Noname manuscript No.
(will be inserted by the editor)

A Practical Guide on Conducting Eye Tracking Studies in Software Engineering

Zohreh Sharafi · Bonita Sharif · Yann-Gaël Guéhéneuc · Andrew Begel · Roman Bednarik · Martha Crosby

Received: date / Accepted: date

Abstract For several years, the software engineering research community used eye trackers to study program comprehension, bug localization, pair programming, and other software engineering tasks. Eye trackers provide researchers with insights on software engineers' cognitive processes, data that can augment those acquired through other means, such as on-line surveys and questionnaires. While there are many ways to take advantage of eye trackers, advancing their use requires defining standards for experimental design, execution, and reporting. We begin by presenting the foundations of eye tracking to provide context and perspective. Based on previous surveys of eye tracking for programming and software engineering tasks and our collective, extensive experience with eye trackers, we discuss *when* and *why* researchers should use eye trackers as well as *how* they should use them. We compile a list of typical use cases—real and anticipated—of eye trackers, as well as metrics, visualizations, and statistical analyses to analyze and report eye-tracking data. We also discuss the pragmatics of eye tracking studies. Finally, we offer lessons learned about using eye trackers to study software engineering tasks. This paper is intended to be a one-stop resource for researchers interested in designing, executing, and reporting eye tracking studies of software engineering tasks.

Zohreh Sharafi
University of Michigan, E-mail: zohrehsh@umich.edu

Bonita Sharif
University of Nebraska–Lincoln, E-mail: bsharif@unl.edu

Yann-Gaël Guéhéneuc
Concordia University, E-mail: yann-gael.gueheneuc@concordia.ca

Andrew Begel
Microsoft Research, E-mail: andrew.begel@microsoft.com

Roman Bednarik
University of Eastern Finland, E-mail: roman.bednarik@uef.fi

Martha Crosby
University of Hawai'i at Mānoa, E-mail: crosby@hawaii.edu

Keywords eye tracking · practical guide · empirical software engineering · program comprehension

1 Introduction

Eye trackers have evolved from invasive, costly, and difficult-to-use tools (Bergstrom and Schall, 2014) into versatile devices used to study diverse topics, such as driver–vehicle interfaces (Grace et al., 1998; Zhang and Zhang, 2010), airplane-cockpit usability (Duchowski, 2002), human–computer interactions (Strandvall, 2009; Poole and Ball, 2005), gaming (Sundstedt, 2010; Alkan and Cagiltay, 2007), and software development (Sharafi et al., 2015b).

Eye trackers allow researchers to record significant and substantial evidence about participants' ways of interacting with visual information, including reading patterns (Rayner, 1998), visual cues during search (Crosby et al., 2002), and interactions and engagement during oral conversations (Bednarik et al., 2012). They are also used in software engineering research to study various tasks, including, but not limited to, source code reading and debugging, comprehension of software artifacts *e.g.,* source code and UML class diagrams, and software traceability (Sharafi et al., 2015b). A handful of studies combined neuroimaging and biometric techniques (*e.g.,* EEG, fMRI, and fNIRS) with eye-tracking to measure task difficulty and cognitive load (Fritz et al., 2014; Peitek et al., 2018a; Fakhoury et al., 2018; Lee et al., 2018).

Researchers have used a large variety of eye tracking tools, techniques, measures, and analyses. This variety reduces the chances for successfully comparing and reproducing others' research methods, which impedes progress in eye-tracking and software engineering research and confuses novice researchers (as we have experienced ourselves with our students and collaborators). Researchers also face methodological, practical, and ethical challenges when using eye trackers.

Based on our collective, extensive experience in using eye trackers to study software engineering tasks for over 10 years, we offer this paper as a one-stop resource for researchers interested in designing, executing, and reporting eye-tracking studies of software engineering tasks.

We organize the paper as follows:

- Section 2 provides a brief background on eye-tracking technology, the main theories on which it is based, and how it should be used to collect reliable eye movement data.
- Section 3 discusses *when* and *why* use eye trackers in software engineering research with examples of prior eye-tracking studies from the literature and summaries of their research questions and results.
- Section 4 provides exhaustive definitions of the metrics associated with eye-movement data.
- Section 5 reports examples of typical studies from the literature that used eye trackers in software engineering.

– Section 6 provides practical advice on designing, setting up, and specifying eye-tracking tasks, selecting a target population, recruiting participants, analyzing data, and reporting results. It also describes typical threats to the validity of eye-tracking studies and ethical considerations.
– Section 7 presents how to analyze eye-tracking data along with set of visualizations and statistical analyses.
– Section 8 presents conclusions followed by future work.

This work complements our previous systematic literature review (Sharafi et al., 2015b) and a systematic mapping study by Obaidellah et al. (2018) which cover 63 research papers using eye tracking in software engineering research. These studies provide an exhaustive list of experiments and areas that were studied using eye trackers in software engineering. They also provide detailed information on the programming tasks, materials, metrics, and participants of previous studies. Obaidellah concluded, however, that there was a lack "of a methodology in conducting such experiments, which resulted in terminological inconsistency in the names and formulas of metrics used, which needs to be addressed in a future qualitative review." With this paper in particular, we strive to address this gap and help newcomers with practical advice on how to design and perform eye-tracking studies in software engineering. We promote the use of eye trackers in the study of program comprehension and provide a resource to help researchers use eye trackers in software-engineering research.

2 Foundations of Eye Tracking

Eye tracking involves collecting a participant's overt visual attention by recording eye gaze data (Rayner, 1978; Duchowski, 2007). Visual attention triggers the cognitive processes required for comprehension and problem solving, while cognitive processes guide visual attention to specific locations. Therefore, eye tracking is useful to study the participant's cognitive processes and effort while performing software engineering tasks (Duchowski, 2007).

A visual stimulus is any object, *e.g.,* a piece of source code, that is necessary to perform a task and whose visual perception by the participant triggers the participant's cognitive processes, and ultimately, some actions, *e.g.,* an edit of a statement in a source code file.

Eye gaze data is studied with respect to certain areas of a stimulus called Areas of Interest (AOIs). An AOI can be relevant to the participant while answering a particular question and can be irrelevant for another participant and question. For example, in a source code editor, an irrelevant AOI could be the class comment, while a relevant AOI could be the class name.

According to indicators of ocular behavior, eye gaze data—obtained by processing raw data recorded by an eye tracker with an event detection algorithm—belongs to the following categories (Rayner, 1978; Duchowski, 2007):

– Fixation: a spatially-stable eye-gaze that lasts for 100 to 300ms. During a fixation, the participant's visual attention is focused on a specific area of

the stimulus and triggers cognitive processes (Just and Carpenter, 1980). Fixation duration changes with task and participant's characteristics.
- Saccade: common, continuous, and rapid eye movements, lasting 40–50ms, occurring between fixations, but providing only limited visual perception.
- Pupil dilation and constriction: the pupil is the aperture through which light enters the eye, whose dilation is controlled by the iris muscle. A larger pupil may indicate increased cognitive effort (Poole and Ball, 2005).
- Scan path: through saccades, eyes fixate different parts of a stimulus, forming series of fixations, or visited AOIs, ordered chronologically.

Researchers in psychology report that information acquisition and processing mostly occurs during fixations. They also report that only a small set of fixations is necessary for a participant to acquire and process a complex visual stimulus (Privitera and Stark, 2000).

The meaning of fixations is context-dependent. A higher fixation rate on a specific AOI may indicate greater interest in its content, such as when reading some statements in a source code file. However, a cluster of fixations may also indicate effort/difficulties in understanding (Poole and Ball, 2005).

2.1 Eye Tracker Evolution

Figure 1 shows a brief history of eye tracking. Starting from 1879, Louis Émile Javal studied text-reading patterns via naked-eye observations. He reported that readers do not skim across words in texts smoothly but rather through a set of quick movements—saccades—and short pauses—fixations.

In 1898, Edmund Huey built the first eye tracker (Huey, 1908). This eye tracker was intrusive and required participants to wear a kind of primitive contact lens with a hole for the pupil. In 1901, Raymond Dodge and Thomas Sparks Cline used light reflected from cornea to develop the first non-invasive and precise eye tracker. However, this eye tracker required the participant's head to be absolutely still. In 1937, Guy Thomas Buswell performed the first recordings of eye movements on film. He performed a set of experiments with 200 participants looking at pictures and gathered about 2,000 eye-movement records, each consisting of a large number of fixations.

In 1948, Hamilton Hartridge and Landsborough C. Thomson proposed the first head-mounted eye tracker (Hartridge and Thomson, 1948), which was subsequently improved (Shackel, 1960; Mackworth and Thomas, 1962) to mitigate the constraints on head movements (Jacob and Karn, 2003).

In 1965, Alfred L. Yarbus (Yarbus, 1967) reported one of the first comprehensive accounts of the use of eye tracking, in his landmark book, "Eye Movements and Vision". In this book, translated in 1967 from Russian to English, Yarbus describes research results showing that eye movements depend on the tasks at hand, as shown in his famous image reproduced in Figure 2.

During the 1970s and 1980s, eye-tracking research flourished. Eye trackers became more accurate and less intrusive. Psychologists formulated different

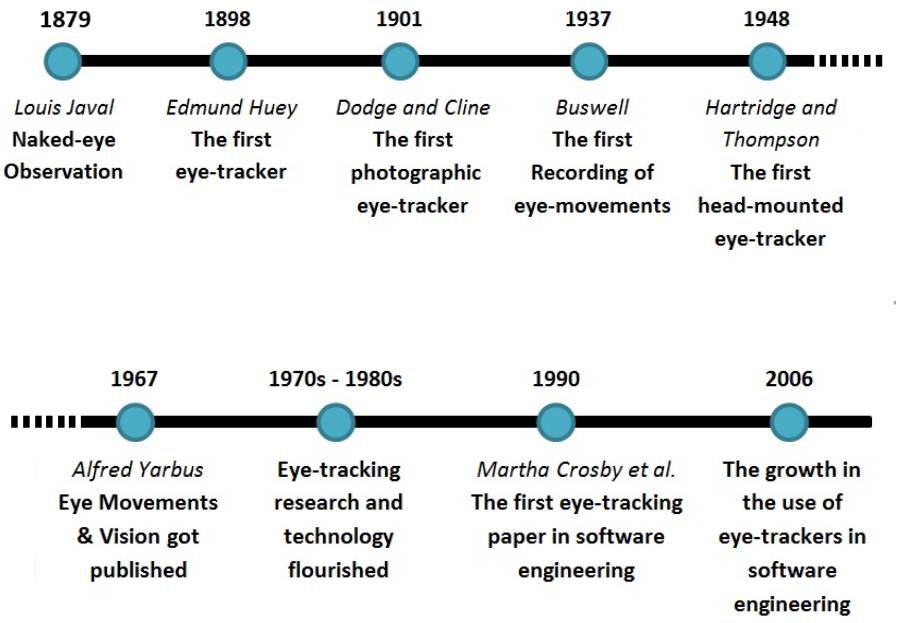


Fig. 1: A brief history of eye tracking.

theories to link eye gaze to cognitive processes (Jacob and Karn, 2003), including the influential strong eye-mind hypothesis by Just and Carpenter (1980). Eye-tracking technology continued to evolve and its applicability expanded to various business and scientific purposes.

The emergence of video-based eye trackers in the 1990s drastically improved their access and use in various research. In 1990, Crosby and Stelovsky (1990) performed the first eye-tracking study in software engineering. They investigated participants' reading strategies and their impact on the comprehension of procedural code.

Between 1990 and 2006 there was little work using eye tracking in software engineering. We posit this to be the case because (1) Crosby and Stelovskys work in 1990 was pioneering at the time and others did not consider using eye-tracking in software engineering research; (2) eye-tracking technology was not convenient until 2000s for its use in software engineering research, instead (3) research efforts were spent on underlying domains of perception, cognition, and reading.

Since 2006, the use of eye trackers in software engineering has shown modest but steady growth for the study of various topics, including collaborative

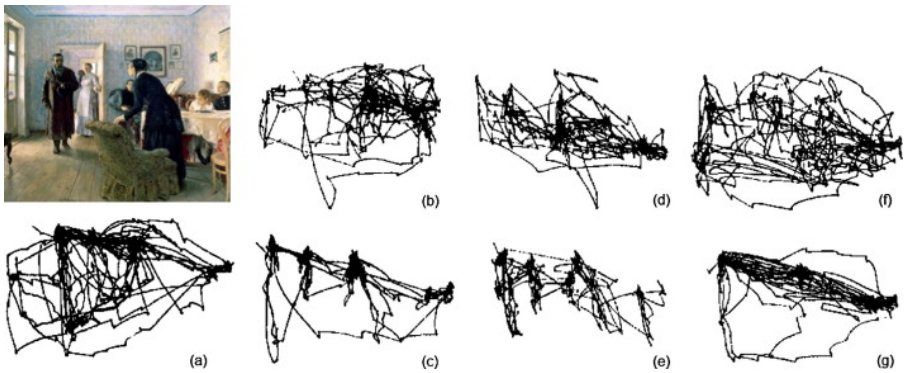


Fig. 2: Set of images from Yarbus with superimposed fixations and links between them. This shows that the participant's task changes their eye movements, from (Haji-Abolhassani and Clark, 2014; Yarbus, 1967).

interactions, program/model comprehension, code review, debugging, maintenance, and traceability (Sharafi et al., 2015b).

2.2 Eye Tracker Operation

A large variety of eye trackers are available on the market for business and scientific purposes (Sharafi et al., 2015b). These eye trackers vary in their physical forms and the methods used to track eye gaze (Bojko, 2005). An eye tracker usually includes the following hardware and software components:

- One or more cameras (usually infrared).
- One or more light sources (usually infrared).
- Image-processing software that detects and locates the eyes and the pupils and maps eye motion and the stimulus.
- Data collection software to collect and store real-time eye gaze data.
- Real-time display showing the location of the eyes' focus.

Currently available eye trackers mostly use the corneal-reflection/pupil-center method. An emitter of (typically invisible infrared) light is directed toward the eyes, entering the pupils. A significant amount of light is reflected back, causing the pupils to light up and appear bright. Another amount of light is reflected by the eyes and appears as glints on their surface.

Cameras can detect and track these reflections of the light source, along with other features such as the center of the pupil. Using a pre-established calibration and trigonometric calculations, and employing a variety of modeling approaches (Hansen and Ji, 2009), the image-processing software identifies the eye gaze (independent of head position and motion) (Jacob and Karn, 2003; Poole and Ball, 2005; Duchowski, 2007).

2.3 Eye Tracking Assumptions

The relation between eye gaze and cognitive processing is based on two assumptions from the theory of reading: the immediacy assumption and the eye-mind assumption (Just and Carpenter, 1980). The immediacy assumption proposes that interpretation of the stimuli begins immediately as a participant sees it, *e.g.,* as soon as a reader reads a word. The eye-mind assumption states that participants fixate their attention only on the part of the stimulus that is being currently processed (Just and Carpenter, 1980).

These two assumptions are the foundation of how eye gaze represents the participant's cognitive processes. Eye-gaze data indicates both the target of the participant's attention and the effort (or lack thereof) and length of time used to understand the stimulus. In addition, based on physiological studies, psychologists assume that participants do not have conscious control over many attributes of their eye gaze, *e.g.,* their pupil size, other than that for the location of their attention.

