Information and Software Technology 67 (2015) 79–107

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

# A systematic literature review on the usage of eye-tracking in software engineering

Zohreh Sharafi [1], Zéphyrin Soh *, Yann-Gaël Guéhéneuc *

*Ptidej Team, Département de Génie Informatique et Génie Logiciel, Polytechnique Montréal, Québec, Canada*

ARTICLE INFO

*Article history:*
Received 8 December 2014
Received in revised form 20 June 2015
Accepted 22 June 2015
Available online 6 July 2015

*Keywords:*
Eye-tracking
Software engineering
Experiment

ABSTRACT

*Context:* Eye-tracking is a mean to collect evidence regarding some participants' cognitive processes. Eye-trackers monitor participants' visual attention by collecting eye-movement data. These data are useful to get insights into participants' cognitive processes during reasoning tasks.
*Objective:* The Evidence-based Software Engineering (EBSE) paradigm has been proposed in 2004 and, since then, has been used to provide detailed insights regarding different topics in software engineering research and practice. Systematic Literature Reviews (SLR) are also useful in the context of EBSE by bringing together all existing evidence of research and results about a particular topic. This SLR evaluates the current state of the art of using eye-trackers in software engineering and provides evidence on the uses and contributions of eye-trackers to empirical studies in software engineering.
*Method:* We perform a SLR covering eye-tracking studies in software engineering published from 1990 up to the end of 2014. To search all recognised resources, instead of applying manual search, we perform an extensive automated search using Engineering Village. We identify 36 relevant publications, including nine journal papers, two workshop papers, and 25 conference papers.
*Results:* The software engineering community started using eye-trackers in the 1990s and they have become increasingly recognised as useful tools to conduct empirical studies from 2006. We observe that researchers use eye-trackers to study model comprehension, code comprehension, debugging, collaborative interaction, and traceability. Moreover, we find that studies use different metrics based on eye-movement data to obtain quantitative measures. We also report the limitations of current eye-tracking technology, which threaten the validity of previous studies, along with suggestions to mitigate these limitations.
*Conclusion:* However, not withstanding these limitations and threats, we conclude that the advent of new eye-trackers makes the use of these tools easier and less obtrusive and that the software engineering community could benefit more from this technology.



**Contents**



* Corresponding author.
*E-mail addresses:* zohreh.sharafi@polymtl.ca (Z. Sharafi), zephyrin.soh@polymtl.ca (Z. Soh), yann-gael.gueheneuc@polymtl.ca (Y.-G. Guéhéneuc).
[1] Principal corresponding author.

http://dx.doi.org/10.1016/j.infsof.2015.06.008
0950-5849/

## 1. Introduction

The evidence-based paradigm was first employed in 1992 in clinical medicine to integrate "the best research evidence with clinical expertise and patient values" [16]. Kitchenham et al. [13] were the first researchers to adapt the evidence-based paradigm in software engineering in 2004 and called it "Evidence-based Software Engineering" (EBSE). Since then, EBSE has been used to provide insights on different topics in software engineering research and practice. Performing Systematic Literature Reviews (SLR) is recognized as the main approach to realize EBSE with the goal of bringing together all existing evidence on a topic and providing evidence-based guidelines to push froward that topic [14].

In cognitive psychology, researchers have traditionally used eye-trackers to study information processing tasks [40]. Eye-trackers are also increasingly used in other domains, such as marketing, industrial design, and computer science (e.g., graphic data processing and human–computer interactions) [39]. In the early nineties, the software engineering community started to show interest in eye-tracking technology to study the reading strategies involved in program comprehension. In 1990, Crosby et al. [Crosby and Stelovsky, 1990] performed the first eye-tracking study in software engineering. They investigated reading strategies and their impact on the comprehension of procedural code.

However, eye-trackers have not been used a lot in software engineering because of the high price of accurate devices and the difficulties associated with using these devices [Bednarik and Tukiainen, 2007]. As an alternative solution, some previous studies used the Restricted Focus Viewer approach (RFV) [42] and its modified, enhanced version to study debugging strategies [Romero et al., 2002, Romero et al., 2003, 43]. RFV tracks the movements of a computer mouse over a stimulus and their coordinates and durations [42,53]. Yet, since 2006, software engineering researchers started to leverage the availability of unobtrusive eye-trackers to perform different studies.

This paper presents a SLR of 36 papers, collecting, evaluating, and interpreting all studies relevant to the uses of eye-tracking technology in software engineering from 1990 up to the end of 2014. We believe that conducting this SLR is beneficial at this point in time because it brings together all the studies that have been done in the past and could help researchers to avoid misusing eye-tracking technology in software engineering research. Moreover, this SLR provides an overview of all the different eye-trackers, metrics, visualization techniques, activities, and artifacts used in previous eye-tracking studies. It also discusses the limitations associated with eye-tracking technology. Therefore, it can be a starting points for researchers who are interested to perform eye-tracking studies to become acquainted with this technology and its limitations, to find related works, and to decide whether or not to use this modern technology.

In summary, the contributions of this SLR are the following:

1. To provide descriptive statistics and overviews on the uses of eye-tracking technology in software engineering.
2. To present an annotated bibliography and provide and discuss exhaustive lists of topics and artifacts studied using eye-trackers.
3. To summarize all the metrics and tools available for the analysis of eye-tracking data to study developers' cognitive processes.

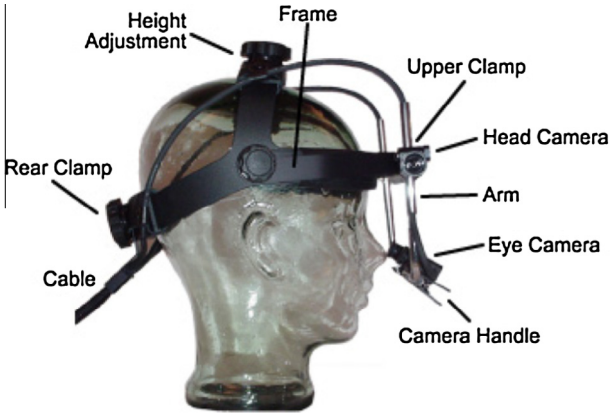


**Fig. 1.** Example of goggle used in intrusive video-based eye-trackers. This image is adapted from [39].

4. To allow researchers to use a unified and consistent terminology (e.g., metrics names), when conducting and reporting eye-tracking studies.

The annotated bibliography presents information about the selected studies in a structured way. It allows researchers to compare studies with one another with respect to the selection of material, the participants' selection, and the study variables. Researchers with particular topics, activities, or artifacts in mind, can find out whether their topics, activities, or artifacts have been studied and, if they were, relevant information from the annotated bibliography. The SLR also concludes with the needs for new metrics and tools to analyze eye-tracking data as well as advices to overcome or limit threats to the validity of eye-tracking studies.

The reminder of this paper is organized as follows: In Section 2, we provide necessary background on eye-tracking technology. In Section 3, we discuss the methodology that we follow in conducting the SLR and present the summary of evidence in Section 4. We provide the complete answers to our research questions in Section 5. We discuss threats to the validity of this SLR in Section 6 and conclude this paper in Section 7.

## 2. Eye-tracking technology

We now provide the necessary background on eye-tracking technology as well as the assumptions related to eye-movements that underly the building and usage of eye-trackers (in Section 2.1). We also summarize the different types of eye-trackers (in Section 2.2).

Eye-trackers are designed to monitor a participant's visual attention by collecting eye-movement data when the participant looks at a stimulus while working on a task [39,40]. In human vision, eye-movements are essential to cognitive processes as they carry the visual attention to the specific parts of some stimuli that are processed by the brain. A stimulus is an object (e.g., source code and diagram) necessary to perform the task.

Eye-movement data are studied with respect to certain areas of the stimuli, which are called Areas of Interest (AOIs). An AOI can either be relevant or not to a participant while performing a task. For example, if we consider a class diagram as stimulus, a relevant AOI could be a specific class that is used by the participant to perform the task, while irrelevant AOI would be any other classes.

