Extending analytics for eye tracking data linked to source code based on iTrace

Dennis Bøgelund Olesen

Kongens Lyngby 2018

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Richard Petersens Plads, building 324,
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary (English)

The goal of the thesis is to create a tool for visualizing data gathered by the gaze to source code mapping Eclipse plugin, iTrace. The solution covers the steps necessary to analyze the data collected using iTrace. Starting with updating the iTrace interface connecting to the eye tracker to use the current state of the art Tobii eye tracker software development kit. Followed by extending the data storage of iTrace to support a database setup making several iTrace sessions readily available for the analysis tools. A python library, iTraceAnalyser is created to convert the iTrace gaze data into fixations while retaining the source code information. Finally the thesis propose two visualization solutions. The first being simple data visualizations using bar plots to quickly view data for single sessions. The second being a method of creating event logs making comparison of several sessions available through existing process mining tools.

Summary (Danish)

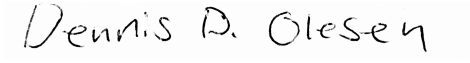
Målet for denne afhandling er at bygge et værktøj for visualisering af data samlet af Eclipse pluginnet iTrace som forbinder blikke til kildekode. Løsningen dækker de nødvendige trin for at analysere dataen samlet ved brug af iTrace. Første delmål er at opdatere interfacet mellem iTrace og eye trackeren således at iTrace kan bruges med moderne Tobii eye tracker software development kits. Herefter skal dataopbevaringen udvides så iTrace støtter en database opsætning. Dette vil gøre flere iTrace sessioner tilgængelige samtidig for analyse værktøjet. Et python bibliotek, iTraceAnalyser, er implementeret for at konvertere iTrace blik data til fikseringer uden tab af kildekode information. Til sidst forslår afhandlingen to visualiseringer til analyse af dataen. Den første visualisering bruger bar plots til at skabe et hurtigt overblik over dataen for enkelte sessioner. Den anden foreslåede løsning skaber event logs ud fra fikseringerne og bruger dette til at mine processen med eksisterende værktøjer.

Preface

This thesis was prepared at DTU Compute in fulfillment of the requirements for acquiring an M.Sc. in Computer Science and Engineering under the supervision of Barbara Weber and assistant supervisor Andrea Burattin

The thesis deals with state of the art eye tracking data visualization and analysis using process mining on data collected using the gaze-aware Eclipse plugin iTrace.

Lyngby, 14-January-2018



Dennis Bøgelund Olesen

I would like to acknowledge the following people for their contributions:

Barbara Weber, my supervisor for providing meaningful input to the direction of the project through regular meetings.

Andrea Burattin, assistant supervisor for providing input and ideas primarily regarding the practical implementations.

Contents

Chapter 1

Introduction

With eye tracking becoming more commonplace and businesses using eye tracking to determine behaviours of users in examples such as advertising [JL] it would make sense to look into using eye tracking for other purposes. One such purpose was made visible with the release of an Eclipse plugin, iTrace [TRS15], which couples eye tracking to source code. This presents information on not only the coordinates where the user was looking but also of the underlying source. Through this iTrace has provided a tool for gathering information on the behaviour of developers reading and attempting to understand code. However the iTrace data anlysis performed by others are limited to static representations of single sessions through visualizations such as heatmaps and gaze maps [BC17]. As such this thesis will attempt to provide ideas and solutions for how to transition from the data gathered using iTrace to an understanding of the process the user has gone through while developing, primarily focusing on process mining for data visualisation. The intention is for this to be used primarily in teaching environments where it would be worth noting certain patterns in the way the student works. In order to provide insight and provide assistance in the areas where they are struggling.

In order to set up a system for performing an analysis of iTrace data a couple of goals needs to be met. First the interface connecting the eye tracker to iTrace needs to be updated as the current implementation uses an old software development kit which is no longer available. The rebuilding of this is going to

be covered in order to allow for actual eye tracking information and can be seen as a prerequisite for the analysis.

Additionally the data storage was changed as iTrace stores the file separately which makes comparison across sessions and users inconvenient. As such a database setup will be proposed and the remaining analysis will use the data placed there. This also assists in providing additional insight into the structure of iTrace.

Before the data can be used for comprehension purposes it will be necessary to clean and structure the data into fixations to determine which parts of the data is relevant.

The data analysis will be split into two parts. One part which focuses on representing the values in static plots which provide information on the time spent on different elements of the source code and additional information of that sort. This helps provide a quick overview of a specific session without going into a more in depth exploratory analysis.

The second part will involve treating the task of understanding the source code as a process and consider each gaze as an activity where the user is trying to understand the related code. As such the fixation data will be converted into event logs which can be read by process mining tools and thereby provides the base for exploratory and comparison of several sessions. Doing this allow finding users who work in ways that are outside the expected and the norm. Providing a base for helping the person correct some of the issues and knowing where to put in additional work for improvement.

Chapter 2

Theory

This section will go over the theoretical knowledge which builds the foundation of the thesis. There will be an overview of the information related to eye tracking in Section 2.1, iTrace in Section 2.2 and process mining in Section 2.3 which is intended as relevant background knowledge for the systems created in the method Chapter 3 and the evaluation of the systems along with analysis performed with the systems at Chapter 4. The discussion of the systems and their results in the Discussion Chapter 5. For a brief overview of the some of the used terms see the glossary in Appendix A

2.1 Eye tracking

2.1.1 Introduction

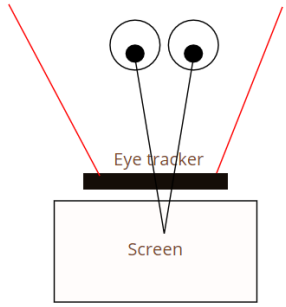
Eye tracking[AS14] is the act of tracking the eye movement in relation what is being observed, such as a computer screen, to determine where the person is looking. Eye tracking is being developed for several devices such as computers and wearable devices. This thesis will be focusing on eye tracking coupled to a stationary computer screen using the Tobii eye tracker 4C[Tob17a].

Eye tracking is the act of tracking eye movements and collecting gazes. Gazes are the raw output collected from eye tracking. Each gaze is a point on the screen with an X,Y coordinate set relating to the pixel position. A gaze may be attributed more values depending on the tool. Most relevant for this thesis are the recorded time for the gaze and the source code data related to the gaze point especially the line attribute.

In order to collect the gazes it is necessary to track the eyes. This can be done in several ways [HC84]. Four methods mentioned by H.R Chennamma are Electro-Oculography, Sceleral Search Coils, Infrared Oculography, Video Oculography. The Electro-Oculography method uses sensors placed on the skin around the eyes to estimate the position of the eye. The Sceleral Search Coils method places wires on a contact lens which allows for tracking eye movement very accurately. Infrared Oculography uses infrared light to calculate how the eye moves based on the amount of light reflected. The last method is Video Oculography which uses cameras and light to collect and process images of the eye to estimate movement and gaze locations.

In the case of the Tobii eye trackers, the tracking is done using a near infrared light along with a camera. The eye tracker then process the images taken of the eyes and use the position of the reflected light to determine where on the screen the subject is looking. a quick description of the setup can be seen in Figure 2.1

Figure 2.1: Eye tracking setup



As people's eyes are slightly different [RvdL] and the characteristics of the eyes and the way the light reflects may vary it is typically necessary to perform a calibration test [Tob17c] before starting any tracking session. These calibrations typically use points on the screen which the user then will focus on for a few seconds, to map out what the eyes look like when looking at different areas, and

thereby changing the settings so the eye tracker is more accurate.

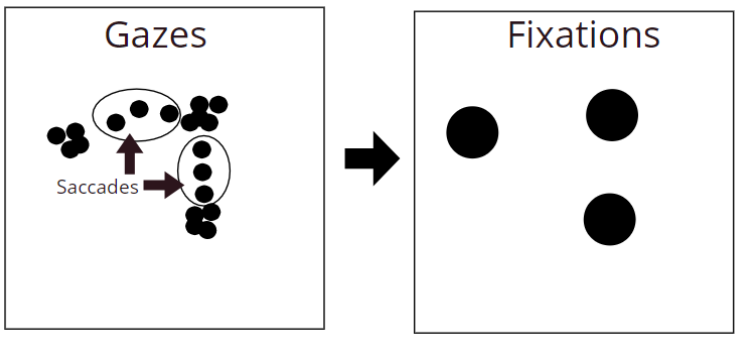
The procedure of determining where the user is looking is then performed as fast as is allowed by the technology. For one of the newer models, the Tobii 4C tracker by Tobii, this is at 90 Hz [Tob17b]. Each recording is then transformed into data which is commonly referred to as a Gaze. In a very simple setup a gaze would just consist of the x and y coordinates relating to the screen in pixels. However for analytic purposes gazes also include device time stamps as well as information on pupil including dilation and position and a validation, which indicates the correctness of the recorded gaze. Among other data. This data allows us to not only detect where the person is looking, but sort out bad gazes and use pupil information to perform additional analysis.

2.1.2 Fixations

Gazes are by themselves not particularly interesting, as it has been found that a person only comprehends what is being looked at after fixating on it for about 100 to 200ms [Wid84].

As such an important of step of cleaning the data is to transition from gazes to fixations. Fixations [AS14] are points where the user concentrates for an extended time. As such a fixation relates to gazes as fixations can be considered a cluster of gazes around the same point over an extended time. Fixation points typically contains X and Y coordinates and a start time and an end time. This can be done using grouping algorithms to group neighbouring gazes, which are collected immediately after each other, and are sufficiently close. These groups can then be considered fixations, and the transition from one fixation to another is considered a saccade. Saccades are the movement between fixations. When the areas are moving from one point of focus to another the gazes in between are considered the saccade. An illustration of going from gazes to fixations can be seen on figure 4.2

Figure 2.2: Gazes to fixations



It is however worth considering how to group these fixations as it can have a big impact on the rest of the analysis. Some of the issues which may be expected to arise if there is bad grouping are too few fixations if the algorithm is being too strict and too many fixations if it is not strict enough. On top of this some algorithms are more robust than others and handle issues such as fluctuations in the gazes better than others.

When looking at fixation identification algorithms [DDS00], which are the algorithms used for identifying the fixations in a group of gazes, it is possible to use a taxonomy considering some of the important criteria . One proposed way to define these criteria can be as accuracy, speed, robustness and ease of implementation. Here one may consider accuracy by how good the algorithm is in determining whether a gaze is part of the current fixation or if it belongs to a new fixation or saccade. The speed defines how quickly the algorithm groups the fixations. this is primarily important in any analysis done online and when large amounts of data are considered. Robustness is graded by how resistant the algorithm is to wrongful fluctuations in the data. The ease of implementation is worth considering when deciding upon the right algorithm for your product, as you may consider whether implementing a stronger algorithm is worth the effort depending on the precision and quality with which you collect your data.

Before deciding which fixation identification algorithm will be the right one for an analysis it is worth looking into a few different ones. This section will be looking at a velocity based algorithm Velocity-Threshold identification (I-VT). A dispersion based algorithm, Dispersion-Threshold identification (I-DT). There are other alternatives such as a model using probabilistic methods, Hidden Markov model identification (I-HMM), however these are not covered in this thesis. These models have slightly different approaches to how to determine the

fixations. However the basics of the algorithm remains the same and can be described as the pseudo code shown in figure 2.3

Figure 2.3: Fixation identification pseudo

```
Loop over the gazes and for each gaze.
  decide whether the current gaze is
  part of the same fixation as the previous gaze or gazes.

  if a gaze is not part of the fixation:
    Group the previous gazes as a fixation
    and start over from the remaining data.
    Remove fixation if duration is less than threshold
```

The challenge is then to decide whether the gazes are part of the same fixation. Additionally it may be worth mentioning as previously stated, a fixation needs to have a minimum length in order to avoid single gazes along a saccade to be considered a fixation. This also abides to the idea that a person only records what is being looked at after a short time, this time is usually considered to be around 100 to 200ms.[Wid84]

2.1.2.1 Velocity-Threshold fixation identification

The I-VT approach [DDS00] looks at the velocity between two gazes. This uses the idea that a fixation is steady within an area and as such it has only very small movements in the eyes. Where as any saccade would involve rapid movement from one point to another. Calculating this velocity can be done as the distance in pixels over time, or using the movement of the eyes in degrees to achieve a degrees/secon velocity. These values are then used to assess whether the gaze is fixated at a point near the previous gaze or if the movement can be considered rapid. It is generally considered sufficient to determine a static criteria for the velocity thresholds. Defining the threshold will depend whether you are using the angular velocities or if you only know the point to point velocities. If you are using the angular velocities previous research has approximated 20 deg/seconds to be a good threshold [TS84]. If these values are not available andt he point to point velocities are being used, a suitable threshold may be found using explorative analysis for the specific setup. The threshold may vary depending on resolution and the screen size, as the pixels are closer together

in high resolutions, as well as the fluctuations in the recorded data. When the velocity is known the algorithm can be described as seen on figure 2.4 [DDS00]

Figure 2.4: Velocity-Threshold identification, Adopted from [DDS00]

```
Calculate point to point velocity for each gaze

Label velocities under threshold as a fixation point

Collapse consecutive fixation points into a fixation group
Remove saccades

For each fixation group,
  determine the start time and the end time
  find the centroid of the points

Return the fixations
```

2.1.2.2 Dispersion-Threshold fixation identification

The dispersion based identification algorithm [DDS00] calculates the dispersion between the points in order to determine whether they belong in the same fixation. This relies on the idea that any fixation would consist of several gazes clustered together, allowing us to group together nearby gazes. This differs from the I-VT as the dispersion is based on the maximum and minimum values for all the gazes positions in the fixation window rather than only comparing two points at a time. However in the same manner as the I-VT in order to use the I-DT it is necessary to determine the a suitable threshold. The dispersion itself is calculated as $(max(x) - min(x)) + (max(y) - min(y))$.

The dispersion threshold can be found using exploratory analysis, seeing which value provides you with the best data over several sets. This is necessary as it may depend on resolution, screen size and the data fluctuation as well as the individual users fixation characteristics [RvdL]. It is also considered that if you can calculate the angle to the screen a dispersion of roughly 0.5 to 1 degree [P09] can be used as a general threshold. Additionally to avoid describing saccades as fixations we make sure any fixation is at least 100-200 ms long The algorithm itself works as described in figure 2.5

Figure 2.5: Velocity-Threshold identification, Adopted from [DDS00]

```
Loop over each gaze
  Add points until the duration threshold is covered

  If dispersion of window points <= threshold
    Add next gazes until dispersion threshold > threshold
    Calculate the centroid
    Collapse into a fixation.
  If dispersion of window points > threshold
    remove first point in window.
```

2.2 iTrace

2.2.1 Introduction

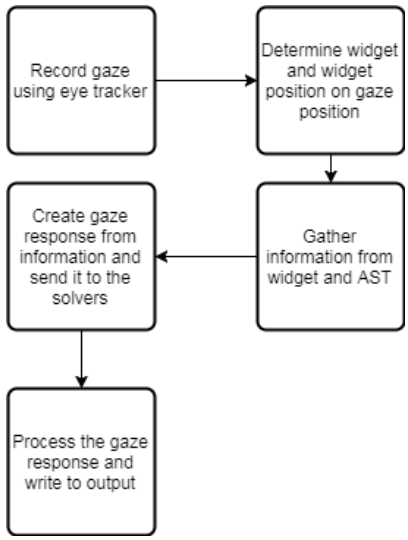
iTrace [TRS15] is an Eclipse plugin by the Software Engineering Research and Empirical Studies Lab, which enables eye tracking in a programming environment and allow the gazes within Eclipse to be linked to corresponding source code entities, SCE. The SCEs are defined as entities which are represented in the Eclipse editor as styled text. This includes declarations, invocations, statements, methods and comments.

The tool exclusively works for the Eclipse environment and as such it does not offer support for gaze tracking outside the Eclipse window. The data we receive from iTrace provide information on both the gaze itself which includes data about the eyes, time and positions as well as data about the corresponding area in Eclipse. The Eclipse data includes information about the source code entity in the area such as type and a unique name. SCEs are parts of the source code that can be categorized. These include highlighted words and comments. Additionally it also provides us with the line and column which can be useful for code sections that do not fit on a regular screen and require scrolling. [TRS15]

2.2.2 Structure

The structure of iTrace is segmented into parts handling the different parts of the process. The process consists of capturing a gaze using the eye tracker, converting the recordings from the eye tracker into a gaze containing the base data such as position, time and pupil information. Then iTrace need to handle the data which makes the gaze into a gaze response which is a data type that defines both the gaze and the corresponding part in the editor. This includes lines as well as SCE depending on the type of gaze response. Once you have the gaze response it will be passed to the solver which is responsible for outputting the gaze as readable data, by default XML and JSON. These different parts are split into the classes IGazeHandler, iGazeResponse and ISolver. The IGazeHandler is the component of iTrace which is responsible for collecting the data from the tracker and collecting further information about the connected SCEs, line numbers and more. The IGazeResponse is the structured gaze data which includes the information collected by the IGazeHandler. The ISolver is the super class responsible for outputting the IGazeResponses to other external sources such as XML and JSON. The actions of these classes are then done in parallel in threads using gaze and gaze response queues to avoid waiting in between them. As the solvers are writing the output every time they receive a gaze response the data is gathered online. The process for a single gaze can be described as seen on Figure 2.6. This process is repeated once the tracking is started, and ends when the user presses the Stop tracking button in the iTrace controller.

Figure 2.6: iTrace gaze handling process



In order to get the information that is at the gaze point iTrace opens Eclipse as a plugin which gives them access to the widget class allowing them to segment the Eclipse window as well as accessing information such as offset and relative positions. Additionally iTrace maintains an abstract syntax tree (AST) of the content within the editor. This syntax tree allow for mapping absolute positions in the widget to the content using the ASTParser. In relation to iTrace the abstract syntax tree holds the structure for the source code in order to find information on the specific SCEs. The AST also allow for information about the surrounding content.

The data outputted by the iTrace solvers are made available as JSON and XML files in seperate folders for each session. Additionally a JSON description of the session itself is created beside the gaze data.

In order to connect to the eye trackers an IEyeTracker class has been defined [TRS15] which defines the necessary functionality of the gaze collection and calibration. In the case of the Tobii eye trackers the SDKs are not directly available for java implementation, and as such a java native interface, JNI, is implemented for using the SDK in a different language. JNI is a java library used for calling C and C++ code in a java environment. In this thesis the term JNI or driver is used to describe the C++ module connecting to the eye trackers software development kit (SDK). In this case the Tobii eye tracker is implemented as a C++ driver using the old Tobii Analytics SDK . This SDK is no longer available and a new or updated implementation will be needed prior to use. The newest SDK is the Tobii Pro SDK.

2.2.3 Additional features

Besides capturing gazes and providing data iTrace include some additional features which assist in making it friendly for use in a test environment. The first thing when beginning an experiment iTrace will ask you to define the session. This session definition allow for intuitive separation of the data collected over several individual experiments, or sessions. Additionally it provides the opportunity to define users, the purpose of the session and creates a unique session id.

iTrace acknowledges that the precision of the eye tracking may vary depending on the user, despite an initial calibration [TRS15]. To combat this issue it is possible to manually change the X and Y position displacement drift. as well as a marker on the screen where you are looking, which works to give you an impression of the accuracy of the tracking

2.2.4 Related Work

One of the modern proposals on how to analyse and visualize the data collected with iTrace is the iTraceVis [BC17] solution. iTraceVis supports visualizations of the gaze data within the Eclipse plugin and supports heatmaps of the source code entities values in the editor, gaze maps which show how the user looked around the file, a gaze skyline visualization showing which lines were viewed over time. These visualizations were found to be useful for information regarding a single user and session.

2.3 Process mining

2.3.1 Business process models

Business process models, BPMs,[dA11] are being used to provide a meaningful way of describing the processes within many different business applications and work flows. BPMs are a way of describing a process in such detail and structure that it is often possible to recognize if there exists waste within the process. It is often used in regards to automating and optimizing processes and rely on defining actions and the flow with which these actions are done, in order to know how an event is to be handled. Regular business process models can help provide insight into the flow of the process as well as provide a guide for those who are part of the process.

2.3.2 The three stages of process mining

In some cases there is no detailed model that the users are aware of and as such they do not necessarily follow a specific flow. It is however still possible that there may be similarities between the way the process is performed across sessions. In these cases, if adequate log data exists process mining can be used to get further insight into the process.

The use of process mining is often put in three categories. Discovery, conformance and enhancement. These three categories can be used depending on the goal of the task.

Discovery is the act of gathering log events and producing a graph showing events and transitions explaining the behaviour. This will present you with

model showing how the process have been performed and make available information which can be investigated to acquire insight into the process.

Conformance is the act of comparing an existing BPM to an event log. This will allow you to determine whether the actual performed process matches the described model. This comparison can then be used in either direction, as it may mean the existing BPM is missing something, or simply not in tune with the real world. Or check if the event log represents a process which is wrong either by choice, someone finding the process bad, or by mistake.