2.4 Eye Tracking Limitations

Eye trackers come with intrinsic limitations. We discuss the most important ones that exist at the time of writing this paper. If past history is to be our guide, we believe that much of these limitations will lessen or disappear as newer technologies and algorithms are invented in the years to come.

Accuracy. Accuracy (sometimes referred to as offset) is the difference between true and measured gaze data in degrees of the visual angle (Holmqvist et al., 2012). Current, popular eye trackers report accuracy values between 0.5 and 1 degree. An accuracy of 1 degree means that if the distance between the participant and the stimulus is 50 cm, the eye tracker could locate the eye gaze anywhere within a radius of 0.87 mm $\approx$ 1 cm of the actual, true position.

Although newer eye trackers have accuracy values below 0.5 degree, manufacturers usually compute this reported accuracy in an ideal situation, measured either directly after calibration or with artificial eyes (Sharafi et al., 2015b). They also avoid any obstacles that can interrupt the normal path of (infrared) light, such as thick eyelashes, hard contact lenses, or eye glasses.

Precision. Precision reports how well an eye tracker can reproduce the same results for two successive eye gazes at the same location. The precision values of common eye trackers range from 0.01 degree to 1 degree.

Drift. Drift is the gradual decrease over time of the accuracy of the eye-tracking data, when compared to the true locations of the eye gaze. Drift is caused by the deterioration of calibration over time due to the physiology of the eye, *e.g.,* changes in wetness, and other factors (Sajaniemi, 2004) .

Extrafoveal Vision. Extrafoveal vision (*i.e.,* parafoveal and the peripheral vision) makes up 98% of the human visual field and is not captured by eye trackers. Eye trackers only record foveal fixations, which are fixations corresponding to the central region of sight with the best visual acuity. Consequently, a lack of fixations on parts of a stimulus does not mean that participants did not see these parts, only that they chose not to direct their attention on them.

Although extrafoveal processing plays an important role in comprehension and has been studied in other fields, it has rarely been studied in software engineering research. Orlov and Bednarik (2017) performed the first study in software engineering that looked into the impact of extrafoveal information processing on source-code comprehension. They found that extrafoveal information was utilized more by expert programmers than novices.

3 Usage

Some software engineering tasks are better suited to the use of eye trackers than others. Before researchers design empirical studies, they should first answer the question, "Are eye trackers necessary and/or useful for this study?" This section presents various reasons when and why to use an eye tracker in software engineering research.

3.1 When?

Researchers can use eye trackers when they want to understand the impact of some visual stimuli on their participants' thought processes, *e.g.,* comprehension, collaboration, emotion, etc. Eye trackers can complement and enhance data collected using automated tools, *e.g.,* Mylyn (Murphy et al., 2006), semi-objective collection methods, *e.g.,* screen and audio recordings, and subjective collection methods, *e.g.,* surveys and questionnaires.

Any eye-tracking study will suffer from all common perceived limitations of human studies reported by software engineering researchers (Buse et al., 2011). It will also incur some extra research overhead. In the following, we discuss the actual costs of carrying out eye-tracking study.

Recruiting. Eye tracking constrains remote participation, such as via Amazon's Mechanical Turk crowdsourcing. There are webcam-based eye trackers available on the market but they may have low accuracy and precision.

Time and Cost. Experiment time and cost are significant concerns for eye tracking studies. The total cost include (1) infrastructure, hardware, and software purchase and maintenance, (2) training staff to manage and perform eye-tracking experiments, (3) time and effort to conduct the eye tracking experiments and (4) to analyze the resulting eye tracking data.

A typical eye tracking experiment lasts between 15 minutes to one hour and at least one experimenter must supervise the process. It usually involves one eye tracker and a PC with two screens (in a dual-screen configuration). One screen is used to present visual stimuli to the participants and is installed in front of the participants. The other screen, installed away from the participants' field of view, is used by the experimenter to perform calibration, control stimuli presentation, and monitor eye-tracking quality during the study. This dual-screen setup allows the experimenter to detect any issues with the equipment during the experiment, without interfering with the participants. Because the goal is to gather the data when the participant is engaged with the task, experiments must be performed in a quiet, dedicated room to avoid distractions and other confounding factors.

3.2 Why?

Eye tracking is recommended when researchers wish to understand their participants' cognitive processes as well as the intentions that motivate their actions. Researchers could use think-aloud protocols, interviews, questionnaires, or surveys to understand these processes and intentions, however, these methods depend on the participants' memory and communication skills, and–or subjective judgment to provide insight into their processes and intentions. Moreover, studies in cognitive science showed that the participants' perception of their own behavior does not always agree with their underlying processes and intentions (Bergstrom and Schall, 2014).

Eye trackers provide an objective, real-time, quantitative measure of eye gaze, without conscious filtering. They help researchers to study processes and intentions that participants cannot articulate (Ross, 2009). An eye tracker provides additional insights into what participants were doing and why based on where they focused their attention during a task.

Eye trackers help researchers determine (1) why participants have problems finishing a task, (2) where participants expect to find certain elements, (3) whether elements are distracting, (4) how efficiently a design, layout or artifact guides participants through a task, (5) whether there are differences in the participants' efficiency, based on their demographics or expertise, and (6) whether participants focus on details or briefly scanned the stimuli (Obaidellah et al., 2018; Sharafi et al., 2015b). All of this is done objectively while the task is being executed.

However, using eye trackers correctly is vital. Conducting an eye tracking study requires dedication to fine details to make sure the data is collected correctly and accurately. The collected data must be analyzed carefully to relate participants' fixations with their cognitive processes and intentions (Jacob and Karn, 2003; Karn et al., 1999). In particular, currently, there is no absolute way of knowing whether participants indeed understood parts of the stimuli on which they fixated.

4 Metrics

Analysis of eye-tracking data is challenging (Jacob and Karn, 2003) and the same applies in software engineering eye-tracking research, such as program comprehension (Bednarik, 2007). We now present definitions and metrics that can help analysing this data.

Karn et al. (1999) and Jacob and Karn (2003) classified eye tracking metrics as follows:

First Order Data. They are raw data, *i.e.,* unfiltered eye tracking outputs:

- **X,Y position**: the spatial coordinates of each gaze point, mapped to a location on the stimulus. These coordinates indicate the participants' focus of attention but not their understanding of the stimulus.
- **Pupil diameter**: the physical size of the pupil, usually its diameter in millimeters. Pupil size variations are more important than actual sizes because they vary across participants. Variations in pupil size, however, depend on cognitive workload and task difficulty, *i.e.,* increased effort and heavier cognitive workloads are related to larger pupil sizes (Beatty, 1982).
- **Eye blinks**: the number of blinks per unit of time, *e.g.,* per minute. These are associated by psychologists with cognitive workload. Lower blink rates indicate more attention (Poole and Ball, 2005; Beatty, 1982). Blink rates are not a common eye-tracking output, however. Blinks require vision algorithms to calculate them, and only certain trackers, such as the Smart Eye trackers, provide blinks as part of their output. Additional methods are necessary to accurately detect blinks in realtime, *e.g.,* using a video-based eye tracker (Divjak and Bischof, 2008).

X,Y positions, pupil diameters, and eye blinks, like other biometric data, are inherently noisy and contain outliers and invalid data. Therefore, this data must be cleaned before analysis (Soh et al., 2018). Researchers can clean this data visually, *e.g.,* by replaying the fixations and saccades and removing those obviously off, or statistically, *e.g.,* by removing outliers long fixations.

Several factors, including ambient light levels, participants' emotional and cognitive states, distance to the eye-tracker, and image quality of the camera impact this data. For example, blink rates increase with stress and anxiety but decrease with intense concentration.

Second Order Data. they include fixations and saccades, derived from the first order data using physiological thresholds. Eye trackers implement event detection algorithms to distinguish fixations from saccades using spatial and temporal criteria. These algorithms may impact the results of the analyses of the data (Salvucci and Goldberg, 2000).

Fixations can be voluntary or involuntary. Involuntary fixations stem from reflexes, *e.g.,* the optokinetic reflex that leads the eyes (and therefore, attention) to focus on moving objects. Eye tracking researchers are mostly concerned with voluntary fixations, although involuntary fixations may happen

in software engineering tasks, too, *e.g.,* when a window pops up to alert the participant to an event.

Third Order Data. They are obtained by eye tracking software through analyses of fixations and saccades:

- **Fixation count**: the number of fixations in an area of interest (AOI) or the whole stimulus.
- **Fixation duration or fixation time**: duration of all the fixations on an AOI or the stimulus.
- **Percentage of fixations or fixation rate**: ratio of the total number of fixations on one AOI or stimulus to another.
- **Time to the first fixation in an AOI**: time from the beginning of an experiment until the participant fixates on a given AOI.
- **All fixations within a selected time**: the number of fixations on an AOI or the stimulus in a given period of time.

Previous eye tracking studies used fixation count, fixation duration, and fixation rate to find the AOIs that attract more attention (Crosby and Stelovsky, 1990; Crosby et al., 2002; Uwano et al., 2006) and to measure the efficiency of participants' task-solving strategies (Soh et al., 2013).

A smaller fixation rate indicates a lower efficiency in search tasks: participants spend more effort to find relevant areas (Poole and Ball, 2005). Higher rates indicate that more effort is required to complete tasks, *i.e.,* find defects (Bednarik, 2012; Sharif et al., 2012) or fix bugs (Sharif et al., 2013), understand source-code statements (Binkley et al., 2013), recall the names of identifiers (Sharif and Maletic, 2010a), or explore different stimulus layouts (Guéhéneuc, 2006a; Yusuf et al., 2007).

When using these metrics to compare two AOIs or stimuli, the values must be adjusted by the sizes of the AOIs/stimuli to perform fair comparisons. For example, when working with text, the fixation count must be divided by the number of words in each AOI to compare two AOIs that do not contain the same numbers of words.

Fixation counts and durations are not correlated with one another (Sharafi et al., 2015a). Previous studies used both fixation counts and durations together to characterize participants' efforts:

- **Average Fixation Duration (AFD)** is also referred to as Mean Fixation Duration (MFD) is an average of fixation duration over all the fixations in an AOI, with respect to the fixations counts in all the AOIs or stimulus.
- **Ratio of On-target to All-target Fixations (ROAF)**: the sum of the fixation durations of all the fixations in an AOI divided by the fixation counts in all the AOIs or stimulus.

AFD has been proposed and used for relevant and non-relevant AOIs separately: Average Duration of Relevant Fixations (ADRF) and Average Duration of Non-Relevant Fixations (ADNRF) (Jeanmart et al., 2009; De Smet et al.,

2014; Soh et al., 2012). Higher ROAF values indicate higher efficiency associated with lower effort. They also indicate the importance of an AOI relative to other AOIs or the stimulus.

To compare fairly two stimuli with each other, the size of each stimulus must be taken into account. Jeanmart et al. (2009) proposed the Normalized Rate of Relevant Fixations (NRRF) to compare two (or more) stimulus with each other. Higher values of NRRF indicate increased effort to understand the corresponding stimulus.

AFD, ROAF, and related metrics were used to measure and compare the amount of visual effort (or difficulty) to perform a task (Crosby and Stelovsky, 1990; Bednarik and Tukiainen, 2005, 2006; Jeanmart et al., 2009; Cepeda and Guéhéneuc, 2010; Busjahn et al., 2011; Bednarik, 2012; Soh et al., 2012; Sharafi et al., 2012; Petrusel and Mendling, 2012; Binkley et al., 2013; Cagiltay et al., 2013; Sharafi et al., 2013; De Smet et al., 2014) and to find the AOIs that are most important for the participants to perform their tasks (Bednarik and Tukiainen, 2006; Jeanmart et al., 2009; Cepeda and Guéhéneuc, 2010; De Smet et al., 2014).

Similarly to fixations, several third order metrics exist based on saccades:

- **Saccade count**: the total number of saccades in an AOI or the stimulus.
- **Saccade duration or saccade time**: the duration of all the saccades in an AOI or the stimulus.
- **Regression rate**: the percentage of backward or regressive saccades, *e.g.,* leftward in left-to-right source-code reading, over the total number of saccades (Poole and Ball, 2005; Busjahn et al., 2011).

Higher regression rates indicate increased difficulty in performing and completing a task (Goldberg and Kotval, 1999; Poole and Ball, 2005). Busjahn et al. (2011) reported higher regression rates for source-code reading compared to natural-language text reading. Fritz et al. (2014) used saccades to study the impact of the difficulty of some stimuli on participants.

Fourth Order Data. Sequences of fixations or AOIs are called scan paths. Scan paths describe the durations and lengths of eye gazes. They are indicators of search efficiency. Longer and longer-lasting scan paths indicate that the participants took more time or effort to explore a stimulus to find relevant AOIs, which in turn indicate less efficient scanning and searching.

Scan paths naturally become longer as participants spend time in an experiment, which make them difficult to analyze and compare. They must be studied by taking into account the numbers and locations of fixations as well as their temporal order and duration.

Scan paths can be studied using the following algorithmic tools:

- **Transition matrix**: a tabular representation of transition frequencies between AOIs. The matrix density can be computed as the number of nonzero cells divided by the total number of cells to compare two transition matrices with each other. Figure 3 shows an example of a scan path on a visual

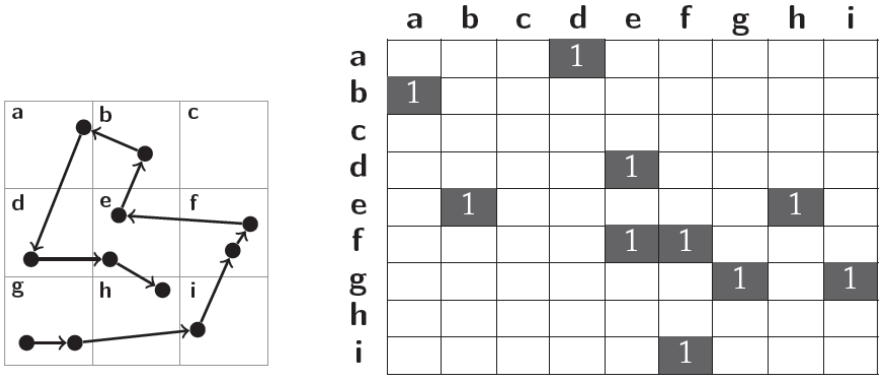


Fig. 3: Example of scan path and corresponding transition matrix, from (De Smet et al., 2014). A 1 in each matrix cell indicates a directed edge in the scan path between the points labeled by the row and column.

grid and its transition matrix with a spatial density of 12% (10 cells out of 81 are filled). Increased spatial density indicates more extensive search with inefficient scanning (Sharafi et al., 2015a).

- **Scan path recall, precision, F-measure**: measures of the relations between AOIs and scan paths. Scan path recall is the number of fixated, relevant AOIs divided by the number of all relevant AOIs. Scan path precision is the number of fixated, relevant AOIs divided by the number of all AOIs. Scan path F-measure is a weighted average of scan path precision and recall (Petrusel and Mendling, 2012).
- **Edit distance**: uses the Levenshtein algorithm to compute the minimum editing cost of transforming one scan path to another with basic operations, such as insertion, deletion, and substitution (Levenshtein, 1966).
- **Sequential PAttern Mining (SPAM)**: a depth-first algorithm that can be used to compare scan paths based on the fixation locations and durations (Ayres et al., 2002).
- **ScanMatch**: based on the Needleman-Wunsch algorithm used in bioinformatics to compare sequences of DNA (Cristino et al., 2010). It uses temporal binning to adjust the length of two (or more) scan paths based on fixation durations. It outputs a similarity value of the two scan paths.