Eye-movement data are classified based on the following significant indicators of ocular behavior:

1. A fixation is the stabilization of the eye on a part of the stimulus for a period of time between 200 and 300 ms. All eye-trackers provide fixation data, which include a time stamp and $x$ and $y$ coordinates.
   Researchers in psychology claim that most of the information acquisition and cognitive processing occur during fixations. Moreover, they showed that only a small set of fixations is required for participants to acquire and process a complex visual input [41].
2. A saccade is the sudden, quick movement from one fixation to another. Saccadic eye-movements are extremely rapid, within 40–50 ms. Researchers in psychology claim that the information encoding and cognitive processing that occur during a saccade is very limited [39,41].
3. A scan-path is a series of fixations or AOIs, in chronological order. An AOI is visited if there is at least one fixation in it. Fig. 3 shows two distinct scan-paths on a same stimulus, with four areas of interest: A, B, C, and D. Researchers in psychology showed that scan-paths are representative of the tasks being performed by participants.

### 2.1. Eye-tracking assumptions

The relation between eye-movements and comprehension is based on two assumptions [22]:

- The immediacy assumption states that, as soon as a participant sees a stimulus, she tries to interpret it.
- The eye–mind assumption states that a participant fixates her attention on a stimulus until she understands it.

Previous works presented in this SLR support these assumptions and use them to analyze and interpret eye-movement data. They also further assume that participants are actively engage in performing their tasks and are not day-dreaming during the studies.

### 2.2. Devices

Eye-trackers are divided into two main categories. While many eye-trackers are available, we list only those that are used in the selected papers retained for our SLR:

1. Intrusive eye-trackers typically contain three miniature cameras that are mounted on a padded headband that participants wear during the studies. Two of these cameras capture eye-movements using infrared lights reflecting on the participants' pupils while the third one is optional and used for head tracking. The major limitation of these devices is instability because a light head movement may result in an inaccurate eye-movement coordinates. In addition, the padded headband, which is a heavy goggle with several wires, as shown in Fig. 1, may worry participants. The *Eye-link II* from *SR Research* [54] is an example of intrusive eye-trackers.
2. Non-intrusive eye-trackers divide in two generations. Those of the older generation typically used beams of light that are reflected on the participants' eyes and captured while returning. Such eye-trackers have low resolutions and precisions. Those of the newer generation—the video-based eye-trackers—typically consist of one computer, two miniature cameras, and one infrared pad, as shown in Fig. 2. The computer detects and tracks the participants' eye-movements by tracking the positions of the participant's heads using eye-brows, noses, and lips. It also uses corneal reflection (infrared) and pupil center to distinguish head- and eye-movements. *FaceLAB* from *Seeing Machines* [55] and *Tobii 1750* are examples of non-intrusive, video-based eye-trackers.

**Fig. 2.** (Left) A participant sits in front of the computer and a video-based eye-tracker captures eye-movement data. (Right) FaceLAB is an example of video-based eye-trackers. This image is adapted from FaceLAB manual [54].

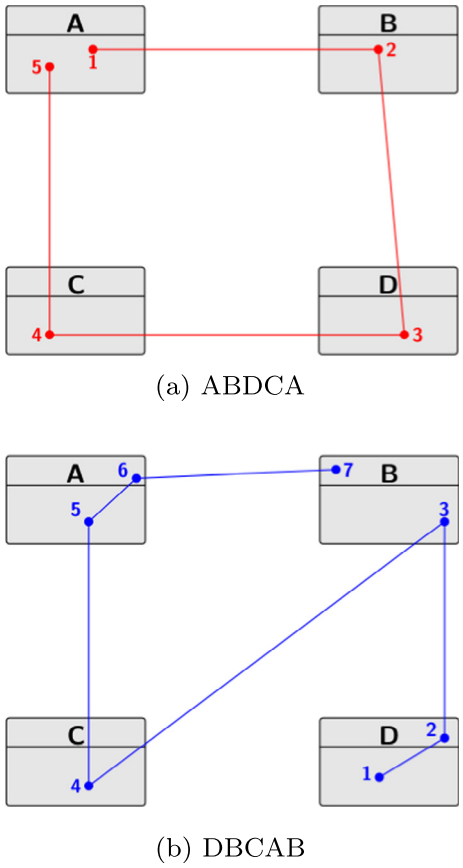


**Fig. 3.** Two distinct scan-paths on the same stimulus with four areas of interest: A, B, C, and D. This image is adapted from [De Smet et al., 2014].

## 3. Methodology

We carry out this SLR following Kitchenham's guidelines [17].

### 3.1. Research questions

This SLR answers the following research questions:

RQ 1. How many articles have been published using eye-trackers in software engineering research since 1990?
RQ 2. What research topics have been explored and examined in eye-tracking studies?
RQ 3. How much have eye-tracking studies contributed to software engineering?
RQ 4. How have researchers used eye-trackers to collect and visualize quantitative measurements?
RQ 5. What are the limitations of current eye-tracking studies?
RQ 6. What eye-trackers are most frequently used in eye-tracking studies?

To address RQ1, we identify the numbers of journal, conference, and–or workshop papers published starting from 1990 up to the end of 2014 and which report one of more empirical studies using an eye-tracker in the field of software engineering. We choose 1990 as starting point because the first eye-tracking study in software engineering was carried out in 1990 by Crosby et al. [Crosby and Stelovsky, 1990]. We confirmed this observation by performing a complementary search looking for relevant papers published between 1980 and 1990. This complementary search returned only the paper by Crosby et al. Regarding RQ2, we consider the scope of each study based on categories for which we found at least two publications: model comprehension, debugging, code comprehension, collaborative interaction, and traceability.

To answer RQ3, we summarize the outcomes of the selected studies. With respect to RQ4, we provide a list of all metrics based on eye-tracking data as well as their definitions and applicability. To answer RQ5, we explain the limitations of current eye-trackers for software engineering studies along with the solutions deployed by researchers to mitigate these limitations. Finally, we provide a list of commonly used eye-trackers to answer RQ6.

### 3.2. Search process

We use Engineering Village[2] to search for papers that present one or more eye-tracking studies in the field of software engineering. Engineering Village is an information discovery platform that is connected to several trusted engineering literature databases. If we had performed our search using individual, independent electronic databases, such as the ACM[3] and IEEE Xplore[4] digital libraries separately, we would have needed different search queries, potentially threatening the internal validity of our results. Engineering Village gives us the ability to search in all recognised scholarly journals, conference and workshop proceedings together with a unique search query.

Because our main goal is **to study the usage of** eye-tracking technology in software engineering, we assume that the **RFV approach** and–or **eye-trackers** are used to collect participants' eye-movements while performing **tasks** using software engineering-related **stimuli**. Thus, we define three sets of keywords presented as follows, where a "*" at the end of each word is a truncation that can be replaced with zero or more characters:

[2] http://www.ei.org/engineering-village.
[3] http://dl.acm.org/.
[4] http://ieeexplore.ieee.org.

```
("eye-track*" OR "eye track" OR "RFV" OR "Restricted Focus Viewer") AND
("source code" OR program* OR UML OR model* OR representation*) AND
(comprehen* OR understand* OR debug* OR navigat* OR read* OR scan*)
```

**Fig. 4.** Search query.

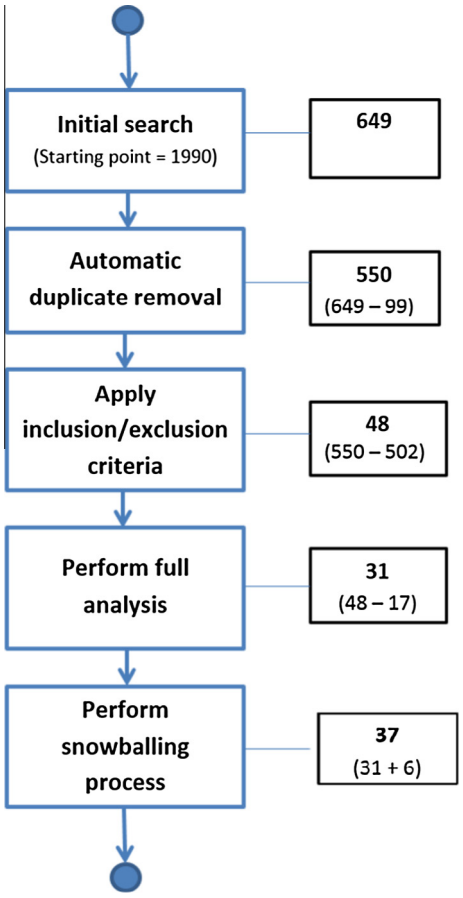


**Fig. 5.** Selection process adapted to choose appropriate related papers. The numbers on the left are the numbers of papers remaining after each step has been performed.

