read a document and view session analyses and feedback in real time without waiting for the program to calculate or load data. For a prototype the software runs smoothly, the GUI is fluid and intuitive thanks to Java FX (especially compared to traditional Java Swing GUIs), and the nearly instant session analyses provide quantitative evaluations of the user’s reading patterns. Testing the software with text snippets designed specifically to exaggerate certain results does yield the expected resulting analyses.

3.2 Basic Metrics Calculations

Because every reader is unique and reading performance can vary from session to session, many of the comprehensive analyses are achieved by comparing metrics for a specific interest area within a document to the average statistics from the entire reading session. Therefore, in order to come up with more complicated output parameters, some basic metrics must first be calculated. The basis for most of the output analysis relies on being able to properly distinguish between saccades and fixations. A fixation is a point of relatively stationary eye movement, while a saccade is a quick darting motion of the eye from one point to another. According to authors Salvucci and Goldberg, one of the best methods of distinguishing saccades and fixations is a dispersion-based identification. In this method, a dynamic window is set, usually about the width of one character, and all fixations within that window are combined into one fixation, located around their centroid. Once a fixation falls outside of this window, it is no longer considered as part of the previous group of fixations[1]. The movement between the last point in the previous group of small fixations and the new group of fixations is considered to be saccade. While this is a very robust method of distinguishing fixations and saccades when proper parameters are established, the EyeTribe hardware used for the software has a parameter which outputs whether or not a gaze data point is a fixation or not. The group decided to use the EyeTribe's method of distinguishing the eye movements because of time constraints which may have limited the full advantages of using the dispersion based identification. However, this method is still mentioned as it provided guidance for some other forms of analysis.

The duration of a reading session is a simple calculation based on the total time the specific text was open. Likewise, the speed in words per minute is merely a fraction consisting of the number of words in the text file divided by the duration. The total number of saccades and fixations can be determined from the Eyetribe's output information indicating fixation, and the total number of points in the session gaze data. Each fixation is clearly identified in a stored boolean variable, and each saccade can be defined as any point which is not a fixation but while the gaze remains on screen. The fraction of a session spent focused on the text can be calculated as the duration of both saccades and fixations divided by the total duration. As fixations are more clearly defined, it is actually more reliable to calculate saccade parameters based on fixations. Therefore, the spatial length of each saccade can be defined as the number of characters between two fixations. Likewise, the temporal length of each saccade can be calculated as the duration of time between the end of one fixation and the beginning of another. The temporal length of a fixation can also be calculated easily. During reading there are often small eye movements know as nystagmus (a constant tremor of the eye), drifts (small slow movements due to imperfect ocularmotor control), and microsacades[2]. As a result, there may be points within an area identified as fixations though there seem to be no saccades between them. For the sake of simplification, all of these fixations may be considered together as one fixation[1]. Therefore, the temporal length of a fixation may be considered as the duration of all local fixations not separated by a saccade. For mapping purposes, the center of this fixation would be considered as the centroid of all fixation points considered together. This is a modification of the method of Dispersion-Based Identification described by Salvucci and Goldberg.

3.3 Comprehensive Output Analysis

3.3.1 Interest Area

Though an interest area is not itself a user directed output, it is necessary to define interest areas in order to perform a robust analysis and provide other meaningful outputs to the user. The interest area is the area within a piece of text for which the eye movement is being specifically examined. Interest areas provide a standardized unit of measure which makes it possible to ascertain the eye movement patterns for different sections of text.