Studies compared scan paths to identify and analyze participants' viewing strategies to explore stimuli and solve tasks (De Smet et al., 2014; Sharafi et al., 2013; Hejmady and Narayanan, 2012; Busjahn et al., 2015). Lower edit-distance and SPAM values indicate similarity among participants and show that they used similar reading strategies (De Smet et al., 2014; Sharafi et al., 2013; Hejmady and Narayanan, 2012).

Other fourth order data include:

- **Attention switching**: the total number of switches between a list of AOIs per unit time, *e.g.,* one minute.

– **Fixation Spatial Density (SD)**: represents the stimulus coverage, the dispersion of the participants' fixations (Goldberg and Kotval, 1999). If a stimulus is divided into equal cells to form a grid, then SD is the number of visited cells, *i.e.,* a cell that received at least one fixation. Smaller spatial density values indicate less coverage.
– A **convex-hull area**: the smallest convex set of fixations that contains all of participants' fixations (Goldberg and Kotval, 1999). A smaller value indicates that the fixations are close together and that the participants spent less effort to find relevant areas in a stimulus.
– **Linearity**: associated with participants' search strategies (Poole and Ball, 2005). It is defined using eye gaze linearity, *e.g.,* left-to-right and top-to bottom for readers of Latin-based natural languages.

Studies have used SD and convex-hull area to study the coverage of fixations (Sharafi et al., 2012; Soh et al., 2012; Sharafi et al., 2013), which relates to the efficiency of the search strategies used by participants. It also indicates the preferred parts of visual stimuli.

In source code, "linearity represents how closely readers follow a texts natural reading order" (Busjahn et al., 2011).

Metrics based on the spatial distributions of fixations are sensitive to invalid data. For the convex-hull area, just one fixation deviating from its actual location can change the shape and the area of the convex hull significantly. Thus, noise removal and data cleaning is necessary to use fourth order data.

5 Typical Eye-tracking Studies in Software Engineering

Based on the current literature and state-of-the art, we now describe examples of studies that used eye trackers for software engineering tasks. These examples cover the main usages of eye trackers in software engineering research and some representative studies in terms of objectives, designs, metrics, etc.

5.1 Program Comprehension

Turner et al. (2014a) studied the effects of programming languages on developers' performance in a number of programming tasks. One of their research questions asked if there was a significant difference in visual effort in overview and bug localization tasks between C++ and Python programs. They recruited 38 undergraduate and graduate students, some with C++ and some with Python experience to explain what C++ and Python code samples did (overview) and to look for and explain any error they could find in the programs (bug localization). To measure visual effort, the authors recorded four metrics: (1) fixation count and (2) duration for the entire program and (3) fixation count and (4) duration on the buggy lines of the programs. Each of these metrics should increase with the participants' visual effort. The authors compared the metrics recorded for the median of each group who studied C++ and

Python programs using the non-parametric Mann-Whitney test. They found no significant differences between any of the metrics but more fixations on the buggy lines of the Python programs than on the C++ programs.

Binkley et al. (2013) conducted two eye-tracking studies on the impact of identifier styles on program comprehension. They asked 169 students to recall English words and comprehend C++ source code. They analyzed fixation rate, average fixation duration, ratio of on-target to all-target fixations (ROAF) with a set of statistical methods, including linear mixed-effects regression, Generalized Linear Mixed Models, and logistic regressions. They concluded that experts are less affected by the identifier styles than novices. They also concluded that source-code reading and comprehension are different from that of natural text because the effects of style on quality assessment and thinking time varied between natural text and source code.

5.2 Diagram Comprehension

Guéhéneuc (2006a) and Yusuf et al. (2007) independently investigated the impact of layout, color, and stereotypes of UML diagrams on comprehension. The later asked 12 students and faculty members to work on three UML class diagrams with different layouts (orthogonal, three-cluster, and multi-cluster layouts). After calculating fixation count, first fixation time, and comparing the distribution of fixations via heat maps and gaze plots, they concluded that experts use extra information, *e.g.,* color, layout, and stereotypes, more efficiently than novices to browse UML diagrams. They also showed that the layouts with additional, design, semantic information are more effective.

5.3 Code Review

Uwano et al. (2006) investigated the impact of scan time on source code review, *i.e.,* reading the entire code before investigating the review. After recording the eye movements of 5 students debugging C code, they computed the fixation count. They used an algorithm to generate a set of graphs depicting the time sequence of focused code lines. They used these graphs to compare the participants' viewing strategies and fixation counts to compare the scan times. Results showed that a longer scan time leads to faster bug finding. They showed the tendency of novices to go back and forth between code and graphical representations.

Begel and Vrzakova (2018) studied participants' source-code scanning behaviour during code review to identify how suspicious patterns of code are recognized. They studied of 35 developers performing 40 code reviews. By playing back the eye gazes, they observed that code review was mainly a code-scanning task in which the majority of the code is skimmed rapidly.

5.4 Traceability

Ali et al. (2015) performed an eye-tracking study to understand how participants verify requirement traceability links and identify the most used source code entities (SCEs). They asked 26 undergraduate and graduate students to read a set of Java code snippets and answer one comprehension question about each. They used the total fixation duration to calculate the time spent on each SCE, including class, method, variable names, and comments, and ranked the developers preferred SCE. Then, they used the ranked SCEs to propose two new weighting schemes to recover RT links with an IR technique. One is called SE/IDF (source code entity/inverse document frequency) and the other is DOI/IDF (domain or implementation/inverse document frequency). They reported that participants have distinct preferences for different SCEs, method names and comments over others, and that the proposed weighting schemes statistically improve the accuracy of their IR-based techniques.

5.5 Education

Busjahn et al. (2014) performed a case study to evaluate the feasibility of the use of eye trackers in computing education and teaching. They recruited two developers to read Java code and answer a comprehension question while recording their eye movements. Then, they provided the participants with two videos showing the Java code overlaid with their eye movements and asked them to encode them according to a multi-tier coding scheme, *e.g.,* blocks vs. lines. They presented the lessons and challenges learned from the data analysis and their participants' comprehension of the encoded scan paths.

5.6 Eye tracking and Other Psycho-physiological Measures

Fritz et al. (2014) performed an eye tracking study while gathering other psycho-physiological measures using electroencephalography (EEG), electrodermal activity (EDA), and NASA TLX scores, to evaluate task difficulty. While previous studies focused on *post hoc* data analysis, they proposed a new approach to detect when developers experience difficulty while working with source code. They recruited 15 professionals to work on ten short tasks. They trained a Naive Bayes classifier using a combination of these measures to predict whether a participant would feel a task was difficult. Their classifier achieved over 70% precision and over 62% recall. They confirmed that the duration of saccades is related to the participants' cognitive effort.

5.7 Source Code Summarization

Rodeghero et al. (2014) developed a code summarization tool based on eye movements of ten developers, asked to read Java code and wrote English sum-

maries of Java methods in the snippets. They extracted common keywords from the code based on the amount of fixation time spent by participants on these keywords. They then used Vector Space Model Summarization (Haiduc et al., 2010) (VSM TF/IDF) to extract keywords and compare the results with the keywords obtained by their proposed method and those from the developers. They showed that developers focused more on the keywords that they found relevant and also used these keywords to write their summaries. Developers spent a different amount of visual attention on different keywords. Method signatures attracted more attention than method invocations, which in turn attracted more attention than control flows.

Abid et al. (2019b) recruited 18 developers to work on 63 methods from five different systems. In contrast with previous studies that used short methods in isolation, they asked developers to work in Eclipse, using scrolling in files and switching between files. They collected eye-gaze data, written summaries, and the time spent by the participants to complete each summary. They showed that keywords in the control flows of the methods were revisited frequently rather than read for a long period of time. They also compared experts and novices and found that the sizes of the methods mattered: as their sizes increased, experts revisited the method bodies more frequently than their signatures. They also compared mental cognition models (*i.e.,* bottom–up or top–down) during code summarization (Abid et al., 2019a) and reported that both experts and novices using the bottom–up mental model: they read methods closely rather than browsing methods. However, novices needed more gaze time then experts to apply the bottom–up strategy.

6 Pragmatics of a Typical Eye Tracking Study

This section presents a practical approach to eye-tracking experimental design and setup. We direct the reader to general guides on controlled experiments (Ko et al., 2015), which we extend to include other important setup and pragmatic issues that arise when using an eye tracker as part of data collection. Some of these issues were reported in previous works while others were identified through our own experiences while yet others could pertain to any empirical studies. Whenever possible, we provide references to the works, if any, that reported the issues first.

6.1 Experimental Setup

We now discuss setting up experiments with eye-trackers, from their purchase to the recording of their data.

6.1.1 Eye Tracking Device

Eye trackers have improved greatly since their beginning, and are now both accurate and readily available. There are differences between affordable eye

trackers and high-end eye trackers on the market. Sampling rate, accuracy, and freedom of movement are key factors that determine the quality and, ultimately, the price of an eye tracker.

Webcam-based eye trackers provide a practical, low-cost, or even free solution (by getting eye-tracking data from a "normal" camera, already installed in almost all laptops). The main advantage of these eye trackers is to collect gaze data on any population quickly, just like sending out a typical online survey. However, researchers do not use these devices often because they are inaccurate when compared to infrared eye trackers[1]. The environment is also not as controlled (with regards to the noise level and lightning and ambiance, cf. Section 6.1.2) as appropriate for precise study. If the goal of the study is to find out whether a specific part of the screen has been looked at by participants, a webcam-based eye tracker suffices. However, if precise temporal or spatial resolutions are required (*e.g.,* a line by line or word by word comparison of the source code or text), then an infrared eye tracker is needed.

There is a large range of prices for the various models of video-based remote eye trackers. Low-end eye-trackers cost from $100 to $2,000 and are not generally used for advanced research, especially if the researchers are interested in spatio-temporal resolution for saccade detection. Mid-end eye trackers cost from $2,000 to $10,000, while high-end ones can cost over $10,000.

To decide which eye tracker to purchase, we recommend checking the two following resources: (1) Obaidellah et al. (2018) list a variety of manufacturers and eye trackers for software engineering research and (2) Farnsworth (2019a) presents an overview of the price points for various eye trackers. Some eye trackers are more extensible, creating a 3D model of the world around them, whereas others do not allow this setup. Researchers must choose the right eye tracking device for their study.

6.1.2 Eye Tracking Environment

The environment in which an eye tracking study is performed is important. Researchers should conduct their studies in quiet, windowless rooms with good lighting. The room should be calm and with a stable lighting that does not produce glare on the screens or interfere with the infrared light of the eye tracker. Environmental changes (*e.g.,* light conditions and humidity) may result in drift and inaccurate data (Pernice and Nielsen, 2009; Sajaniemi, 2004).

To avoid inaccurate data due to participants' head and body movements, researchers should place participants in a stationary seat with a headrest but without wheels or leaning capability. Slight head movements are acceptable and participants should sit with a normal posture in front of the eye-tracking screen. A chair and desk with vertical adjustment capability are useful to accommodate different participants' heights. Some eye tracking software tools provide indication of the optimal distance and head placement.

[1] https://imotions.com/blog/webcam-eye-tracking-vs-an-eye-tracker/

Most studies use only a small number of participants as shown in previous works, *e.g.,* (Sharafi et al., 2015b), and only one participant can use an eye tracker at a time. Therefore, the room and eye tracker should be straightforward and simple in accommodating one participant at a time.

6.1.3 Overview and Calibration

At the start of a study, researchers must provide participants with relevant information including:

1. The procedure and policy for the data analysis, storing, and discarding, in particular whether the data is anonymous or not.
2. The number of tasks that must be completed, including the number of questions for each task, and an estimation of the task duration.
3. The procedure for a participant to inform researchers when a task is completed or when a task is abandoned.
4. The means by which a participant can relax and work as if they are alone. Participants should not explain what they are doing (*i.e.,* no think-aloud as the cognition required alters low-level eye movements).

Then, researchers must calibrate the eye tracker to participant's eyes:

1. Researchers must inform the participants at all times about their actions, *e.g.,* when adjusting the participant's chair or the screen.
2. Researchers must ensure that the participant's head appears in the middle of the screen and at a distance of about 50–60 cm, which may vary with the specifications of the eye tracker and the participant's height.

6.1.4 Pilot Study

We recommend to conduct a pilot study with at least one participant to identify any potential problem in the experimental design/setup:

1. Check that the eye tracker and room are set up correctly.
2. Check that recording properly acquires and saves data to disk.
3. Check the quality of recorded data to make sure that the light conditions are appropriate to capture eye movements.
4. Observe how the participant reacts to the research questions, setup, stimuli, and tasks.
5. Record the time taken by the participant to complete the study.
6. Analyze the data to evaluate the results and avoid any data loss.

6.1.5 Recording

Researchers must check participants at all time to ensure the quality of the data, including avoiding:

– Holding any material in between the eye tracker and the participant's face, *e.g.,* an answer sheet or the participants' hands.
– Leaning back, forward, or sideways in a manner that makes the eyes move out of the tracked zone.
– Squinting or closing eyes at length and–or repeatedly.

If such events happen, the researchers must record the time of the event (timestamp) to analyze later whether the data (or part thereof) can still be used or should be discarded entirely. If the researcher notices that the participant is moving too much, they should consider re-running the eye tracking calibration procedure after each task.

6.2 Stimuli and Tasks

Stimuli. In most previous eye tracking studies, participants worked with a set of static stimuli. A static stimulus is an image shown on a screen and on which participants have no control. The majority of previous studies used small source-code snippets that fit on one screen with appropriate font size and type for reading. We provide guidelines for designing a static stimulus:

– Ask only one question per stimulus. A static stimulus limits the number of elements that can be displayed. We recommend presenting one question per image to conserve display space and simplify data analysis. If several questions are necessary, then the stimulus can be repeated.
– Show the question on the top-left corner of the stimulus to avoid that elements placed there receive undue attention: previous work showed participants tendency to look to the top-left corner (Goldberg et al., 2002).
– Avoid over-crowding the stimulus. Eye trackers have a specific resolution below which it is not possible to distinguish whether attention was focused on one element or another of the stimuli.
– Use fixed-width fonts (mono-spaced and mono-type), *e.g.,* Courier, for the stimulus. These fonts provide the same horizontal space for all characters and, thus, better control over the visual stimuli.
– Use appropriate font size: smaller fonts have the advantage of allowing more text on the screen, but hinder capturing fixations of participants natural, smooth reading. The font size needs to be big enough to support mapping of gaze to words. A trial/test needs to be done prior to conducting the study to determine what size works best.

Font types and sizes can make fixation positions less accurate, data noisier, and data analysis more difficult. Choosing a proper font type and size is particularly important if word-level analysis must be done (*e.g.,* source code and identifiers understanding).

The participants' viewing distance from the screen, the eye-tracker accuracy, and ecologically valid study design are all critical factors in choosing a font size (Godfroid, 2019). Results in vision research reported a range of 4pt to

40pt for eye-tracking studies (Godfroid, 2019). We recommend choosing a size closer to the middle of this range. If a fixed-width font is used and participants sit at approximately 50 cm from the screen, then a 16 to 18 point font is a good choice.