1. Eye-tracking: "eye track*", "eye-track*", "RFV", and "Restricted Focus Viewer". We use these keywords to find papers that use an eye-tracker or the RFV approach. Therefore, the returned set of papers contains two main terms related to eye-trackers, i.e., "eye" and "track" and RFV-related terms.
2. Stimuli: "source code", "program*", "UML", "model*", and "re presentation*". We define this set of keywords to find papers focusing on the different types of stimuli used in software engineering studies.
3. Task: "comprehen*", "understand*", "debug*", "explorat*", "navigat*", "read*", "scan*", and "analy*". We define this set of keywords to find papers pertaining to the different activities performed by software developers on a daily basis and that have been studied by researchers.

First, we included all keywords in a prelimiary query. We evaluated and refined the query as follows. First, we executed the preliminary query. The search engine was set to search from 1990 to 2014. We automatically applied duplicate removal. We used both Inspec[5] and Compendex[6] databases. We sorted the results of the preliminary query by relevance. Second, the two first authors, who are familiar with the uses of eye-trackers in software engineering checked the first ten found papers and verified the number of relevant papers to evaluate the quality of the search results. We found five papers out of ten that are relevant to the goal of this study. Third, we evaluated the impact of each keyword by removing the keywords one by one and checking the results. If by removing a keyword, at least one relevant result went missing, we considered that keyword essential and kept it. If after keyword removal, there was no missing paper, we removed that keyword. Based on the three sets of keywords and this evaluation process, we obtain the final search query shown in Fig. 4.

### 3.3. Selection process

The process that we adopt to select the relevant papers is presented in Fig. 5. On the left, we present the set of activities that we undertook while, on the right, we present the numbers of remaining papers after each activity.

1. Select related papers: we execute our query in Engineering Village. The search engine searches into the title, the abstract, and the keywords sections of the papers looking for the keywords that are defined in our query to find matching papers.
2. Apply automatic process of duplicate removal: we use Engineering Village duplicate removal feature to automatically find and remove duplicate papers among the set of papers returned by its engine.
3. Apply inclusion and exclusion criteria: we perform this step to check whether the resulting papers are relevant or not to our goal. We present and explain these criteria in details in the next section.
4. Analyze selected papers for duplicate removal: for each selected paper, one author performs its full analysis. Using title, keywords, abstract, and body, we check whether the paper is a duplicate because, although Engineering Village features a duplicate-removal process, there are still duplicated papers in the results; for example, because different publishers use different formats to display authors' names.
5. Perform the snowballing process: for each paper, either the first or second author of this SLR went through the list of all its references, checked each reference starting from its title, the conference proceeding and journal names, and then checked the full paper if necessary to decide whether to include the paper in the study or not.

### 3.4. Inclusion and exclusion criteria

We adopt the following inclusion/exclusion criteria. We go through the abstract and the body of each paper to ensure their relevance according to these criteria.

Inclusion criteria:

1. Primary studies published in journals or conference and workshop proceedings in the form of experiments, surveys, case studies, reports, and observation papers using eye-tracking technology to study and investigate software engineering activities.

[5] http://www.theiet.org/resources/inspec/.
[6] http://adat.crl.edu/databases/about/compendex.

**Table 1**
Journals, conference and workshop proceedings of the selected papers.

| Source (Acronym) | Total | Papers |
|---|---|---|
| ACM Computer Supported Cooperative Work (CSCW) | 1 | [Jeanmart et al., 2009] |
| Conference of the Center for Advanced Studies on Collaborative Research (CASCON) | 1 | [Guéhéneuc, 2006] |
| Computer Supported Collaborative Learning Conference (CSCL) | 1 | [Sharma et al., 2013] |
| Conference on Advanced Information Systems Engineering (CAiSE) | 1 | [Petrusel and Mendling, 2012] |
| Conference on Computing Education Research (Koli Calling) | 1 | [Busjahn et al., 2011] |
| Conference on Software Maintenance (ICSM) | 2 | [Sharif and Maletic, 2010]<br>[Ali et al., 2012] |
| Conference on Program Comprehension (ICPC) | 5 | [Yusuf et al., 2007]<br>[Sharafi et al., 2012]<br>[Soh et al., 2012]<br>[Sharafi et al., 2013]<br>[Walters et al., 2014] |
| Empirical Software Engineering Journal (EMSE) | 2 | [Cepeda and Guéhéneuc, 2010]<br>[Binkley et al., 2013] |
| Eye-tracking Research and Application (ETRA) | 7 | [Bednarik and Tukiainen, 2006]<br>[Uwano et al., 2006]<br>[Bednarik and Tukiainen, 2008]<br>[Hejmady and Narayanan, 2012]<br>[Sharif et al., 2012]<br>[Busjahn et al., 2014]<br>[Turner et al., 2014] |
| Hawaii International Conference on System Sciences (HICSS) | 1 | [Aschwanden and Crosby, 2006] |
| IEEE Computer Journal (IEEE Computer) | 1 | [Crosby and Stelovsky, 1990] |
| International Conference on Software Engineering (ICSE) | 2 | [Fritz et al., 2014]<br>[Rodeghero et al., 2014] |
| Journal of Behavior research methods (BRM) | 1 | [Bednarik and Tukiainen, 2007] |
| Science of Computer Programming Journal (SCP) | 1 | [De Smet et al., 2014] |
| Journal of Human Computer Studies (HUMCOMPUT) | 1 | [Bednarik, 2012] |
| International Journal of Human Computer Interaction (IJHCI) | 1 | [Duru et al., 2013] |
| International Conference on Multimodel Interaction (ICMI) | 1 | [Stein, 2004] |
| Journal of Systems and Software (JSS) | 1 | [Bednarik, 2012] |
| Lecture Notes in Computer Science (LNCS) | 1 | [Cagiltay et al., 2013] |
| Symposium on Software Engineering and Measurement (ESEM) | 1 | [Romero et al., 2002] |
| Working Conference on Software Visualization (VISSOFT) | 1 | [Sharif et al., 2013] |
| Workshop of Psychology of Programming Interest Group (PPIG) | 2 | [Crosby et al., 2002]<br>[Romero et al., 2003] |

2. Primary studies that present the more detailed and complete results, if there are more than one published versions of a specific study.

Exclusion criteria:

1. Papers that do not use an eye-tracker.
2. Papers that are not related to software engineering. We remove papers related to other fields (i.e., biomedical optics and imaging) by checking the body of the selected papers.
3. Papers in "grey" literature, which are not published by trusted, well-known publishers, and–or which did not go through a well-defined referring process [19]. (These papers are automatically excluded by Engineering Village.)
4. Papers that are not published in English. (These papers were removed by Engineering Village through its publication language feature and using only the "English" language.)

We did not include dissertations in this SLR because they are not returned by Engineering Village a it considers them "grey" literature. Moreover, a dissertation may contain several eye-tracking studies and, therefore, we prefer to consider the individual papers presenting each study.

### 3.5. Data collection

The pieces of information extracted from each paper are the following:

- Authors and their affiliations.
- Years of publication.
- Names of the eye-trackers.
- Configurations of the devices in the study, e.g., the distance between the participants and the screens, the sizes and resolutions of the screens.
- Main topics of the publication.
- Main research questions and results.
- All dependent, independent, and mitigating variables of the experiment.
- Metrics defined based on eye-tracking data.
- Studied artifacts, e.g., source code, diagrams, text, etc.
- Studied tasks: e.g., reading, scanning/exploration, comprehension, etc.

The first and second authors were responsible for extracting the data from the papers and verifying the extraction. The papers, whose data was extracted by the first author were verified by the second author and vice versa. Although using one extractor and one checker is not recommended by medical standards for performing SLR [17], the usability of this procedure in software engineering is confirmed by Brereton et al. [18]. In case of disagreement, all the authors held a discussion until agreement. To preserve the consistency of the data extraction, we used a data extraction form that we designed for this study. We provide a copy of this form in Table 17.