Finally when considering using process mining for enhancement it is often done through the use of event logs to enhance the BPM. This can be used for repairing the BPM by adding flows that were missed in the creation but exists within the event logs. Or adding additional information to the model, for instance by using new information from the even logs, such as timings, to extend the quality of the model.[dA11]

2.3.3 Disco

With the event log information available, there exists tools for visualizing the flow and provide process mining information. One of these tools, Disco [Flu17], provide a platform for discovering the process behind the event logs. Disco allow us to look at the individual cases as well as getting an overview of the process as a combination of all events. It will allow us to get information of which events have been performed the most times as well as the duration of these events.

Chapter 3

Method

3.1 The process

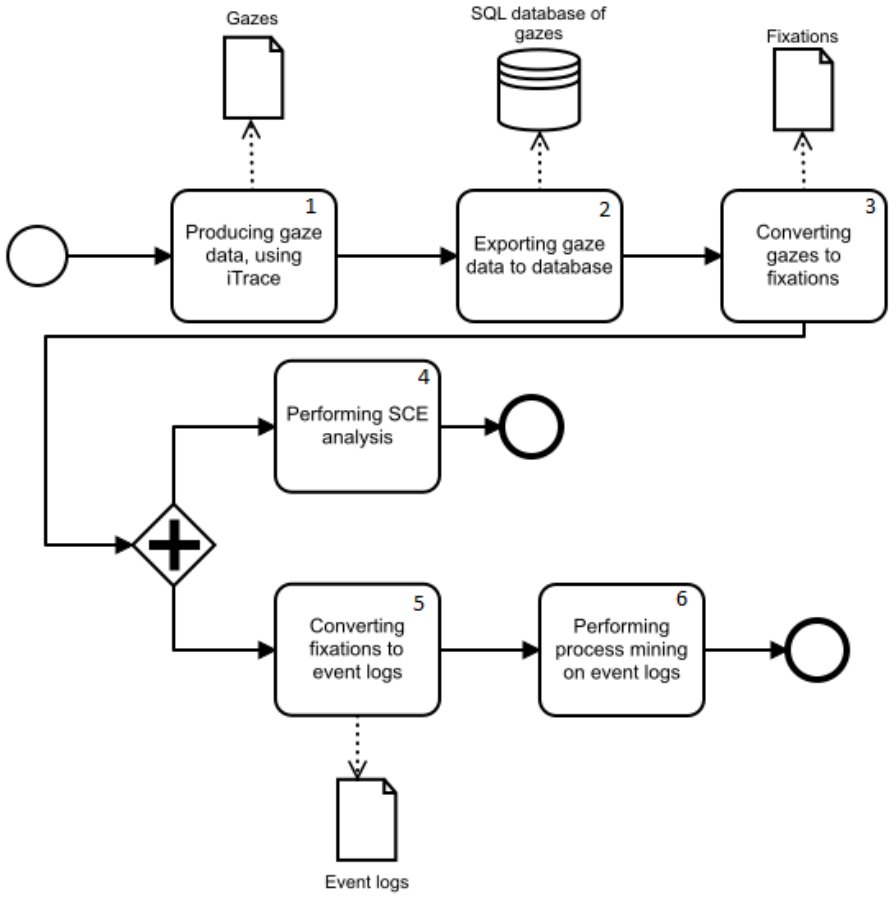
In this chapter we are looking at the process of building the system with the purpose of providing the tools for creating an extended analysis of the iTrace data in a testing and teaching environment. The goal of this section is to understand how the system is built and how the different components operate together, this includes the process from start to finish as well as the interaction and responsibility of each component and sub process in the created system. On top of this we hope to understand the reasoning behind the decisions made in regards to what is built, what is left out and why certain methods were chosen over others.

A model for the process of main aspects of the system going from the start to the end can be seen on figure 3.1.

The first step of the system requires a re-write of the java native interface as the Tobii SDK used in the official release of iTrace is outdated.
The second step of the process includes an update to the data storage to support future potential cross machine sharing and to centralize the data in one collection rather than the previous separate file setup as this will make analysis and comparison of several sessions more available.

The third step consists of having to clean the data by converting from gazes to fixations and removing saccades. This ensures that only the areas which the user has focused on is passed along for analysis.
The fourth step is a simple analysis of the fixation data created in step three, intended to get a quick overview of the values of a single session.
Step five and six is a more in depth analysis using process mining as visualization. The process mining tools are available however the data needs to be structured as event logs in order to use existing tools such as Disco [Flu17].

Figure 3.1: Analysis process, start to end



As we can see on the model the process has to go through a number of steps. It is worth noting that neither of these steps are readily available before this

extension.

To produce gaze data using iTrace in step one we need to update the JNI with the new SDK to allow use of the eye tracker. This sub process is described in section 3.2.1.

To export the gaze data to database in step two it is necessary to set up a database, as well as add a new solver which works with the database. This sub process is described in section 3.3.2.

To perform step three we must implement a fixation identification algorithm which maintains data of interest from the iTrace gaze responses. This sub process is described in section 3.4.

Performing the SCE analysis in step four requires presenting and structuring the fixation data, with respect to the source code entity attributes. This sub process is described in section 3.5.1.

Converting fixations to event logs in steps five require a definition of events, as well as a conversion implementation. Performing the process mining on event logs in step six can be done drawing conclusions from a process mining tool such as disco. This sub process is described in section 3.5.2.

Each of these steps can be considered separately but in concession as they depend on findings from previous steps in the way defined in figure 3.1 .

For installing the implementation see Appendix B.

3.2 Producing gaze data

3.2.1 Java native interface

The basis of the project relies on using iTrace for the gaze tracking. The tools provided in iTrace are by themselves complete for gathering the gazes as presented in section 2.2. However as the company developing the eye trackers continue to improve the development kits the drivers used in the current version of iTrace, Tobii analytics 3.0, are out of date and are no longer freely available. As such a rewrite of the java native interface is necessary to connect to the newer tobii devices.

3.2.1.1 Java Native Interface Responsibilities

The JNI is responsible for all communication with the eye tracker both in regards to data sent from the eye tracker to the plugin, as well as instructions sent from the plugin to the eye tracker.

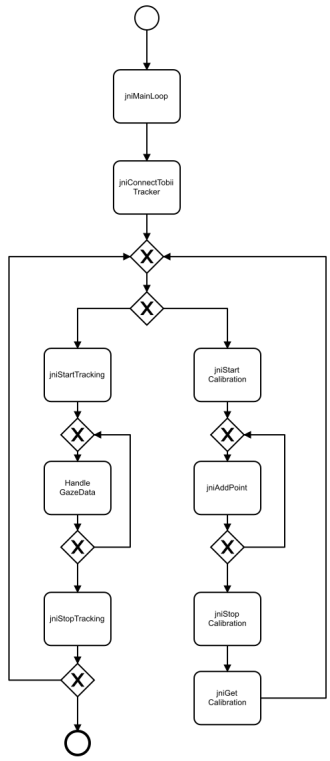
First it is necessary to understand what has changed since the previous version of the SDK. The rework meant that the C++ libraries in Tobii Analytics 3.0 has been replaced by C libraries in the new Tobii Pro SDK.

This does however not prevent a reuse of the old JNI implementation, as with minor changes to the SDK headers the C libraries are compatible with the C++ JNI implementation, however without the addition of an external c definition in the C headers, the compilers will not read the function calls in the libraries. Once the libraries are made compatible with the old JNI it is then possible to rewrite the Tobii eye tracker JNI to function with newer Tobii trackers.

Using the eye trackers with the JNI requires a couple of steps. First iTrace need to create a background thread for the eye tracker to run on. Afterwards iTrace connects to the eye tracker. Once these two steps are completed it is then possible to either start the tracking immediately, or set up a calibration. The calibration requires setting the eye tracker in a calibration mode, and then add points as the user is focusing on them. When enough points are added the eye tracker leaves the calibration mode and computes the data gathered from the points. At this point the calibration can be repeated or the tracking can be started. At this point the eye tracker repeatedly handles each registered gaze, and passes it along to the plugin.

The flow of these steps, named by their corresponding function in the JNI implementation can be seen in Figure 3.2

Figure 3.2: iTrace JNI flow



In order to update the old implementation with a new SDK it would not be enough to rely on the method names alone as these have changed with the new SDK and as such it is worth looking into the responsibilities of each method.

The jniTobiiMainLoop method is responsible for initializing the native data, which holds information about the eye tracker and the eye tracking main loop functionality. The mainloop and the jni function has the functionality which is often seen in initialization functions.

The jniConnectTobiiTracker methods responsibility is to connect to the tobii tracker. In order to do this it is required to find which eye trackers are available on the system. This is done using the library functionality of an eye tracking browser which scans and waits for any information on any eye trackers plugged into the system. Afterwards the eye tracker in the program is created using the information from the browser.

The jniStartTracking method sets the eye tracker in tracking mode and maintains a listener responding to captured gazes. Every time a gaze is captured the HandleGazeData method is called. The HandleGazeData processes the gaze and forwards a wrapped version of the gaze to the Eclipse plugin.

The jniStopTracking method leaves the tracking state and resets the tracking data to clean up after the tracking session.

The calibration part consists of four methods, **jniStartCalibration**, **jniStopCalibration**, **jniAddPoint** and **jniGetCalibration**. The method jniStartCalibration enters the eye tracker into calibration mode and clears the current calibration data, to free up for a new one. jniStopCalibration is responsible for leaving the calibration mode and additionally it computes the calibration data and applies the calibration on the eye tracker. In between these two it is necessary to add calibration points during the calibration. The calibration is done using points on the screen which the user is asked to focus on for a few seconds. While the user is focusing on these points the method jniAddPoint is called to collect the gaze position relative to the point they were supposed to focus. The system has to collect enough of these points in order to compute a proper calibration in the end. When the calibration is done iTrace can call jniGetCalibration to get the calibrated points, for visualizing them as a result to the user to provide an overview of how accurate the calibration were.

3.2.1.2 Updating the Java Native Interface

With the understanding of how the previous JNI implementation works it is necessary looking into what has changed since this version of the SDK. As the required tasks are the same as previously it is expected for the new SDK implementation to contain all the same functionality. The task is then to determine which methods are changed, and what impact the structural changes have on the implementation.

The major changes since the previous version is the change previously mentioned transition from C++ to C. On top of this the old JNI uses a main loop to form a frame around the eye tracking process. This loop took control of the eye tracking and was responsible for providing the browsing information used to gather information on the eye tracker and create the eye tracker. As such the mainloop structure played a large role in the old implementation. This has since been removed and no immediate replacement exists. This appear to be due to a change in the structure where they no longer rely on the mainloop class to access the eye tracking information, but rather have them all available in C methods, which in turn take eye tracking information as arguments rather than

inheriting it from the mainloop class.

This change has a structural impact on the way we are storing calibration information as well, as it used to all be kept within the classes, and calibration data remained within the eye tracker class. The new implementation require keeping the calibration data outside and passing them into the calibration computation methods alongside the eye tracker class.

An example of this change can be seen in figure 3.3. From this it should be visible that much of the data has been moved from the class structures to variables outside the classes, as well as the methods are now called independent of a class in addition to the name changes.

Figure 3.3: Calibration implementation example

```
//Original implementation
eye_tracker->computeCalibration();
eye_tracker->stopCalibration();

//New implementation
TobiiResearchStatus status =
tobii_research_screen_based_calibration_compute_and_apply(
eye_tracker, calibration_result);

tobii_research_screen_based_calibration_leave_calibration_mode(
eye_tracker);
```

Additionally some methods have had their functionality combined, an example of this is the start tracking which used to consist of a method for entering tracking mode, and setting up a listener which calls a helper function whenever a gaze is recorded. This has been moved into a single method, which takes a callback method and while in the tracking state this will function like the previous listener.

These changes were implemented in a new version of the tobiitracker.cpp file with the structural and convention wise changes described and can be seen in Appendix C.

3.3 Exporting gaze data

3.3.1 Solvers

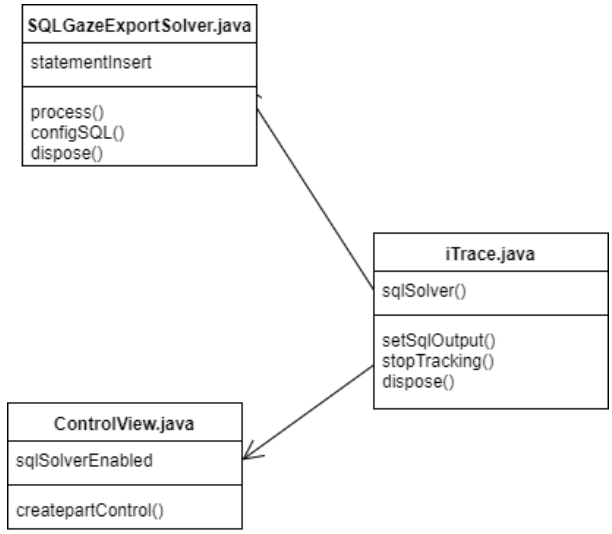
Due to the segmented structure of iTrace the export of data is mainly performed in the solver class as mentioned in section 2.2.2. The solver classes receive gaze responses which allows for collecting all the information created in the process. As such the solver is responsible for step five in figure 2.6 extracting the values in the responses and structure and forward it to an output file or database.

The implementation from iTrace saves XML and JSON files with the data in a folder named after the session id. As such accessing the data from several sessions at once is not intuitive. Additionally this setup does not support having a centralized data storage outside the computer where the eye tracking experiment is performed.

To combat these issues a new solver supporting SQL has been implemented and a database structure has been proposed. It is worth noting that for any cross machine sharing a networking setup would be necessary, but this has been deemed out of scope for this thesis. However as this provides the foundation for the remaining data analysis the changes required should be limited. The majority of the implementation resides within the solver class, where a gaze response is passed. It is worth noting that iTrace implements several types of gazes. Of these types the focus will only be on gazes on lines with no source code entities, and styled text gazes which include information regarding the SCEs on the line. This is done by readying a SQL statement for updating the database with the new data, depending on the gaze type, and appending all the SCEs afterwards and. At the same time the solver needs to connect to a database which is currently hard coded within the solver.

For the solver to work it is not enough to build the new SQL solver. The program still needs the solver integrated with the rest of the system. By this the gaze responses must be passed to an instance of the class, and additionally the user should still be able to decide whether or not to have the SQL solver toggled.

Figure 3.4: iTrace gaze exportation



To integrate a new solver involves a couple of additions to the controlling classes of iTrace. as seen on figure 3.4 it requires coupling the solver to the iTrace main class. In iTrace.java along side the previous existing xmlSolver and jsonSolver a new instance of the sqlSolver is needed. With the new sqlSolver instance it is then possible to pass along the responses as they are made. As a change from the XML and JSON solvers the sqlSolver requires a specific config call to insert the session information in the database. The session information is the developer user name and name along with the purpose of the session and the session description. These are originally stored in a file separately from the XML and JSON files and are as such not immediately available without specifically adding an additional sql config method. The implementation can be seen in Appendix F

The ControlView is responsible for adding new buttons to the console panel and handles the user interface aspect of the implementation. It is in this UI the user sets the sqlSolverEnabled variable which is used by iTrace.java to determine whether or not to do the sql export. When these aspects are set up iTrace will be able to export into the database. The next subsection will take a close look at the database and the data being exported.

3.3.2 Database

Previously to the database implementation iTrace saved the data as either JSON or XML files in separate folders for each session. By looking at a gaze from the JSON data it can be seen that the data consists of the base gaze information such as coordinates, pupil etc. Additionally it contains a variable amount of source code entities. The full list and the structure can be seen in Figure 3.5

Figure 3.5: Example JSON data


```
"gazes": [
  {
    "name": "example.java",
    "type": "java",
    "x": 775,
    "y": 452,
    "left_validation": 0.75,
    "right_validation": 0.75,
    "left_pupil_diameter": 3.4242401123046875,
    "right_pupil_diameter": 3.9959869384765625,
    "timestamp": "2017-10-05T13:14:41.332+01:00",
    "session_time": 973895205,
    "tracker_time": 240730,
    "system_time": 1507202081332,
    "nano_time": 1709448837018574,
    "path": "C:\\test\\src\\test\\example.java",
    "line_height": 28,
    "font_height": 18,
    "line": 18,
    "col": 13,
    "line_base_x": 424,
    "line_base_y": 449,
    "sces": [
      {
        "name": "java.io.PrintStream.println(java.lang.String)",
        "type": "METHOD",
        "how": "USE",
        "total_length": 30,
        "start_line": 18,
        "end_line": 18,
        "start_col": 4,
        "end_col": 34
      },
      {
```


The database structure were then created with the goals of being able to hold several sessions with several different users. As well as making sure the data available in the previous XML and JSON implementation are still available in the database implementation.

In the original iTrace structure there is also a separate file containing the sessions information. The session information contains a unique session id. The session purpose which is entered upon beginning a session. The session description. The developer user name and the developer name.

The goal is then to combine these into a database. By looking at the data it resembles tables already as there is a session information file which all the gazes belong to. Also for each gaze there are a group of SCEs with identical attributes. These different SCEs are the source code entity which the gaze is located at and the surrounding SCEs in sorted order from nearest to furthest. The surrounding SCEs are limited to the SCEs related to the entity gazed at. In other words if looking at a statement in a loop within a class the SCEs would cover the statement, the loop statement, the main declaration and the class declaration. As such depending on the depth of the SCE any gaze needs to contain a variable amount of SCEs. In order to keep the SCEs sorted and thus keep the information about the depth of SCE an attribute has been added which defines the depth of each SCE. As such if the user is interested in the tree like SCE structure it can be recreated by sorting the depth attribute for SCEs with the same ID. However if the analysis is done entirely on the object at the gaze point all sces with a depth of one or higher can be discarded.

Figure 3.6: iTrace database schema

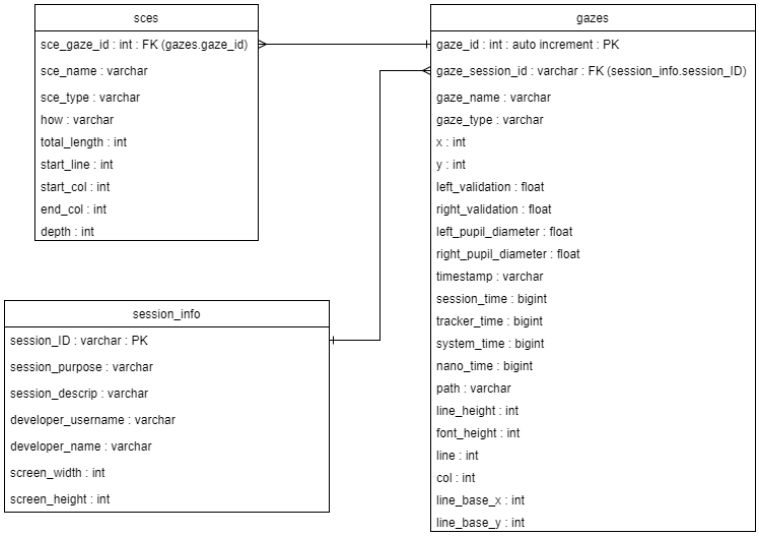


Figure 3.6 shows the database schema for the implemented solution. This solution reuses the unique session IDs created by iTrace to ensure the uniqueness of

the individual sessions. In order to couple several source code entity entries to an entry in the gaze table each gaze must have a unique primary key the SCE table entries can use to relate to their parent gaze. In addition to this each gaze belongs to a session and as such have been given the session ID as a foreign key to ensure the relation. With this setup the three tables are related and the data is ready to be cleaned and analysed. The iTrace.sql file for building the database can be seen in Appendix E.

3.4 Converting gazes to fixations

Before any valuable analysis can be done on the data it is necessary to determine fixations from the gaze data. To do this the algorithms discussed in section 2.1.2 can be used. From these the choice was reduced to either the velocity based identification algorithm or the dispersion based. These were chosen as they both scored well by Dario D. Salvucci and Joseph H. Goldberg [DDS00] as they are both fast and robust while not requiring any unnecessarily complicated implementations. From the velocity based algorithm and the dispersion based algorithm the dispersion based identification algorithm were arbitrarily chosen as either would work for the data.

As seen in section 2.1.2.2 the dispersion based algorithm depends on defining a threshold for how large of a dispersion is allowed for gazes to be considered part of the same fixaton. To calculate this dispersion threshold needed to determine whether gazes are part of a fixation or a saccade this implementation relies purely on the dispersion of pixels, as there is no static setup to ensure the distance to user and the angle from user to tracker, which would allow us to use angles instead of pixels. As such it also requires the user to pick a dispersion threshold for the algorithm measured in pixels. This dispersion threshold may have to vary between different uses and tasks so the ability to change this has been given to the user.