In some previous eye tracking studies, participants worked with dynamic stimuli, *i.e.,* stimuli with which they could interact. In particular, Clark and Sharif (2017) developed *iTrace* and *iTraceVis*, which provide an automatic mapping of eye-gaze data on source-code elements displayed in an IDE, such as Eclipse or Visual Studio, even with scrolling in and switching between files. Thus, researchers can study participants' complex interactions with IDEs and the elements displayed in these IDEs.

Tasks. The tasks should be engaging and easy to understand. Appropriate tasks must trigger the participants' cognitive processes when performing their tasks. There is currently no absolute way of telling whether participants understood the elements on which they fixated. Therefore, studies must include comprehension questions/measures to asses whether participants understood the elements presented by the stimuli.

To avoid fatigue, it is important to strike a balance when preparing the stimuli between completeness and duration and to avoid long sessions. An eye-tracking session should not last longer than 90 minutes. Changes in the physiology of the eye over time, *e.g.,* dryness caused by fatigue, may result in measurement errors. As a general rule, if a session is longer than 30 minutes, then participants should be given time to relax their eyes between successive stimuli, for example, by working on questions printed on paper.

6.3 Recruiting Participants

Researchers must define the population from which they will select the participants. The ideal target population may be that of all software engineers who perform development and–or maintenance activities. However, around 77% of previous eye tracking studies used populations of students and–or faculty members (Sharafi et al., 2015b). Indeed, they argued that students are akin to junior software engineers, while faculty members may have considerable development/maintenance experience.

Researchers should consider the following benefits when selecting participants from a population of students:

- Students come to the school regularly and they are more accessible (schedules, willingness) to academic researchers than professionals who work in industry. Researchers already in industry will find that their software development colleagues may be easier to recruit.
- Students in the same year often have comparable experience and expertise, potentially increasing the homogeneity of the population and comparability of the data collected from different participants.

Researchers should recruit professionals if their research questions pertain to the impact of expertise and experience Kitchenham et al. (2002). One way to increase the number of participants is to run the study for a long time. Another possibility is to bring eye trackers to conference venues with many professional developer attendees. A third possibility is to visit and recruit participants from the local area beyond the researchers' organizations.

Researchers must strive to recruit enough participants to obtain statistically significant results. There is no unique sample size for eye-tracking studies because the size depends on many factors, including the research questions and the experimental design (*e.g.,* within subjects vs. between subjects) (Bojko, 2005). Previous eye tracking studies had from 5 to 169 participants, with a mean value of 56.9 and median value of 18. 56% of these studies had fewer than 20 participants.

Researchers must define exclusion criteria to reject participants who cannot participate in an eye-tracking experiment. These criteria mostly pertain to the use of visual aids. While modern eye trackers can collect eye gaze data even if participants wear eyeglasses, some cautions (Pernice and Nielsen, 2009) should be taken with other visual aids/vision impairments, in particular:

- Bifocal or progressive glasses.
- Dirty or damaged glasses.
- Dyslexia and other such disorder.
- Thick rimmed glasses.
- Droopy eye and/or lazy eyelids.
- Heavy eyelashes or mascara.
- Fringes covering eyes, hats, or other artifacts.
- Eye problems such as uncorrected astigmatism.
- Photosensitive epilepsy.

McChesney and Bond (2019), for example, compared 28 developers, with and without dyslexia, performing program comprehension tasks. They reported that dyslexic developers had a different gaze behaviour than non-dyslexic developers and that their gaze behaviour was also different from what was expected from the literature on dyslexia and natural text.

Prior to the session, the experimenters might include instructions to mitigate some of the aforementioned barriers, including: wearing minimal or no eye makeup to the session, bringing or wearing corrective optics, such as single-vision glasses or contact lenses, checking the cleanliness of the eye-glasses, and having headbands/hairpins to put up long hair and bangs to provide the eye tracker a clear view of the eyes. Some eye trackers also require a view of the participant's ears to build the appropriate models necessary for eye detection. So hair should be moved behind ears before beginning the study.

6.4 Background Questionnaire

It is common to ask participants to fill out a survey regarding their experience or knowledge of software development and maintenance. The questionnaire should ask for any existence of eye problems, epilepsy, or reading disorders.

Questionnaires are typically asked before the study begins. However, care should be taken to avoid the stereotype threat (Spencer et al., 1999; Shapiro and Neuberg, 2007). Women and underrepresented minorities are at higher risk of being judged by the negative stereotype that they have weaker ability (Steele and Aronson, 1995; Spencer et al., 1999). To alleviate this threat, we recommend asking questions that might interfere with the participants' performance, such as ones related to expertise and proficiency, at the end of the study. Moreover, unless it is part of the design, researchers must avoid priming in the questions with information that may lead to heightening the salience of participants' personal identity.

6.5 Experimental Design

We now discuss the design of an eye tracking experiment, including defining research questions and hypotheses, identifying dependent, independent, and mitigating variables and calibration.

6.5.1 Research Questions

After reviewing previous eye tracking studies in software engineering (Sharafi et al., 2015b), we classify research questions into the following categories:

- To evaluate the usefulness of some systems, artifacts, or tools when participants perform a specific task with one of these.
- To evaluate the participants' effectiveness and efficiency when performing a specific task while using some systems, artifacts, or tools.
- To find the areas of interest in some stimulus by studying the distribution and the intensity of participants' visual attention.
- To detect navigation strategies used by participants by studying their scan paths when performing software engineering tasks.

Table 1 summarizes the types of research questions asked by researchers and answered with eye-tracking studies. Tables 2, 3, and 4 provide examples of research questions of eye-tracking studies in software engineering; most common are questions to evaluate usability, efficiency, and–or effectiveness.

6.5.2 Variables and Measures

After expressing research questions and hypotheses, researchers must define dependent, independent, and mitigating variables. In eye-tracking studies, the

Table 1: Types of research questions in eye-tracking software engineering experiments.

Categories	Types of Questions
Usability, Efficiency, and Effectiveness Evaluation	Is (X) useful for task (Y)?
	What types of (X) are most effective for task (Y)?
	What types of tasks (Y) benefit from artifacts (X)?
	Do the participants' individual characteristics (Z) impact their effectiveness and efficiency when performing task (Y)?
Finding the Areas of Interest	What items or what parts of artifact (X), do participants view when performing task (Y)?
Navigation Strategies	How do participants navigate through artifact/system (X) when performing task (Y)?
	Does the type of artifact (X) impact the participants' navigation strategies when they perform task (Y)?
	Do the participants' individual characteristics (Z) impact their strategies when performing task (Y)?
	Do developers follow any pattern when they are working on task (Y)?

independent variables are the elements presented in the stimuli while the dependent variables are mainly the measures of the participants' eye-gaze data and how well the participants answered the questions asked in the study.

The choice of the stimuli and that of the dependent variables depends on the research questions. Eye-gaze data has been extensively used to measure the visual (cognitive) effort that is representative of the tasks and stimuli being assessed. Sharafi et al. (2015a) provided a list of visual effort metrics while discussing how previous studies used and interpreted them. Table 5 presents an examples of variables used in previous eye-tracking studies.

6.5.3 Calibration between Saccades and Fixations

Researchers study a variety of stimuli and tasks in software engineering, such as diagrams, requirements documents, and source code. Depending on stimuli and tasks, they must use appropriate fixation identification algorithms (FIAs). Indeed, a FIA used for natural text might not work well for source code.

Salvucci and Goldberg (2000) compared FIAs. They categorized algorithms with respect to their spatial and temporal characteristics. They found that the dispersion-threshold identification (I-DT) and Hidden Markov Model fixation identification (I-HMM) algorithms are the FIAs, independent of the domain, *e.g.,* image scanning, driving, etc.

FIAs require researchers to set some parameters. Two main parameters are (1) the number of raw data samples that should be considered a fixation and (2) the maximum distance in pixels between a raw data point and the average fixation point to be still considered a fixation. Most eye trackers and data analysis tools come with default parameters values that can be changed. Default parameters must be used with caution and we advise running pilot studies to tune these parameters.

Table 2: Examples of research questions for usability, efficiency, and effectiveness.

Does SeeIT 3D help a participant in solving overview, new feature, and bug fixing tasks? (Sharif et al., 2013)
Is there an improvement in the comprehension of certain software-maintenance tasks for stereotyped class diagram layouts vs. layouts based on pure aesthetics? (Sharif and Maletic, 2010b)
Which software comprehension tasks benefit most from stereotyped class diagram layouts? (Sharif and Maletic, 2010b)
Does the layout of a class diagram affect the visual effort needed during a software maintenance task? (Sharif and Maletic, 2010b)
Does the type of requirement representations (graphical vs. textual) impact the participants effort, time, and answer accuracy in requirements comprehension tasks? (Sharafi et al., 2013)
Does identifier style impact readability? (Binkley et al., 2013)
Assuming a difference in readability, does identifier style affect higher-level comprehension activities? (Binkley et al., 2013)
Does programming language affect the effectiveness and efficiency of solving overview and bug-finding tasks? (Turner et al., 2014a)
Do the participants' genders impact their effort, their required time, and their ability to recall identifiers in source-code reading? (Sharafi et al., 2012)
What is the relation between a participant's professional status and her class diagram comprehension? (Soh et al., 2012)
What is the relation between the expertise of a participant and class diagram comprehension? (Soh et al., 2012)
Is there a difference between experts and novices with respect to the SeeIT 3D tool? (Sharif et al., 2013)
Do stereotyped layouts help design experts and novices in the same way? (Sharif and Maletic, 2010b)
Is there a difference in visual effort while reading and analyzing source code in C++ vs. Python? (Turner et al., 2014a)
Do blind programmers take less time to complete a task using AudioHighlight as compared to StructJumper? (Armaly et al., 2018)

Table 3: Examples of research questions for finding areas of interest.

What do participants really look at in class diagrams? (Yusuf et al., 2007)
On which items in class diagrams do participant fixate the most? (Yusuf et al., 2007)
Does experience influence a participant's focus on critical areas of the algorithm? (Crosby and Stelovsky, 1990)
What are the important source-code entities (SCEs) to which participants pay attention when verifying traceability links? (Ali et al., 2015)
Do developers read error messages? (Barik et al., 2017)

6.6 Definition of the Areas of Interest

While designing an eye-tracking study, researchers must define the Areas of Interest (AOIs) based on the research questions, hypotheses, and variables. AOIs are used to describe visual stimuli but there are no standard method for defining AOIs in terms of size and granularity (Goldberg and Helfman, 2010). In the following, based on our collective experience and guidelines by Goldberg and Helfman (2010), we provide our recommendations:

Table 4: Examples of research questions to identify navigation strategies.

How do participants navigate through class diagrams? (Yusuf et al., 2007)
Does the layout of a class diagram affect the eye gaze behavior of experts and novices? (Sharif and Maletic, 2010b)
Does the structure of the representations lead participants to use specific task-solving strategies (top-down vs. bottom-up) during requirements comprehension tasks? (Sharafi et al., 2013)
Is there a difference between reading simple text and complex text, such as algorithms? (Crosby and Stelovsky, 1990)
Do the viewing patterns of experienced participants differ from those of novices? (Crosby and Stelovsky, 1990)
Is there a difference in eye gaze behavior between novices and non-novices between C++ and Python? (Turner et al., 2014a)
Do developers follow any pattern when they are required to comprehend regular code? In particular, are their efforts equally divided among regular segments? (Jbara and Feitelson, 2017)

Table 5: Examples of dependent, independent, and mitigating variables.

Dependent Variables	Number or the percentage of correct answers (accuracy) (Turner et al., 2014b)
	The amount of time spent on the stimulus (Binkley et al., 2013)
	The amount of visual effort spent (Sharafi et al., 2013)
Independent Variables	The presence of a tool, an artifact, or an issue (*e.g.,* the presence of defects in the code vs. no defects (Uwano et al., 2006))
	Identifier styles (camel case vs. underscore) (Binkley et al., 2013)
	Different types *e.g.,* graphical vs. textual representations, or Python code vs. C++ code (Sharafi et al., 2013)
Mitigating Variables	Participant's knowledge and experience (Sharif et al., 2012)
	Language proficiency (Sharafi et al., 2013)
	Study level (Ali et al., 2015)

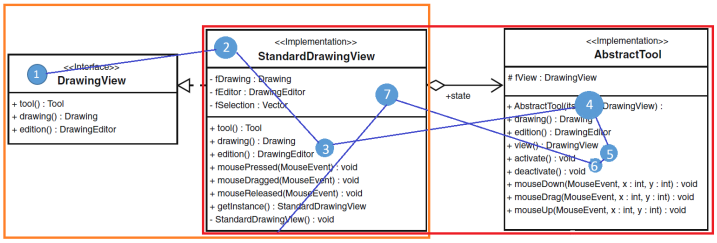


Fig. 4: The rectangular orange AOI overlaps the rectangular red AOI. Try not to do this because it makes analysis much more confusing.

Sizes and Positions of AOIs. Eye tracking accuracy and precision impact the size of AOIs. To limit the impact of fixation precision and accuracy, researchers must define AOIs large enough to capture all relevant fixations. They can add

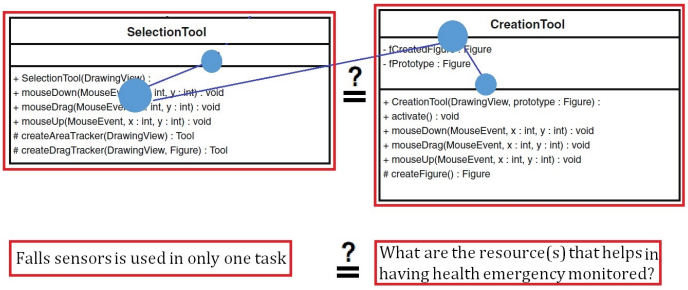


Fig. 5: When comparing two AOIs of different sizes, normalization is required.

extra space, padding, around AOIs to ensure that all relevant fixations are attributed to the appropriate AOIs.

Although participants fixate on specific elements of stimuli, they may not fully perceive these elements. A source of noise in eye-gaze data is incidental fixations. On the one hand, participants can fixate within a 1° visual angle and still encode the information displayed by the stimuli. On the other hand, participants may perceive information with their peripheral vision. Consequently, researchers should define AOIs separated by at least 1°. Different lines of code should be set apart appropriately so AOIs are clearly separated.

Overlapping AOIs. Researchers should not define overlapping or nested AOIs because they complicate data analysis. For example, as illustrated in Figure 4, three fixations are in both the red and orange rectangular AOIs and Tobii Studio, a popular experiment analysis tool, counts the shared fixations twice. In addition, researchers must re-define the concept of transition between AOIs. An AOI should not encompass the entire stimulus because an AOI should capture just one area of interest *in* a stimulus.

Edges of Calibrated Area. Usually, the calibration process includes displaying known points (typically five to nine points) on a screen and mapping their locations with the coordinates of the participants' fixations on these points. Eye trackers typically perform calibration based on both eyes and use the average display location to improve accuracy. If AOIs are located towards the edges of the calibrated area, then error increases because only one eye is used (Goldberg and Helfman, 2010). For participants with only one working eye, single eye calibration is required.

Normalized AOIs. When comparing two AOIs, researchers must normalize the measured value (*e.g.,* fixation duration) based on the sizes of the AOIs to ensure a fair comparison, as illustrated by Figure 5. With graphical stimuli, researchers can divide the measured value by the area of an AOI. With text stimuli, researchers can use the numbers of words or lines in each AOI.