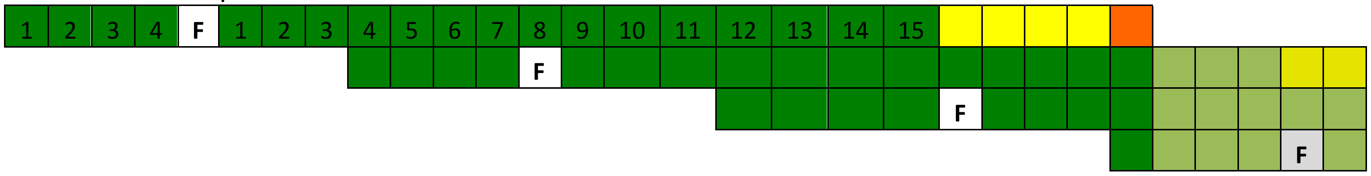
There are many ways to define an interest area. One of the simplest is by defining each line as an interest area. The code for this project was originally designed with this definition, and it provides usable and reliable outputs. However, an entire line is slightly too long to serve as the base unit of analysis. As result, a dynamic interest area, which changes based on fixation location, was agreed as a beneficial improvement for future revisions. In this case the dynamic interest area would be chosen four spaces to the left of fixation and twenty spaces to the right of fixation, resulting in an interest area twenty-five spaces long. The average field of perception while reading in English extends three or four spaces to the left and fourteen or fifteen spaces to the right[2]. This length of twenty-five spaces was chosen so that one fixation per interest area would be insufficient to perceive all of the letters, based on the average field of perception. Rayner also explains that the average saccade is approximately eight spaces long, and this measure is largely independent of font or letter size. Since the average field of perception is twenty spaces long, it indicates that most people do not make one fixation within the field of perception while reading. This indicates that readers with longer spans of perception will not have one fixation within every twenty-five letters. If the reader is reading with an average saccade length, there should be three saccades per interest area. Even if the saccades made by the reader are longer than normal, there will still be at least two saccades per interest area. An average of fewer than two saccades per interest area suggests not everything in a section of text was perceived. The analysis is shown more graphically below:

Key:

F	Point of Fixation
🟩	Within average field of perception
🟨	Within next field of perception
🟧	Fixation at this point results in total coverage of adjacent fields of perception
🟥	Not Perceived

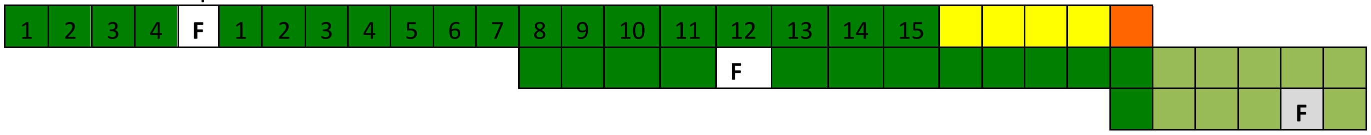
Average Saccade

Three fixations per interest area



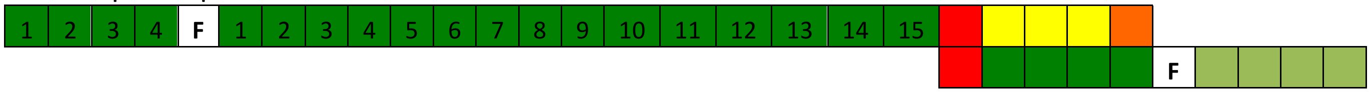
Long Saccade (12 spaces)

Two fixations per interest area



Too Long Saccade

Not all points perceived



3.3.2 Speed

The measure of speed used is words per minute as that is the most applicable and the easiest to compare across different sources. This metric was calculated in the simplest way possible to ensure accuracy and reliability. Reduced to the simplest form, the calculation consisted of the number of words in the document divided by the total session time.

3.3.3 Skimming and Skipping

It is very important to consider interest area in the calculation of skipping and skimming. This analysis is based off of the fact that two fixations per interest area must occur in order for the section to be fully perceived. As a result, it is reasonable to state that a reader is skimming the document if they have one or fewer fixations per interest area for two or more consecutive interest areas. This is further backed up by the fact that saccades over fifteen characters long are very rare in reading[2]. It is unlikely that a person actually reading would have only one fixation per twenty-character interest area. Once the reader increases their number of fixations for two or more interest areas, it is reasonable to state that the reader has stopped skimming.