The algorithms were implemented in python from scratch in order to get full control over which parts of the data are passed during the conversion. In order to make the system usable for a user the fixation gathering has been coupled directly into a method which takes the database information as input and outputs a list of fixations. Each of these fixations contains a start time, end time, x and y coordinates. Additionally it is implemented in such a way additional data from the database can be added as well.

When the gazes are grouped as fixations, each fixation consists of a list of gazes. This presents an issue with knowing which data to use if the fixation is on the

edge of a line and as such consist information from different lines. For example at times some gazes would suggest the SCE type is a for statement and other gazes suggest the line is a print statement. To combat this the current solution picks the attributes repeated the most time. Different approaches could be to determine it based on the line used the most times or basing it off which data values has the most milliseconds of view rather than number of gazes. This happens in much the same way as the calculation of X and Y coordinate centroids.

All of this is being done in the implemented library class iTraceAnalyser which handles connecting to the data base, creating the fixation and the event logs discussed in section 3.5.2. The import is done by the user by calling the get session function bringing along the database information and the unique session id as seen in Figure 3.7.

Figure 3.7: iTraceAnalyser Get Session

```
import iTraceAnalyser as IA
#Importing the test subjects data
iTrace1 = IA.get_session("localhost","root",
"1234","iTrace",'20171201T092946-0684+0100',15)
iTrace2 = IA.get_session("localhost","root"
,"1234","iTrace",'20171201T101056-0578+0100',15)
iTrace3 = IA.get_session("localhost","root"
,"1234","iTrace",'20171205T100741-0299+0100',15)
iTrace4 = IA.get_session("localhost","root"
,"1234","iTrace",'20171205T102737-0035+0100',15)
iTrace5 = IA.get_session("localhost","root"
,"1234","iTrace",'20171205T105857-0765+0100',15)
```

When this is called the variables iTrace1 to 5 are lists of fixations that are ready to be used in other methods. It is also possible to get the raw gazes with the *get_raw_gazes*() method and the same arguments, if those are of interest.

3.5 Analysing The Data

3.5.1 Performing Static Source Code Entity Analysis

Before looking into the process mining for comparison between several sessions, a quick view of specific sessions and their data may be of interest. As such a couple of methods were implemented to work with the fixation data collected in the previous section 3.4. These methods are intended to function as a quick overview of the fixations for a single session.

The main idea behind using this functionality over the process mining is not the quality of the data shown, or it being more in depth than the process mining data. Rather most of what can be seen from these methods in the iTraceAnalyser class seen in Appendix D can also be discovered using the process mining suggested in section 3.5.2. The idea is to give quick access to certain values and views that may be interesting before going into a more comprehensive analysis.

To reach this goal the methods were designed to keep them generalized enough that they will work for any attribute passed along with the fixations acquired in the get session methods seen in Section 3.4 Figure 3.7. This helps in case further work is done on the iTraceAnalyser class where other attributes is being placed in the fixations.

The methods implemented covers bar plots of the following:

- number of fixations.
- Total fixation duration.
- Average fixation duration.
- Maximum fixation duration.

Additionally a method has been made for the total duration of the session in seconds.

In Figure 3.8 the methods are called for creating the bar plots with the fixations created in Figure 3.7. These plots are used and shown and in section 4.2.2.1.

Figure 3.8: iTraceAnalyser, plots and duration

```
print IA.session_duration(iTrace1)

#Creating bar plots for Chapter 4 section 3.
IA.fixation_count_bar(iTrace1,"count_lines","line")
IA.fixation_count_bar(iTrace1,"count_types","sce_type")

IA.fixation_duration_bar(iTrace1,"dur_lines","line")
IA.fixation_duration_bar(iTrace1,"dur_types","sce_type")

IA.fixation_average_duration_bar(iTrace1,"avg_dur_lines","line")
IA.fixation_average_duration_bar(iTrace1,"avg_dur_types"
,"sce_type")

IA.fixation_max_duration_bar(iTrace1,"dur_max_lines","line")
IA.fixation_max_duration_bar(iTrace1,"dur_max_types","sce_type")
```

These methods were chosen as they provide insight into what areas the user has been the most interested in four aspects. Where did the user spend the majority of the time. Which area did the user repeatedly come back to and which areas did the user on average spend the longest. Finally which location did the user spend the longest single fixation, as important areas may be re-visited many times, which lowers the average duration of a fixation at that area.

3.5.2 Performing Process Mining Analysis

This section covers the tools created to perform a proper process mining analysis of the fixation data. This part of the process consists of converting the fixations to events which can be read by the process mining tool Disco. Additionally it will contain a few remarks about the tools Disco provides which can be used.

Process mining was chosen as the primary source of analysis for the data collected using iTrace due to how effectively it can present information regarding how the users have behaved. Additionally it allows for much smoother comparison between several users than individual visualizations such as heatmaps would. For longer sessions it is also less cluttered than would be expected from scan path visualizations. The process mining provides strong tools for comparing the events completed in each session.

In order to perform the process mining analysis the data needs to exist in an event log form. This form needs to include a unique identifier which separates the different processes. In addition each entry needs to be considered as an activity and as such needs an activity name. The activities are not necessarily unique and typically represent one or more actions that are done together. Finally it makes sense to have a start and end time for each event as they are represented as fixations. This presents knowledge about both the duration of each event and it also let the user know the time in transition between events. When working with this fixation data the unique identifier may very well be the session ID as this has been brought along throughout the process and it uniquely identifies each session.

Which value to use as the event descriptor is more interesting as the analysis changes depending on how the event is defined. First thing to notice is that events in this case can be considered to be fixating on a point. The difference then depend on what you decide to take from the fixation. One way could be using the X and Y coordinates and create an event for each X Y fixation combination. This presents a couple of issues in regards to the readability of the mined processes. It would not be trivial for the reader to couple screen coordinates to the code viewed. This becomes increasingly difficult when the code is large enough that scrolling is involved and position X,Y is no longer deterministic as the scrolling offset is unknown.

3.5.2.1 Lines

To get around these issues each line could be considered an event. In this way the offset is no longer a problem and with having the code next to the diagram the line can easily be coupled to a section on the screen. An example use of this is if the user spent an extraordinary amount of time on a line, the reader can look up the line in the code and judge if this makes sense. This allows for an exploratory analysis of the session.

In order to make it easier to relate the activities seen in the process mined data to the content of the lines in the source code the SCE types that have been viewed on that fixation point has been appended to each event. As such an event would be named "34 - FORSTATEMENT" rather than just 34. This makes it easier to understand what exists on the line without changing the tree.

An example of the mapping from source code to the activities in the mined process for the line wise method can be seen in Figure 3.9

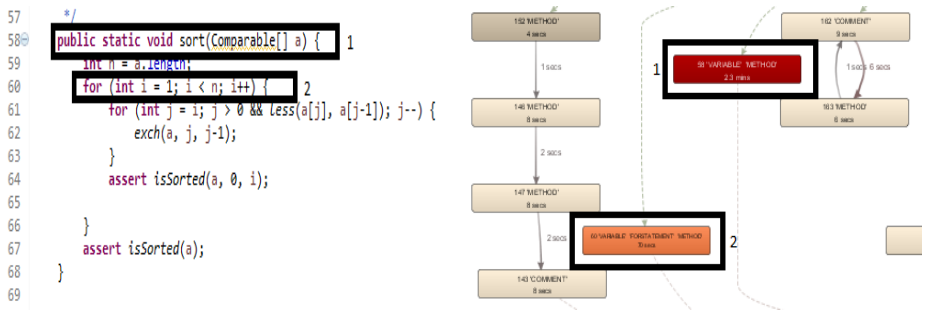


Figure 3.9: Source code to Event log mapping, lines

The proposed solution works as a general implementation and requires no outside interference from the reader and provides a process which can be analyzed and hopefully clarify differences in approaches and behaviour of the user. However an additional method is proposed to help alleviate a few issues. The first issue is when the eye tracking is not accurate enough to do pixel, or line, perfect gaze capture. Secondly the more events, in this case lines, the process contains the harder the analysis is to read. In addition the naming using lines requires a lot of back and forth between the code and the process mined diagram.

3.5.2.2 Area of Interest

The alternative method allows the user to define areas of interest. The area of interest, AOI, are then used as events. An example AOI could be line 1-10, which may consist of an introduction to the file and could be named by this. This would allow the user to define larger areas and name them in ways that are easily relatable to the source code and lessen the need for looking up in the source files. As for the accuracy issue as the areas are intended to be larger and in some cases a slight padding can be added it prevents issues where accuracy makes the user leave the AOI which he is actually looking at.

Figure 3.10 shows the area of the source code which is mapped to its matching activity in the mined process. In this example the Sort method is defined as an activity. This makes the entire section a single activity in the process mining diagram.

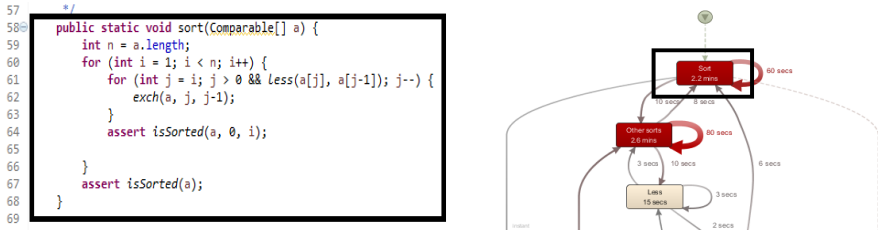


Figure 3.10: Source code to Event log mapping

When creating these events a few decisions were made in regards to implementation. When two or more consecutive fixations are on the same event, assuming the transition is less than a second they are collapsed into one event. This may happen due to spikes in data or moving from one point within the area to another. This allows for longer events, while still allowing catching cases where the user is looking away from the screen and then returns to the same area.

These solutions has been implemented as a python library, iTraceAnalyser Appendix D which handles collecting data from the db, converting it to fixations as well as creating event logs and bar plots of the fixation data.

Chapter 4

Evaluation

4.1 Setup

When the data was collected a few goals were set with respect to the extent of the report. It would not make sense to use example data as this does not provide any real insight into what kind of results could be seen using the proposed analysis process. As such one goal was to gather data from outside sources. At the same time it was decided not to formalize the testing entirely and the test task was kept small and relatively simple with only one file. The testing environment was kept informal and was done using students around campus at the Technical University of Denmark.

The physical setup consists of a laptop with a Tobii 4C tracker placed in the section between screen and keyboard using adhesives to keep the tracker in place facing towards the user. The users was also given a pen and some paper, in case they needed to note down the list to be sorted.

To enhance the quality of the tracking each session began with the user looking at a test file with a letter they were asked to focus on. Using the crosshair in iTrace it was possible to determine how close it was to the letter. Small deviations could be solved using the drift tool in iTrace and larger deviations ran the calibration.

The test itself was done using an implementation of Insertion sort available from and authored by Robert Sedgewick and Kevin Wayne [RS17]. Minor changes was done to this file in order to make the test more extensive and can be seen in Appendix G. First the name was changed to avoid the user immediately seeing which sort is being used. The name was however not removed from the comment fields. Additionally the sorting loop was made to begin the iterations from i=1 as opposed to the original i=0 as the first iteration would otherwise do nothing.

The Insertion sort file consists of several methods. The ones to consider are
Sort with one argument is the sort method the user is asked to work on, this sort takes only one argument.
Sort with multiple arguments is the sorts which take different arguments. The user is not supposed to look at these. It is however a large section covering 70 lines which means that several short fixations within this area is expected.
Less is a method comparing two values. Less is called in the Sort method.
Exch is a method switching around the position of two values. Exchange is called in the Sort method
isSorted is a method used in asserting that the list is sorted for a given interval. isSorted is called in the Sort method. **Show** is a method printing the list.
Main is the main method calling Sort with one argument.

The test subjects were then asked to first figure out which method was called as the implementation consists of several sorts with different amounts of arguments. The sort is called by the main method using one argument containing the list [4,2,1,3]. This question was used to ensure they all worked on the same method. They were also asked what the array would look like after the first iteration of the outer loop. As the algorithm is insertion sort the first iteration will compare the first two indexes and swap if the first element is lower than the second element. The sorting algorithm is written as seen in 4.1 and shows two other functions, less and exch. These exists in other sections of the code.

Figure 4.1: Insertion sort

```
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 1; i < n; i++) {
        for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
            exch(a, j, j-1);
        }
        assert isSorted(a, 0, i);
    }
    assert isSorted(a);
}
```

These questions were asked to five testers where four of them completed the task.

4.2 Analysis

This analysis section goes over the data collected using the methods described in the previous chapter and performs an analysis of what is seen at the results in order to provide an example of the kind of analysis which can be done using the system.

Results and analysis will be seen of the following parts of the system.

- Fixations, accuracy and choice of dispersion threshold. From Section 3.4
- Fixation data analysis with python plots. From Section 3.5.1
- Process Mined data of event logs. Both line by line analysis and User Defined area of interest analysis. From Section 3.5.2

4.2.1 Fixations

In the setup in Section 4.1 it was described how gaze data from five students were gathered during a small test. With the collected gazes it can be of interest

to determine how well the gazes are translated to fixations. This also allows for checking the output depending on a few different dispersion threshold settings. In Figure 4.2 a comparison of the raw gazes and the fixations achieved can be seen with a dispersion threshold of 15 pixels which appears to be a decent value for the gazes collected as it covers the expected areas without creating fixations in the saccade areas. The gaze data used is not related to the test setup mentioned in Section 4.1 as the coordinates do not take into account scrolling and a smaller data section help provide a more clear image of the gaze to fixation transformation.

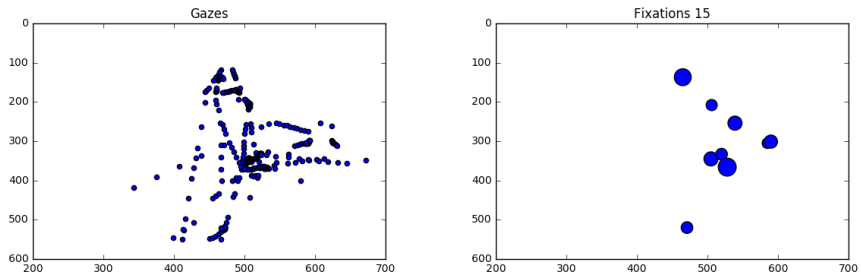


Figure 4.2: Gazes to fixations

From the images seen in Figure 4.2 it is clear that a lot of gaze data is produced when looking around the screen. As such the fixation identification algorithm effectively removes a lot of the noisy data and present a clearer image of which areas were actually being focused at. For this example the size of the dots depend on the duration of the fixation.

Besides the transformation it can also be seen how the choice of dispersion threshold changes the results drastically. According to a study published by the National Center for Biotechnology Information they found that
Quote: *"there are considerable differences in the characteristics of fixations not only between these tasks, but also between individuals."* [RvdL].
They point out that the eye movements differ depending on the type of task and from person to person. As seen on Figure 4.3 the fixations vary greatly based on the dispersion threshold and it should provide an idea of how the analysis may change depending on the choice.

Here the dispersion threshold of 15 pixels were chosen as it covers the same fixational areas as the the others without creating additional fixations such as the 25 and 30 threshold.

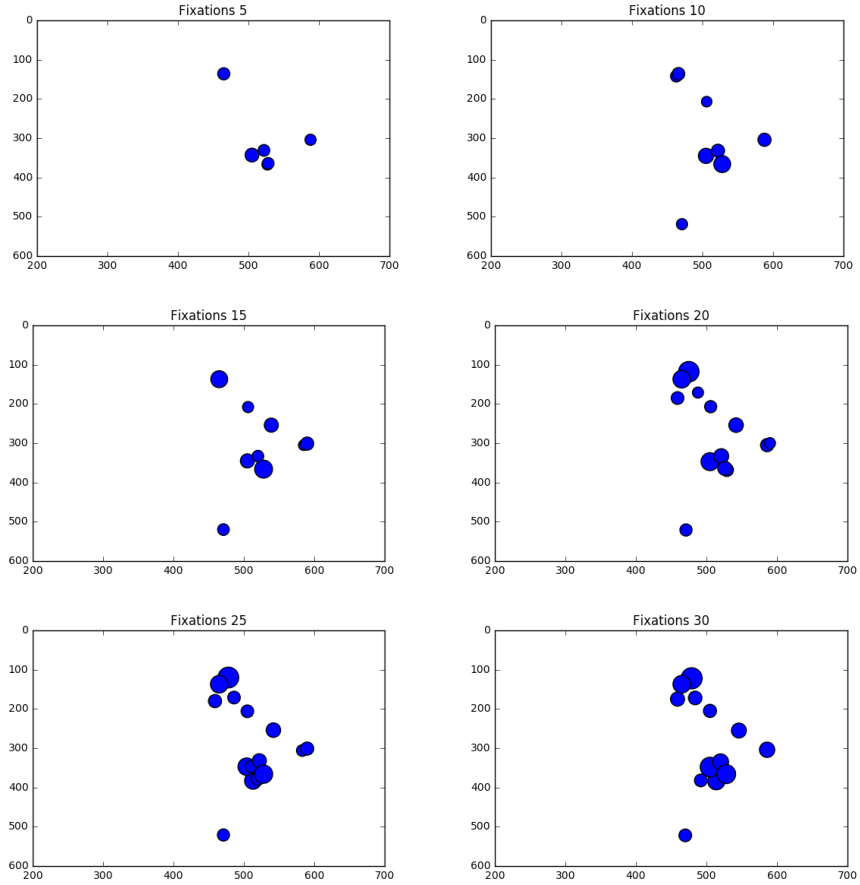


Figure 4.3: Gazes to fixations

As previously stated the cut off point for threshold between gazes is not trivially found for all cases of the tracking and need to be considered for the individual setup. As such a look is taken at the behaviour of the fixation algorithm for data gathered with the setup used throughout the experiments. The results of these thresholds can be seen in Figure 4.3. From these it can be seen that the gazes which are not clustered together, the saccaddes, are removed for all the tested values. It is also seen that with thresholds between 15 and 30 the areas remain the same while the number of fixations identified differs. This gives the impression that values between these would offer decent results for further analysis. It is however not immediately clear which value is the most accurate without a more rigorous analysis.

4.2.2 Bar Plots Fixation Data Analysis

4.2.2.1 Bar Plots Fixation Line Analysis

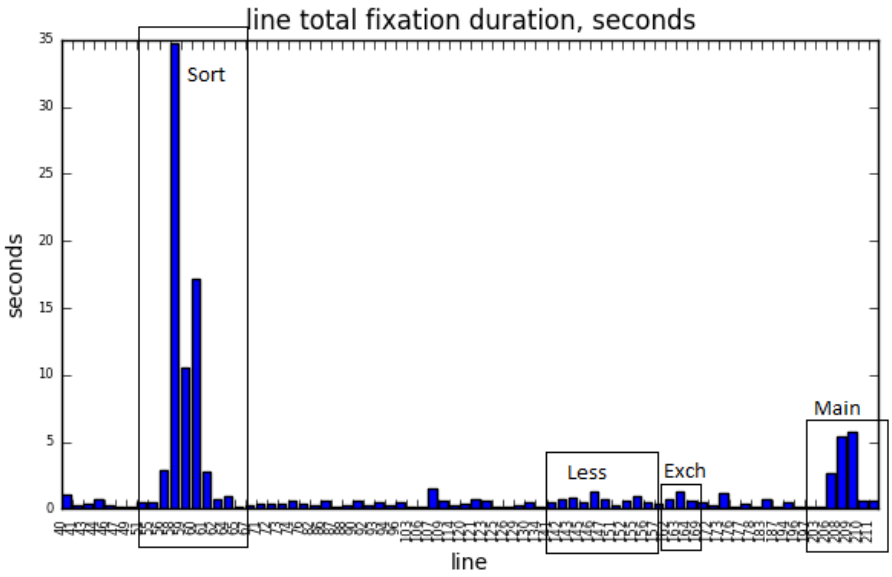
In this section we will look at the data for a single session. The user was a masters student at the Computer Science and Engineering study line at DTU and were put to the task defined in Section 4.1. The goal here is to see if the data can be used to get an impression of how the user worked. In particular it is interesting to see if the user was able to determine the critical parts in the source code for answering the questions.

To assist in reading the plots the methods and their lines are:

- The correct Sort method line 54 to 69.
- Other sort methods line 70 to 140.
- Less method line 141 to 160.
- Exchange method line 161 to 168.
- isSorted method line 169 to 192.
- Show method line 194 to 201.
- Main method line 202 to 214.

With these lines in mind the first step in seeing how the user worked could be to look at the total duration spent on each line. This can be seen in Figure 4.4 with added boxes to better see which lines belong to which methods.