6.7 Ethics Approval

When performing an experiment involving people, researchers must guarantee and preserve the participants' dignity and rights. Universities and–or governments have their own ethical guidelines and codes of conduct, *e.g.,* a Research Ethics Board (REB) or an Institutional Review Board (IRB), that govern the recruitment and the studies.

In software engineering, researchers do not study participants themselves but rather want to understand participants' uses of some systems, artifacts, or tools when performing some tasks. Therefore, with some ethics boards, researchers can apply for an umbrella agreement to perform a set of experiments instead of applying for each study individually.

Researchers must preserve the confidentiality of the participants' data at all times. They must assure participants that their information/data is confidential. They must also assure participants that an eye tracker does not collect any images or videos of the participants. Therefore, they must explain to participants the functioning and output of the eye tracker as well as, if appropriate, the analyses that they will perform on the data. The General Data Protection Regulations of the European Union (GDPR) is a good source of information regarding participants' privacy rights and researchers' responsibilities.

6.8 Discussion of Threats to Validity

We now discuss threats that may influence the validity of the results of an eye tracking study.

Internal Validity. Internal validity relates to the quality of the study. The following biases may jeopardize the internal validity of an eye tracking study.

- **Order effect**: in a within-subjects experimental design, in which each participant works with all conditions, some participant may show better performance in the second task because they practiced on the first. Participants may also perform worse because they are tired. Researchers can minimize the order effect by using a factorial design or randomization using the Latin-square method, with the need for more participants.
- **Instrument bias**: the eye tracker used in a study may change its measurements in time. The use of video-based eye trackers reduces instrument bias because participants can move their heads without decalibration.
- **Hawthorne effect**: researchers must provide guidance to the participants, calibrate the eye tracker, and check the recording. The researcher's presence may bias the data because participants may feel being watched. Researchers should sit inconspicuously away from the participants.
- **Experimenter bias**: researchers may unintentionally influence the participants to achieve certain outcomes. The experimenter bias can be mitigated either by minimizing the interaction between researchers and participants or by implementing a double-blind procedure.

Construct Validity. Researchers should not inform the participants about the precise goals of the study to avoid hypothesis guessing. They should clearly explain to the participants the process of the study, the number and duration of the sessions, and the type of questions before running the experiment.

External Validity. This validity is related to the generalization of the results from the participants to the population as a whole. Researchers must consider individual differences while selecting and assigning participants. Researchers can assign randomly participants to different groups or use stratified sampling. Participants drawn from a population of students reduce the researchers' ability to generalize to a wider population, as discussed in Section 6.3.

Conclusion Validity. Conclusion validity is related to incorrect conclusions about relationships between measures. Researchers analyze the eye-gaze data to find relationships between dependent and independent variables. Calibrating the eye tracker for each participant and using well-documented measures can mitigate this threat. Also, any results must be discussed and if possible, explained using some theories of cognition from psychology.

6.9 Results Presentation

After analyzing the results, researchers must present their results. We suggest to start by explaining the definitions of eye tracking and related concepts, *i.e.,* first order data, fixations, saccades, scan paths, AOIs, and stimuli. Apart from these concepts, few other eye-tracking concepts have well accepted names such as the metrics defined in Section 4. We recommend to avoid excessive eye-tracking jargon, and instead communicate findings in a way that is comprehensible to those outside of the field.

It is also beneficial to use visual representations, such as heat maps and gaze plots, to describe the data. They must be accompanied with proper and complete explanations of the data that they present.

Provide replication packages is crucial to improve external validity Kitchenham (2004). We refer the avid reader to these work for an in-depth discussion and guidelines on the replication of empirical studies in software engineering (Siegmund et al., 2015; Lung et al., 2008; Kitchenham et al., 2002).

No specific standard format exists to provide replication packages for eye-tracking studies. Previous work uses popular hosting services, such as GitHub[2], or team Web sites to offer replication packages[3]. We recommend that replication packages report the following information to facilitate replications:

– Description of the dataset, including: (1) raw and processed eye-tracking data (gaze, events, and pupil dilation), (2) demographic data (age, programming experience, gender, etc.), and (3) responses to the various questionnaires, surveys, and tasks.

[2] `https://github.com/brains-on-code`

[3] `http://www.ptidej.net/downloads/replications/`

– Stimuli, code snippets, and any other artifacts presented before, during, or after the eye-tracking experiment to participants.
– Setup information, including: (1) screen layout, (2) participants' viewing distance, (3) font sizes and font types, and (4) screen size and resolution.
– Data analysis results and scripts, including: (1) eye-tracking metrics used in the study, 2) types and results of the statistical analyses, along with their scripts.

6.10 Combining Eye Tracking with Other Physiological Measures

Over the last 20 years, the software engineering research community has benefited from the use of eye trackers. However, eye trackers are not without limitations and, unlike neuroimaging devices, they do not provide insights into the brain activities, only a proxy to cognitive processes, through the mind–eye hypothesis. As a result, some researchers use eye tracking simultaneously with electroencephalography (EEG) Fritz et al. (2014), fMRI (Peitek et al., 2018b), and fNIRS (Fakhoury et al., 2018).

From a participants perspective, there is almost no extra effort with incorporating eye tracking into EEG, fMRI, or fNIRS studies. The majority of fMRI devices come with built-in eye trackers. Also, eye trackers can be installed in front of a monitor while participants are wearing EEG or fNIRS sensors. Only extra minutes are required to calibrate and validate the eye tracker at the beginning of such studies. Peitek et al. (2018b) showed that adding eye tracking to fMRI studies results in more fine-grained fMRI analyses.

However, adding fMRI or fNIRS as an additional modality to eye-tracking experiments brings many difficulties. fMRI and fNIRS rely on the participants' hemodynamic response, which is a metabolic change (*e.g.,* oxygen, glucose) in neuronal blood flow to active brain regions (Buxton et al., 2004). This response saturates over time, which imposes a stringent limitation on the amount of time that a stimulus is shown to the participants (commonly 30 seconds). They also require robust mathematical analyses to avoid false discovery. Finally, fMRI and fNIRS are expensive per se and their uses are expensive as well, about $500 to $600 per hour for a fMRI.

7 Data Analyses and Interpretation of the Results

Eye trackers typically generate a massive amount of data, so it is important to make sure that the data used for analysis is accurate and reliable. It is our experience that due to the lack of standardized protocols and tools, most of the advanced analyses must be customized and implemented on a case by case basis. This lack causes variations in the analytical approaches and make comparisons across studies difficult.

Most eye tracking software packages do not come with advanced data analysis tools for software engineering tasks. Therefore, researchers must leverage

specialized tools to obtain insights into the software engineers' cognitive processes and intentions from their eye movement data. We strongly recommend that researchers perform a preprocessing step to assess the quality of the data before visualizing and analyzing statistically the data.

7.1 Data Quality Assessment

Good quality eye gaze data is essential for the validity of the research. Eye gaze data contains noise and errors. Holmqvist et al. (2012) discussed the magnitude and the importance of the effect of data quality on eye-tracking study results with examples of accuracy, precision, and data loss.

Although associations, like COGAIN[4], work on standardizing eye-gaze data quality, there are no guidelines for evaluating the quality of eye-tracking data. In the following, we provide our recommendations:

- Replay the eye gaze using the analysis tools provided with the eye tracker. The replay can roughly show when, where, and for how long a participant viewed different parts of the stimuli. Observing eye gaze behaviors can also reveal parts in which no recorded data is available to be displayed on the screen, a situation which could potentially make the dataset sparse.
  Some analysis tools offer automatic evaluation of data quality, for example, in the form of a percentage of the time eyes could be reliably tracked during a trial. In our studies, we considered only trials with at least 70% trackability. However, trackability depends on the length of the recordings and other factors so a replay must be performed to verify where missing data occurred and determine whether the loss could impact the study at hand.
  Several reasons can explain missing data. Participants may have moved and their eyes/heads went out of the range of the eye tracker, which caused a total data loss. Extensive head movements also may lead to decalibration, which may result in offsets or data loss. By replaying the captured eye movements, researchers can visually identify time frames during which data is missing or offset to exclude these frames or correct the offsets.
- Look for offsets in the eye gaze data and apply corrections. Offsets happen when a participant moved beyond the capability of an eye tracker to follow or when decalibration occured. Researchers can use offset-correction algorithms provided by some analysis tools or third-party tools, *e.g.,* Taupe (De Smet et al., 2014), to correct offsets.

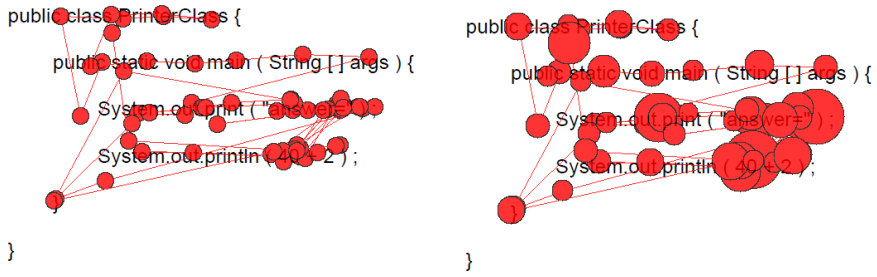
7.2 Visualizations

An eye-tracking experiment generates a large amount of data. Visualizations allow researchers to explore the temporal and spatial characteristics of the eye-tracking data.

[4] `http://www.cogain.org/eye-tracking/`

Blascheck et al. (2017) presented a taxonomy of existing visualizations. This comprehensive overview classifies visualizations based on the granularity of the data (fixation-based or AOI-based) and the representation of data (temporal, spatial, or spatio-temporal).

We now discuss visualizations that have been used by the software engineering community. In addition to presenting a detailed description of these visualizations, we compare and discuss various aspects of these techniques.



(a) Fixed diameter of the fixation circles (b) Varied diameter of the fixation circles

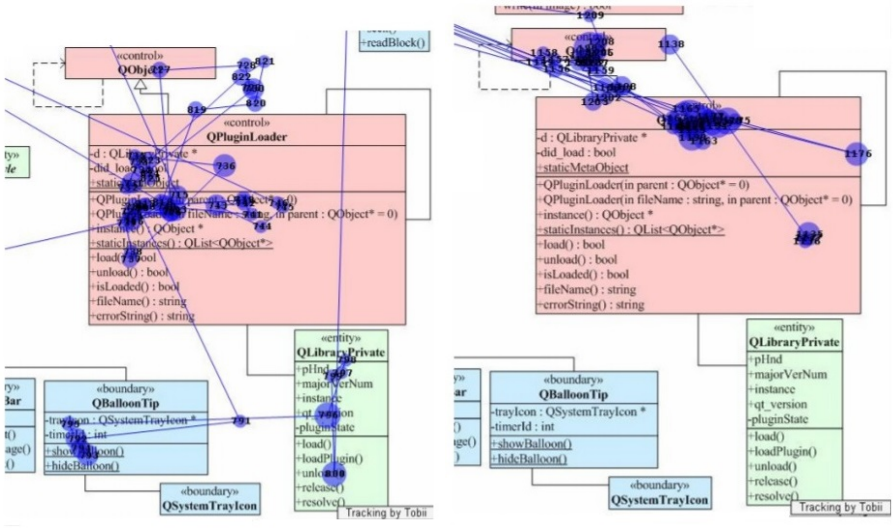
Fig. 6: Examples of gaze plots plotted with EyeCode. (a) uses the same size for all fixation circles, (b) takes fixation duration into account.

Gaze Plots. Gaze plots provide a static view of the eye-gaze data and show the time sequence of looking using the locations, orders, and duration of fixations on stimuli. Each fixation is represented as a circle. Some gaze plots use the same size for all fixation circles. Others take fixation duration into account, correlating the circle's radius with the fixation duration. The longer the fixation, the larger the circle as shown using EyeCode[5] in Figure 6). As shown in Figure 7, Sharif and Maletic (2010b) used gaze plots to compare experts and novices performing design pattern comprehension tasks.

Heat Maps. A heat map is a color spectrum that represents the intensity of a measure, for example, fixations. Heat maps are the most common visualizations in eye tracking studies (Kitchenham, 2004). They show the distribution and focus of visual attention over the stimuli. However, in contrast to gaze plots, they do not provide any information about the order of the fixations.

A heat map is usually superimposed on top of a stimulus to highlight the areas at which participants looked, as illustrated by Figure 9. The colors red, orange, green, and blue indicate the fixation counts or duration from highest to lowest; thus, the longer the observation, the warmer (redder) the color.

[5] `https://github.com/synesthesiam/eyecode`



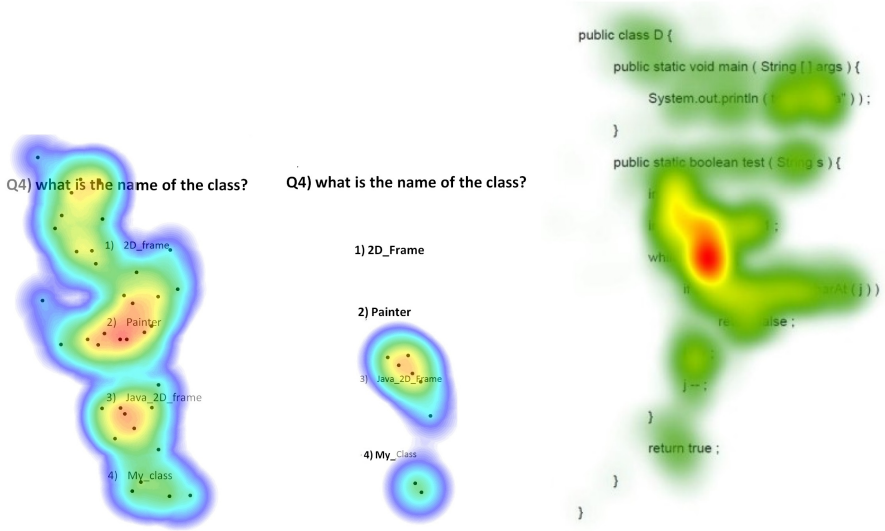
(a) Expert (b) Novice

Fig. 7: Gaze plots on portion of a stimulus comparing an expert and novice for Singleton pattern comprehension task (Sharif and Maletic, 2010b).

(a) Multi-cluster layout (b) Orthogonal layout

Fig. 8: The heat map of a participant working with (a) multi-cluster and (b) orthogonal layouts (Sharif and Maletic, 2010b).

A heat map can be generated based on fixation counts or fixation duration. When using fixation counts, as illustrated by Figure 8, it treats all fixation duration equally, even though fixation duration play an important role in understanding eye tracking data (Henderson and Pierce, 2008). Bojko (2009) presents various types of attention maps while providing guidelines on the usage of heat maps to avoid common misuses and pitfalls.



(a) Male vs. female participants

(b) Attention distribution of various code areas

Fig. 9: (a) A heat map of (Left) a female participant and (Right) a male participant in a study asking each to recall the name of identifiers (Sharafi et al., 2012). (b) Areas of source code that attract higher interests (Busjahn et al., 2011).