However, strict values do not provide a thorough enough analysis since a reader who averages more than three fixations per interest area may skim the document at a different rate. This would especially tend to be the case for readers with a smaller field of perception. To correct for these readers, a value based on average reading speed must be set in place. Readers who average three saccades for the average field of perception are assumed to be skimming when the fixation density drops to one-third of the original value, or one fixation per interest area. It is assumed that they stopped skimming when the fixation density increases above this level. To prevent against transient changes in skimming rate, it may be helpful to set the skimming end point as the point when the fixation density reaches half of its original value. The lower beginning value is to prevent against classifying an increase in reading speed as skimming to quickly. However, once a reader begins to skim they may slow down, despite still skimming and therefore not absorbing full information, for a section before they begin to read again.

Skipping text is more easily defined as it the total lack of reading a particular section. It could be simply defined by comparing reading speeds to the maximum reading speed of 1600 words per minute[3], and classifying all sections read above that speed as skipped text. However, this is too high a parameter as many readers read significantly below the maximum speed and may actually skip text at a slower rate. As a result, it is best to define skipping as any segment where there are no fixations for at least one full line.

3.3.4 Rereading

Support for the detection of rereading has not yet been implemented, but is in progress at the time of writing this report. Rereading can be difficult to define since backward saccades, known as regressions, are normal parts of the reading process. However, these normal corrective regressions tend to be very small, and consequently, it is possible to classify any section where the reader begins to read from ten or more spaces back as rereading[2]. Therefore, passages which were reread can be identified by the following boundaries. They begin at any point where a saccade landed ten or more spaces back from the farthest point read and end at the point from which they initially made the regression. Not all regressions go immediately back to the desired point in the passage. In fact, poor readers especially tend to search more to find the point from which they wish to reread[2]. To simplify this complication, the farthest regression will count as the beginning of the reread segment. Furthermore, it will be possible to sort the reread passages by estimated difficulty based on the amount of searching which occurred during rereading. The searching can be defined based on how many regressive saccades occurred during rereading.

3.3.5 Periods of Extended Focus

Periods of extended focus are meant to highlight the areas where a reader's attention lingered for a longer than expected amount of time. This metric is meant to identify any point in the passage which may have been a source of confusion. It can be calculated through length of fixations as well as regression behavior. If the fixations within multiple interest areas are longer than average and there are more regressions than average it is an indication that that section was challenging. A list of sections for which a there were both more regressions and longer fixations than average could provide the users with an indication of the points where they struggled. This list is currently being defined by fixation data point count per interested area (by line), generating sorted lists of full lines of text where the user demonstrated extended focus. In the graphical information presented to the user, this sorted list is arranged in order of the most time spent in order to give some idea of most to least difficulty.

This analysis has also been applied to the much smaller interest area of single characters, generating a list of individual words where the user demonstrated extended focus. This functionally populates a vocabulary list for the user. For future improvement, intra-word regressions as well as secondary fixations on a word are a sign that the reader is struggling with a particular word[2]. A vocab list can therefore be created for all of the words containing any secondary fixation. However, this should not be the only factor as secondary fixations may occur for a variety of reasons. The strongest way to output a vocab list would be to include a combination of secondary fixations, intra-word regressions, and fixation durations. An improved vocabulary list would contain words which have an intra-word regression and a longer than normal total gaze length. An intra-word saccade would intrinsically result in a secondary fixation, and gaze length is a sum of total fixation duration. This output list could then be sorted by total gaze length to get an idea of difficulty. Due to the fact this is still a relatively broad parameter, during testing it was beneficial to limit this vocab list to a relatively small quantity of words with the longest gaze duration.

3.3.6 Points of Distraction

The points of distraction are easily identified as the points when the user is looking off screen, and are noted as such in the injection query to the database upon gaze update. By making the application and text full screen, it is doubtful a user will get totally distracted by a different point on screen when using the software. Therefore if the reader does get distracted by a different point in the text this would show up as rereading or skipping rather than distraction.