Figure 4.4: Total duration, lines



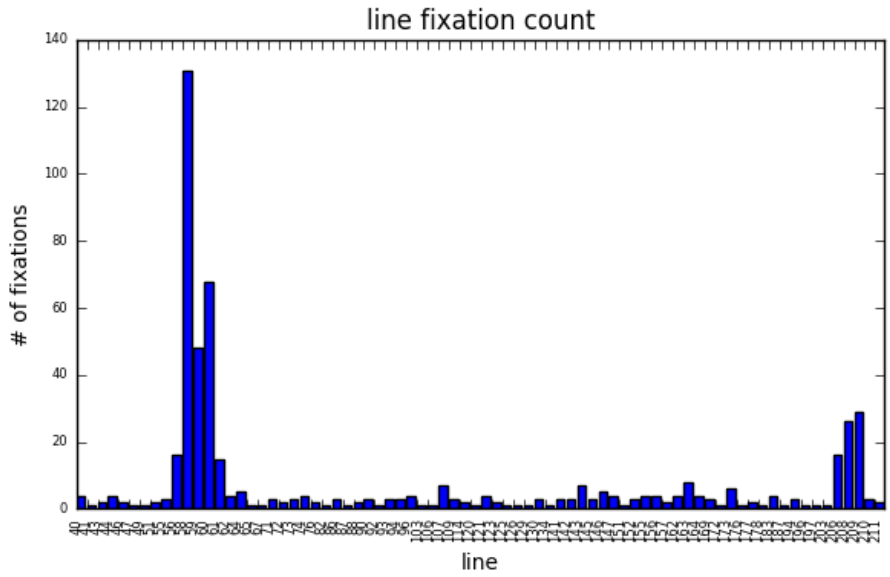
One thing to notice here is that this type of visualization is limited by the number of the bars shown. It is clear that if the file had been larger or if the user had fixated on each line it would quickly become unreadable. An alternative is then to use a histogram rather than a bar plot. This would provide readability at the cost of precision. Information specific to each line would be lost using a histogram.

For the specific case however it appears line 56 to 62 has been focused at a lot. These lines are part of the Sort method we expected the user to focus on. Additionally line 141 to 173 stands out with a small peak compared to the surrounding lines. These lines are within the Less and exchange method. Finally there is another peak at line 206 to 210, which represents the main method, this peak is significantly more noticeable than the less and exchange peak.

From this it can be seen that over all the user was primarily interested in the Sort method, the Less and to some extend the exchange method and noticibly interested in the Main method with a very large peak around the sort method.

In order to get an idea of the flow of the process it is worth looking at how many fixations were on each line. This can be seen in Figure 4.5

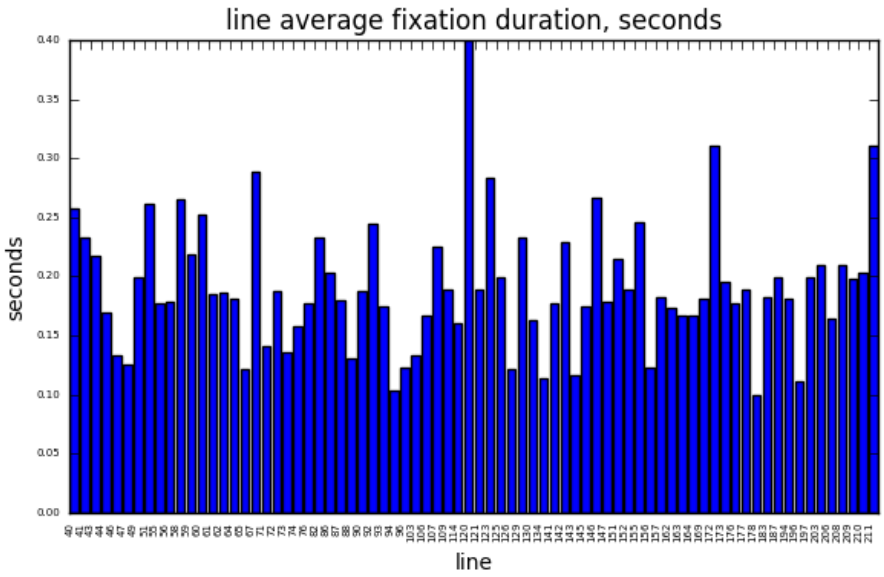
Figure 4.5: Fixation Count, lines



From the fixation counts the first thing to notice is that it shows almost the same picture as the total duration bars. It evens out the areas in between the peaks but otherwise the same is shown. The peaks are still around the Sort as the largest peak. The Less and exchange method as a peak area and the main method having three lines that are visited a lot. This makes sense as the Sort method is the method performing the logical operations and are as such expected to be the most complex method. The main method must be viewed in order to know which method is called and the values being sorted. Less and exchange are both called from the Sort method, and specifically Less may be interesting as its operations could be going both ways, in the sense of which argument is less than which argument.

In order to get an idea of how long all the fixations were on average at a place we can compare to the average fixation duration in Figure 4.6 to the number of fixations.

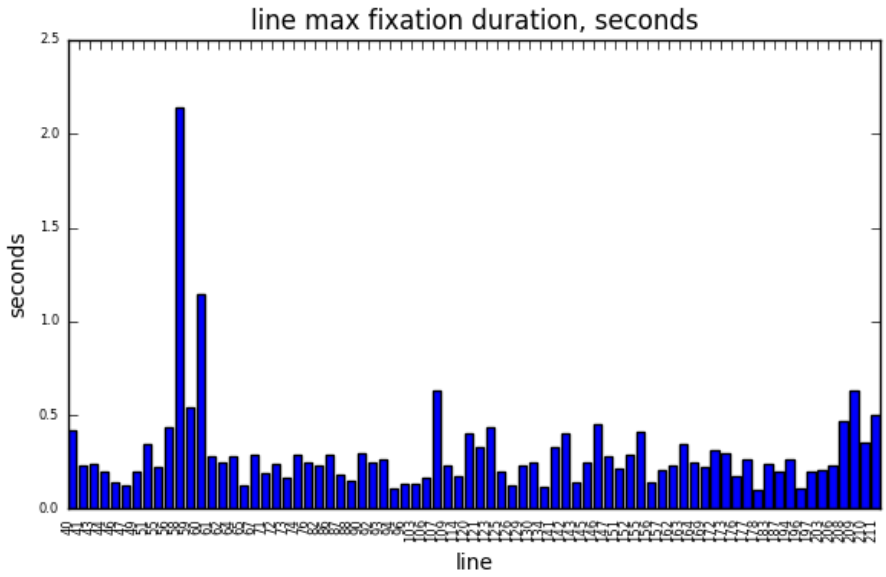
Figure 4.6: Average Fixation Duration, lines



Looking at the average duration diagram it does not appear to present much information about the process. It does however show that, not accounting for the rare cases with large averages such as line 120, the fixations were mostly between 100 and 300 ms. with one line going up to 400 ms. These numbers are within the expected range for fixations. The values show little impact from areas which were more complex than others. This is likely due to the complex areas having a large number of fixations which dilutes the maximum values.

Next a look will be taken at the maximum fixation duration of each line in order to see if the complexity of areas meant the user had longer fixations on these areas. The maximum duration for each line can be seen in Figure 4.7

Figure 4.7: Maximum Fixation Duration, lines



Here it can be seen that line 60 and 58 have the longest fixations. Where 60 are the outer for-statement and 58 is the Sort method declaration. There are also some slightly longer fixations at the main method.

The fixation of over two seconds at line 58 appear too long in comparison to the remaining fixations. However looking at the number of fixations and the average fixation duration, the value at 58 appear to be out of the ordinary and is not a typical occurrence as the average duration is not larger than other lines.

These values shows that the Sort method must have had a degree of complexity as the user dwelled long within a short area. The length is however much longer than expected of fixations and may be due to small saccades that was not caught by the threshold we used. On the rest of the data we see that the main method has a peak as well.

4.2.3 Bar Plots Type Analysis

From the bar plots in Figure 4.8 it can be seen that despite method declarations being the most viewed types in total time and number of fixations, the for

statement type has the longest fixations on average. This gives the impression that the hardest to read parts of the code was happening within for statements despite method statements being more common.

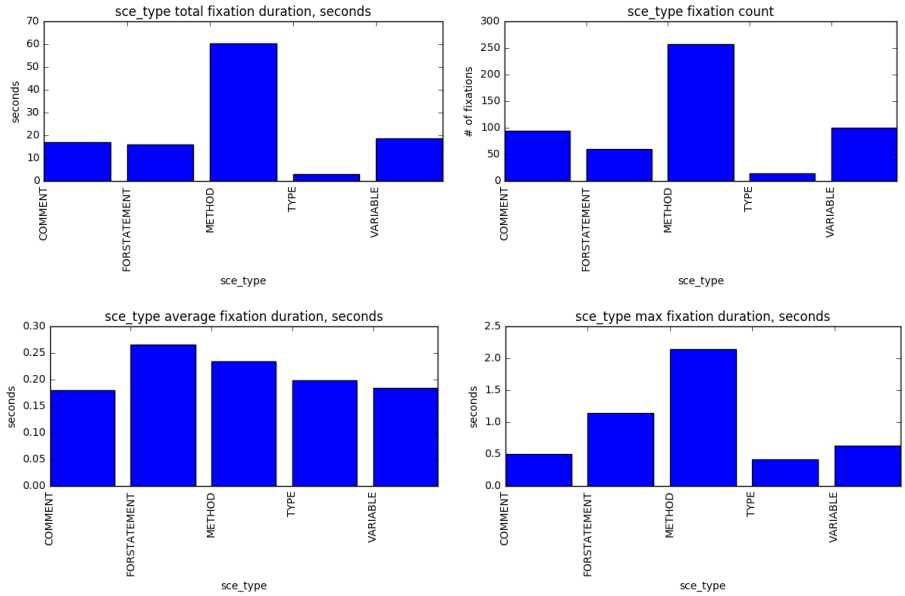


Figure 4.8: Bar plots For Source Code Entity Types

4.2.3.1 Bar Plots Analysis Conclusion

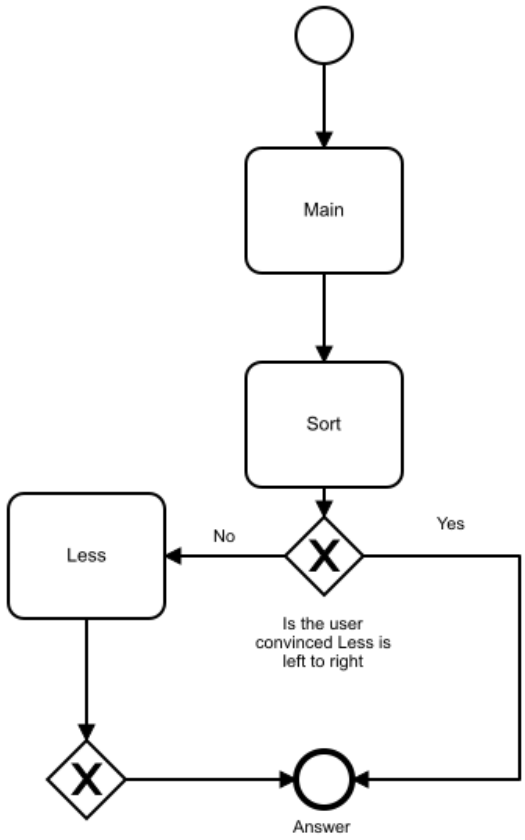
By simply looking at these data plots it seems likely that the user understood the task and was able to find the most important sections in the source code which would assist in answering the questions. The analysis shows that the user worked primarily on the Sort method, the Main method and to a lesser extend the Less method, while there was no discernible peaks during the other sorts. It shows that the Sort method was the most challenging of the three, with the For statement being the line looked at the longest in one fixation. The analysis does however not provide any insight into the order of the fixations apart from the fixation count which presents how many times the area was re-visited in the sense of how many fixations were at the area.

4.2.4 Process Mining

Before looking at the process mined data for the five users we will take a look at what the expected process looks like.

While it is very unlikely that the users will follow a specific process of reading the source code, it can be expected in order for them to answer the questions, to follow the process described in Figure 4.9 at some sub path of reading process. This does not take into account anything visited while looking for the specific methods, or any re-reading in case the user becomes unsure.

Figure 4.9: Insertion Sort reading sub process



To create the event logs the script in figure 4.10 was used along with the fixations created in Figure 3.7. The full script used to create plots and event logs can be

found in Appendix H

Figure 4.10: iTrace analysis script

```
IA.process_mining_data([iTrace1,iTrace2,iTrace3
,iTrace4,iTrace5],"test_sort_lines.csv",'lines')
IA.process_mining_data([iTrace1,iTrace2,iTrace3,iTrace4
,iTrace5],"test_sort_userdef.csv"
,'user_defined',"insertion.csv")
```

For all the figures in the analysis it is worth noting that all the Disco diagrams shown section has had their activities and paths lowered, meaning that the least visited areas and transition paths are removed. This is to avoid having upwards of 200 nodes with an equal amount of paths going to them.

4.2.4.1 Disco Lines Analysis

In this part of the analysis a look will be taken at the event logs where each line is considered an activity. As mentioned in 3.5.2. The lines have the SCE types which exists on the line written as well as the line number. This is intended to assist in readability to make the reader less dependent of having the source code on the side. An overview of what methods are on which lines can be seen in Section 4.2.2.1

Looking at Figure 4.11 it can be seen that similar information as we saw in the bar plots can be gathered here. However Figure 4.11 represents all four users who finished the task, and consecutive fixations on the same line has been aggregated. Looking at the diagram for the total duration spent on each activity combined for all the users, it can be seen that lines 58 to 61 are the most visited lines. This matches what we saw in the Bar plot analysis in Section 4.2.2. It is also expected that the Main method, line 202 to 214, and the Less method, line 141 to 160, had significant time spent on them.

It can be seen that line 209,206 and 208 each made it through the exclusion made by Disco. While they have significantly less time spent on them, as also seen in the bar plot analysis, they still appear significant.

On the left side in Figure 4.11 we see the lines 143,146,147 and 152 next to each other. These are the important part of the Less method which contains

the method declaration and the return statement.

It is also seen that the less statement is typically read in concession and is thereby consecutive in the diagram. Whereas the Sort and main method are split up in different sections of the diagram as the Sort method is visited more often from various sections of the code.

Figure 4.11: Line wise, total duration

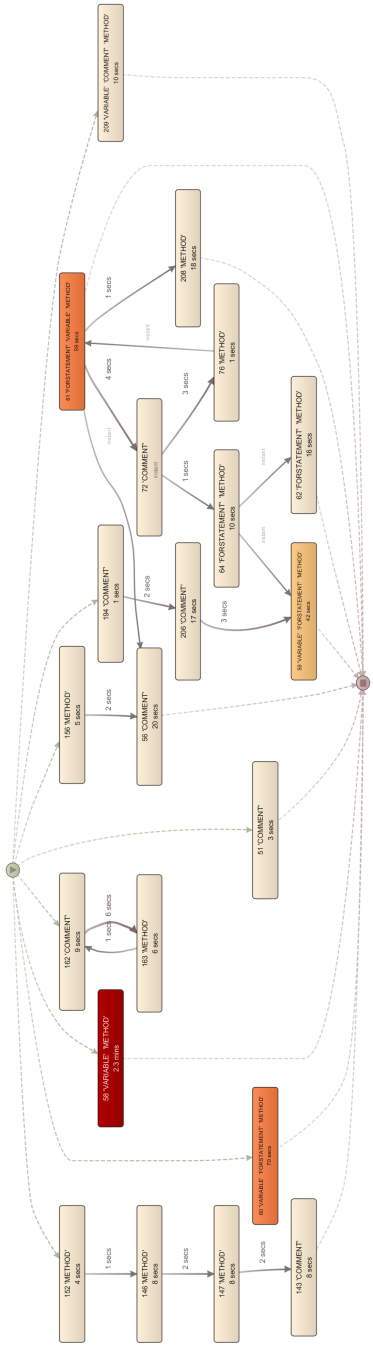


Figure 4.12 shows how long the longest views within the lines were. This figure shows that line 58 and 60 have the longest views. This is the same result as seen in the Bar plot analysis, however here it is for all four users, providing even more assurance that these lines take longer to understand than other lines in the source code.

Figure 4.12: Line wise, max duration

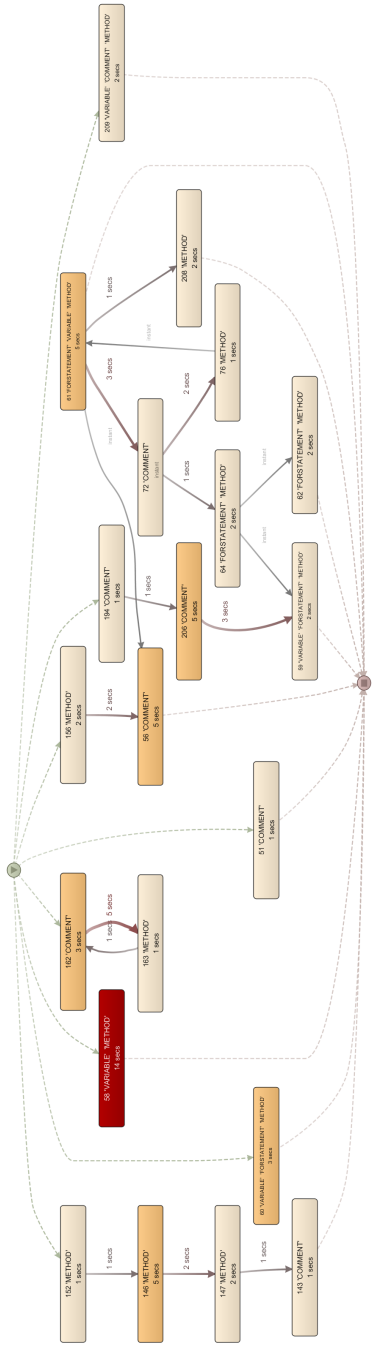
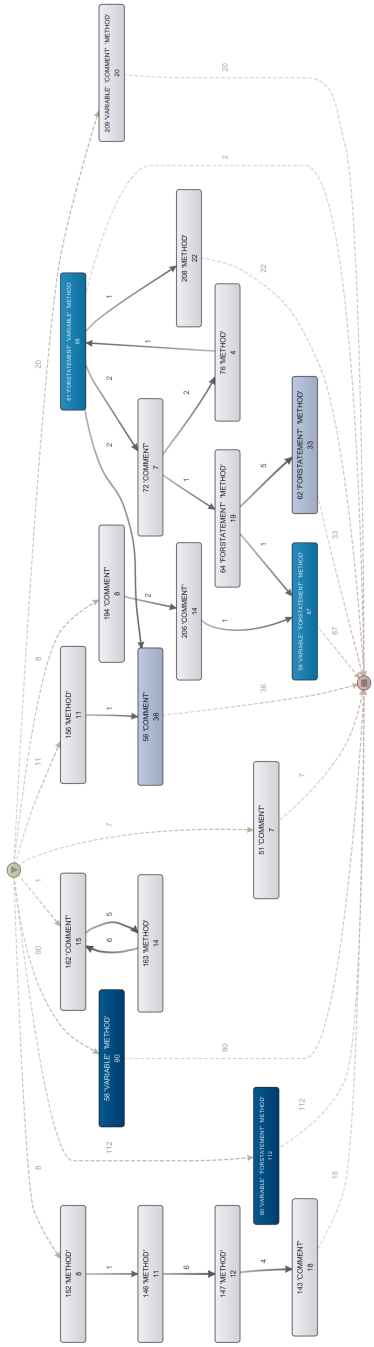


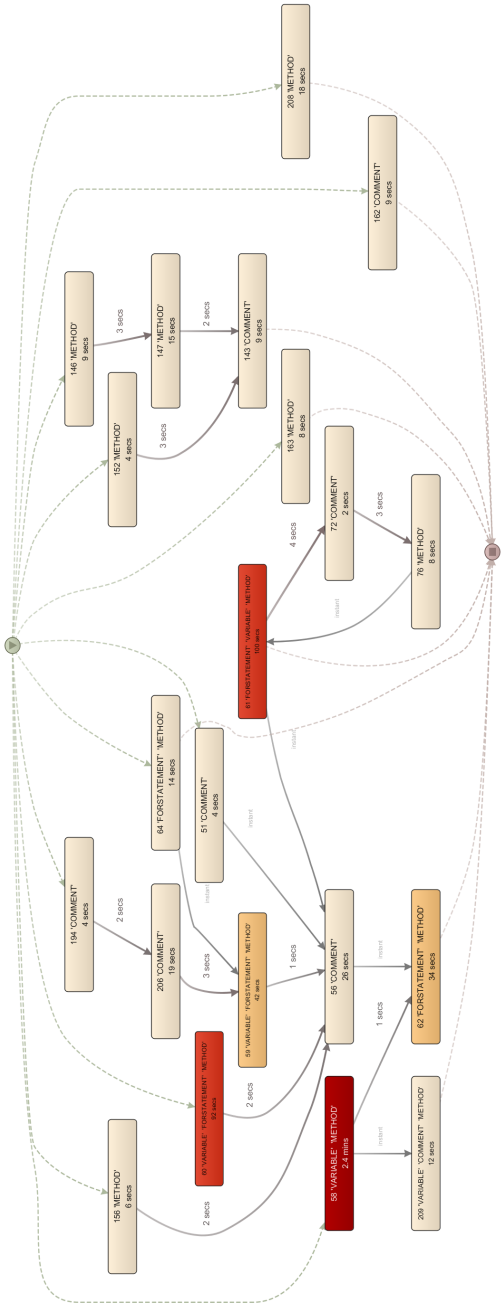
Figure 4.13 shows the frequencies for each action. The frequencies are the number of times an action is performed. The figure shows the same results as the previous two Disco diagrams in the sense that the activities with the most time spent also tend to have been visited the most times.