Sharif and Maletic (2010b) used heat maps to compare multi-cluster vs. orthogonal layout for design pattern comprehension, as shown in Figure 8. Busjahn et al. (2011) and Ali et al. (2015) used heat maps to illustrate areas in the source code that attract more visual attention, as shown in Figure 9. Sharafi et al. (2012) used heat maps to compare the different viewing strategies deployed by a small number of male and female developers while recalling the names of identifiers. Jbara and Feitelson (2017) compared the attention distribution of average participant for regular code (code with repetitions of the same basic pattern) vs. irregular one, using heat-maps.

Color Coded Attention Allocation Map. A color-coded attention-allocation map is generated for a textual stimulus based on either the fixation counts or duration. It assigns a color to each word separately from a color spectrum between light green (lowest attention level) to light red (highest attention level) (Busjahn et al., 2011), as illustrated in Figure 10. Busjahn et al. (2011) used color-coded attention-allocation maps to identify and study the different parts of source code that attracts different levels of attention (based on the fixation numbers). Ali et al. (2015) used similar maps to identify the parts of texts and source code used in traceability tasks.

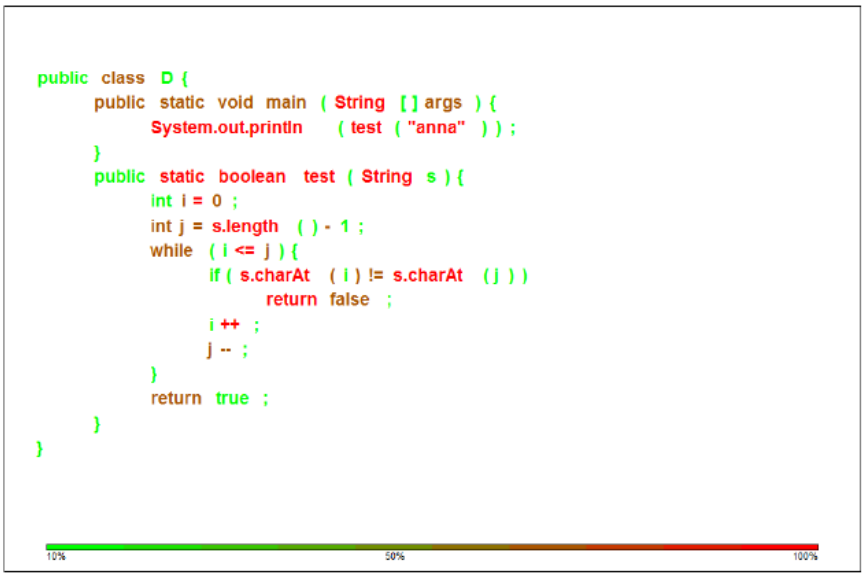


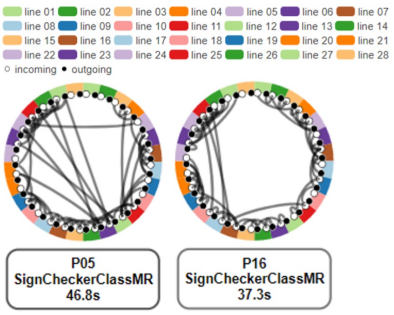
Fig. 10: Color-coded attention allocation map based on the number of fixations per word (Busjahn et al., 2011).

Radial Transition Graph. A radial transition graph is a circular heat map (Blascheck et al., 2017). AOIs are shown on a circle in which the size of each circle segment specifies the total fixation duration within an AOI. The circles are color-coded based on the fixation numbers inside the AOIs while arrows indicate the transitions between AOIs. The thickness of the arrows represents the numbers of transitions between AOIs.

Blascheck and Sharif (2019) used radial transition graphs to compare participants' viewing strategies while reading natural text and source code. Peterson et al. (2019) used radial transition graphs to compare participants' line reading behavior between novice and expert developers. As shown in Figure 11, a novice developer, P16, goes back and forth between lines that are relatively close by. In contrast, an expert developer, P05, transitions between lines of code that are further away.

All these visualizations (and the many others that exist in and are developed by various research communities) have strengths and weaknesses for eye-tracking data. Various factors impact the choice of a visualization.

Generally, independent of a particular task, researchers can use visualizations as a starting point for analyzing eye-tracking data and later as illustrations (Pernice and Nielsen, 2009). They can help identify patterns, trends, and outliers in eye-tracking data. However, they must also use statistical analyses to support their conclusions about the participants' eye gaze data.




```
1.  #include <iostream>
2.  #include <string>
3.  using namespace std;

4.  class SignChecker {
5.  private :
6.          int number ;
7.  public :
8.          SignChecker ( int ) ;
9.          string check ( ) ;
10. };

11. SignChecker::SignChecker ( int currentNumber ) { number = currentNumber ; }

12. string SignChecker::check ( ) {
13.         string theSign = "" ;
14.         if ( number < 0 ) {
15.                 theSign = "negative" ;
16.         } else if ( number > 0 ) {
17.                 theSign = "positive" ;
18.         } else {
19.                 theSign = "null" ;
20.         }
21.         return theSign ;
22. }

23. int main ( ) {
24.     SignChecker number1 ( 10 ) ;
25.     cout << "Actual: " << number1.check ( ) << "Expected: positive" << endl ;
26.     SignChecker number2 ( 0 ) ;
27.     cout << "Actual: " << number2.check ( ) << "Expected: null" << endl ;
28.     return 0 ;
29. }
```

Fig. 11: **Left**: Radial transition graphs for P05, an expert, and P16, a novice. **Right**: `SignCheckerClassMR`, which returns the sign of the input integer (Peterson et al., 2019).

Specifically, the task at hand is a key factor to identify which visualizations work best. Gaze plots are useful for scan paths comparisons to identify and compare some participants' viewing strategies. Heat maps work best for comparing some participants' attention distributions and identifying whether the fixated areas either are difficult to process or contain relevant content.

However, heat maps and gaze plots must be used with care (Holmqvist et al., 2011). Pernice and Nielsen (2009) showed that different factors, including tasks, motivation, familiarity with the stimuli, affect how heat maps and gaze plots are drawn. Thus, researchers must consider these mitigating factors when comparing different heat maps and gaze plots to draw conclusions.

We recommend using a combination of visualizations to analyze the eye-tracking data. For example, researchers can use heat maps and gaze together to identify participants' viewing trends through the distribution of visual attention along with the visual paths. Then, further statistical analyses of eye-tracking metrics would support or refute these findings.

Classic attention maps, such as heat map and gaze plot, only take into account the spatial aspect of the eye gaze data and ignore its temporal distribution. Thus, they have been mainly used by the research community for static stimuli, *e.g.,* images. However, as eye-tracking research advances, reseachers proposed new visualizations for dynamic stimuli, *e.g.,* videos, by adjusting fixation data based on a moving object (Blascheck et al., 2017).

The number of AOI-based visualizations is limited, especially if researchers are interested in spatio-temporal analyses. Radial transition graphs became popular recently and are being adopted by the research community. However, they only support small numbers of AOIs. In general, scalability is the main

limitation of AOI-based visualizations and there is a need for new visualizations to overcome this limitation (Blascheck et al., 2017). Achieving scalability in eye tracking visualization on large systems is a non-trivial problem which visualization researchers are working on (Blascheck et al., 2017). Various types of filters and focus plus context approaches are needed.

7.3 Statistical Analyses

Table 6: Summary of the statistical tests used for eye-tracking data analysis. (B) and (W) are for between- and within-subjects designs, respectively. The distribution (Dist.) of the data is either normal (N) or non-normal (NN).

Purpose		**Dist.**	**Factor (Level)**	**Statistical Test**
To confirm a theory		N	1 (2)	Student's t-test (B)
			1 (2)	Paired samples t-test (B)
			1 (>2)	F-statistics (One-way ANOVA) (W)
			1 (>2)	One-way repeated-measure ANOVA (W)
		NN	1 (2)	Mann-Whitney U (B)
			1 (2)	KruskalWallis H (B)
			1 (>2)	Wilcoxon signed-rank (W)
			1 (>2)	Friedman test (W)
To quantify an association	**Measure of association**	N	1 (2)	Spearman's correlation coefficient
		NN	1 (2)	χ^2 test
	Equation of association	N	>=1 (>=2)	Linear models (W) Factorial ANOVA (B)
			>=1 (>=2)	Linear mixed model (W) Factorial repeated-measure ANOVA (W)
		NN	>=1 (>=2)	Aligned Rank Test (ART) (B)
			>=1 (>=2)	ART or Generalized linear mixed model (W)

This section discusses statistical analyses applicable to eye-tracking data and how to choose appropriate ones.

The first step in analyzing eye-tracking data is to explore the data and to search for any relationship between two or more variables. Box plots, histograms, and scatter plots are statistical graphs that researchers can use to analyze eye-tracking data at first. Although there were not developed for eye-tracking data, they provide valuable, visual information about such data.

A box plot shows the distribution of some data using their minimum, first quartile, median, third quartile, and maximum. It can be used by researchers to identify outliers and compare distributions. Sharafi et al. (2012) provided box plots for three effort metrics: fixation counts, fixation rates, and fixation counts on relevant and irrelevant AOIs to compare male and female participants.

A scatter plot displays the correlation between two set of data. It also shows a line of best fit (regression line) highlighting the spread or dispersion of the

data (closeness to the line). Sharafi et al. (2012) used scatter plots to visualize efficiency and accuracy trade offs between male and female participants.

After qualitatively examining the data using some statistical plots, researchers must perform quantitative analyses to confirm their theory and–or to quantify associations between sets of data. They must consider different factors when choosing statistical tests, including the type and distribution of the data, the purpose of an experiment, and the experimental design (Pfleeger, 1995). Table 6 presents a set of statistical tests and the factors impacting their usages, *e.g.,* distribution of the data.

To find a valid statistical test, researchers must consider the normality of the distribution of the data. Researchers can ascertain the normality of some data using various statistical tests, including the Shapiro–Wilk test or Pearson's χ^2 test. Wohlin et al. (2012) explains in great detail, in the context of software engineering research, how to study data distributions. We recommend to adhere to their guidelines. If the data is normally distributed, then parametric tests can be used, which are more "powerful" than their non-parametric counterparts. Non-parametric tests are usually appropriate only with small sample sizes (Kitchenham et al., 2002), which are common in eye-tracking studies (Sharafi et al., 2015b).

The type of experimental design is also a factor in choosing a valid test. In between-subjects (or between-groups) study designs, different participants are being exposed to each levels of some treatments. In within-subjects (or repeated-measures) study designs, the same participants test all the treatments. For example, Sharif and Maletic (2010b) investigated the effectiveness of three different UML layout techniques on comprehension in a within-subjects design in which all participants worked with all three layouts.

Then, researchers must use the tests that help them with their studies: either to confirm a theory or to quantify an association between two data sets.

To Confirm a Theory. The goal of an experiment may be to validate the truth of a theory, *e.g.,* to investigate the impact of a technique on participants' effectiveness. It is usually formally expressed using a set of hypotheses and researchers can use a test to verify their hypotheses and confirm their theory:

1. *The data is normally distributed.* If the comparison is between two treatments, the Student's t-test is appropriate (Wohlin et al., 2012). For example, Bednarik (2012) compared experts and novices looking for bugs. Their null hypothesis was the lack of statistically significant difference between novices and experts regarding the number of bugs found. They used the t-test and concluded that experts found more bugs than novices.
   If there are more than two treatments, then ANalysis Of VAriance (ANOVA) is appropriate to determine whether the means of three or more groups are different. ANOVA uses a F-test to assess the equality of means. An F-test returns a F-statistics, its related degrees of freedom, and its p-value, used to reject the null hypothesis. For example, Stein and Brennan (2004) divided their participants into two groups, recorded the eye gaze of those

in the first group, and displayed the records to those in the other group. They analyzed the impact of watching other participants' eye gazes on the performance of the second group using ANOVA. They concluded that a participant's eye gaze provides useful cues to other participants and improves their performance. (Linear models are increasingly used as a robust alternative to ANOVA.)

2. *The data is not normally distributed.* If the comparison is between two treatments, the Mann-Whitney U or Wilcoxon signed-rank tests are appropriate. Sharafi et al. (2012) compared male and female developers' accuracy and efficiency in time when performing a program comprehension task. They applied the Wilcoxon test separately on accuracy and time data and reported that there was no statistically significant difference between male and female developers.
   If there are more than two treatments, then the Kruskal–Wallis test (One-Way ANOVA) (Wohlin et al., 2012) can tell whether the data originated from a same distribution. For example, Ali et al. (2015) computed the sum of the fixation duration that participants applied to specific source-code entities (SCEs), *e.g.,* class names. They used Kruskal–Wallis test to assess whether participants had equal preferences for the different SCEs.

To Quantify Associations between Two Sets of Data. Researchers can use a correlation analysis to find and quantify the association between two sets of data, for example to analyze the impact of a factor X on participants' characteristics Y. A correlation analysis is obtained either by generating a measure of association or by providing an equation that describes the association. Linear regression can also be used to understand the type of the association. Cagiltay et al. (2013) used Pearson correlation coefficients to investigate the relationship between defect difficulty levels and fixation duration. First, they computed the defect difficulty levels for all participants based on the time that they took to find the defects and the order in which they found the defects. Second, they assessed the normality of the data. Finally, they applied Pearson's correlation coefficients to identify a significant correlation between the two variables: a longer mean fixation duration is associated with a higher defect difficulty level.

7.4 Tools

Various tools are available for analysing eye tracking data. Commercial eye trackers suppliers sells, *e.g.,* FaceLAB and Tobii are selling their own data analysis tools. In addition, there are free-of-charge tools that offer some capabilities required and–or work with a subset of commercial eye trackers. Farnsworth (2019b) also listed ten eye-tracking software tools, along with their functions and accessibility. The majority of these tools support testing of stimuli through non-commercial Webcam-based eye trackers.

Based on our previous experience with commercial, high-precision eye trackers, we feature a non-exhaustive list of six open-source data analysis tools. These tools work well with the commercial eye trackers that have been frequently used by software engineering researchers. Table 7 captures the essentials of the eye-tracking analysis tools discussed below.

Table 7: A rundown of capabilities of eye tracking analysis tools. ● = Provides capability, ◐ = Partially provided capability, ○ = Does not provide capability.

Capabilities	Tools					
	Ogama	Taupe	iTrace	EyeCode	PandasEye	PyGaze
AOI Analysis	●	●	●	●	○	●
Plots	●	●	●	●	○	●
Metrics	●	●	○	●	○	●
ML Analysis	○	○	○	○	○	●
Realtime Recording	●	○	●	○	○	○
Support Scrolling	○	○	●	○	○	○
Programming Required	○	○	○	●	●	●
Ongoing Support	○	●	●	●	●	●
Hardware Compatibility	●	◐	◐	●	●	●
Multi-input Integration	●	○	○	○	○	●
Open source	●	●	●	●	●	●

Ogama[6] is an open-source software, published under GPL license that allows simultaneous recording and analyzing eye-gaze and mouse-tracking data. It supports filtering of eye gaze and mouse data, creating attention maps, defining and modifying AOIs, calculation of saliency and Levenshtein distance. In addition, the replay of the recording is available while recording is also possible with various commercial and open-source eye trackers.

Taupe[7] stands for Thoroughly Analyzing the Understanding of Programs through Eyesight. Published under GPL license, It was introduced by Guéhéneuc (2006b) and extended by De Smet et al. (2014) as an open-source software system designed for analyzing eye-tracking data. Taupe supports the data coming from various commercial eye trackers, including FaceLAB and Tobii eye trackers. A set of well-known software engineering practices, such as design patterns and a plug-in architecture, were used to make Taupe extensible by developers.