4.0 Specific Design Details

The specific design details include a detailed explanation of the broad software architecture, class structure and flow path (including input and output), as well as database and GUI designs and descriptions of the various 3rd party software packages that were used to create the software. Overall the key points of the design that yield a working result are an easy query-able database, a prototype friendly GUI framework (JavaFX), and existing Java TextLayout.

4.1 Overall Software Architecture

The overall software architecture is kept relatively simple. A flow chart is located in Appendix A. A main controller handles the chronological organization and flow between all other software components. A database controller handles the injection and retrieval of user, reading session and gaze data information which are all held in an SQL database. Text that will be used in a reading session is formatted and displayed, and real time hit detection calculates where in the text (with character precision) the user is looking. This hit detection uses the gaze data generated by The Eye Tribe SDK and API. Gaze data is analyzed for basic metrics and useful feedback is generated. All interaction with the user is handled through a GUI (graphical user interface). Each page of the GUI is composed of a page controller to instantiate the page, dynamically update information and respond to user input, as well as a format FXML (Function Extensible Markup Language) file to handle the layout of the page.

The application begins from the mainApp class, which coordinates, organizes and connects all the other software components. The mainApp instantiates a new databaseController which connects to an existing database or creates a new database if one does not already exist. See the appendix for SQL schema and a list of the many table definitions. After connecting to a database, a login box prompts the user to enter their credentials and if the information is valid the databaseController logs in the user. The mainApp then launches a Main Menu GUI by creating a new instance of a landingPageController. The graphic display of the Main Menu is determined by the landingPage FXML layout of the landingPageController. Different pages can be selected from the Main Menu, and are opened through instantiating a new controller for each page. The display of dynamic user information, required for some of the pages, is accomplished through the page's controller which accesses the databaseController via the mainApp and then calls its query methods to retrieve the necessary data.

Through another GUI page, a user can select a particular text to read, in which case the selectTextController will call a method from the mainApp which instantiates a new TextFormat with the selected text. The TextFormat creates a simple TET (The EyeTribe) controller instance and displays properly formatted text, allowing the user to begin reading. The TET sends updated gaze data to the TextFormat class where an interrupt driven listener receives the data. The reader's gaze data and hit detection results are injected into the database by calling methods from the databaseController. TextFormat begins the analysis function when the user closes the text display. The metrics and analyses for the reading session are generated and stored in the database. These metrics are drawn to the GUI page when the user chooses to view statistics for a certain section.

Again, a more visual flow path of the software is included in Appendix A. The details of the component parts are explained in the following sections.

4.1.1 Main App Controller

The mainApp controller is the main class that is launched to begin the application. It instantiates the GUI, begins a new database controller, and mediates between all other classes. mainApp contains very little actual control logic, rather it organizes and connects all other classes in a chronological order. Each class must communicate through the mainApp in order to inject or retrieve information from the database.

4.1.2 Database Controller

The database controller checks for an existing database. If one is found, the database controller links to it, if not, it instantiates tables of a new database. Once the database controller has linked to a database, it handles all insertions and queries to and from the database. Thus in order to store or retrieve information, from the database, a method from the database controller must be called. All database interactions occur through this controller, including the login. At login, a method from the database controller is used to check if a user exists in the database and sets the current user for the rest of the software classes to update dynamic information accordingly.

4.1.3 Text Format

The text format handles the layout and display of the text graphically. It also contains the callback from the gaze data updates in TET, handling hit detection on updates from the gaze tracker. To format text, the class subdivides the measured text into a list of line layouts with defined upper and lower y-coordinate (vertical) boundaries. Through these boundaries, the y coordinate of the reader's current gaze can be matched to a specific line. The line layout for that specific line is then retrieved, and the character index of that line's layout is calculated through single line hit detection. It is possible to find the paragraph index by adding the offset index found within the line to the paragraph index of the first character in the line.