## 4. Search results

Applying our search query returned an original set of 649 papers. We then perform the following actions to obtain the final set of relevant papers, as illustrated in Fig. 5:

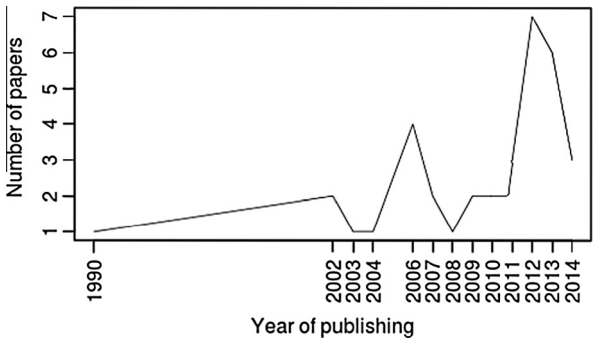


**Fig. 6.** Number of published papers per year.

1. We use the Engineering Village search engine and remove duplicates automatically. 550 papers remain in the set.
2. We apply the inclusion/exclusion criteria and reduce the number of articles to 48. We use the abstract and the body of the papers to remove several papers that are not related to software engineering. Several papers pertain to different fields, including cognitive science, natural language processing, computational science, mathematics, geoinformatics, bioinformatics, robotics, etc. Two proceedings of the "Eye-tracking Research and Application Conference" (ETRA) of 2008 and 2012 are also removed because we already selected their papers related to software engineering.
3. We analyze the remaining set of 48 papers and remove 17 more papers, leaving 31 papers. Seven out of the 17 removed paper present new methods to compute, represent, or predict participants' scan-paths and do not perform any software engineering-related study. There is also one paper whose text we could not find online. In addition, we remove [43] because [Romero et al., 2002] is its extended journal paper version.
   We also remove [21,58] because they do not contain any eye-tracking study. We also remove six duplicated papers. Out of these six papers, one removed paper is a replication of another eye-tracking study [Yusuf et al., 2007] using a new questionnaire. Another [32] is not a study and is only used in a working session of a conference. Another [29] is a collection of four studies including two eye-tracking ones. Therefore, we remove this paper and consider one of the original eye-tracking studies [Sharif and Maletic, 2010]. Regarding the second eye-tracking study, we remove the original study [31] and use a more complete version published in 2013 [Binkley et al., 2013]. We replace [33,34] with a journal paper that presents a more complete study [Bednarik and Tukiainen, 2007].
4. After snowballing, we find five more papers [Crosby et al., 2002, Aschwanden and Crosby, 2006, Guéhéneuc, 2006, Fritz et al., 2014, Rodeghero et al., 2014].

We have not found paper [Crosby et al., 2002] by applying our search query because it does not include, in its title or abstract, occurrences of the words in our first set of keywords, related to eye-tracking. Paper [Aschwanden and Crosby, 2006] is not listed in Engineering Village because it is published in a local conference, Hawaii International Conference on System Sciences, yet this conference conforms to our inclusion/exclusion criteria.

Overall, we find 36 papers related to the use of eye-tracking technology in software engineering between 1990 and 2014. Table 1 shows the names of journals and conference and workshop proceedings of the papers along with the total numbers of papers per venue. It also shows the set of selected papers from each source. ETRA published seven articles, which is the highest number in comparison to other sources, because this conference is dedicated to eye-tracking technology and its applications. Consequently, several researchers target this conference to present their research work. ICPC published five articles. This last number shows that several software-engineering researchers used the eye-tracking technology to study program comprehension. Because program comprehension is a problem-solving task, eye-trackers provide useful information about the developers' cognitive processes. Finally, ICSE, ICSM, and ESEM, each published two articles.

## 5. Discussion

We can now answer our research questions and discuss their answers.

### *5.1. How many articles have been published using eye-trackers in software engineering research since 1990?*

Overall, we find 36 relevant papers, summarized in Tables 15 and 16. All of these papers report at least one study with some eye-tracker. Fig. 6 presents the numbers of papers published per year starting from 1990 to 2014. Only 13.8% of these papers were published between 1990 and 2006 and the rest were published in the last 10 years. This table shows that, in the software engineering community, eye-trackers have become increasingly accepted as useful tools to perform empirical studies from 2006.

### *5.2. What research topics have been explored and examined in eye-tracking studies?*

We categorize the selected papers in five groups, shown in Table 2. Classifying papers is a challenging task. We could not define categories in advance because they are dependent on the selected papers. Therefore, we first identify a set of categories based on titles and abstracts. Then, after performing their analyses,

**Table 2**
Classification of the selected papers based on category.

| Aspect | Total | Papers |
|---|---|---|
| Model Comprehension | 10 | [Guéhéneuc, 2006, Yusuf et al., 2007, Jeanmart et al., 2009] [Cepeda and Guéhéneuc, 2010, Sharif and Maletic, 2010, De Smet et al., 2014, Soh et al., 2012, Cagiltay et al., 2013] [Petrusel and Mendling, 2012, Sharafi et al., 2013] |
| Code comprehension | 12 | [Crosby and Stelovsky, 1990, Crosby et al., 2002, Aschwanden and Crosby, 2006, Bednarik and Tukiainen, 2006] [Busjahn et al., 2011, Sharafi et al., 2012, Binkley et al., 2013] [Duru et al., 2013, Busjahn et al., 2014, Turner et al., 2014, Fritz et al., 2014, Rodeghero et al., 2014] |
| Debugging | 9 | [Romero et al., 2002, Romero et al., 2003, Uwano et al., 2006] [Bednarik and Tukiainen, 2007, Bednarik and Tukiainen, 2008, Bednarik, 2012, Hejmady and Narayanan, 2012] [Sharif et al., 2012, Sharif et al., 2013] |
| Collaborative interactions | 3 | [Stein, 2004],[Jermann and Nüssli, 2012, Sharma et al., 2013] |
| Traceability | 2 | [Ali et al., 2012, Walters et al., 2014] |

we finalize the categories based on the tasks performed by participants and assign papers to different categories so that each retained category includes at least two papers.

For each category, we now systematically report each paper in a structured way by presenting the following information: (1) artifacts; (2) number and participants' types (students, faculty members, and–or professionals); (3) eye-tracker; and (4) dependent variables (DV), independent variables (IV), and mitigating variables (MV). If one piece of information is missing in a paper, we put "not mentioned". We explain the metrics used for calculating dependent and independent variables are in Section 5.4. We use this systematic, structured way to provide an minimal annotated bibliography of the selected studies. Using this bibliography, researchers can compare different studies regarding different pieces of information.

Selected papers refer to developers participating in the eye-tracking studies as "subjects" or "participants" interchangeably. Yet, the word "participant" better fits eye-tracking studies because researchers do not want to understand developers to change their behaviors but rather to understand their *uses* of some artifacts and tools when performing of tasks to help them in their work.

#### 5.2.1. Model comprehension

We find ten papers pertaining to model comprehension. All of the papers selected for this category perform comprehension tasks.

[Guéhéneuc, 2006] uses eye-trackers to study the comprehension of UML class diagrams. It introduces a visualization technique to aggregate and display eye-tracking data (fixations and saccades). This visualization technique superimposes aggregations of the fixations and saccades for all participants to highlight the most visited AOIs.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams | |
| *Participants* | 12 students | |
| *Eye-tracker* | EyeLink II | |
| *Variables* | *DV* | Not mentioned |
| | *IV* | Not mentioned |
| | *MV* | Not mentioned |

[Yusuf et al., 2007] investigates the impact of different characteristics of UML class diagrams, including layout, color, and stereotypes. It uses UML class diagrams of HippoDraw[7] as artifacts.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams of the open source called HippoDraw | |
| *Participants* | 12 students and faculty members | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Layout (orthogonal, three-cluster, and multi-cluster) |
| | *MV* | Not mentioned |

[Jeanmart et al., 2009] investigates the impact of the Visitor design pattern on comprehension and maintenance. It uses UML class diagrams of JHotDraw,[8] JRefactory,[9] and PADL[10] as artifacts.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams of three open source programs | |
| *Participants* | 24 students | |
| *Eye-tracker* | EyeLink II | |
| *Variables* | *DV* | Time and effort |
| | *IV* | Design alternative: no pattern, canonical, and modified |
| | *MV* | UML and design pattern knowledge |

[Cepeda and Guéhéneuc, 2010] compares different UML representations of design patterns.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams | |
| *Participants* | 24 students | |
| *Eye-tracker* | EyeLink II | |
| *Variables* | *DV* | Time, accuracy, ratio of on-target:all-target fixation time, and ratio of on-target:all-target fixation |
| | *IV* | Representations of patterns (Schauer [8], Gamma [9], and Dong [10]) and tasks (participation, composition, and role). |
| | *MV* | JHotdraw and design pattern knowledge |

[Sharif and Maletic, 2010] analyses the impacts of orthogonal and multi-clustered layouts on the comprehension of design patterns. It uses UML class diagrams of JHotDraw, JUnit,[11] and Qt[12] as artifacts.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams of three open-source programs | |
| *Participants* | 12 students and faculty members | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Reading behavior |
| | *MV* | |

[De Smet et al., 2014] investigates the impact of different design patterns, including Observer, Composite, and Model-View-Controller on comprehension. It uses UML class diagrams of JUnit, Quick-UML,[13] and ArgoUML[14] as artifacts.

| | | |
|---|---|---|
| *Artifacts* | UML class diagrams of three open source programs | |
| *Participants* | 26 students and faculty members; 18 students | |
| *Eye-tracker* | EyeLink II | |
| *Variables* | *DV* | Spatial density, transitional matrix, average fixation duration; time and scan-path distance |
| | *IV* | Presence of patterns (observer and composite); different variants of MVC pattern |
| | *MV* | Not mentioned |

[Soh et al., 2012] studies the relation between expertise and professional status for UML class diagram comprehension.