Figure 4.13: Line wise, frequencies



Lastly we look at the disco diagram for all the users, including the user who did not finish. This can be seen in Figure 4.14. It would make sense to look at the user who did not finish the task by himself, however Disco does not remove paths and activities if there is only one case. The user who did not finish fixated on 118 lines. In order to get an idea of the idea of the difference between the the fifth user and the remaining four, we will explore the statistics.

Figure 4.14: Line wise, All users



Some statistics to be seen on the different users in the statistics part of Disco: Number of activities, unique activities not including re visits and session duration.

- User 1: 76 Activities, Duration: 4m50s
- User 2: 65 Activities, Duration: 4m25s
- User 3: 76 Activities, Duration: 6m58s
- User 4: 68 Activities, Duration: 3m15s
- User 5: 118 Activities, Duration: 11m9s

As such it can be seen that the first four user fixated on 65 to 76 unique lines. Additionally by looking at the total number of unique activities for the four users who finished it can be seen that had a total of 114 unique lines in total.

This shows that the user who did not finish was significantly more thorough in reading most of the code than the users who finished were. This along with the time spent gives the impression the user was trying to find something in the code for longer, resulting thorough scanning of most of the lines and thereby more time spent.

4.2.4.2 Disco Lines Analysis Conclusion

The first thing to notice was that the most common lines were represented in the diagram, which presented the same image as we saw in Section 4.2.2. Which is that the Main method, the less method and the Sort method were the three most viewed sections in the source code. This holds true for the Disco analysis and as such the same values were available. It was seen that the majority of the time was spent on the Sort method, the lines in the Sort method was the most frequently visited parts and had the longest single fixations.

As for the process it was hard to get a full idea of how they worked due to the large number of activities and paths as there were 114 activities for the four who finished. This forced us to look at a very low number of paths and activities which lost a lot of the context. It may be possible to look into these individually however the complexity makes it undesirable.

4.2.4.3 Disco User Defined Area of Interests Analysis

To make the process less complex and provide a higher level analysis rather than a line by line analysis the iTraceAnalyser was made to allow the user to define areas of interest. On the same set of fixations as the Line analysis was conducted, an event log has been created with a set of user defined activities created for the specific experiment.

These activities are made to represent areas of interests, AoIs. The idea of an AoI is to group together lines which represent the same section. The example of sections used here is the methods we are interested in. The areas in between these methods.

As seen in Figure 4.9 the areas we are interested in are the Main method, the Sort method and the Less method. In addition to these it may be interest to see if the users also looked at the Exchange method.

Thus in this analysis each AoI is considered an activity instead of considering each line an activity. Figure 4.15 shows the .csv file used to defined the AoIs.

Figure 4.15: Insertion.csv rules for areas of interest

```
Sort,54,69
Other sorts,70,140
Less,141,160
Exchange,161,168
isSorted,169,192
Show,194,201
Main,202,214
```

With these rules an event log with seven actions are created based on the methods described in 4.1.
Sort is the sort method the user is asked to work on.
Other sorts is the sorts which take different arguments. The user is not supposed to look at these. It is however a large section covering 70 lines which means that several short fixations within this area is expected.
Less is a method comparing two values. Less is called in the Sort method.
Exchange is a method switching around the position of two values
isSorted is a method used in asserting that the list is sorted for a given interval.
Show is a method printing the list.

Main is the main method calling Sort.

Looking at Figure 4.16 it can be seen similarly to the Disco line analysis and the Bar plot analysis, that the majority of the time was spent on the Sort method, with the Main and Less methods also receiving some focus. The Show, exchange and isSorted have all had less than 10 seconds of fixations on average per user.

Due to the low number of activities the diagram can easily contain all the activities without being cluttered. The complexity will still increase if all paths were added so for this example it will be left at the lowest number of paths. This should also help see which paths were the most general across the users.

This solution provides a clearer representation of the general process flow. A couple of observations to be seen:

- Sort transitions to Other sorts
- Other sorts transition to either Show or Exchange
- Show transitions to Main.
- Main transitions isSorted
- isSorted transitions to Less
- Exchange transitions to Sort
- isSorted transitions to Main
- Less transitions to Sort.
- Sort, Less and Main all transition to themselves.

The diagram as such essentially shows that Sort is the primary area the user fixations resides. From here it moves down to Main and Less and back up to Sort. Alternately it moves from Other sorts down to Exchange and back up to Sort again.

This reasonably matches the assumption of the user going to the Main method to see which values are in the list, and then back to Sort until they decide to check the Less method. As such this diagram gives the impression that the users are following the flow of the method calls. The sub process seen in 4.9 is also seen in the movement of the process.

The loop transitions in the Sort activity is most likely due to the user looking away from the screen and then back. In this setup the users was allowed to use paper and pencils which will have caused the long transitions where they look at the Sort method followed by looking at the paper and then finally looking back at the Sort method.

Figure 4.16: User defined, total duration

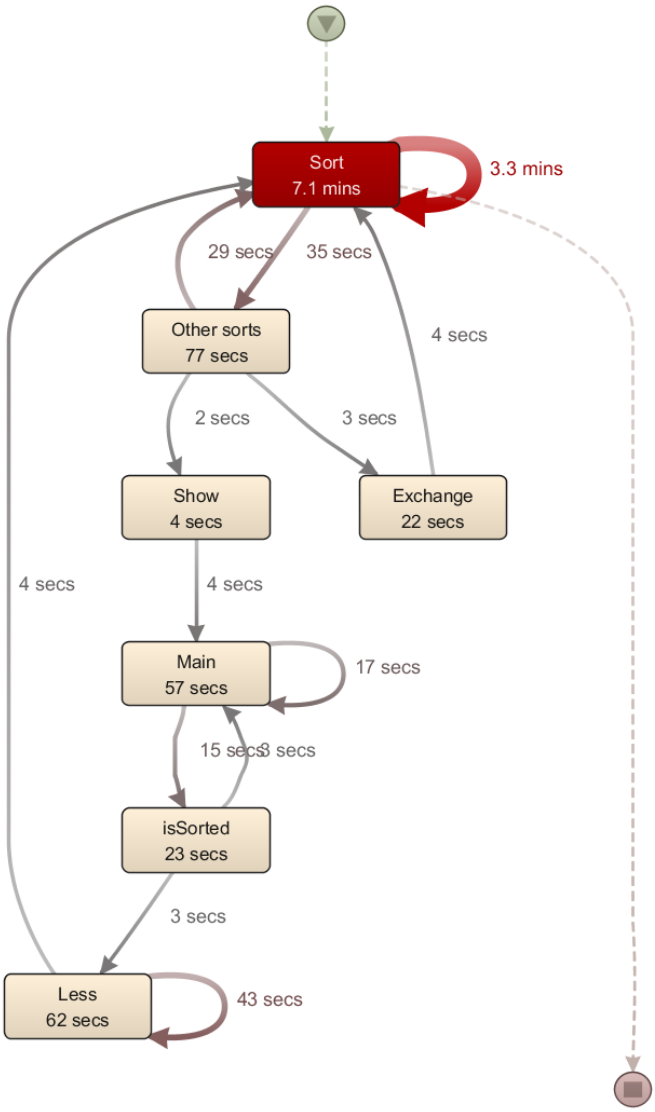


Figure 4.17 shows the graph depicting the maximum duration. Here it can be seen that the Sort method has the longest single viewing. The values here differs from the longest view in the Line wise analysis. This is due to the collapsing of consecutive fixations on the same AoI within the same area with less than a second between the fixations. This is happening over a significantly larger area than it did in the line wise analysis. As mentioned in Section 3.5.2 this is made due to an interest in knowing how long they focused within a specific area, and the areas with this method may be large enough that you can change area of fixation and remain within the same area of interest.

Figure 4.17: User defined, max duration

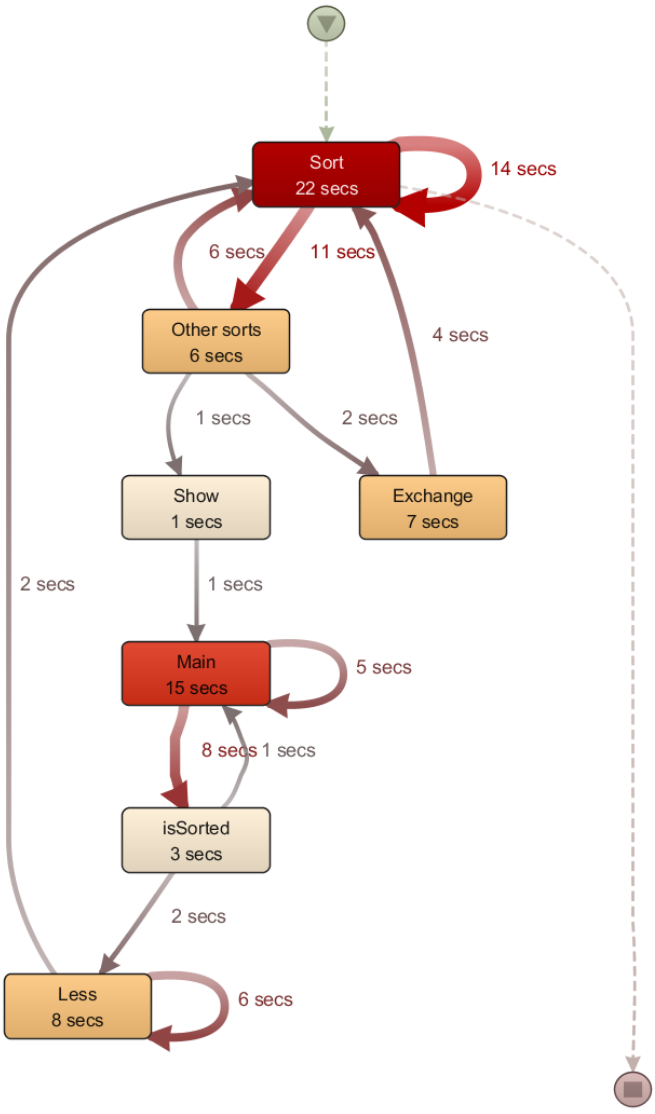
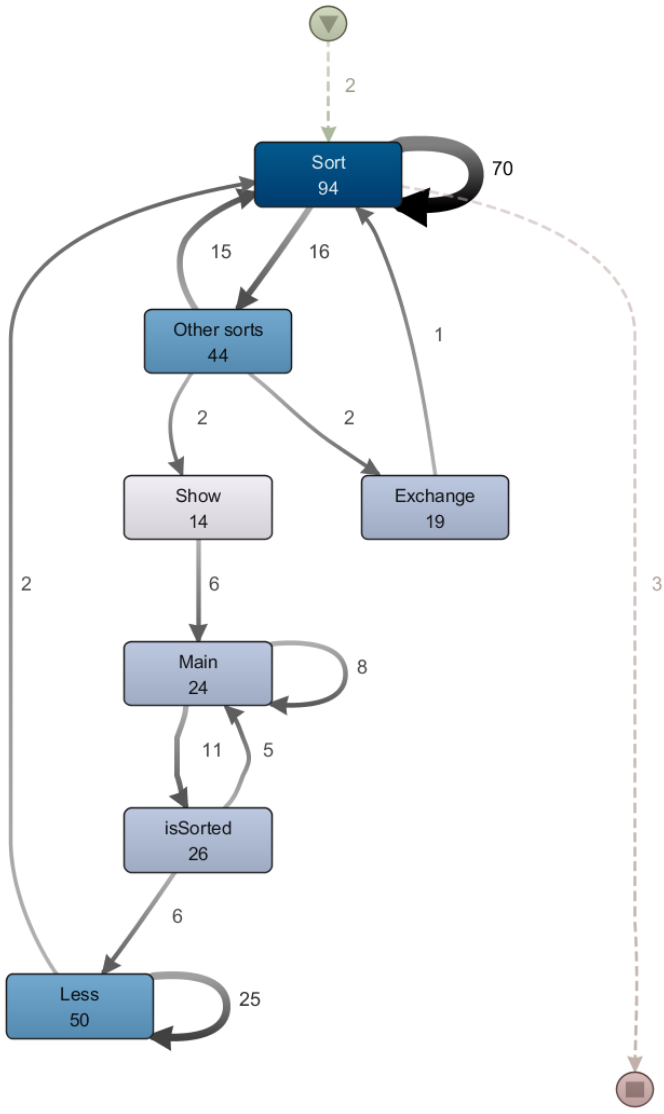


Figure 4.18 shows the frequencies of each activity. The major take away here which was not as visible in the previous analysis is how often the the user transitioned within the Sort method. The figure also shows that less had almost twice the revisits of the Main method, despite the duration of both only being five seconds apart.

Figure 4.18: User defined, frequencies

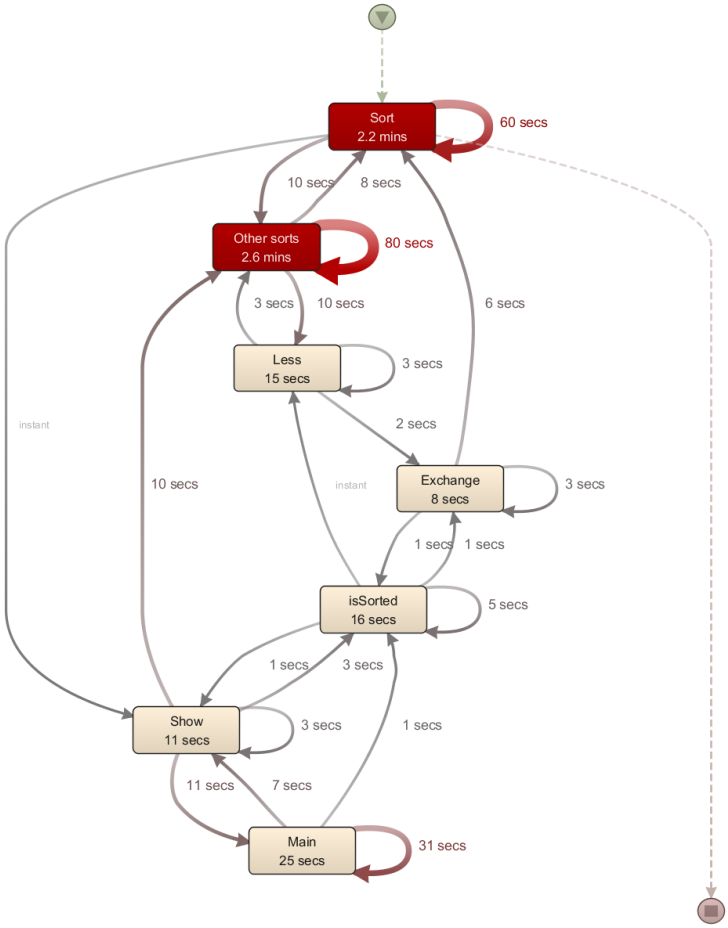


Due to the low number of activities it is possible to look at the process for the user who did not finish the task. This process can be seen in Figure 4.19. Here it can be seen that including transitions the user spent four minutes inside the Other sort section. Compared to the 77 seconds from the other four users in total, this is a very significant time. The user also spent almost a minute reading the main method.

Connecting this with the number found in the line wise analysis where it was seen that the user looked at more lines than the other four users in total. This gives the idea that the user who did not complete may have attempted reading the entire file before answering the question.

The process of moving from Sort to main and back to Sort is also seen here, however the durations tell us that this process happend over a significant time compared to the other users.

Figure 4.19: User defined, Did not complete the task



4.2.4.4 Disco User Defined Analysis Conclusion

The user defined sections makes it easier to read the Disco results as it abstracts away from the source code and provides a higher level analysis. While this solution did not present any information about the individual source code entities or lines, it made the overall process easier to read as the number of activities

and paths were significantly reduced.

Additionally the analysis revealed how the user who did not finish, read the code different to the other users and did not appear to follow the flow of the code but rather read all the sort methods. Making it appear the user did not immediately understand which sections of the file was necessary to answer the questions.

Chapter 5

Discussion

5.1 Data Preparation

5.1.1 Data Collection

The data collection mostly worked as expected in the test cases used with the eye tracking working within a lines accuracy for all the users and for most users the crosshair was close enough to the actual point which the user looked at that there did not appear to be any issues with the accuracy of the tracking.

The export to the database also worked seamlessly and all were saved in the first try, that is to say no tests were discarded due to bad calibration, bad saves or other crashes.

These results were as expected as the work was done with iTrace which already presented itself as a working piece of software, and the implementation of the new native interface was done in way matching the old implementation and should not have any loss of functionality.

The number of tests and test subjects are very limited in that each session were 3-7 minutes. With the user who did not finish spending 11 minutes. As such

it was done in short sessions and on a low number of users. This means a lot more work can be done on collecting and analyzing real world data. This thesis does however provide some insight into what the analysis can look like. As well as providing the tools for collecting and storing the data.

5.1.2 Fixations

Fixation identification of the gaze data remains an important aspect of the analysis as the data loses value and credibility if the user cannot trust that the identified fixations are the true fixations. As such the implementation relied on a trusted algorithm for the identification of the fixations.

In Section 2.1.2 we see an analysis of the implemented fixation identification algorithm in use. While it is not clear which threshold provides the most accurate results, it is clear that while the threshold is not set to an extreme such as 5 pixels, the fixations are covering the same areas but with different amounts and duration of the fixations.

So while further work can be done to ensure the most accurate results the fixations collected does to some extent match what is expected from seeing the gaze clusters.

5.2 Fixation Data Analysis

The analysis of the fixation data in Section 4.2.2.1 provides a quick view of some statistics of the fixation data. This serves the purpose giving quick access to certain metrics of the data.

While it does allow the user to understand where the majority of the time and fixations were placed, as well as which areas required the most focus. This type of analysis does not appear to provide the same degree of insight and possibilities for further analysis as the process mining method. This may in part be due to the size of the data and how the plots does not make it immediately relateable to a source code editor. For this other visualizations should provide more useful, and for this thesis the focus was then moved to considering the fixations as processes.

5.3 Process Mining

The idea of using process mining as the primary visualization method for analyzing the fixation data of a source code reading session provided a very valid way of seeing how the users approached the source code. By grouping neighbouring fixations at the same areas of interest it helps raise the level of abstraction to a point where it is more immediately clear what the user was working on at a given time in the process.

For the analysis in Section 4.2.4 two separate methods were used. The analysis approach remains mostly the same with the major difference being in the way the event logs are created.

Of the line wise events and the user defined area of interest events both provide insight and statistics on the cases. It is however clear that for analysis of larger scale tests it may be a clear advantage to raise the abstraction to larger areas and create more meaningful names to each area.

For smaller parts, such as understanding which part of a given source code area the user spent the time and focus on, it may however be an advantage to see it line by line. In some cases even SCE by SCE.

In this thesis the conformance and compliance checking was done very informally with merely having an idea of what was expected to be seen, which four of the users to a large extent lived up too and the fifth user varying from the assumed approach.

There may be a lot to gain by formalizing the compliance testing to find if there exists sub routines which conforms to the expected process.

This may have the potential for use in regards to using heuristics to score each user and get immediate approximations of how well the user did before performing exploratory analysis on each user in detail.

5.4 Future Work

This thesis covered the proposed tools for and methods for creating an analysis of code comprehension tasks by developers. The examples used in the evaluation in Section 4, are limited in number of users and complexity of the task as well as number of tasks. Larger scale testing would be necessary in order to be more

certain of the usefullness of the solution.

Additionally conformance testing and compliance checking could be used for automating estimates of how well a user is doing. This along with the right heuristics may make it possible to detect developers who had issues automatically.

Chapter 6

Conclusion

This thesis presents a system for creating visualizations of eye tracking data gathered using the gaze to source code mapping Eclipse plugin, iTrace. The first task of the thesis serves to update the native driver connecting to the eye tracker to function with the current versions of the Tobii SDK. The second task implements a solution which extends the iTrace system with a MySQL database and the ability to transfer gaze data seamlessly into the local database. The proposed system includes a two part python analysis library, iTraceAnalyser. The tool loads a session from the database and converts the gazes into fixations. The analyzer provides an easy way of creating simple visualizations representing the fixation data in Bar plots for quick overview of single sessions. The primary purpose of the tool is to convert the data gathered with iTrace into event logs compatible with process mining tools such as Disco by Celonis. The evaluations of the tool has shown that the event logs, which come as both a general line wise activity definition and activities which can be defined by the user, serves as a good all around tool for analyzing and comparing several sessions at once. The larger the areas of interests are the higher level of analysis can be done on the processes by the users.