All text retrieval is also handled in TextFormat since it holds the list of layouts. TextFormat contains methods to retrieve words or an entire line of text given the proper character index or line number. Other text retrieval methods were coded, but are not used in this program. TextFormat halts logging gaze data when the text display is closed. Upon closing of the window, the basicMetricsGenerator is called followed by the comprehensiveAnalysisGenerator to calculate the specific output statistics for the user.

4.1.4 Basic Metrics Generator

The basic metrics generator calculates simple metrics including duration of session, average reading speed in words per minute, the total number of saccades, the total number of fixations, the average spatial length of a saccade in characters, the average temporal length of a saccade in milliseconds, the average temporal length of a fixation in milliseconds, and the faction of the session spent focused on the text. The methods of calculation are included in the previous analysis section 3.3.

4.1.5 Comprehensive Analysis Generator

The comprehensive analysis generator calculates the number of fixations per definable interest area (per line, and per character index in the whole paragraph). It then sorts the fixations per interest area from highest to lowest so it is possible to identify the areas with the highest and lowest fixation densities. The method also retrieves the text associated with these areas in order to aid in further analysis. It is then trivial to compare the fixation counts per interest area, as well as speed through interest area, to the session averages calculated by the basicMetricsGenerator as well as the user's historical averages. These comparisons are useful in defining sections of extended focus, vocab lists where the user struggled as well as sections skimmed or skipped. This feedback can then be provided to the user later in the GUI.

4.1.6 Heat Map Generator

The heatMapGenerator leverages the power of an external HeatMap class to create a visual representation of the user's general reading patterns. First the gaze data is arranged into lists used by the HeatMap, which proceeds to rank the fixation durations by location and uses corresponding colors to visually map the data. The heatMapGenerator created by the team produces two heat maps – one including all gaze data and one including only fixation data. The generator uses a screenshot taken immediately before the closing of the TextFormat and overlays either heat map on top of the screenshot. It then saves the overlaid file as an image so it can be retrieved by the GUI at a later time.

4.2 GUI and Design

4.2.1 Individual GUI Controllers

The GUI has multiple pages to allow users to navigate selecting a text to read and viewing statistics. Each page has an individual controller, which handles both the page construction and the actions of the page elements. Some such actions include button function and dynamic list population. The controller also contains the logic for retrieving dynamic user information. Such user content includes a list of the user's previous sessions, the users previous performance, and data comparing the user to local and global performance averages.

4.2.2 Individual GUI Layouts

The individual pages of the GUI each have an FXML file which provides the graphical layout map for the page. While the controller logic is handled by the controller, as described previously, the FXML file handles formatting and layout of the GUI page. This relationship is similar to the functioning of webpages where HTML handles the layout while PHP or JavaScript handles the backend functionality.

5.0 Conclusions

The goal was to create consumer accessible software that quantifies reading patterns and outputs parameters useful for the average consumer. The software allows a user to read a page of text, and successfully outputs indications of areas where the reader may have struggled. The primary cost of the product is the EyeTribe device, which remains one of the cheaper eye tracking devices on the market. Furthermore, the GUI interface is well designed and easy to use for the average consumer, including children who could use the product for school. According to the design safe, all potential risks are low or negligible, with the exception of a moderate risk presented by the EyeTribe cable to infants. The included cable measures six feet long, and could pose the risk of strangulation for infants. There is no way to fully prevent against this other than provide a warning so adults are aware of the risk and keep the device away from children unless supervised. Due to the relatively low cost, functionality, and consumer friendly interfacing, the project was ultimately successful in its aims.

While the project was successful, there are still many avenues for development and improvement. In the future, it would be extremely beneficial to shift the project database online so global statistics can be compared across users. Furthermore, many of the measurements could be made more robust, adding parameters that cut out noise and random artifacts. Planned revisions include using a more precise system of distinguishing fixations and saccades, establishing a hit detection that is not based on single line detection and allows scrolling, and using a dynamic interest area which was described in the analysis but not yet included in the code. In the future some of the software could be altered to make it easier to add new features. A long-term goal would be to package the software as a stand-alone application, and market it more universally. The applications could extend beyond education to uses as extreme as employee contract liability insurance. However the primary target audience would be grade school classrooms without funding for reading examinations and progress tracking, as well as parents who wish to track their own child’s reading performance.