[7] www.slac.stanford.edu/grp/ek/hippodraw.
[8] http://www.jhotdraw.org/.
[9] http://jrefactory.sourceforge.net/.
[10] http://wiki.ptidej.net/doku.php?id=padl.
[11] http://junit.org/.
[12] http://qt-project.org/.
[13] http://sourceforge.net/projects/quj/.
[14] http://argouml.tigris.org/.

| | | |
|---|---|---|
| *Artifacts* | | UML class diagrams |
| *Participants* | | 21 students and faculty members |
| *Eye-tracker* | | EyeLink II |
| *Variables* | *DV* | Accuracy, time, effort |
| | *IV* | Status (practitioner, student) and expertise (expert, novice) |
| | *MV* | Question precision |

[Petrusel and Mendling, 2012] focuses on the understanding of business-process models (BPMN[15] diagrams) and investigates the factors that influence their comprehension.

| | | |
|---|---|---|
| *Artifacts* | | BPMN diagrams |
| *Participants* | | 26 professionals |
| *Eye-tracker* | | Not mentioned |
| *Variables* | *DV* | Accuracy |
| | *IV* | Number of elements in the relevant region, and time |
| | *MV* | Not mentioned |

[Cagiltay et al., 2013] studies non-formal inspections of entity-relationship diagram (ERD). It proposes two measures of defect detection to measure how developers comprehend ERD.

| | | |
|---|---|---|
| *Artifacts* | | ER diagrams |
| *Participants* | | 4 professionals |
| *Eye-tracker* | | Tobii (model not mentioned) |
| *Variables* | *DV* | Defect detection difficulty level and defect detection performance. |
| | *IV* | Search pattern |
| | *MV* | Not mentioned |

[Sharafi et al., 2013] investigates the efficiency of the graphical vs. textual representations of the TROPOS [51] notation in modeling and presenting software requirements.

| | | |
|---|---|---|
| *Artifacts* | | TROPOS diagrams |
| *Participants* | | 28 students |
| *Eye-tracker* | | FaceLAB |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Representation type (graphical vs. textual) |
| | *MV* | English language proficiency and type preferences |

*5.2.2. Code comprehension*

We identify 12 papers pertaining to code comprehension. All of the papers in this category perform comprehension tasks by reading pieces of source code to answer comprehension questions. As types of artifacts, we report the programming languages, the main functionalities of the pieces of source code, and the numbers of lines of code (LOC), if available.

[Crosby and Stelovsky, 1990] studies the source code reading to assess the impact of expertise on the developers' comprehension strategies.

| | | |
|---|---|---|
| *Artifacts* | | Pascal source codes of binary search algorithm |
| *Participants* | | 18 students and faculty members |
| *Eye-tracker* | | Not mentioned |
| *Variables* | *DV* | Number of fixations and time |
| | *IV* | Expertise |
| | *MV* | Not mentioned |

[Crosby et al., 2002] defines beacons as important features in source code that "serve as keys to facilitate program comprehension". It investigates how experts and novices used these beacons to read and understand programs.

| | | |
|---|---|---|
| *Artifacts* | | Lines of code from binary search program written in Pascal and shown in random order |
| *Participants* | | 18 students |
| *Eye-tracker* | | ASL |
| *Variables* | *DV* | Accuracy and time |
| | *IV* | Expertise and number of lines |
| | *MV* | Not mentioned |

[Bednarik and Tukiainen, 2006] provides a visualization technique of Java source code and studies how developers use source code and the visualization interchangeably.

| | | |
|---|---|---|
| *Artifacts* | | Three Java source codes of factorial (15 LOC), recursive binary-search (34 LOC), and naive string matching (38 LOC) |
| *Participants* | | 14 students |
| *Eye-tracker* | | Tobii 1750 |
| *Variables* | *DV* | Time and attention switching |
| | *IV* | Code vs. visualization |
| | *MV* | Not mentioned |

[Aschwanden and Crosby, 2006] focuses on participants' expertise and reveals that experts and novices spend different amounts of visual attention on different parts of the source code.

| | | |
|---|---|---|
| *Artifacts* | | Java source codes of an algorithm presented in recursive and non-recursive versions |
| *Participants* | | 15 (not mentioned the type) |
| *Eye-tracker* | | ASL |
| *Variables* | *DV* | Accuracy and number of lines |
| | *IV* | Algorithm type (recursive vs. non-recursive) |
| | *MV* | Expertise |

[Busjahn et al., 2011] performs an experiment to investigate the differences between source code reading and natural text reading.

| | | |
|---|---|---|
| *Artifacts* | | Java source codes |
| *Participants* | | 15 (not mentioned the type) |
| *Eye-tracker* | | Tobii T120 |
| *Variables* | *DV* | Time, number of characters, and number of elements |
| | *IV* | Source code vs. natural language text, and different source code parts including: operator, keywords, identifiers, and numbers |
| | *MV* | Not mentioned |

[15] http://www.bpmn.org/.

[Sharafi et al., 2012] investigates the impact of identifier styles (camel case vs. underscore) on developers recalling the names of identifiers. It also compares different strategies deployed by male and female developers.

| | | |
|---|---|---|
| *Artifacts* | Three Java sources code of 2D graphical frame (30 LOC), database tester (36 LOC), and nprime number calculator (44 LOC) | |
| *Participants* | 26 students | |
| *Eye-tracker* | FaceLAB | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Gender and identifier style (camel case vs underscore) |
| | *MV* | Study level and style preferences |

[Binkley et al., 2013] also investigates the impact of identifier styles on comprehension with two studies (recall and multiple choice questions).

| | | |
|---|---|---|
| *Artifacts* | English words and C++ source codes | |
| *Participants* | 169 students | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Identifier style (camel case vs underscore) |
| | *MV* | Length of the phrase, phrase origin (code vs. Non-code), time demography (Applet-Cloud only), training, and Experience |

[Duru et al., 2013] studies visualization techniques to understand the reason why such techniques are not being used in industry.

| | | |
|---|---|---|
| *Artifacts* | An e-commerce small-scale enterprise.NET application | |
| *Participants* | 13 professionals | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Accuracy and time |
| | *IV* | Presence of visualization provided by NDEPEND software visualization tool |
| | *MV* | Not mentioned |

[Turner et al., 2014] investigates the impact of programming language (C++ vs. Python) on source code comprehension by comparing experts' and novices' scan-paths.

| | | |
|---|---|---|
| *Artifacts* | Five C++ source code and five Python source code | |
| *Participants* | 38 Students | |
| *Eye-tracker* | Not mentioned | |
| *Variables* | *DV* | Accuracy, time, and visual effort |
| | *IV* | Programming language (C++ vs. Python) |
| | *MV* | Expertise |

[Busjahn et al., 2014] studies attention distribution on code elements to differentiate experts' and novices' code reading strategies.

| | | |
|---|---|---|
| *Artifacts* | Eleven Java source codes | |
| *Participants* | 15 Professionals | |
| *Eye-tracker* | Tobii T120 | |
| *Variables* | *DV* | Time |
| | *IV* | Code elements (identifiers, operators, keywords, and literals) |
| | *MV* | Expertise |

[Fritz et al., 2014] uses psycho-physiological measures, including eye-gaze data, electroencephalography (EEG), electrodermal activity (EDA), and NASA TLX scores [1] to study task difficulty.

| | | |
|---|---|---|
| *Artifacts* | Eight C♯ source code | |
| *Participants* | 15 Professionals | |
| *Eye-tracker* | Tobii TX300 | |
| *Variables* | *DV* | Time, effort (Eye-gaze data (Pupil size), EEG data (and Blink rate), and EDA data) |
| | *IV* | Task difficulty (easy and hard) |
| | *MV* | Not mentioned |

[Rodeghero et al., 2014] conducts an eye-tracking study and use its findings to build a code summarization tool.

| | | |
|---|---|---|
| *Artifacts* | 67 Java methods from six different applications: NanoXML, Siena, JTopas, Jajuk, JEdit, and JHotdraw | |
| *Participants* | 10 Professionals | |
| *Eye-tracker* | Tobii TX300 | |
| *Variables* | *DV* | Fixation number and duration, Fixation Time, and Number of regression |
| | *IV* | Task difficulty (easy and hard) |
| | *MV* | Not mentioned |

#### 5.2.3. Debugging

We find 9 studies related to debugging. In all of the selected papers, participants read source code and perform debugging tasks.