Appendix A

Glossary

Abstract syntax tree. In relation to iTrace the abstract syntax tree holds the structure for the source code in order to find information on the specific SCEs.

Calibration. In eye tracking calibration is the act of updating the eye trackers knowledge of the characteristics of the users eye, such as shape, light refraction and reflection properties of parts of the eye. [Tob17c]

Eye tracker. The eye tracker is the hardware used for eye tracking and is a camera with near infrared lights used to take images of the eyes for use in eye tracking.

Eye tracking, using a camera to detect where on the screen the users eyes are looking. It maps the position of the users eye to a position on the screen and returns X and Y pixel coordinates.

Fixation. Fixations are points where the user concentrates for an extended time. As such a fixation relates to gazes as fixations can be considered a cluster of gazes around the same point over an extended time. Fixations at the very minimum contains X and Y coordinates and a start time and an end time.

Gaze. Gazes are the raw outputs from the eye tracking. Each gaze is an X,Y coordinate set relating to a pixel area on the screen. A gaze may be attributed

more values depending on the tool most relevant for this thesis among these are the recorded time for the gaze and the source code data related to the gaze point.

Saccades. Saccades are the movement between fixations. When the areas are moving from one point of focus to another the gazes in between are considered the saccade.

Fixation identification algorithms. The algorithms used for converting gazes into fixations.

IGazeHandler is the component of iTrace which is responsible for collecting the data from the tracker and collecting further information about the connected SCEs, line numbers etc. Outputs the gaze data as an IGazeResponse

IGazeResponse is the structured gaze data which includes the information collected by the IGazeHandler.

ISolver is the super class responsible for outputting the IGazeResponses to other external sources such as XML and JSON.

iTrace. An Eclipse plugin which couples eye tracking gazes with the source code entities in Eclipse.

Java native interface is a java library used for calling C and C++ code in a java environment. In this thesis the term JNI or driver is used to describe the C++ module connecting to the eye trackers software development kit (SDK).

Source code entity. SCEs are parts of the source code that can be categorized. These include highlighted words and comments.

Tobii Analytics 3.0 is the old Tobii SDK which is no longer available since the release of the Tobii Pro SDK.

Tobii Pro is the current SDK for Tobii eye trackers as of the composing of the thesis.

Appendix B

Components installation

B.1 JNI

If you download and install iTrace from their Github, you can just replace their version of the file at iTrace-Archive/tree/master/jni/TobiiTracker With the one in appendix C.

You will need to manually change the location of your license on line 209 to the location of your license.:

```
license_success = apply_licenses_example(
native_data->eye_tracker,"C:/Users/Dennis/Desktop/Tobii_license");
```

Additionally when you have downloaded the SDK from
https://www.tobiipro.com/product-listing/tobii-pro-sdk/Download
(You must download 32bit C)

You need to add

```
#ifdef __cplusplus
extern "C" {
#endif
```

At the top of the each of the headers in the include folder before the code section.

And:

```
#ifdef __cplusplus
}
#endif __cplusplus
```

at the bottom of the file.

To install iTrace you can follow the guide on the readme at https://github.com/SERESLab/iTrace-Archive Or build it in an IDE such as visual studio. (This will allow you to explicitly tell the compiler where all the libraries are in case the makefiles have trouble finding them)

As a checklist for installing it, make sure you have installed: - Ivy plugin for Eclipse,

- Have the SDK somewhere your compiler can see it. If you choose to use a compiler in something like Visual studio you can explicitly tell it where the header and library files are. (With the extern C lines in the headers)

- An up-to-date Java development kit

- Also you need the JNI It should be installed along side the JDK. but not all versions have it. Mine was found in jdk1.8.0$_1$4432$which ineeded to explicitly tell my visual studio compiler$.

B.2 Database

To get the database working to use the python class iTraceAnalyser the following is needed: MySQL, run the SQL file in Appendix E
Replace the Controlview.java and iTraceJava with the new versions of the files.
Add the SQLGazeExportSolver.java in Appendix F

Appendix C

JNI, New SDK

edu_ysu_itrace_trackers_TobiiTracker.cpp
Note. The lines commented out are parts of the old implementation left in for comparison.

```
#define _CRT_SECURE_NO_DEPRECATE
#include <ctime>
#include <exception>
#include <tobii_research.h>
#include <tobii_research_calibration.h>
#include <tobii_research_eyetracker.h>
#include <tobii_research_streams.h>
#include <iostream>

/**include <tobii/sdk/cpp/MainLoop.hpp>
#include <tobii/sdk/cpp/EyeTracker.hpp>
#include <tobii/sdk/cpp/EyeTrackerInfo.hpp>
#include <tobii/sdk/cpp/EyeTrackerException.hpp>
#include <tobii/sdk/cpp/EyeTrackerBrowser.hpp>
#include <tobii/sdk/cpp/EyeTrackerBrowserFactory.hpp>
#include <tobii/sdk/cpp/EyeTrackerFactory.hpp>
#include <tobii/sdk/cpp/GazeDataItem.hpp>
**/
```

```
19 #include "edu_ysu_itrace_trackers_TobiiTracker.h"
20 #include
    ↪ "edu_ysu_itrace_trackers_TobiiTracker_BackgroundThread.h"
21 #include "edu_ysu_itrace_trackers_TobiiTracker_Calibrator.h"
22
23 //using namespace tobii::sdk::cpp;
24
25 struct TobiiNativeData
26 {
27         JavaVM* jvm;
28         jobject j_tobii_tracker;
29         jobject j_background_thread;
30         TobiiResearchEyeTracker* eye_tracker;
31         TobiiResearchCalibrationData* calibration_data = NULL;
32         TobiiResearchCalibrationResult* calibration_result =
          ↪ NULL;
33         //MainLoop main_loop;
34
35
36 };
37
38
39
40 //Only one TobiiTracker can be active at one time.
41 bool g_already_initialised = false;
42 //Sort of ugly but necessary. When connecting to eye tracker,
    ↪ this is used to
43 //pass the tracker information to the main thread.
44 //EyeTrackerInfo::pointer_t g_et_info =
    ↪ EyeTrackerInfo::pointer_t();
45 //Also not very clean.
46 TobiiNativeData* g_native_data_current = NULL;
47
48 void throwJException(JNIEnv* env, const char* jclass_name, const
    ↪ char* msg)
49 {
50         jclass jclass = env->FindClass(jclass_name);
51         env->ThrowNew(jclass, msg);
52         env->DeleteLocalRef(jclass);
53 }
54
55 jfieldID getFieldID(JNIEnv* env, jobject obj, const char* name,
    ↪ const char* sig)
56 {
```

```
57          jclass jclass = env->GetObjectClass(obj);
58          if (jclass == NULL)
59                  return NULL;
60          jfieldID jfid = env->GetFieldID(jclass, name, sig);
61          if (jfid == NULL)
62                  return NULL;
63          return jfid;
64  }
65
66  TobiiNativeData* getTobiiNativeData(JNIEnv* env, jobject obj)
67  {
68          jfieldID jfid_native_data = getFieldID(env, obj,
            ↪  "native_data",
69                  "Ljava/nio/ByteBuffer;");
70
71          if (jfid_native_data == NULL)
72          {
73                  return NULL;
74          }
75
76          jobject native_data_bb = env->GetObjectField(obj,
            ↪  jfid_native_data);
77
78          return (TobiiNativeData*)
            ↪  env->GetDirectBufferAddress(native_data_bb);
79  }
80
81  JNIEXPORT jboolean JNICALL
82          Java_edu_ysu_itrace_trackers_TobiiTracker_00024BackgroundThread_
83            jniBeginTobiiMainloop
84          (JNIEnv* env, jobject obj)
85  {
86
87          //Initialise Tobii SDK if not yet initialised, else an
            ↪  error condition has
88          //occurred.
89          if (g_already_initialised)
90                  return JNI_FALSE;
91          else
92          {
93                  //tobii::sdk::cpp::Library::init();
94                  g_already_initialised = true;
95          }
```

```
96          //Get native data ByteBuffer field in TobiiTracker
            ↪  object.
97          jfieldID jfid_parent = getFieldID(env, obj, "parent",
98                  "Ledu/ysu/itrace/trackers/TobiiTracker;");
99          if (jfid_parent == NULL)
100                 return JNI_FALSE;
101         jobject parent_tobii_tracker = env->GetObjectField(obj,
            ↪  jfid_parent);
102         jfieldID jfid_native_data = getFieldID(env,
            ↪  parent_tobii_tracker,
103                 "native_data", "Ljava/nio/ByteBuffer;");
104         if (jfid_native_data == NULL)
105                 return JNI_FALSE;
106         //Create structure to hold instance-specific data.
107
108         TobiiNativeData* native_data = new TobiiNativeData();
109
110
111         jobject native_data_bb = env->NewDirectByteBuffer((void*)
            ↪  native_data,
112                 sizeof(TobiiNativeData));
113         //Set java virtual machine and BackgroundThread
            ↪  reference.
114         env->GetJavaVM(&native_data->jvm);
115         native_data->j_background_thread =
            ↪  env->NewGlobalRef(obj);
116         //Store structure reference in Java object.
117         env->SetObjectField(parent_tobii_tracker,
            ↪  jfid_native_data, native_data_bb);
118
119
120         //Run!
121         //native_data->main_loop.run();
122
123         //This code does not execute until the main loop has been
            ↪  stopped.
124         //delete native_data;
125
126         //return JNI_TRUE;
127 }
128
129 /**void handleBrowserEvent(EyeTrackerBrowser::event_type_t type,
130         EyeTrackerInfo::pointer_t et_info)
131 {
```

```
132         if (type == EyeTrackerBrowser::TRACKER_FOUND)
133                 g_et_info = et_info;
134 }**/
135 int apply_licenses_example(TobiiResearchEyeTracker* eyetracker,
    ↪  const char* license_file_path) {
136 #define NUM_OF_LICENSES 1
137         char* license_key_ring[NUM_OF_LICENSES];
138         FILE *license_file = fopen(license_file_path, "rb");
139         if (!license_file) {
140                 printf("License not found!\n");
141                 return 0;
142         }
143         fseek(license_file, 0, SEEK_END);
144         size_t file_size = (size_t)ftell(license_file);
145         rewind(license_file);
146         if (file_size <= 0) {
147                 printf("License is empty!\n");
148                 return 0;
149         }
150         license_key_ring[0] = (char*)malloc(file_size);
151         if (license_key_ring[0]) {
152                 fread(license_key_ring[0], sizeof(char),
                    ↪  file_size, license_file);
153         }
154         fclose(license_file);
155         printf("Applying license from %s.\n", license_file_path);
156         fflush(stdout);
157         TobiiResearchLicenseValidationResult validation_results;
158         TobiiResearchStatus retval =
            ↪  tobii_research_apply_licenses(eyetracker, (const
            ↪  void**)license_key_ring, &file_size,
            ↪  &validation_results, NUM_OF_LICENSES);
159         free(license_key_ring[0]);
160         if (retval == TOBII_RESEARCH_STATUS_OK &&
            ↪  validation_results ==
            ↪  TOBII_RESEARCH_LICENSE_VALIDATION_RESULT_OK) {
161                 printf("Successfully applied license from list of
                    ↪  keys.\n");
162                 fflush(stdout);
163                 return 1;
164         }
165         return 0;
166 }
167 JNIEXPORT jboolean JNICALL
```

```
168 Java_edu_ysu_itrace_trackers_TobiiTracker_jniConnectTobiiTracker(
169         JNIEnv* env, jobject obj, jint timeout_seconds)
170 {
171         //Get native data from object.
172 
173         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪  obj);
174 
175         if (native_data == NULL)
176         {
177                 return JNI_FALSE;
178         }
179         //Set TobiiTracker reference.
180 
181         native_data->j_tobii_tracker = env->NewGlobalRef(obj);
182         //Find Tobii trackers.
183         /**EyeTrackerBrowser::pointer_t browser =
184                 EyeTrackerBrowserFactory::createBrowser(native_data->main_loop);**/
185         /**browser->start();
186         browser->addEventListener(handleBrowserEvent);**/
187         time_t start_time = time(NULL);
188         //Wait until found or timeout occurs.
189         //Currently does not have this
190         while (0)//g_et_info == NULL)
191         {
192                 if (time(NULL) > start_time + timeout_seconds)
193                 {
194                         //browser->stop();
195                         return JNI_FALSE;
196                 }
197         }
198         //EyeTrackerInfo::pointer_t et_info = g_et_info;
199         //browser->stop();
200 
201         //Connect eye tracker
202         /**cker = et_info->getEyeTrackerFactory()->
203                 createEyeTracker(native_data->main_loop); **/
204         TobiiResearchEyeTrackers* eyetrackers = NULL;
205     TobiiResearchStatus result =
        ↪  tobii_research_find_all_eyetrackers(&eyetrackers);
206         native_data->eye_tracker = eyetrackers->eyetrackers[0];
207         //Setting license
208         int license_success;
209         //HARD CODED. BAD
```

```
210         license_success =
            ↪ apply_licenses_example(native_data->eye_tracker,
            ↪ "C:/Path/To/Tobii_license");
211
212         //Needs a check for whether eyetracker is found.
213
214
215     // int64_t system_time_stamp;
216     // TobiiResearchStatus status =
        ↪ tobii_research_get_system_time_stamp(&system_time_stamp);
217
218         return JNI_TRUE;
219 }
220
221
222 JNIEXPORT void JNICALL
    ↪ Java_edu_ysu_itrace_trackers_TobiiTracker_close
223         (JNIEnv* env, jobject obj)
224 {
225         //Get native data from object.
226         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪ obj);
227
228
229         if (native_data == NULL)
230         {
231                 throwJException(env,
                    ↪ "java/lang/RuntimeException",
232                         "Cannot find native data.");
233                 return;
234         }
235
236         //Shut down main loop
237         //native_data->main_loop.quit();
238 }
239
240 void handleGazeData(TobiiResearchGazeData* gaze_data)
241 {
242         JNIEnv* env = NULL;
243         g_native_data_current->jvm->AttachCurrentThread((void**)
            ↪ &env, NULL);
244         jint rs = g_native_data_current->jvm->GetEnv((void**)
            ↪ &env, JNI_VERSION_1_6);
245         if (rs != JNI_OK || env == NULL) {
```

```
246                 return;
247         }
248         jobject obj = g_native_data_current->j_tobii_tracker;
249 
250         jclass tobii_tracker_class = env->GetObjectClass(obj);
251         if (tobii_tracker_class == NULL)
252                 return;
253         jmethodID jmid_new_gaze_point =
        ↪  env->GetMethodID(tobii_tracker_class,
254                 "newGazePoint", "(JDDDDIIDD)V");
255         //Just pretend nothing happened.
256         if (jmid_new_gaze_point == NULL)
257                 return;
258         //Call newGazePoint.
259         env->CallVoidMethod(obj, jmid_new_gaze_point,
        ↪  (jlong)gaze_data->device_time_stamp,
260                 gaze_data->left_eye.gaze_point.position_on_display_area.x,
                 ↪  gaze_data->right_eye.gaze_point.position_on_display_area.y,
261                 gaze_data->right_eye.gaze_point.position_on_display_area.x,
                 ↪  gaze_data->right_eye.gaze_point.position_on_display_area.y,
262                 gaze_data->left_eye.pupil_data.validity,
                 ↪  gaze_data->right_eye.pupil_data.validity,
263                 gaze_data->left_eye.pupil_data.diameter,
                 ↪  gaze_data->right_eye.pupil_data.diameter);
264 
265         /**env->CallVoidMethod(obj, jmid_new_gaze_point, (jlong)
        ↪  gaze_data->time_stamp,
266                 gaze_data->leftGazePoint2d.x,
    ↪  gaze_data->rightGazePoint2d.y,
267                 gaze_data->rightGazePoint2d.x,
    ↪  gaze_data->rightGazePoint2d.y,
268                 gaze_data->leftValidity,
    ↪  gaze_data->rightValidity,
269                 gaze_data->leftPupilDiameter,
    ↪  gaze_data->rightPupilDiameter);**/
270 }
271 
272 void gaze_data_callback(TobiiResearchGazeData* gaze_data, void*
    ↪  user_data) {
273         handleGazeData(gaze_data);
274         //memcpy(user_data, gaze_data, sizeof(*gaze_data));
275 }
276 
277 
```

```
278 JNIEXPORT void JNICALL
    ↪ Java_edu_ysu_itrace_trackers_TobiiTracker_startTracking
279         (JNIEnv* env, jobject obj)
280 {
281
282         //Do not continue if already tracking
283         if (g_native_data_current != NULL)
284         {
285                 throwJException(env, "java/io/IOException",
                    ↪ "Already tracking.");
286                 return;
287         }
288
289
290         //Get native data from object.
291         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪ obj);
292         if (native_data == NULL)
293         {
294                 throwJException(env,
                    ↪ "java/lang/RuntimeException",
295                         "Cannot find native data.");
296                 return;
297         }
298         //Set native data for current tracking TobiiTracker.
299         g_native_data_current = native_data;
300         //Setting gaze data
301         TobiiResearchGazeData gaze_data;
302         try
303         {
304                 tobii_research_subscribe_to_gaze_data(native_data->eye_tracker,
                    ↪ gaze_data_callback, &gaze_data);
305                 //handleGazeData(&gaze_data);
306                 //native_data->eye_tracker->startTracking();
307                 //native_data->eye_tracker->addGazeDataReceivedListener(handleGaz
308         }
309         catch (const std::invalid_argument& e)
310         {
311                 throwJException(env, "java/io/IOException",
                    ↪ e.what());
312                 return;
313         }
314 }
315
```

```
316
317 JNIEXPORT void JNICALL
    ↪ Java_edu_ysu_itrace_trackers_TobiiTracker_stopTracking
318         (JNIEnv* env, jobject obj)
319 {
320
321         try
322         {
323                 tobii_research_unsubscribe_from_gaze_data(g_native_data_current->eye_
                    ↪ gaze_data_callback);
324                 g_native_data_current = NULL;
325         }
326         catch (const std::invalid_argument& e)
327         {
328                 throwJException(env, "java/io/IOException",
                    ↪ e.what());
329                 return;
330         }
331 }
332
333 JNIEXPORT void
334         JNICALL
            ↪ Java_edu_ysu_itrace_trackers_TobiiTracker_00024Calibrator_jniAddPoint
335         (JNIEnv* env, jobject obj, jdouble x, jdouble y)
336 {
337
338         //Get native data from parent TobiiTracker
339         jfieldID jfid_parent = getFieldID(env, obj, "parent",
340                 "Ledu/ysu/itrace/trackers/TobiiTracker;");
341         if (jfid_parent == NULL)
342         {
343                 throwJException(env,
                        ↪ "java/lang/RuntimeException",
344                         "Parent TobiiTracker not found.");
345                 return;
346         }
347         jobject parent_tobii_tracker = env->GetObjectField(obj,
            ↪ jfid_parent);
348         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪ parent_tobii_tracker);
349
350         try
351         {
352                 TobiiResearchNormalizedPoint2D point[1] = {x,y};
```

```
353                 //Add new calibration point
354                 tobii_research_screen_based_calibration_collect_data(native_data-
                    ↪  point->x, point->y);
355
356 /**                 if
    ↪  (tobii_research_screen_based_calibration_collect_data(native_data->eye_tracke
    ↪  point->x, point->y) != TOBII_RESEARCH_STATUS_OK) {
357                         /* Try again if it didn't go well the
    ↪  first time. */
358                         /* Not all eye tracker models will fail
                            ↪  at this point, but instead fail on
                            ↪  ComputeAndApply. */
359                         //tobii_research_screen_based_calibration_collect_data(na
                            ↪  point->x, point->y);
360                 //}
361
362
363                 //native_data->eye_tracker->addCalibrationPoint(Point2d(
364                 //        (double) x, (double) y));
365         }
366         catch (const std::invalid_argument& e)
367         {
368                 throwJException(env, "java/io/IOException",
                    ↪  e.what());
369                 return;
370         }
371 }
372
373 JNIEXPORT void JNICALL
374         Java_edu_ysu_itrace_trackers_TobiiTracker_00024Calibrator_jniStartCalibra
375         (JNIEnv* env, jobject obj)
376 {
377
378         //Get native data from parent TobiiTracker
379         jfieldID jfid_parent = getFieldID(env, obj, "parent",
380                 "Ledu/ysu/itrace/trackers/TobiiTracker;");
381         if (jfid_parent == NULL)
382         {
383                 throwJException(env,
                    ↪  "java/lang/RuntimeException",
384                         "Parent TobiiTracker not found.");
385                 return;
386         }
```

```
387         jobject parent_tobii_tracker = env->GetObjectField(obj,
            ↪ jfid_parent);
388         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪ parent_tobii_tracker);
389         try
390         {
391                 //Start and clear
392                 tobii_research_screen_based_calibration_enter_calibration_mode(native
393                 //native_data->eye_tracker->startCalibration();
394                 //native_data->eye_tracker->clearCalibration();
395         }
396         catch (const std::invalid_argument& e)
397         {
398                 throwJException(env, "java/io/IOException",
                    ↪ e.what());
399                 return;
400         }
401 }
402
403 JNIEXPORT void JNICALL
404         Java_edu_ysu_itrace_trackers_TobiiTracker_00024Calibrator_jniStopCalibration
405         (JNIEnv* env, jobject obj)
406 {
407
408         //Get native data from parent TobiiTracker
409         jfieldID jfid_parent = getFieldID(env, obj, "parent",
410                 "Ledu/ysu/itrace/trackers/TobiiTracker;");
411         if (jfid_parent == NULL)
412         {
413                 throwJException(env,
                    ↪ "java/lang/RuntimeException",
414                         "Parent TobiiTracker not found.");
415                 return;
416         }
417         jobject parent_tobii_tracker = env->GetObjectField(obj,
            ↪ jfid_parent);
418         TobiiNativeData* native_data = getTobiiNativeData(env,
            ↪ parent_tobii_tracker);
419
420         try
421         {
422                 //Compute and stop calibration
423
424
```

```
425                 TobiiResearchStatus status =
                    ↪ tobii_research_screen_based_calibration_compute_and_apply(nat
                    ↪ &native_data->calibration_result);
426                 tobii_research_screen_based_calibration_leave_calibration_mode(na
427                 //old
428                 //native_data->eye_tracker->computeCalibration();
429                 //native_data->eye_tracker->stopCalibration();
430         }
431         catch (const std::invalid_argument& e)
432         {
433                 throwJException(env, "java/io/IOException",
                    ↪ e.what());
434                 return;
435         }
436 }
437
438 JNIEXPORT jdoubleArray JNICALL
439         Java_edu_ysu_itrace_trackers_TobiiTracker_00024Calibrator_jniGetCalibrati
440           (JNIEnv *env, jobject obj)
441 {
442         //Get native data from parent TobiiTracker
443         jfieldID jfid_parent = getFieldID(env, obj, "parent",
444                 "Ledu/ysu/itrace/trackers/TobiiTracker;");
445         if (jfid_parent == NULL)
446         {
447                 throwJException(env,
                    ↪ "java/lang/RuntimeException",
448                         "Parent TobiiTracker not found.");
449                 return NULL;
450         }
451         jobject parent_tobii_tracker = env->GetObjectField(obj,
                ↪ jfid_parent);
452         TobiiNativeData* native_data = getTobiiNativeData(env,
                ↪ parent_tobii_tracker);
453
454         try
455         {
456                 //Get calibration
457                 //Calibration::pointer_t calibrationData =
458                 //                native_data->eye_tracker->getCalibration();
459                 //                Calibration::plot_data_vector_t
                    ↪ calibrationPlotData =
                    ↪ calibrationData->getPlotData();
```

```
460                             //TobiiResearchCalibrationResult*
                                ↪ calibration_result = NULL;
461                             //status =
                                ↪ tobii_research_screen_based_calibration_compute_
                                ↪ &calibration_result);
462
463                             //int itemCount =
                                ↪ static_cast<int>(calibration_result->calibration_
464
465                 int itemCount =
                    ↪ static_cast<int>(native_data->calibration_result->calibration_poi
466                 jdoubleArray calibrationPoints =
                    ↪ env->NewDoubleArray(4 * itemCount);  //
                    ↪ allocate
467
468                 if (NULL == calibrationPoints) return NULL;
469
470                 jdouble *points =
                    ↪ env->GetDoubleArrayElements(calibrationPoints,
                    ↪ 0);
471
472                 TobiiResearchCalibrationPoint item;
473                 //tobii_research_apply_calibration_data(native_data->eye_tracker,
                    ↪ calibration_data);
474
475                 //Writing data. for test
476
477                 for(int i = 0; i < itemCount; i++)
478                 {
479
480                         item =
                            ↪ native_data->calibration_result->calibration_points[i];
481                         points[i] =
                            ↪ item.calibration_samples->left_eye.position_on_display_ar
482                         points[itemCount + i] =
                            ↪ item.calibration_samples->left_eye.position_on_display_ar
483                         points[2 * itemCount + i] =
                            ↪ item.calibration_samples->right_eye.position_on_display_a
484                         points[3*itemCount+i] =
                            ↪ item.calibration_samples->right_eye.position_on_display_a
485
486                         //points[i] = item.leftMapPosition.x;
487                         //points[itemCount+i] =
                            ↪ item.leftMapPosition.y;
```

```
488                         //points[2*itemCount+i] =
                            ↪  item.rightMapPosition.x;
489                         //points[3*itemCount+i] =
                            ↪  item.rightMapPosition.y;
490                 }
491
492                 env->ReleaseDoubleArrayElements(calibrationPoints,
                    ↪  points, 0);
493                 return calibrationPoints;
494         }
495         catch (const std::invalid_argument& e)
496         {
497                 throwJException(env, "java/io/IOException",
                    ↪  e.what());
498                 return NULL;
499         }
500 }
```