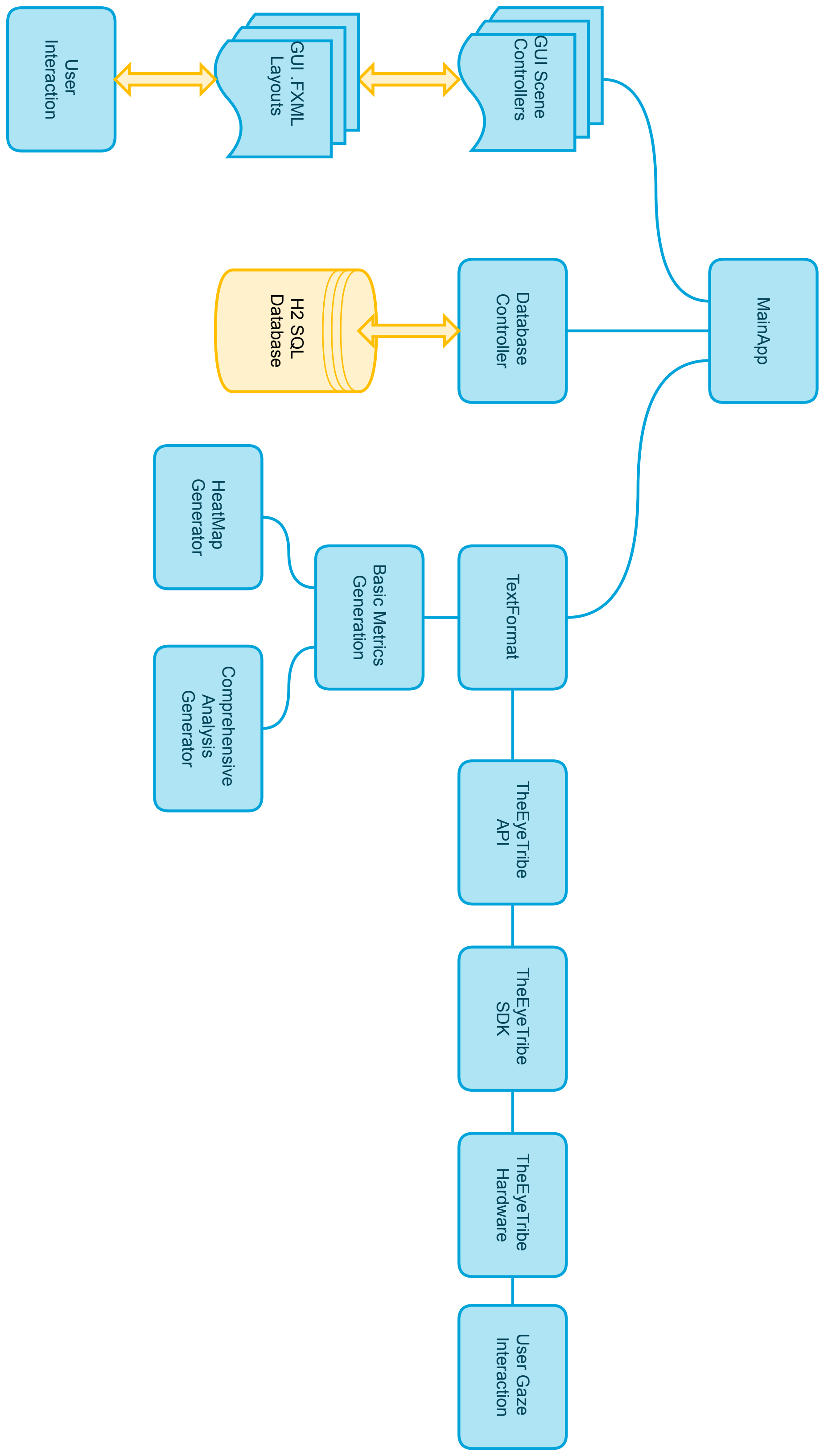
As the project currently stands there are not many ethical issues which arise. However, they may become an issue with further development. If, in the future, the database is put online in order to create a central global database, there will be issues with keeping user data confidential. There will also be ethical issues with how much user information is needed for general comparisons. Furthermore, if the product ever becomes marketable there will be the added ethical issue of how much to charge for the software. Ideally, it should be available free of charge so it can be used by underfunded schools to help support their reading programs.