[Romero et al., 2002] focuses on the use of different representations by participants while performing debugging tasks and whether these representations bring higher performance.

| | | |
|---|---|---|
| *Artifacts* | Java source codes of two programs | |
| *Participants* | 4 students and 1 professionals | |
| *Eye-tracker* | RFV | |
| *Variables* | *DV* | Accuracy and time |
| | *IV* | The presence of RFV (RFV-on vs. RFV-off) |
| | *MV* | Not mentioned |

[Romero et al., 2003] characterizes the participants' strategies in debugging tasks.

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 49 students | |
| *Eye-tracker* | RFV | |
| *Variables* | *DV* | Accuracy, time, and switching frequency |
| | *IV* | Visualization type (graphical vs. textual), visualization perspective (data structure vs. control flow), and type of error |
| | *MV* | Not mentioned |

[Uwano et al., 2006] focuses on source code reviews by developers to find defects.

| | | |
|---|---|---|
| *Artifacts* | Five C source codes (12–23 LOC) | |
| *Participants* | 5 students | |
| *Eye-tracker* | Eye Mark Tracker (EMR-NC) | |
| *Variables* | *DV* | Time |
| | *IV* | Presence of defects |
| | *MV* | Not mentioned |

[Bednarik and Tukiainen, 2007] uses both an eye-tracker and the RFV approach to investigate the impact of RFV and of expertise on attention.

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 18 students and faculty members | |
| *Eye-tracker* | Tobii 1750 and RFV | |
| *Variables* | *DV* | Accuracy, time, and switching frequency |
| | *IV* | Presence of RFV (RFV-on vs. RFV-off) and level of experience |
| | *MV* | Not mentioned |

[Bednarik and Tukiainen, 2008] reanalyses the data reported in [Bednarik and Tukiainen, 2007] and, by segmenting eye-tracking data into smaller chunks, reports more accurate differences between experts and novices.

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 14 students | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Time and switching frequency |
| | *IV* | Expertise |
| | *MV* | Not mentioned |

[Sharif et al., 2012] partially replicates [Uwano et al., 2006] with a larger number of participants and additional eye-tracking metrics. It analyses participants' eye-movements captured while performing defect finding tasks.

| | | |
|---|---|---|
| *Artifacts* | C source codes | |
| *Participants* | 15 students and faculty members | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Presence of defects |
| | *MV* | Expertise |

[Bednarik, 2012, Hejmady and Narayanan, 2012] use a multi-representational integrated development environment (IDE) to display source code. [Hejmady and Narayanan, 2012] analyses how experts and novices find defects and their uses of different representations.

| | | |
|---|---|---|
| *Artifacts* | Three Java source codes (in average 100 LOC) | |
| *Participants* | 18 students and professionals | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Time, switching frequency, and type of switches |
| | *IV* | Expertise and strategies |
| | *MV* | Not mentioned |

[Hejmady and Narayanan, 2012] studies the effectiveness and the role of multiple representations during debugging.

| | | |
|---|---|---|
| *Artifacts* | One Java source code implementing bubble sort algorithm and seeded with three bugs | |
| *Participants* | 19 students | |
| *Eye-tracker* | Tobii T60 XL | |
| *Variables* | *DV* | Experience, familiarity with jGrasp, and time |
| | *IV* | Strategy |
| | *MV* | Not mentioned |

[Sharif et al., 2013] assesses the usability of a visualization tool, called SeeIT, for performing code overview and bug-fixing tasks.

| | | |
|---|---|---|
| *Artifacts* | Java source code of an open source system: Gantt Project[a] | |
| *Participants* | 97 students | |
| *Eye-tracker* | Tobii X60 | |
| *Variables* | *DV* | Accuracy, time, and effort |
| | *IV* | Presence of 3D visualization tool (SeeIT 3D, No-SeeIT 3D) |
| | *MV* | Expertise. |

[a] http://www.ganttproject.biz.

#### 5.2.4. Collaborative interactions

We find three studies on collaborative interactions.

[Stein, 2004] records some participants' focus of attention and display them to other participants performing the same debugging task. It evaluates the usefulness of eye-movements as a cue for productive collaborations.

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 10 students and professionals | |
| *Eye-tracker* | Not mentioned | |
| *Variables* | *DV* | Accuracy and time |
| | *IV* | Stimuli with or without another person's gaze info |
| | *MV* | Not mentioned |

[Jermann and Nuüssli, 2012, Sharma et al., 2013] focus on pair programming tasks. [Jermann and Nüssli, 2012] analyses participants' focus of attention to observe how pair programmers share sequences of AOIs.

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 82 students | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Speech, selection, and gaze cross-recurrence |
| | *IV* | Selection type (individual, dual, and shared) |
| | *MV* | Not mentioned |

[Sharma et al., 2013] studies participants' interactions with one another and its impacts in a pair programming task.

**Table 3**
Types of artifacts used in eye-tracking studies.

| Code | | | | | Model | | | | English text | Other |
|---|---|---|---|---|---|---|---|---|---|---|
| Pascal | C/C++ | Java | C♯ | Python | UML | ER | Tropos | BPMN | | |
| 2 | 3 | 16 | 1 | 1 | 7 | 1 | 1 | 1 | 2 | 3 applications |

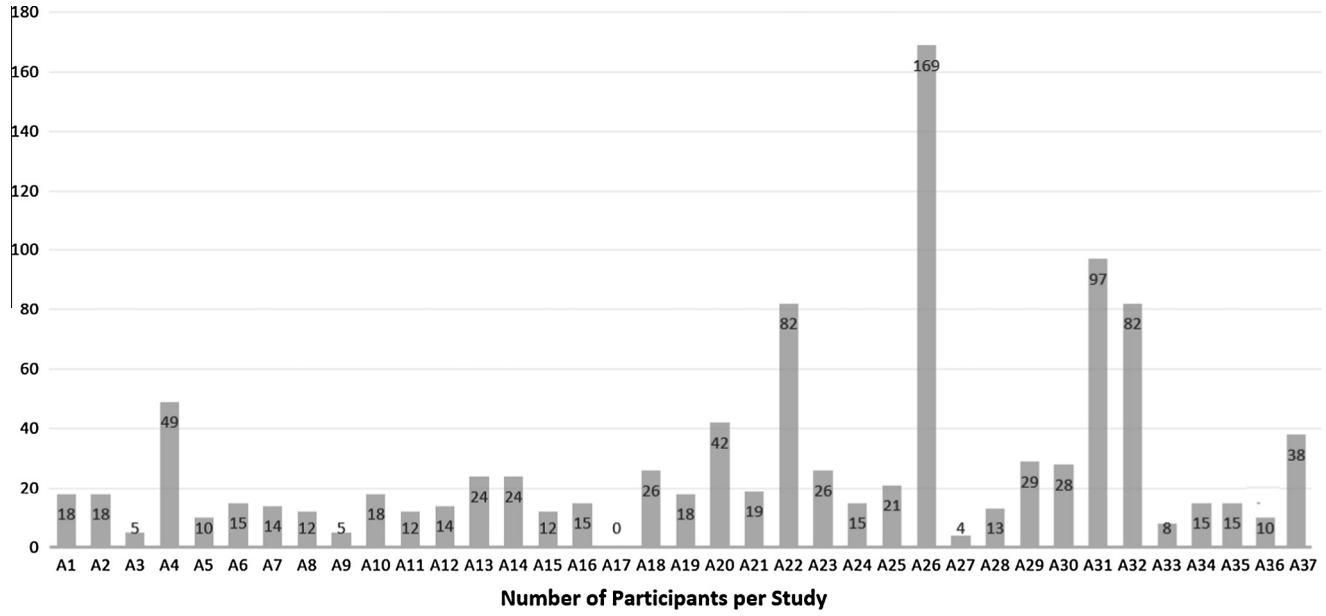


**Fig. 7.** Total number of participants for the selected papers.

**Table 4**
Summary of the results for model comprehension.

| Model comprehension |
|---|
| Experts vs. novices<br>1. Experts use extra information, e.g., color, layout, and stereotypes, more efficiently to browse UML diagrams [Yusuf et al., 2007]<br>2. Expertise impacts the speed and accuracy more than experience. Also, practitioners are more accurate than students [Soh et al., 2012] |
| Design patterns<br>1. The canonical representation of the Visitor design pattern has a negative impact [Jeanmart et al., 2009]<br>2. For composition and role tasks, Dong's representation is more efficient while Gamma's and Schauer's are more efficient for the participation task [Cepeda and Guéhéneuc, 2010]<br>3. Compared to the canonical form of the MVC and the Model-Delegate, MVC variant is easier to understand and participants have higher level of accuracy [De Smet et al., 2014] |
| UML representations:<br>1. Using similar visual notations enhances diagram comprehension and reduces effort [Yusuf et al., 2007]<br>2. Using multi-cluster layout leads to a significant improvement in the accuracy, time, and effort compared to an orthogonal layout [Sharif and Maletic, 2010] |