Appendix D

iTraceAnalyser.py

```
import MySQLdb
import pandas
import matplotlib
import matplotlib.pyplot as plt
import math
import csv
from matplotlib import rcParams
rcParams.update({'figure.autolayout': True})

#Returns a dictionary of Fixations

def get_session(host,user,passwd,db, session_id, threshold):
        db = MySQLdb.connect(host=host,
                        user=user,
                        passwd=passwd,
                        db=db)

        cursor = db.cursor()
        session_info = cursor.execute("SELECT * FROM session_info
        ↪  WHERE session_id='"+session_id+"'")
```

```
20          df = pandas.read_sql("SELECT * FROM (SELECT * FROM gazes
            ↪  JOIN sces ON gazes.gaze_id=sces.sce_gaze_id) a WHERE
            ↪  a.gaze_session_id='" + session_id + "';", db)
21          #df has all sces. dfu only has sce of depth 0.
22          dfu = df.drop_duplicates(subset="gaze_id")
23
24          # I-VT. let's find some fixations!
25          fixations = []
26          saccades = []
27
28          fix_list = []
29          fixation = []
30          prev_index = []
31
32
33          for index, row in dfu.iterrows():
34              curr_index = {'session_time' : row['session_time'],
35                            'x':row['x'], 'y':row['y'],
                              ↪  'sce_name':row['sce_name'],
36                            'depth':row['depth'],
                              ↪  'gaze_name':row['gaze_name'],
                              ↪  'sce_type':row['sce_type'],
37                            'gaze_session_id':row['gaze_session_id'],
38                            'line':row['line']}
39              if not fixation:
40                  fixation.append(curr_index)
41              #If time is less than 100
42              if (fixation[len(fixation)-1]['session_time']/1e6 -
                ↪  fixation[0]['session_time']/1e6) < 100:
43                  fixation.append(curr_index)
44                  #Ensure that while time is less than 100,
                    ↪  threshold is met.
45                  while get_dispersion(fixation) > threshold:
46                      fixation.pop(0)
47                  continue
48              #If time is over 100, but threshold is less than
                ↪  threshold
49
50
51
52              if get_dispersion(fixation) <= threshold:
53                  fixation.append(curr_index)
54
55                  if get_dispersion(fixation) > threshold:
```

```
56                      fixation.pop(len(fixation)-1)
57                      #print "fixation dispersion " +
                        ↪  str(get_dispersion(fixation))
58                      fix_list.append(fixation)
59                      fixation = [curr_index]
60
61          for fix in fix_list:
62              #Getting the x and y values, for centroid
63              x_list = []
64              y_list = []
65              sce_list = []
66              sce_type_list = []
67              line_list = []
68              id_list = []
69              for i in fix:
70                  x_list.append(i['x'])
71                  y_list.append(i['y'])
72                  sce_list.append(i['sce_name'])
73                  sce_type_list.append(i['sce_type'])
74                  line_list.append(i['line'])
75                  id_list.append(i['gaze_session_id'])
76
77
78              centroid_x = sum(x_list)/len(x_list)
79              centroid_y = sum(y_list)/len(y_list)
80
81              #startTime, endTime and duration in seconds.
82              startTime = fix[0]['session_time']/1e9
83              endTime = fix[len(fix)-1]['session_time']/1e9
84              duration = endTime - startTime
85
86              depth = 0 #Not supported, but some day, maybe?
87
88              fileName = fix[0]['gaze_name']
89              #Sce name, if two source code entities are very
                ↪  close, we pick the one with the most gazes.
90              sce_name =  max(set(sce_list), key=sce_list.count)
91              sce_type =  max(set(sce_type_list),
                ↪  key=sce_type_list.count)
92              line =  max(set(line_list), key=line_list.count)
93              gaze_session_id =  max(set(id_list),
                ↪  key=id_list.count)
94
95
```

```
96              #Returns a dictionary of Fixations (startTime,
                ↪  endTime, duration, X, Y, sceName, depth, file)
97              fixations.append({'start_time':startTime,
98                                'end_time':endTime,
                                  ↪  'duration':duration,
99                                'x':centroid_x, 'y':centroid_y,
100                               'sce_name':sce_name, 'depth':depth,
101                               'file_name':fileName,
102                               'sce_type':sce_type, 'line':line,
103                               'gaze_session_id':gaze_session_id})
104         return fixations#, saccades
105
106 def get_dispersion(fixation):
107     #Finding distance between the points
108     min_x = 999999
109     max_x = 0;
110     max_y = 0;
111     min_y = 999999;
112     min_line = 999999;
113     max_line = 0
114     for i in fixation:
115         if i['x'] > max_x:
116             max_x = i['x']
117         if i['x'] < min_x:
118             min_x = i['x']
119         if i['y'] > max_y:
120             max_y = i['y']
121         if i['y'] < min_y:
122             min_y = i['y']
123         if i['line'] < min_line:
124             min_line = i['line']
125         if i['line'] > max_line:
126             max_line = i['line']
127
128
129     dispersion = (max_x-min_x) + (max_y-min_y)
130     if (max_line - min_line) > 2:
131         dispersion += 1000
132     return dispersion
133
134
135
136 def get_raw_gazes(host,user,passwd,db, session_id):
137                 db = MySQLdb.connect(host=host,
```

```
138                     user=user,
139                     passwd=passwd,
140                     db=db)
141                 cursor = db.cursor()
142                 session_info = cursor.execute("SELECT * FROM
                    ↪  session_info WHERE
                    ↪  session_id='"+session_id+"'")
143                 df = pandas.read_sql("SELECT * FROM (SELECT *
                    ↪  FROM gazes JOIN sces ON
                    ↪  gazes.gaze_id=sces.sce_gaze_id) a WHERE
                    ↪  a.gaze_session_id='" + session_id + "';", db)
144                 #df has all sces. dfu only has sce of depth 0.
145                 dfu = df.drop_duplicates(subset="gaze_id")
146                 return dfu
147
148 def fixation_scatter_plot(fixations, fig, title):
149     x_list = []
150     y_list = []
151     s_list = []
152
153     for point in fixations:
154         x_list.append(point['x'])
155         y_list.append(point['y'])
156         s_list.append(point['duration']*1000)
157         plt.xlim(200, 700)
158         plt.ylim(0, 600)
159         plt.title(title)
160     plt.gca().invert_yaxis()
161     plt.scatter(x_list,y_list, s=s_list)
162         #Uncomment for numbers in order.
163     #i = 0
164     #for x, y in zip(x_list, y_list):
165     #    plt.text(x, y, str(i), color="red", fontsize=12)
166     #    i += 1
167     plt.savefig(fig + ".png", format='png')
168
169     plt.show()
170
171 def gaze_scatter_plot(gazes, fig,title):
172         x_list = []
173         y_list = []
174         for index, point in gazes.iterrows():
175                 x_list.append(point['x'])
176                 y_list.append(point['y'])
```

```
177         plt.xlim(200, 700)
178         plt.ylim(0, 600)
179         plt.title(title)
180         plt.gca().invert_yaxis()
181         plt.scatter(x_list,y_list)
182         plt.savefig(fig + ".png", format='png')
183 
184         plt.show()
185 
186 
187 def process_mining_data(fixations_list, csv_name, aoi,
    ↪  user_def='No'):
188     #Make buckets, and define name of AoIs
189     if(aoi == 'lines'):
190         area = 'line'
191 
192     if (aoi == 'sce_grouped'):
193         area = 'sce_type'
194     if (aoi == 'sce'):
195         area = 'sce_name'
196 
197     if (aoi == 'user_defined'):
198         aoi_defs = []
199         with open(user_def, 'rb') as csvfile:
200             csv_reader = csv.reader(csvfile)
201             for line in csv_reader:
202                 start = int(line[1])
203                 end = int(line[2])
204                 aoi_defs.append([line[0],[i for i in
                    ↪  range(start,end+1)]])
205 
206 
207     with open(csv_name, 'w') as csvfile:
208         #initialising csv file
209         csv_writer = csv.writer(csvfile, delimiter=',')
210         csv_writer.writerow(["subject", "area of interest",
            ↪  "start", "end"])
211 
212         #creating a name dictionary
213         if aoi == 'lines':
214             name_dict = {}
215             for fixations in fixations_list:
216                 for fixation in fixations:
217                     if fixation['line'] in name_dict:
```

```
218                         name_dict[fixation['line']] =
                            ↪ name_dict[fixation['line']] +
                            ↪ [fixation['sce_type']]
219                     else:
220                         name_dict[fixation['line']] =
                            ↪ [fixation['sce_type']]
221         for fixations in fixations_list:
222             #go through fixations and see if it matches a bucket.
                ↪ "empty" buckets whenever you find something from
                ↪ a new bucket.
223             subject = ''
224             name_of_aoi = ''
225             start = 0
226             end = 0
227             aoi_list = []
228             area_of_interest = []
229             prev_fix = {}
230             prev_group = ""
231             fix_group = ""
232             inAoI = False
233             for fix in fixations:
234                 if(aoi == 'user_defined'):
235                     for aoi_index in aoi_defs:
236                         if (fix['line'] in aoi_index[1]):
237                             inAoI = True
238                     if (inAoI == False):
239                         if not(area_of_interest == []):
240                             aoi_list.append(area_of_interest)
241                             area_of_interest = []
242                             prev_fix = {}
243                             prev_group = ""
244                         continue
245
246
247                 if prev_fix == {}:
248                     prev_fix = fix
249                     if(aoi == 'user_defined'):
250                         for aoi_index in aoi_defs:
251                             if (prev_fix['line'] in
                                ↪ aoi_index[1]):
252                                 prev_group = aoi_index[0]
253                     continue
254
255                 area_of_interest.append(prev_fix)
```

```
256
257                 if(aoi == 'user_defined'):
258                     #if we don't set previous group (doesn't
                        ↪  belong in any AoI), we dont append it.
259                     #finding group for each fix
260                     for aoi_index in aoi_defs:
261                         if (fix['line'] in aoi_index[1]):
262                             fix_group = aoi_index[0]
263                     if (prev_group == fix_group and not
                        ↪  (prev_fix['end_time']+1 <
                        ↪  fix['start_time'])):
264                         prev_fix = fix
265                         prev_group = fix_group
266
267                     else:
268                         aoi_list.append(area_of_interest)
269                         area_of_interest = []
270                         prev_fix = fix
271                         prev_group = fix_group
272
273                 #Not user_defined
274                 else:
275                     if (prev_fix[area] == fix[area] and not
                        ↪  (prev_fix['end_time']+1 <
                        ↪  fix['start_time'])):
276                                             area_of_interest.append(prev_fix)
277                                             prev_fix = fix
278
279                     else:
280                         aoi_list.append(area_of_interest)
281                         area_of_interest = []
282                         prev_fix = fix
283                 #special case for last
284                 if (fix == fixations[len(fixations)-1]):
285                     area_of_interest.append(fix)
286                     aoi_list.append(area_of_interest)
287
288             for l in aoi_list:
289                 name_of_aoi = ""
290                 subject = l[0]['gaze_session_id']
291                 start = l[0]['start_time']
292                 end = l[len(l)-1]['end_time']
293                 if (aoi == 'user_defined'):
294                     for aoi_index in aoi_defs:
```

```
295                         if (l[0]['line'] in aoi_index[1]):
296                             name_of_aoi = aoi_index[0]
297                     #That special case, where the last fixation,
                        ↪ which we had to add
298                     if name_of_aoi == "":
299                         continue
300                 else:
301                     #print
                        ↪ str(set(name_dict[l[0]['line']]))[5:-2].
302                     print
                        ↪ str(set(name_dict[l[0]['line']]))[5:-2].replace(",",
                        ↪ " ")
303                     if aoi == 'lines':
304                         name_of_aoi = str(l[0][area]) + " " +
                            ↪ str(set(name_dict[l[0]['line']]))[5:-2].replace(",",
                            ↪ " ")
305                     else:
306                         name_of_aoi = str(l[0][area])
307
308
                    ↪ csv_writer.writerow([subject,name_of_aoi,start,end])
309
310
311 def fixation_average_duration_bar(fixations, filename,
    ↪ attribute):
312     x = []
313     y = []
314     x_ticks = []
315     #fixations = sorted(fixations, key=lambda k: k[attribute])
316
317     for i in fixations:
318         x_ticks.append(i[attribute])
319
320     x_ticks = sorted(list(set(x_ticks)))
321     print x_ticks
322     durr = 0
323     for ftype in x_ticks:
324         avg_counter = 0
325         for i in fixations:
326             if i[attribute] == ftype:
327                 avg_counter += 1
328                 durr += i['duration']
329         y.append(durr/avg_counter)
330         durr = 0
```

```
331     x = [i for i in range(len(x_ticks))]
332     plt.xticks(x,x_ticks,rotation=90)
333     if(len(x) > 15):
334         plt.tick_params(axis='both', which='major', labelsize=5)
335     plt.title(attribute + ' average fixation duration, seconds')
336     #plt.tight_layout()
337     plt.xlabel(str(attribute))
338     plt.ylabel("seconds")
339 
340     plt.bar(x,y)
341     plt.savefig(filename + ".png", format='png')
342 
343     plt.show()
344 
345 
346 
347 def fixation_duration_bar(fixations, filename, attribute):
348     x = []
349     y = []
350     x_ticks = []
351     for i in fixations:
352         x_ticks.append(i[attribute])
353 
354     x_ticks = sorted(list(set(x_ticks)))
355     durr = 0
356     for ftype in x_ticks:
357         for i in fixations:
358             if i[attribute] == ftype:
359                 durr += i['duration']
360         y.append(durr)
361         durr = 0
362     x = [i for i in range(len(x_ticks))]
363     plt.xticks(x,x_ticks,rotation=90)
364     if(len(x) > 15):
365         plt.tick_params(axis='both', which='major', labelsize=6)
366     plt.title( attribute + ' total fixation duration, seconds')
367     #plt.tight_layout()
368     plt.xlabel(str(attribute))
369     plt.ylabel("seconds")
370 
371 
372     plt.bar(x,y)
373     plt.savefig(filename + ".png", format='png')
374 
```

```
375     plt.show()
376 
377 def fixation_max_duration_bar(fixations, filename, attribute):
378     x = []
379     y = []
380     x_ticks = []
381     for i in fixations:
382         x_ticks.append(i[attribute])
383 
384     x_ticks = sorted(list(set(x_ticks)))
385     durr = 0
386     for ftype in x_ticks:
387         for i in fixations:
388             if i[attribute] == ftype:
389                 if i['duration'] > durr:
390                     durr = i['duration']
391         y.append(durr)
392         durr = 0
393     x = [i for i in range(len(x_ticks))]
394     plt.xticks(x,x_ticks,rotation=90)
395     if(len(x) > 15):
396         plt.tick_params(axis='both', which='major', labelsize=6)
397     plt.title( attribute + ' max fixation duration, seconds')
398     #plt.tight_layout()
399     plt.xlabel(str(attribute))
400     plt.ylabel("seconds")
401 
402 
403     plt.bar(x,y)
404     plt.savefig(filename + ".png", format='png')
405 
406     plt.show()
407 
408 
409 
410 def fixation_count_bar(fixations, filename, attribute):
411     x = []
412     y = []
413     x_ticks = []
414     for i in fixations:
415         x_ticks.append(i[attribute])
416 
417     x_ticks = sorted(list(set(x_ticks)))
418     count = 0
```

```
419     for ftype in x_ticks:
420         for i in fixations:
421             if i[attribute] == ftype:
422                 count += 1
423         y.append(count)
424         count = 0
425     x = [i for i in range(len(x_ticks))]
426     plt.xticks(x,x_ticks,rotation=90)
427     if(len(x) > 15):
428         plt.tick_params(axis='both', which='major', labelsize=6)
429     plt.title(attribute + ' fixation count')
430     #plt.tight_layout()
431     plt.xlabel(str(attribute))
432     plt.ylabel("# of fixations")
433
434
435
436     plt.bar(x,y)
437     plt.savefig(filename + ".png", format='png')
438     plt.show()
439
440 def session_duration(fixations):
441     start = int(fixations[0]['start_time'])
442     end = int(fixations[len(fixations)-1]['end_time'])
```