If the project was attempted again, there are a few things that could be done differently. At the starting point of development the EyeTribe was determined to be the best eye tracking hardware option available for the needs of this project. Since that point, more hardware has come out and it is likely that a different hardware would be chosen to accomplish this project. Furthermore, it would have been helpful to establish a version control system for the code early on that would have made collaborations and revision documentation simpler. It also would have been very helpful to begin prototyping the software earlier in development. This may have allowed the inclusion of the more robust and dynamic analysis methods described previously as future revisions.

Through the project, it quickly became clear how challenging it is to make broad scope software. It takes significantly longer to debug than to prototype. As a result, the team learned that it is helpful to plan to begin prototyping much earlier than expected. Additionally, the team learned how important it was to use a proper development environment in order to boost productivity. Ultimately, one of the most important components was laying out a realistic and thoroughly planned schedule, which provided a generous amount of time for software development and prototyping.

Due to the fact that this project is completely a software project, there are not too many Intellectual Property (IP) complications. The only novel, potentially patentable idea is the implementation of the reading software for reading pattern analysis aimed at consumers and focused on educational support. While similar applications do have patents, they focus on industry or research and many have not yet been implemented, as mentioned in our previous report. While this information is potentially patentable, the team will not be pursuing a patent at this time. Some of the code is automatically copyrighted due to original authorship. The only open source software components that was were created originally by one of the team members are the TET API (used to receive information from the hardware), the Java HeatMap (used by the heatmap generator), and the H2Database package. With the exception of these pieces of code, team members have copyright on the remainder of the software.

Appendix A: Software flow chart



Appendix B: EyeTribe Hardware Specs and Company Information

Hardware Specifications:

Manufacturer	**Eyetribe**
Product	Tracker
Grade	consumer
Dimensions	20x1.9x1.9cm
Weight	70g
Operating Distance	45-75cm
Gaze Pos. Accuracy	0.5° – 1°
Spatial Resolution	0.1 degree (RMS)
Calibration Modes	9, 12 or 16 pts
System Latency	<20ms @ 60Hz
Works w/ glasses	yes
Max Monitor Size	24”
API/SDK	C++, C# and Java
OS Support	7,OSX
Sampling Rate	30, 40 or 60Hz
Connection	USB 3.0
Price	$100.00
Availability	available

Company Contact Information:

The Eye Tribe Aps
Amagerfaelledvej 56, Box 34
2300 Copenhagen S
Tel. no. : +45 36 980 580

Appendix C: 3rd Party Software Packages Used in Code Development

1. The Eye Tribe SDK and Java API
2. Java Heatmap class from software-talk.org
3. H2 SQL Database from H2database.com
4. Java TextLayout class from Oracle
5. JavaFX from Oracle

Appendix D: Commented Code (on separate pages, post references)

References

1. Salvucci DD, Goldberg JH. Identifying Fixations and Saccades in Eye Tracking Protocols.
2. Rayner K. Eye Movements in Reading and Information Processing: 20 Years of Research. Psychological Bulletin. 1998; 124(3):372-422.
3. Rubin GS, Turano K. Reading without Saccadic Eye Movements. Vision Res. 1992; 32(5):895-902.