**Table 5**
Summary of the results for model comprehension (continued).

| Model comprehension |
|---|
| Navigation strategies<br>1. Vertical scanning is more efficient than horizontal scanning for ERD comprehension tasks [Cagiltay et al., 2013]<br>2. Scan-paths and relevant areas are connected with the participants' strategies and correctness when understanding BPMN models [Petrusel and Mendling, 2012]<br>3. The structure of representations leads participants to use different navigation strategies [Sharafi et al., 2013] |
| Others:<br>1. We expect that participants follow relations between classes when understanding class diagrams. Surprisingly, an eye-tracking study reports that participants do not seem to follow binary-class relationships [Guéhéneuc, 2006].<br>2. Two different browsing strategies are observed for experts and novices. Experts browse diagrams from their centers while novices use either top-down or bottom-up strategies [Yusuf et al., 2007].<br>3. We expect practitioners to be more efficient and accurate than students, yet students are faster for comprehension tasks [Yusuf et al., 2007].<br>4. Design patterns should improve design quality and, thus, could reduce comprehension effort. Results show that the Visitor design pattern does not reduce effort for comprehension tasks [Jeanmart et al., 2009].<br>5. Participants spend more time and effort working with graphical representations compared to textual ones [Sharafi et al., 2013]. |

**Table 6**
Summary of the results for code comprehension.

| Code comprehension |
|---|
| Experts vs. novices<br>1. Novices spend more visual attention on comments than experts [Crosby and Stelovsky, 1990]<br>2. Novices do not start by reading source code and look at the graphical representation first [Bednarik and Tukiainen, 2006]<br>3. Novices benefit from the use of camel case style regarding accuracy and effort while experts are less affected by the identifier style [Binkley et al., 2013]<br>4. The programming language impacts the amount of efforts spent by novices compared to experts while working with buggy pieces of source code [Turner et al., 2014] |
| Source code vs. natural text:<br>1. Participants have higher fixation times and regression rates when reading source code than when reading natural text [Busjahn et al., 2011].<br>2. Source code reading and comprehension are fundamentally different from natural text reading and comprehension [Binkley et al., 2013]. |
| Identifier styles<br>1. Expertise affects the impact of identifier styles on reading and comprehension [Binkley et al., 2013]<br>2. No difference exists between camel-case and underscore identifier styles regarding accuracy, time, and effort for comprehension tasks [Sharafi et al., 2012]<br>3. No significant difference is observed between male and female participants regarding time, accuracy, and effort when comparing camel-case and underscore identifier styles [Sharafi et al., 2012] |
| Navigation strategies:<br>1. Source code visualization techniques provide more important information for participants at early stages of comprehension and less at later ones [Bednarik and Tukiainen, 2006]<br>2. Visualization techniques improve participants' performance and help them to follow more systematics strategies during program comprehension [Duru et al., 2013]<br>3. Male and female participants follow different strategies when answering identifier recall questions [Sharafi et al., 2012] |

**Table 7**
Summary of the results for debugging.

| Debugging |
|---|
| Navigation strategies:<br>1. Patterns of retracing are observed when participants look back to the declarations of variables [Uwano et al., 2006]<br>2. Debugging strategies change in time. Participants mostly focus on output rather than source code at later stages [Bednarik and Tukiainen, 2008]<br>3. Repetitive patterns (going back and forth between textual and graphical representations) are observed for participants with less expertise and lower performance [Bednarik, 2012]<br>4. No pattern of retracing is observed when participants perform debugging tasks [Sharif et al., 2012]<br>5. Two different debugging strategies are deployed by participants to find bugs [Sharif et al., 2012]. |
| Performances:<br>1. Participants with higher performance in bug finding mainly use graphical representations rather than source code [Uwano et al., 2006]<br>2. Scanning time is highly correlated with the amount of visual effort spent on defect lines [Bednarik and Tukiainen, 2008]<br>3. Participants with lower performance switch more their attention between different representations [Hejmady and Narayanan, 2012]<br>4. Longer scanning times lead participants to find more bugs [Uwano et al., 2006, Sharif et al., 2012]<br>5. Using SeeIT 3D helps participants to have higher performance for overview tasks [Sharif et al., 2013]<br>6. Participants who use SeeIT 3D spend more times on bug fixing tasks [Sharif et al., 2013] |
| Others:<br>1. RFV and eye-tracker report different amounts of time allocations for the same task [Bednarik and Tukiainen, 2007]. |

**Table 8**
Summary of the results for collaborative interactions.

| Collaborative interactions |
|---|
| 1. Eye-movements is a valuable, useful cue for the task of program comprehension [Stein, 2004, Sharma et al., 2013]<br>2. Participants find bugs faster after watching the eye-movements of the developers in another group [Stein, 2004]<br>3. Pairs of participants who spend more focused time together are more accurate for comprehension tasks [Jermann and Nüssli, 2012] |

**Table 9**
Summary of the results for traceability.

| Traceability |
|---|
| 1. Participants prefer to use method names and comments compared to class names and variable names to perform comprehension tasks [Ali et al., 2012]<br>2. The SimpleGraph Gaze-Link algorithm provides traceability links with high recall, automatically [Walters et al., 2014] |

| | | |
|---|---|---|
| *Artifacts* | Java source codes | |
| *Participants* | 82 students | |
| *Eye-tracker* | Tobii 1750 | |
| *Variables* | *DV* | Gaze, speech, and time |
| | *IV* | Not mentioned |
| | *MV* | Not mentioned |

#### 5.2.5. Traceability

We find two studies on building or understanding traceability links among artifacts.

[Ali et al., 2012] uses an eye-tracker to understand how participants verify requirement traceability links. It reports the most used source code entities, i.e., method names, comments, variables, and class names.

| | | |
|---|---|---|
| *Artifacts* | Six Java source codes (19, 18, 19, 18, 24, 28 LOC) | |
| *Participants* | 26 students | |
| *Eye-tracker* | FaceLAB | |
| *Variables* | *DV* | Accuracy and time |
| | *IV* | Source code entity (class name, method name, variables, and comment) |
| | *MV* | Study level |

[Walters et al., 2014] describes SimpleGraph Gaze-Link, an algorithm that automatically recovers links between source code entities. It also presents and studies the usability of iTrace, a tool that supports link generation/recovery and maintenance/evolution.

| | | |
|---|---|---|
| *Artifacts* | A Java application | |
| *Participants* | 8 students and Faculty members | |
| *Eye-tracker* | Tobii X60 | |
| *Variables* | *DV* | Not applicable |
| | *IV* | Not applicable |
| | *MV* | Not applicable |

#### 5.2.6. Discussions

Table 3 summarizes the types of artifacts that have been studied for performing the eye-tracking studies in the selected papers: 16 studies out of 22, which use pieces of source code as stimuli, use the Java programming language. This observation shows the wide use of the object-oriented paradigm and Java. Based on Tiobe Programming Community Index,[16] Java is one of the first three most frequently-used programming languages in the last ten years. One of the studies use Pascal because it was published in 1990. Two other studies use C. One study compares C++ with Python regarding code comprehension.

UML diagrams are the most used modeling artifacts (70%), which is expected because UML has become the defacto standard for describing object-oriented design models. UML is also supported by a wide range of software tools.