Appendix E

iTrace.sql

```
CREATE DATABASE iTrace;

CREATE TABLE session_info
(
    session_ID varchar(255),
    session_purpose varchar(255),
    session_descrip varchar(255),
    developer_username varchar(255),
    developer_name varchar(255),
    screen_width int,
    screen_height int,
    PRIMARY KEY (session_ID)
);

CREATE TABLE gazes
(
    gaze_id int AUTO_INCREMENT,
    gaze_session_id varchar(255),
    name varchar(255),
    type varchar(255),
    x int,
    y int,
```

```
23      left_validation float,
24      right_validation float,
25      left_pupil_diameter float,
26      right_pupil_diameter float,
27      timestamp varchar(255),
28      session_time bigint,
29      tracker_time bigint,
30      system_time bigint,
31      nano_time bigint,
32      path varchar(255),
33      line_height int,
34      font_height int,
35      line int,
36      col int,
37      line_base_x int,
38      line_base_y int,
39      PRIMARY KEY (gaze_id),
40      FOREIGN KEY (gaze_session_id) REFERENCES
        ↪  session_info(session_ID)
41
42  );
43
44  CREATE TABLE sces
45  (
46      sce_gaze_id int,
47      name varchar(255),
48      type varchar(255),
49      how varchar(255),
50      total_length int,
51      start_line int,
52      end_line int,
53      start_col int,
54      end_col int,
55      depth int,
56      FOREIGN KEY (sce_gaze_id) REFERENCES gazes(gaze_id)
57
58  );
```

Appendix F

SQLGazeExportSolver.java

```
package edu.ysu.itrace.solvers;

import java.awt.Dimension;
import java.awt.Toolkit;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import javax.activation.DataSource;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.swing.UIManager;

import org.eclipse.e4.core.services.events.IEventBroker;
import org.eclipse.ui.PlatformUI;
import org.osgi.service.event.Event;
import org.osgi.service.event.EventHandler;
```

```
import edu.ysu.itrace.Gaze;
import edu.ysu.itrace.AstManager.SourceCodeEntity;
import edu.ysu.itrace.gaze.IGazeResponse;
import edu.ysu.itrace.gaze.IStyledTextGazeResponse;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
public class SQLGazeExportSolver implements IFileExportSolver,
↪  EventHandler {

  private Dimension screenRect;
  private String sessionID;
  private String sessionPurpose;
  private String sessionDescrip;
  private String devName;
  private String devUsername;
  private IEventBroker eventBroker;
  private String outFile;
  Connection conn;

  public SQLGazeExportSolver() {
    UIManager.put("swing.boldMetal", new Boolean(false)); //make
    ↪  UI font plain
    eventBroker =
    ↪  PlatformUI.getWorkbench().getService(IEventBroker.class);
  }


  /**
   * Any initialization work with side effects, such as opening
↪  files. This
   * method should very probably be called before calling process
↪  or dispose.
   */
  public void init() {
    screenRect = Toolkit.getDefaultToolkit().getScreenSize();
    outFile = "Set";
    System.out.println("init is called HAHASDHASHDHASDHASDHASD");

  }

  public void setSessionInfo() {
```

```
  try {
    System.out.println("Connecting to DB");
    Class.forName("com.mysql.jdbc.Driver");
    this.conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/iTrace","root","1234");
    // Insert session info into db.
    int rs;
    System.out.println("WE ARE CONNECTED");
    String statement = "INSERT INTO session_info"
       +
        ↪  "(session_ID,session_purpose,session_descrip,developer_username,
        ↪  "
       + "developer_name) VALUES ( "
       + "'" + this.sessionID + "',"
       + "'" + this.sessionPurpose + "',"
       + "'" + this.sessionDescrip + "',"
       + "'" + this.devUsername + "',"
       + "'" + this.devName + "')";
    //   + "'" + String.format("%d",this.screenRect.getWidth())
     ↪  + ","
    //   + "'" +
     ↪  String.format("%d",this.screenRect.getHeight()) + ");";
    rs = conn.createStatement().executeUpdate(statement);
    System.out.println(statement);
  }

  catch(SQLException e) {
    e.printStackTrace();
  } catch (ClassNotFoundException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

}

/**
 * Called to process new gazes.
 */
public void process(IGazeResponse response) {
  int screenX =
          (int) (screenRect.width * response.getGaze().getX());
  int screenY =
          (int) (screenRect.height *
           ↪  response.getGaze().getY());
```

```
Gaze gaze = response.getGaze();
String statementInsert = "INSERT INTO gazes (name, type,
↪  gaze_session_id, x, y,"
    + "left_validation, right_validation,
    ↪  left_pupil_diameter,"
    + "right_pupil_diameter, timestamp, session_time,
    ↪  tracker_time,"
    + "system_time, nano_time";

String statementValues = " VALUES ('"
    + response.getName() + "','"
    + response.getGazeType() + "','"
    + this.sessionID + "','"
    + screenX + "','"
    + screenY + "','"
    + gaze.getLeftValidity() + "','"
    + gaze.getRightValidity() + "','"
    + gaze.getLeftPupilDiameter() + "','"
    + gaze.getRightPupilDiameter() + "','"
    + gaze.getTimestamp() + "','"
    + gaze.getSessionTime() + "','"
    + gaze.getTrackerTime() + "','"
    + gaze.getSystemTime() + "','"
    + gaze.getNanoTime() + "'";

String nostyle = statementInsert + ")" + statementValues +
↪  ");";

// gazes
if (response instanceof IStyledTextGazeResponse) {
  IStyledTextGazeResponse styledResponse =
      (IStyledTextGazeResponse) response;
  statementInsert += ", path, line_height, font_height,
  ↪  line,"
      + "col, line_base_x, line_base_y";

  statementValues += ",'" + styledResponse.getPath() + "','"
      + styledResponse.getLineHeight() + "','"
      + styledResponse.getFontHeight() + "','"
      + styledResponse.getLine() + "','"
      + styledResponse.getCol() + "','"
      + styledResponse.getLineBaseX() + "','"
      + styledResponse.getLineBaseY() + "'";
```

```
  String style = statementInsert + ")" + statementValues +
   ↪  ");";

  try {
    System.out.println("style");
    System.out.println(style);
    this.conn.createStatement().executeUpdate(style);
  } catch (SQLException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
  }

// sces
int depth = 0;
String sce_gaze_id = "";
try {
 ResultSet rs =
  ↪  this.conn.createStatement().executeQuery("SELECT
  ↪  max(gaze_id) FROM gazes WHERE gaze_session_id='" +
  ↪  sessionID +"';");
 rs.next();
 sce_gaze_id = rs.getString("max(gaze_id)");
 System.out.println(sce_gaze_id);
} catch (SQLException e1) {
  // TODO Auto-generated catch block
  e1.printStackTrace();
}
for (SourceCodeEntity sce : styledResponse.getSCEs()) {

  String statementSCE = "INSERT INTO sces (sce_gaze_id, "
      + "name, type, how, total_length,"
      + "start_line, end_line, start_col, end_col, depth)"
      + "VALUES ('" + sce_gaze_id + "','"
      + sce.getName() +"','"
      + sce.type.toString() + "','"
      + sce.how.toString() + "','"
      + sce.totalLength + "','"
      + sce.startLine + "','"
      + sce.endLine + "','"
      + sce.startCol + "','"
      + sce.endCol + "','"
      + depth + "');";
    depth++;
```

```
      try {
       this.conn.createStatement().executeUpdate(statementSCE);
     } catch (SQLException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
     }
     }
   }
   else {
     try {
       System.out.println("nostyle");
       System.out.println(nostyle);
       this.conn.createStatement().executeUpdate(nostyle);
     } catch (SQLException e) {
       // TODO Auto-generated catch block
       e.printStackTrace();
     }
   }


}


public String friendlyName() {
  return "SQL Gaze export";
}

/**
 * Configure the export filename.
 */

public void config(String sessionID, String devUsername) {
  // TODO Auto-generated method stub

}

public void configSQL(String sessionID, String devUsername,
    String devName, String sessionDescrip, String
     ↪  sessionPurpose){
  this.sessionID = sessionID;
  this.devUsername = devUsername;
  this.devName = devName;
  this.devUsername = devUsername;
```

```
    this.sessionDescrip = sessionDescrip;
    this.sessionPurpose = sessionPurpose;
  }

  /**
   * Launch dialog to display the export filename.
   */
  public void displayExportFile() {

  }



  /**
   * Frees any resources. It is very likely a bad idea to process
↪   new data
   * after calling dispose. Not sure if we need this, either.
   */
  public void dispose() {
    this.conn = null;
  }



  @Override
  public void handleEvent(Event event) {
    // TODO Auto-generated method stub
    if(outFile == null) this.init();
    String[] propertyNames = event.getPropertyNames();
    IGazeResponse response =
    ↪  (IGazeResponse)event.getProperty(propertyNames[0]);
    this.process(response);

  }


  @Override
  public String getFilename() {
    // TODO Auto-generated method stub
    return null;
  }
}
```

Appendix G

MagicSort.java

```
import java.util.Comparator;

/**
 *  The {@code Insertion} class provides static methods for
↪  sorting an
 *  array using insertion sort.
 *  <p>
 *  This implementation makes ~ 1/2 n^2 compares and exchanges in
 *  the worst case, so it is not suitable for sorting large
↪  arbitrary arrays.
 *  More precisely, the number of exchanges is exactly equal to
↪  the number
 *  of inversions. So, for example, it sorts a partially-sorted
↪  array
 *  in linear time.
 *  <p>
 *  The sorting algorithm is stable and uses O(1) extra memory.
 *  <p>
 *  See <a
↪  href="https://algs4.cs.princeton.edu/21elementary/InsertionPedantic.java.html
 *  for a version that eliminates the compiler warning.
 *  <p>
```

```
 *  For additional documentation, see <a
↪  href="https://algs4.cs.princeton.edu/21elementary">Section
↪  2.1</a> of
 *  <i>Algorithms, 4th Edition</i> by Robert Sedgewick and Kevin
↪  Wayne.
 *
 *  @author Robert Sedgewick
 *  @author Kevin Wayne
 */
public class MagicSort {

    // This class should not be instantiated.
    private MagicSort() { }

    /**
     * Rearranges the array in ascending order, using the natural
↪   order.
     * @param a the array to be sorted
     */
    public static void sort(Comparable[] a) {
        int n = a.length;
        for (int i = 1; i < n; i++) {
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--) {
                exch(a, j, j-1);
            }
            assert isSorted(a, 0, i);

        }
        assert isSorted(a);
    }

    /**
     * Rearranges the subarray a[lo..hi) in ascending order,
↪   using the natural order.
     * @param a the array to be sorted
     * @param lo left endpoint (inclusive)
     * @param hi right endpoint (exclusive)
     */
    public static void sort(Comparable[] a, int lo, int hi) {
        for (int i = lo; i < hi; i++) {
            for (int j = i; j > lo && less(a[j], a[j-1]); j--) {
                exch(a, j, j-1);
            }
        }
```

```
        assert isSorted(a, lo, hi);
    }

    /**
     * Rearranges the array in ascending order, using a
↪   comparator.
     * @param a the array
     * @param comparator the comparator specifying the order
     */
    public static void sort(Object[] a, Comparator comparator) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            for (int j = i; j > 0 && less(a[j], a[j-1],
            ↪  comparator); j--) {
                exch(a, j, j-1);
            }
            System.out.println(a);
            assert isSorted(a, 0, i, comparator);
        }
        assert isSorted(a, comparator);
    }

    /**
     * Rearranges the subarray a[lo..hi) in ascending order,
↪   using a comparator.
     * @param a the array
     * @param lo left endpoint (inclusive)
     * @param hi right endpoint (exclusive)
     * @param comparator the comparator specifying the order
     */
    public static void sort(Object[] a, int lo, int hi,
    ↪  Comparator comparator) {
        for (int i = lo; i < hi; i++) {
            for (int j = i; j > lo && less(a[j], a[j-1],
            ↪  comparator); j--) {
                exch(a, j, j-1);
            }
            System.out.println(a);
        }
        assert isSorted(a, lo, hi, comparator);
    }
```

```
    // return a permutation that gives the elements in a[] in
    ↪  ascending order
    // do not change the original array a[]
    /**
     * Returns a permutation that gives the elements in the array
↪   in ascending order.
     * @param a the array
     * @return a permutation {@code p[]} such that {@code
↪   a[p[0]]}, {@code a[p[1]]},
     *    ..., {@code a[p[n-1]]} are in ascending order
     */
    public static int[] indexSort(Comparable[] a) {
        int n = a.length;
        int[] index = new int[n];
        for (int i = 0; i < n; i++)
            index[i] = i;

        for (int i = 0; i < n; i++)
            for (int j = i; j > 0 && less(a[index[j]],
            ↪  a[index[j-1]]); j--)
                exch(index, j, j-1);

        return index;
    }


   ↪  /***************************************************************************
    *  Helper sorting functions.

↪   ***************************************************************************/

    // is v < w ?
    private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
    }

    // is v < w ?
    private static boolean less(Object v, Object w, Comparator
    ↪  comparator) {
        return comparator.compare(v, w) < 0;
    }

    // exchange a[i] and a[j]
    private static void exch(Object[] a, int i, int j) {
```

```
        Object swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }

    // exchange a[i] and a[j]  (for indirect sort)
    private static void exch(int[] a, int i, int j) {
        int swap = a[i];
        a[i] = a[j];
        a[j] = swap;
    }


    /***************************************************************************
    *  Check if array is sorted - useful for debugging.
    ***************************************************************************/
    private static boolean isSorted(Comparable[] a) {
        return isSorted(a, 0, a.length);
    }

    // is the array a[lo..hi) sorted
    private static boolean isSorted(Comparable[] a, int lo, int
     ↪  hi) {
        for (int i = lo+1; i < hi; i++)
            if (less(a[i], a[i-1])) return false;
        return true;
    }

    private static boolean isSorted(Object[] a, Comparator
     ↪  comparator) {
        return isSorted(a, 0, a.length, comparator);
    }

    // is the array a[lo..hi) sorted
    private static boolean isSorted(Object[] a, int lo, int hi,
     ↪  Comparator comparator) {
        for (int i = lo+1; i < hi; i++)
            if (less(a[i], a[i-1], comparator)) return false;
        return true;
    }

   // print array to standard output
    private static void show(Comparable[] a) {
```

```
        for (int i = 0; i < a.length; i++) {
                System.out.println(a[i]);
            //StdOut.println(a[i]);
        }
    }

    /**
     * Reads in a sequence of strings from standard input;
↪   insertion sorts them;
     * and prints them to standard output in ascending order.
     *
     * @param args the command-line arguments
     */
    public static void main(String[] args) {
        String[] a = {"4","2","1","3"}; //StdIn.readAllStrings();
        MagicSort.sort(a);
        //show(a);
    }
}
```

Appendix H

iTraceAnalyserScript.py

```
import iTraceAnalyser as IA

#Creating session fixation data for a specific user at different
↪  sensitivities. For Chapter 4 section 2.

rawGazes = IA.get_raw_gazes("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100')
fixations5 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100', 5)
fixations10 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100',10)
fixations15 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100',15)
fixations20 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100',20)
fixations25 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100',25)
fixations30 = IA.get_session("localhost","root","1234"
,"iTrace",'20171219T095731-0417+0100',30)

#Creating scatter plots for the fixation sesnsitivity analysis
↪  Chapter 4 section 2.
```

```
IA.gaze_scatter_plot(rawGazes, "TestGaze", "Gazes")
IA.fixation_scatter_plot(fixations5,"TestFix5", "Fixations 5")
IA.fixation_scatter_plot(fixations10,"TestFix10", "Fixations 10")
IA.fixation_scatter_plot(fixations15,"TestFix15", "Fixations 15")
IA.fixation_scatter_plot(fixations20,"TestFix20", "Fixations 20")
IA.fixation_scatter_plot(fixations25,"TestFix25", "Fixations 25")
IA.fixation_scatter_plot(fixations30,"TestFix30", "Fixations 30")

#Importing the test subjects data
iTrace1 = IA.get_session("localhost","root","1234","iTrace"
,'20171201T092946-0684+0100',15)
iTrace2 = IA.get_session("localhost","root","1234","iTrace"
,'20171201T101056-0578+0100',15)
iTrace3 = IA.get_session("localhost","root","1234","iTrace"
,'20171205T100741-0299+0100',15)
iTrace4 = IA.get_session("localhost","root","1234","iTrace"
,'20171205T102737-0035+0100',15)
iTrace5 = IA.get_session("localhost","root","1234","iTrace"
,'20171205T105857-0765+0100',15)

#Creating bar plots for Chapter 4 section 3.
IA.fixation_count_bar(iTrace1,"count_lines","line")
IA.fixation_count_bar(iTrace1,"count_types","sce_type")

IA.fixation_duration_bar(iTrace1,"dur_lines","line")
IA.fixation_duration_bar(iTrace1,"dur_types","sce_type")

IA.fixation_average_duration_bar(iTrace1,"avg_dur_lines","line")
IA.fixation_average_duration_bar(iTrace1,"avg_dur_types","sce_type")

IA.fixation_max_duration_bar(iTrace1,"dur_max_lines","line")
IA.fixation_max_duration_bar(iTrace1,"dur_max_types","sce_type")

#creating event logs for chapter 4 section 4
IA.process_mining_data([iTrace1,iTrace2,iTrace3,iTrace4,iTrace5]
,"test_sort_lines.csv",'lines')
IA.process_mining_data([iTrace1,iTrace2,iTrace3,iTrace4,iTrace5]
,"test_sort_userdef.csv",'user_defined',"insertion.csv")
```

Bibliography

[AS14] Romano Bergstrom Andrew Schall, Jennifer. *INTRODUCTION TO EYE TRACKING*. Elsevier Inc., 2014.

[BC17] Bonita Sharif Benjamin Clark. *iTraceVis: Visualizing Eye Movement Data Within Eclipse*. IEEE Working Conference on Software Visualization, Cambridge, MA 02142 USA, 2017.

[dA11] Wil M. P. Van der Aalst. *Process mining Discovery, Conformance, and Enhancement of business processes)*. Springer, 2011.

[DDS00] Joseph H. Goldberg Dario D. Salvucci. *Identifying Fixations and Saccades in Eye-Tracking Protocols)*. ACM press, Cambridge, MA 02142 USA, 2000.

[Flu17] Fluxicon. *Disco, product*. Fluxicon, http://www.fluxicon.com/disco/, 2017.

[HC84] Xiaohui Yuan H.R. Chennamma. *A SURVEY ON EYE-GAZE TRACKING TECHNIQUES*. Indian Journal of Computer Science and Engineering (IJCSE)„ Department of MCA, Sri Jayachamarajendra College of Engineering, Mysore, Karnataka, INDIA, 1984.

[JL] Jae-Hyeon Ahn Joowon Lee. *Attention to Banner Ads and Their Effectiveness: An Eye-Tracking Approach*. ResearchGate.

[P09] Blignaut P. *Fixation identification: the optimum threshold for a dispersion algorithm*. National Center for Biotechnology Information, 2009.

[RS17] Kevin Wayne Robert Sedgewick. *Insertion.java*. Princeton, 2017.

[RvdL] Rik Pieters Ralf van der Lans, Michel Wedel. *Defining eye-fixation sequences across individuals and tasks: the Binocular-Individual Threshold (BIT) algorithm.* National Center for Biotechnology Information.

[Tob17a] Tobii. *How do Tobii Eye Trackers work.* Tobii.com, 2017.

[Tob17b] Tobii. *Specifications for the Tobii Eye Tracker 4C.* Tobii.com, 2017.

[Tob17c] Tobii. *What happens during the eye tracker calibration.* Tobii.com, 2017.

[TRS15] Braden M. Walters Sebastian C. Müller Michael Falcone Bonita Sharif Timothy R. Shaffer, Jenna L. Wise. *iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks).* University of Zurich, Switzerland Department of Informatics, 2015.

[TS84] Ted Megaw Tayyar Sen. *The Effects of Task Variables and Prolonged Performance on Saccadic Eye Movement Parameters.* Elsevier Inc., 1984.

[Wid84] Heino Widdel. *Operational Problems in Analysing Eye Movements.* Elsevier Science Publiahels B.V. (North-Holland),, Forschungsinstitut fur Anthropotechnik Wachtberg-Werthhoven FRG, 1984.