After importing eye-movement data, *Taupe* can calculate various fixation statistics including fixation count, fixation count per AOI, fixation duration, normalized fixation count per AOI, fixation rate, ROAF, and spatial density. Also, Taupe supports calculating saccade statistics (*e.g.,* Transition Density), convex hull area, scan paths and their comparisons, accuracy, and duration.

[6] `http://www.ogama.net/`

[7] `http://www.ptidej.net/tools/programcomprehension/`

Moreover, *Taupe* comes with an AOI creation tool allowing users to draw polygons around a set of fixations superimposed on top of a stimulus. Each AOI has a unique ID used by Taupe to calculate AOI-based metrics.

iTrace[8] is an eye-tracking infrastructure designed for experiments on large software artifacts, including source code, bug reports, and requirement documents (Guarnera et al., 2018; Shaffer et al., 2015). Current eye-tracking systems only support static stimuli fixed in place, while *iTrace* allows participants to interact with source code and other artifacts naturally, supporting scrolling in and switching between files. It comes with three plugins to support Visual Studio, Eclipse, and Google Chrome, which are open source under the GPL license.

Two main components[9] of *iTrace* are *Gaze2Src* and *iTraceVis*. *Gaze2Src* processes *iTrace* gaze data offline after the recording. This post-processing tool supports three fixation algorithms, including the basic fixations, based on a method proposed by Olsson (2007), velocity-based fixations (I-VT), and dispersion-based fixations (I-DT). All these fixation algorithms can be adjusted based on the tasks and the stimuli. *Gaze2Src* maps fixations to syntactic tokens of the source code using srcML[10]. Currently, Gaze2Src supports programs written in C, C++, C#, and Java. A submodule of iTrace named iTrace-Toolkit plans to support high-speed trackers along with support for code editing (which is a non-trivial feature to add).

iTraceVis (Clark and Sharif, 2017) supports the visualization of large-scale eye-tracking data in the presence of scrolling and switching between files. It currently supports four types of visualizations: heat maps, gaze skylines, static gaze-maps, and dynamic gaze-maps.

EyeCode[11] (Hansen, 2014) is a Python library for analyzing and visualizing eye tracking data. It is built on top of the of the pandas statistical computing library. It contains specialized functions for processing eye gaze data, creating AOIs, calculating various fixation and saccade metrics (including those based on AOIs), and displaying the data and metrics. The plugin is open source under the GPL license and it is freely available.

PandasEye[12] (Vrzakova, 2019) is a collection of Python libraries for advanced analysis of raw eye tracking data for the purpose of machine learning experiments. The tool has been previously employed for intention detection and multimodal affect recognition, and includes all primary building blocks of a machine learning pipeline for eye tracking data.

PyGaze[13] (Dalmaijer et al., 2014) is an open source package under the GPL license for creating eye-tracking experiments in Python syntax. It supports both visual and auditory stimulus presentation. It collects data from

[8] `http://www.i-trace.org/`

[9] `http://www.i-trace.org/features/`

[10] `http://www.srcml.org/#home`

[11] `http://github.com/synesthesiam/eyecode`

[12] `http://github.com/hanav/PandasEye`

[13] `http://www.pygaze.org/`

various inputs including keyboard, mouse, joystick, and etc. It works with many commercial eye trackers (EyeLink, SMI, and Tobii systems).

8 Conclusions

Eye tracking provides invaluable insights in experimental studies in software engineering by collecting participants' eye-gaze data on visual stimuli. They provide insights to researchers that are not possible to obtain with questionnaires or surveys. However, they are not without shortcomings and they present practical and ethical difficulties.

Based on our collective experience using eye trackers and a previous, systematic literature-review of eye-tracking studies (Sharafi et al., 2015b), we presented the history and technological evolution of eye trackers. We discussed why, when, and how it is appropriate to use eye trackers in software engineering research. We also provided practical suggestions on conducting eye-tracking studies. With this paper, we help software engineering researchers plan, design, and conduct experimental studies with eye trackers.

Eye trackers now represent a middle ground for studies on human factors in software development activities for several reasons. They are a valuable tool that can provide additional insights into participants' cognitive processes than surveys or questionnaires can. They are also cheaper, less invasive, and less complex than fMRI.

First, eye trackers are now inexpensive and convenient to use. Nowadays, webcam-based eye trackers are a good compromise between price and data quality. Most eye trackers come with their own analysis tools, which are usually well-supported. Third-party tools are also available.

Second, all models of eye trackers provide invaluable data to illustrate participants' cognitive processes, following the mind-eye hypothesis. Such data can not be collected easily through surveys, or at cheaper cost with fMRI.

Third, eye trackers fit in well to software engineering research because almost all software engineering activities, from requirements analysis to performance profiling, make use of visually-oriented artifacts, *e.g.,* deployment diagrams or debugging tools.

Fourth, eye trackers allow systematic studies of software activities and artifacts while participants perform representative tasks, albeit in controlled environments. These studies could improve both the quality of the artifacts and the quality of the developers' interactions with these artifacts.

Fifth, however, eye trackers must be carefully used for their data to be correct and relevant. They also provide a wealth of data that must be carefully collected, stored, and analysed to provide valid conclusions.

Following this paper, we suggest four main directions of future work.

1. More eye-tracking studies should be performed in ecologically valid setups to bring in-depth understanding of various software maintenance activities

and to help make developers more effective. These studies should be performed in a variety of settings and with different sets of participants to reduce validity threats.
2. More research on the uses of eye trackers should be performed to improve the tools that support software developers. In addition to the availability of cheaper and more precise eye trackers for research, eye trackers could be embedded directly into the software developers' workstations to provide timely interventions based on their visual attention.
3. More research should incorporate advances in eye trackers to refine the data collected by and analyses performed with eye trackers to further understand the participants' cognitive processes, using metrics, such as pupil dilatation or virtual reality glasses.
4. Suitable tools should be developed to ease the simultaneous recording and analysis of eye tracking and neuroimaging (*i.e.,* fMRI and fNIRS) data. This simultaneous measurement of software engineering tasks is challenging, but promising (Peitek et al., 2018b). The high cost, restrictive environment, and high rate of data loss due to participant motion of fMRI impose limits on the practicality for a broad spectrum of use cases.

Acknowledgements The authors would like to thank the anonymous reviewers for their insightful comments and suggestions. This work has been partly funded by the US NSF under Grant Numbers CCF 18-55756 and CCF 15-53573, as well as the NSERC Discovery Grant program and the Canada Research Chair in Software Patterns and Patterns of Software.

References

Abid NJ, Maletic JI, Sharif B (2019a) Using developer eye movements to externalize the mental model used in code summarization tasks. In: Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '19, pp 13:1–13:9, DOI 10.1145/3314111.3319834, URL http://doi.acm.org/10.1145/3314111.3319834

Abid NJ, Sharif B, Dragan N, Alrasheed H, Maletic JI (2019b) Developer reading behavior while summarizing java methods : Size and context matters. In: Proceedings of the 41th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE 2019, p To Appear

Ali N, Sharafi Z, Guéhéneuc YG, Antoniol G (2015) An empirical study on the importance of source code entities for requirements traceability. Empirical Software Engineering 20(2):442–478

Alkan S, Cagiltay K (2007) Studying computer game learning experience through eye tracking. British Journal of Educational Technology 38(3):538–542

Armaly A, Rodeghero P, McMillan C (2018) Audiohighlight: Code skimming for blind programmers. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp 206–216

Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential pattern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD

International Conference on Knowledge Discovery and Data Mining, ACM, New York, NY, USA, KDD '02, pp 429–435, DOI 10.1145/775047.775109, URL http://doi.acm.org/10.1145/775047.775109

Barik T, Smith J, Lubick K, Holmes E, Feng J, Murphy-Hill E, Parnin C (2017) Do developers read compiler error messages? In: Proceedings of the 39th International Conference on Software Engineering, IEEE Press, Piscataway, NJ, USA, ICSE '17, pp 575–585, DOI 10.1109/ICSE.2017.59, URL https://doi.org/10.1109/ICSE.2017.59

Beatty J (1982) Task-evoked pupillary responses, processing load, and the structure of processing resources. Psychological bulletin 91(2):276

Bednarik R (2007) Methods to analyze visual attention strategies: Applications in the studies of programming. University of Joensuu

Bednarik R (2012) Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. International Journal of Human-Computer Studies 70(2):143–155, DOI 10.1016/j.ijhcs.2011.09.003

Bednarik R, Tukiainen M (2005) Effects of display blurring on the behavior of novices and experts during program debugging. In: CHI '05 Extended Abstracts on Human Factors in Computing Systems, ACM, New York, NY, USA, CHI EA '05, pp 1204–1207, DOI 10.1145/1056808.1056877, URL http://doi.acm.org/10.1145/1056808.1056877

Bednarik R, Tukiainen M (2006) An eye-tracking methodology for characterizing program comprehension processes. In: Proceedings of the 2006 Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '06, pp 125–132

Bednarik R, Eivazi S, Hradis M (2012) Gaze and conversational engagement in multiparty video conversation: an annotation scheme and classification of high and low levels of engagement. In: Proceedings of the 4th workshop on eye gaze in intelligent human machine interaction, ACM, p 10

Begel A, Vrzakova H (2018) Eye movements in code review. In: Proceedings of the Workshop on Eye Movements in Programming, ACM, New York, NY, USA, EMIP '18, pp 5:1–5:5, DOI 10.1145/3216723.3216727, URL http://doi.acm.org/10.1145/3216723.3216727

Bergstrom JR, Schall A (2014) Eye tracking in user experience design. Elsevier

Binkley D, Davis M, Lawrie D, Maletic JI, Morrell C, Sharif B (2013) The impact of identifier style on effort and comprehension. Empirical Software Engineering 18(2):219–276, DOI 10.1007/s10664-012-9201-4

Blascheck T, Sharif B (2019) Visually analyzing eye movements on natural language texts and source code snippets. In: ETRA 2019-ACM Symposium on Eye Tracking Research & Applications

Blascheck T, Kurzhals K, Raschke M, Burch M, Weiskopf D, Ertl T (2017) Visualization of eye tracking data: A taxonomy and survey. Computer Graphics Forum 36(8):260–284

Bojko A (2005) Eye tracking in user experience testing: How to make the most of it. In: Proceedings of the UPA 2005 Conference

Bojko AA (2009) Informative or misleading? heatmaps deconstructed. In: Proceedings of the 13th International Conference on Human-Computer Interaction. Part I: New Trends, Springer-Verlag, Berlin, Heidelberg, pp 30–39, DOI 10.1007/978-3-642-02574-7_4, URL http://dx.doi.org.proxy.lib.umich.edu/10.1007/978-3-642-02574-7_4

Buse RPL, Sadowski C, Weimer W (2011) Benefits and barriers of user evaluation in software engineering research. In: Object-Oriented Programming, Systems, Languages and Applications, pp 643–656

Busjahn T, Schulte C, Busjahn A (2011) Analysis of code reading to gain more insight in program comprehension. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research, ACM, New York, NY, USA, Koli Calling '11, pp 1–9, DOI 10.1145/2094131.2094133

Busjahn T, Schulte C, Sharif B, Simon, Begel A, Hansen M, Bednarik R, Orlov P, Ihantola P, Shchekotova G, Antropova M (2014) Eye tracking in computing education. In: Proceedings of the Tenth Annual Conference on International Computing Education Research, ACM, New York, NY, USA, ICER '14, pp 3–10, DOI 10.1145/2632320.2632344, URL http://doi.acm.org/10.1145/2632320.2632344

Busjahn T, Bednarik R, Begel A, Crosby M, Paterson JH, Schulte C, Sharif B, Tamm S (2015) Eye movements in code reading: Relaxing the linear order. In: Proceedings of 22th International Conference on Program Comprehension, ICPC '15

Buxton RB, Uludağ K, Dubowitz DJ, Liu TT (2004) Modeling the hemodynamic response to brain activation. Neuroimage 23:S220–S233

Cagiltay NE, Tokdemir G, Kilic O, Topalli D (2013) Performing and analyzing non-formal inspections of entity relationship diagram (erd). Journal of Systems and Software 86(8):2184–2195, DOI 10.1016/j.jss.2013.03.106

Cepeda G, Guéhéneuc YG (2010) An empirical study on the efficiency of different design pattern representations in uml class diagrams. Empirical Software Engineering 15(5):493–522, DOI 10.1007/s10664-009-9125-9

Clark B, Sharif B (2017) itracevis: Visualizing eye movement data within Eclipse. In: Working Conference on Software Visualization (VISSOFT), IEEE, pp 22–32

Cristino F, Mathot S, Theeuwes J, Gilchrist ID (2010) Scanmatch: A novel method for comparing fixation sequences. Behaviour Research Method 42:692–700

Crosby ME, Stelovsky J (1990) How do we read algorithms? a case study. Computer 23(1):24–35

Crosby ME, Scholtz J, Wiedenbeck S (2002) The roles beacons play in comprehension for novice and expert programmers. In: Proceeding of Programmers, 14th Workshop of the Psychology of Programming Interest Group, Brunel University, pp 18–21

Dalmaijer ES, Mathôt S, Van der Stigchel S (2014) Pygaze: An open-source, cross-platform toolbox for minimal-effort programming of eyetracking experiments. Behavior research methods 46(4):913–921

De Smet B, Lempereur L, Sharafi Z, Guéhéneuc YG, Antoniol G, Habra N (2014) Taupe: Visualizing and analyzing eye-tracking data. Science of Computer Programming 79:260–278, DOI 10.1016/j.scico.2012.01.004, URL http://dx.doi.org/10.1016/j.scico.2012.01.004

Divjak M, Bischof H (2008) Real-time video-based eye blink analysis for detection of low blink-rate during computer use. In: First International Workshop on Tracking Humans for the Evaluation of their Motion in Image Sequences (THEMIS 2008), pp 99–107

Duchowski AT (2002) A breadth-first survey of eye-tracking applications. Behavior Research Methods, Instruments, & Computers 34(4):455–470

Duchowski AT (2007) Eye tracking methodology: Theory and practice. Springer-Verlag New York Inc

Fakhoury S, Ma Y, Arnaoudova V, Adesope O (2018) The effect of poor source code lexicon and readability on developers' cognitive load. In: Proceedings of the 26th Conference on Program Comprehension, ACM, New York, NY, USA, ICPC '18, pp 286–296, DOI 10.1145/3196321.3196347, URL http://doi.acm.org/10.1145/3196321.3196347

Farnsworth B (2019a) 10 Free Eye Tracking Software Programs [Pros and Cons]. https://imotions.com/blog/free-eye-tracking-software/, [Online; accessed 30-December-2019]

Farnsworth B (2019b) 10 Free Eye Tracking Software Programs [Pros and Cons]. https://imotions.com/blog/free-eye-tracking-software/, [Online; accessed 30-December-2019]

Fritz T, Begel A, Müller SC, Yigit-Elliott S, Züger M (2014) Using psychophysiological measures to assess task difficulty in software development. In: Proceedings of the 36th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE '14, pp 402–413

Godfroid A (2019) Eye tracking in second language acquisition and bilingualism: A research synthesis and methodological guide. Routledge

Goldberg JH, Helfman JI (2010) Comparing information graphics: A critical look at eye tracking. In: Proceedings of the 3rd BELIV'10 Workshop: BEyond Time and Errors: Novel evaLuation Methods for Information Visualization, ACM, New York, NY, USA, BELIV '10, pp 71–78, DOI 10.1145/2110192.2110203, URL http://doi.acm.org/10.1145/2110192.2110203

Goldberg JH, Kotval XP (1999) Computer interface evaluation using eye movements: methods and constructs. International Journal of Industrial Ergonomics 24(6):631–645

Goldberg JH, Stimson MJ, Lewenstein M, Scott N, Wichansky AM (2002) Eye tracking in web search tasks: Design implications. In: Proceedings of the 2002 Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '02, pp 51–58, DOI 10.1145/507072.507082, URL http://doi.acm.org/10.1145/507072.507082

Grace R, Byrne VE, Bierman DM, Legrand JM, Gricourt D, Davis RK, Staszewski JJ, Carnahan B (1998) A drowsy driver detection system for heavy vehicles. In: Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE, IEEE, vol 2, pp I36–1

Guarnera DT, Bryant CA, Mishra A, Maletic JI, Sharif B (2018) itrace: eye tracking infrastructure for development environments. In: Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications, ACM, p 105

Guéhéneuc YG (2006a) Taupe: Towards understanding program comprehension. In: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp., Riverton, NJ, USA, CASCON '06, DOI 10.1145/1188966.1188968, URL http://dx.doi.org/10.1145/1188966.1188968

Guéhéneuc YG (2006b) Taupe: Towards understanding program comprehension. In: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research, IBM Corp., Riverton, NJ, USA, CASCON '06

Haiduc S, Aponte J, Moreno L, Marcus A (2010) On the use of automated text summarization techniques for summarizing source code. In: 2010 17th Working Conference on Reverse Engineering, IEEE, pp 35–44

Haji-Abolhassani A, Clark JJ (2014) An inverse yarbus process: Predicting observers task from eye movement patterns. Vision research 103:127–142

Hansen DW, Ji Q (2009) In the eye of the beholder: A survey of models for eyes and gaze. IEEE transactions on pattern analysis and machine intelligence 32(3):478–500

Hansen M (2014) eyecode: An eye-tracking experimental framework for program comprehension. PhD thesis, School of Informatics and Computing, 2719 E. 10th Street Bloomington, IN 47408 USA

Hartridge H, Thomson L (1948) Methods of investigating eye movements. The British journal of ophthalmology 32(9):581

Hejmady P, Narayanan NH (2012) Visual attention patterns during program debugging with an IDE. In: Proceedings of the 2012 Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '12, pp 197–200, DOI 10.1145/2168556.2168592, URL http://doi.acm.org/10.1145/2168556.2168592

Henderson JM, Pierce GL (2008) Eye movements during scene viewing: Evidence for mixed control of fixation durations. Psychonomic Bulletin & Review 15(3):566–573

Holmqvist K, Nyström M, Andersson R, Dewhurst R, Jarodzka H, Van de Weijer J (2011) Eye tracking: A comprehensive guide to methods and measures. OUP Oxford

Holmqvist K, Nyström M, Mulvey F (2012) Eye tracker data quality: what it is and how to measure it. In: Proceedings of the symposium on eye tracking research and applications, ACM, pp 45–52

Huey EB (1908) The psychology and pedagogy of reading. The Macmillan Company

Jacob RJ, Karn KS (2003) Eye tracking in human-computer interaction and usability research: Ready to deliver the promises. Mind 2(3):4

Jbara A, Feitelson DG (2017) How programmers read regular code: a controlled experiment using eye tracking. Empirical software engineering

22(3):1440–1477
Jeanmart S, Guéhéneuc YG, Sahraoui HA, Habra N (2009) Impact of the visitor pattern on program comprehension and maintenance. In: Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement, pp 69–78
Just MA, Carpenter PA (1980) A theory of reading: from eye fixations to comprehension. Psychological review 87(4):329
Karn KS, Ellis S, Juliano C (1999) The hunt for usability: tracking eye movements. In: CHI'99 extended abstracts on Human factors in computing systems, ACM, pp 173–173
Kitchenham BA (2004) Procedures for undertaking systematic reviews. Tech. rep., Joint Technical Report, Computer Science Department, Keele University (TR/SE- 0401) and National ICT Australia Ltd
Kitchenham BA, Pfleeger SL, Pickard LM, Jones PW, Hoaglin DC, Emam KE, Rosenberg J (2002) Preliminary guidelines for empirical research in software engineering. IEEE Trans Softw Eng 28(8):721–734, DOI 10.1109/TSE.2002.1027796
Ko AJ, Latoza TD, Burnett MM (2015) A practical guide to controlled experiments of software engineering tools with human participants. Empirical Softw Engg 20(1):110–141, DOI 10.1007/s10664-013-9279-3, URL http://dx.doi.org/10.1007/s10664-013-9279-3
Lee S, Hooshyar D, Ji H, Nam K, Lim H (2018) Mining biometric data to predict programmer expertise and task difficulty. Cluster Computing 21(1):1097–1107
Levenshtein V (1966) Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady 10:707
Lung J, Aranda J, Easterbrook SM, Wilson GV (2008) On the difficulty of replicating human subjects studies in software engineering. In: Proceedings of the 30th international conference on Software engineering, ACM, pp 191–200
Mackworth NH, Thomas EL (1962) Head-mounted eye-marker camera. JOSA 52(6):713–716
McChesney I, Bond R (2019) Eye tracking analysis of computer program comprehension in programmers with dyslexia. Empirical Softw Engg 24(3):1109–1154, DOI 10.1007/s10664-018-9649-y, URL https://doi.org/10.1007/s10664-018-9649-y
Murphy GC, Kersten M, Findlater L (2006) How are java software developers using the eclipse ide? IEEE Software 23(4):76–83, DOI 10.1109/MS.2006.105, URL http://dx.doi.org/10.1109/MS.2006.105
Obaidellah U, Al Haek M, Cheng PCH (2018) A survey on the usage of eye-tracking in computer programming. ACM Comput Surv 51(1):5:1–5:58, DOI 10.1145/3145904, URL http://doi.acm.org/10.1145/3145904
Olsson P (2007) Real-time and offline filters for eye tracking
Orlov PA, Bednarik R (2017) The role of extrafoveal vision in source code comprehension. Perception 46(5):541–565

Peitek N, Siegmund J, Apel S, Kstner C, Parnin C, Bethmann A, Leich T, Saake G, Brechmann A (2018a) A look into programmers' heads. IEEE Transactions on Software Engineering pp 1–1

Peitek N, Siegmund J, Parnin C, Apel S, Hofmeister J, Brechmann A (2018b) Simultaneous Measurement of Program Comprehension with fMRI and Eye Tracking: A Case Study. In: Symposium on Empirical Software Engineering and Measurement, to appear.

Pernice K, Nielsen J (2009) Eyetracking methodology: How to conduct and evaluate usability studies using eyetracking. Nielsen Norman Group Technical Report

Peterson C, Saddler J, Blascheck T, Sharif B (2019) Visually analyzing students' gaze on c++ code snippets. In: EMIP 2019-6th International Workshop on Eye Movements in Programming

Petrusel R, Mendling J (2012) Eye-tracking the factors of process model comprehension tasks. In: Proceedings of the Conference on the Advanced Information Systems Engineering, Springer, CAiSE '13, pp 224–239

Pfleeger SL (1995) Experimental design and analysis in software engineering, part 5: Analyzing the data. SIGSOFT Softw Eng Notes 20(5):14–17, DOI 10.1145/217030.217032, URL http://doi.acm.org/10.1145/217030.217032

Poole A, Ball LJ (2005) Eye tracking in human-computer interaction and usability research: Current status and future. In: Prospects, Chapter in C. Ghaoui (Ed.): Encyclopedia of Human-Computer Interaction. Pennsylvania: Idea Group, Inc

Privitera CM, Stark LW (2000) Algorithms for defining visual regions-of-interest: Comparison with eye fixations. IEEE Transactions on Pattern Analysis and Machine Intelligence 22:970–982

Rayner K (1978) Eye movements in reading and information processing. Psychological Bulletin 85(3):618–660

Rayner K (1998) Eye movements in reading and information processing: 20 years of research. Psychological bulletin 124(3):372

Rodeghero P, McMillan C, McBurney PW, Bosch N, D'Mello S (2014) Improving automated source code summarization via an eye-tracking study of programmers. In: Proceedings of the 36th International Conference on Software Engineering, ACM, New York, NY, USA, ICSE 2014, pp 390–401, DOI 10.1145/2568225.2568247, URL http://doi.acm.org/10.1145/2568225.2568247

Ross J (2009) Eyetracking: Is It Worth It? http://www.uxmatters.com/mt/archives/2009/10/eyetracking-is-it-worth-it.php/, [Online; accessed 20-March-2019]

Sajaniemi J (2004) Comparison of three eye tracking devices in psychology of programming research. In: Proceedings of the 16th Annual Psychology of Programming Interest Group Workshop, PPIG '04, pp 151–158

Salvucci DD, Goldberg JH (2000) Identifying fixations and saccades in eye-tracking protocols. In: Proceedings of the 2000 Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '00, pp 71–78, DOI 10.1145/355017.355028, URL http://doi.acm.org/10.1145/

355017.355028
Shackel B (1960) Note on mobile eye viewpoint recording. JOSA 50(8):763–768
Shaffer TR, Wise JL, Walters BM, Müller SC, Falcone M, Sharif B (2015) itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ACM, pp 954–957
Shapiro JR, Neuberg SL (2007) From stereotype threat to stereotype threats: Implications of a multi-threat framework for causes, moderators, mediators, consequences, and interventions. Personality and Social Psychology Review 11(2):107–130
Sharafi Z, Soh Z, Guéhéneuc YG, Antoniol G (2012) Women and men - different but equal: On the impact of identifier style on source code reading. In: Proceedings of 20th International Conference on Program Comprehension, ICPC '13, pp 27–36
Sharafi Z, Marchetto A, Susi A, Antoniol G, Guéhéneuc YG (2013) An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. In: Proceedings of 21st International Conference on Program Comprehension, ICPC '13, pp 33–42
Sharafi Z, Shaffer T, Bonita S, Guéhéneuc YG (2015a) Eye-tracking metrics in software engineering. In: Proceedings of 22nd Asia-Pacific Software Engineering Conference, IEEE CS Press, APSEC '15
Sharafi Z, Soh Z, Guéhéneuc YG (2015b) A systematic literature review on the usage of eye-tracking in software engineering. Information and Software Technology (IST)
Sharif B, Maletic JI (2010a) An eye tracking study on camelcase and under_score identifier styles. In: Proceeding of 18th IEEE International Conference on Program Comprehension, IEEE Computer Society, ICPC '10, pp 196–205
Sharif B, Maletic JI (2010b) An eye tracking study on the effects of layout in understanding the role of design patterns. In: Proceedings of the 26th IEEE International Conference on Software Maintenance, IEEE Computer Society, pp 1–10
Sharif B, Falcone M, Maletic JI (2012) An eye-tracking study on the role of scan time in finding source code defects. In: Proceedings of the Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA'12, pp 381–384, DOI 10.1145/2168556.2168642
Sharif B, Jetty G, Aponte J, Parra E (2013) An empirical study assessing the effect of seeit 3D on comprehension. In: Proceeding of 1st IEEE Working Conference on Software Visualization, IEEE, VISSOFT '13, pp 1–10
Siegmund J, Siegmund N, Apel S (2015) Views on internal and external validity in empirical software engineering. In: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, pp 9–19
Soh Z, Sharafi Z, den Plas BV, Porras GC, Guéhéneuc YG, Antoniol G (2012) Professional status and expertise for UML class diagram comprehension: An empirical study. In: Proceedings of 20th International Conference on Program Comprehension, ICPC '13, pp 163–172

Soh Z, Khomh F, Guéhéneuc YG, Antoniol G, Adams B (2013) On the effect of program exploration on maintenance tasks. In: 2013 20th Working Conference on Reverse Engineering (WCRE), pp 391–400, DOI 10.1109/WCRE.2013.6671314

Soh Z, Khomh F, Guéhéneuc YG, Antoniol G (2018) Noise in mylyn interaction traces and its impact on developers and recommendation systems. Empirical Software Engineering 23(2):645–692, DOI 10.1007/s10664-017-9529-x, URL https://doi.org/10.1007/s10664-017-9529-x

Spencer SJ, Steele CM, Quinn DM (1999) Stereotype threat and women's math performance. Journal of experimental social psychology 35(1):4–28

Steele CM, Aronson J (1995) Stereotype threat and the intellectual test performance of african americans. Journal of personality and social psychology 69(5):797

Stein R, Brennan SE (2004) Another person's eye gaze as a cue in solving programming problems. In: Proceedings of the 6th International Conference on Multimodal Interfaces, ACM, New York, NY, USA, ICMI '04, pp 9–15, DOI 10.1145/1027933.1027936, URL http://doi.acm.org/10.1145/1027933.1027936

Strandvall T (2009) Eye tracking in human-computer interaction and usability research. In: Gross T, Gulliksen J, Kotzé P, Oestreicher L, Palanque P, Prates RO, Winckler M (eds) Human-Computer Interaction – INTERACT 2009: 12th IFIP TC 13 International Conference, Uppsala, Sweden, August 24-28, 2009, Proceedings, Part II, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 936–937

Sundstedt V (2010) Gazing at games: Using eye tracking to control virtual characters. In: ACM SIGGRAPH 2010 Courses, ACM, New York, NY, USA, SIGGRAPH '10, pp 5:1–5:160, DOI 10.1145/1837101.1837106, URL http://doi.acm.org/10.1145/1837101.1837106

Turner R, Falcone M, Sharif B, Lazar A (2014a) An eye-tracking study assessing the comprehension of C++ and Python source code. In: Proceedings of the Symposium on Eye Tracking Research & Applications, ACM, New York, NY, USA, ETRA '14, pp 231–234, DOI 10.1145/2578153.2578218

Turner R, Falcone M, Sharif B, Lazar A (2014b) An eye-tracking study assessing the comprehension of c++ and python source code. In: Proceedings of the Symposium on Eye Tracking Research and Applications, ACM, New York, NY, USA, ETRA '14, pp 231–234, DOI 10.1145/2578153.2578218, URL http://doi.acm.org/10.1145/2578153.2578218

Uwano H, Nakamura M, Monden A, Matsumoto Ki (2006) Analyzing individual performance of source code review using reviewers' eye movement. In: Proceedings of the 2006 symposium on Eye tracking research & applications, ACM, ETRA '06, pp 133–140

Vrzakova H (2019) Machine learning methods in interaction inference from gaze. In: Dissertations in Forestry and Natural Sciences, University of Eastern Finland

Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A (2012) Experimentation in software engineering. Springer Science & Business Media

Yarbus AL (1967) Eye movements during perception of complex objects. Springer

Yusuf S, Kagdi HH, Maletic JI (2007) Assessing the comprehension of UML class diagrams via eye tracking. In: Proceeding of 15th IEEE International Conference on Program Comprehension, IEEE Computer Society, ICPC '07, pp 113–122

Zhang Z, Zhang J (2010) A new real-time eye tracking based on nonlinear unscented kalman filter for monitoring driver fatigue. Journal of Control Theory and Applications 8(2):181–188