In total, 1022 participants took part in the studies reported in the selected papers, as summarized in Fig. 7. The numbers of participants range from 5 to 169 per study whereas the mean value is 56.9. Out of all the selected papers, around 77% use students and faculty members as participants while the rest are either practitioners or no qualitative information about the participants is provided (2 papers out of 36).

[16] http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (Accessed: 2014-04-07).

### 5.3. How much have eye-tracking studies contributed to software engineering?

We look at our categories and discuss the results of the different studies in each category. Tables 4 and 5 summarize the results for model comprehension while Tables 6 and 7 summarize those for code comprehension and debugging, respectively. Finally, we present the summary of the results for collaborative interactions and traceability in Tables 8 and 9, respectively.

#### 5.3.1. Model comprehension

[Guéhéneuc, 2006] reports that developers begin by browsing class diagrams randomly and that, after finding relevant classes, they mostly focus on those. In addition, it reports that participants do not seem to follow binary-class relationships, e.g., inheritance and associations.

After assessing the impact of layout, color, and stereotypes on UML diagram comprehension, [Yusuf et al., 2007] reports that experts and novices use different strategies while navigating class diagrams. Experts use the information provided by coloring, layout, and stereotypes more efficiently than novices. Also, they tend to explore the diagrams from their centers whereas novices use either top-down or left-to-right strategies.

[Soh et al., 2012] discusses the differences between participants' status (practitioners or students) and expertise (experts or novices) when studying their productivity. It reports that expertise is the most important factor for the comprehension of a UML class diagrams in terms of speed and accuracy. In addition, its results show that practitioners are more accurate than students while students are faster.

[Jeanmart et al., 2009] reports that the Visitor design pattern does not reduce the participants' effort for comprehension tasks. Moreover, its canonical representation has negative impact on the comprehension of the class diagrams.

[De Smet et al., 2014] could not find any impact for the Observer and Composite design patterns although, by analyzing participants' scan-paths, it reports that novices systematically browse class diagrams while experts quickly gather relevant information. It also compares three different variants of the Model-View-Controller design pattern on comprehension: the *Canonical form of the MVC style* [7], the *Model-Delegate style*, and the *Model View Presenter style (MVP)* and reports that the MVP variant is usually easier to understand than the two other variants.

By comparing the impact of orthogonal and multi-clustered layouts on the comprehension of design patterns, [Sharif and Maletic, 2010] reports a significant improvement in the accuracy, time, and visual effort for the multi-cluster layout.

[Cepeda and Guéhéneuc, 2010] compares and reports that Dong's representation [10] is more efficient for Composition and Role Tasks while Gamma's [9] and Schauer's [8] are more efficient for the Participation Task.

[Petrusel and Mendling, 2012] targets the factors that impact the comprehension of business process models. It provides a formal notation for correlating the relevant regions and scan-paths. Moreover, it shows that the participants' answers given to questions pertaining to the models are correlated with the relevant regions.

[Cagiltay et al., 2013] proposes two metrics for ERD defect detection: Defect Detection Difficulty Level (DF) and Defect Detection Performance (PP). It reports that a defect with higher DF value leads to higher fixation durations. Moreover, participants who search the ERD vertically are more efficient (higher PP values) than those who perform horizontal search.

**Table 10**
Metrics for visual effort calculation based on the number of fixations.

| Visual effort and efficiency metrics<br>Number of fixation | | | |
|---|---|---|---|
| Names | Formulas | | Papers |
| Fixation Count (FC) | $FC = \text{Total number of fixations in AOI}$ | (1) | [Crosby and Stelovsky, 1990, Crosby et al., 2002, Uwano et al., 2006, Yusuf et al., 2007, Sharif and Maletic, 2010, Sharafi et al., 2012, Sharif et al., 2012, Sharif et al., 2013, Turner et al., 2014] |
| Fixation Rate (FR) | $FR = \frac{\text{Total Number of Fixations in AOI}}{\text{Total Number of Fixations in AOG}}$ | (2) | [Cepeda and Guéhéneuc, 2010, Sharif and Maletic, 2010, De Smet et al., 2014, Sharafi et al., 2012, Sharif et al., 2012, Binkley et al., 2013, Turner et al., 2014] |
| Spatial Density | $SD = \frac{\sum_{i=1}^{n} c_i}{n}$<br>$n$: number of fixations in the specific area (for one cell).<br>$c_i$: equal to 1 if the area number $i$ visited, otherwise 0 | (3) | [De Smet et al., 2014, Soh et al., 2012] |
| Convex hull area | Area of the smallest convex set of fixations | (4) | [Sharafi et al., 2012, Soh et al., 2012, Sharafi et al., 2013] |

[Sharafi et al., 2013] reports that participants spend more time and effort while working with a graphical representation than a textual representation of some requirements, although no significant difference is reported for accuracy and participants state that they prefer the graphical representation. Moreover, it reports that the spatial structure of the graphical representation facilitates the comprehension tasks and leads participants to follow two different navigation strategies (top-down and bottom up) to perform the comprehension task.

*5.3.2. Code comprehension*

[Crosby and Stelovsky, 1990, Crosby et al., 2002] study the impact of expertise on source code reading. [Crosby and Stelovsky, 1990] reports that novices pay more visual attention to comments than experts. Also, [Crosby et al., 2002] reports that experts pay more attention to beacons than novices while novices do not discriminate between different areas of the source code.

[Aschwanden and Crosby, 2006] focuses on participants' behavior during code reading. It suggests that any area of code that exhibits very long fixations, above 1,000 ms, is a beacon.

[Bednarik and Tukiainen, 2006] gives participants access to a graphical representation of source code as well as to the source code itself to perform comprehension tasks. It reports that novices do not start by reading source code and prefer to look at the graphical representation first. It reports that a graphical representation provides more important information at early stages of comprehension rather than at the later stages.

[Busjahn et al., 2011] reports that participants spend more fixation times and have higher regression rates (backward-directed eye-movements) when reading source code than natural text. This result shows that the higher complexity of the source code forces participants to change their focus of attention more frequently.

Moreover, participants spend significantly more time reading identifiers in source code than keywords, numbers, and operands.

[Busjahn et al., 2014] analyses attention distribution when reading source code. This paper confirms the results of the previous paper and reports that identifiers, operators, keywords, and literals receive the most attention, in that order, while separators receive almost none. It also claims that methods from research on natural-language text reading can be applied to source code with some modifications.

Regarding the impact of identifier styles on source code reading, [Sharafi et al., 2012] has not found any significant differences between camel case and underscore. Moreover, it reports no significant differences between male and female participants regarding time, accuracy, and effort. However, it reports that male and female participants follow different strategies while answering recall questions. Female participants spend more time analyzing different options while male participants quickly decide on an answer. Yet, both groups have the same accuracy and the time difference between male and female participants is not significant either.

[Binkley et al., 2013] presents contradictory results using four different studies comparing camel case and underscore identifier styles regarding participants' time and accuracy. Moreover, this paper compares natural text reading with source code reading and suggests that these two activities are different, in agreement with [Busjahn et al., 2011]. It also shows that participants rapidly understand code independent of style and that the choice of the identifier style has less impact on experts than novices. Novices benefit from the use of camel case.

[Duru et al., 2013] reports that the visualization technique provided by NDEPEND improves participants' accuracy and task completion-time. Its results also indicate that the visualization helps participants find relevant entities and–or code snippets and follow more systematic strategies.

[Turner et al., 2014] reports that there is no significant difference between C++ and Python regarding time and accuracy. However, for buggy source code, participants spend more visual effort on Python code. This paper also reports that there is a significant difference between novices and experts with respect to accuracy and effort on buggy source code.

[Fritz et al., 2014] can predict task difficulty with 64.99% precision and 64.58% recall. Its results also demonstrate that probabilistic classifiers, such as Naive Bayes classifiers, can be trained with various biometric data to predict task difficulty. To validate the task difficulty predicted by a Naive Bayes classifier using psycho-physiological measures, the paper compares its results with NASA TLX scores and confirms a high correlation.

**Table 11**
Metrics for visual effort calculation based on the duration of fixations.

| Visual effort and efficiency metrics<br>Duration of fixations | | | |
|---|---|---|---|
| Names | Formulas | | Papers |
| Average Fixation Duration (AFD) | $\mathrm{AFD(AOI)} = \frac{\sum_{i=1}^{n}(ET(F_i) - ST(F_i)) \text{ in AOI}}{n}$<br>$ET(F_i)$ and $ST(F_i)$: the end time and start time for fixation $F_i$ $n$: the total number of fixations in a given AOI | (5) | [Crosby and Stelovsky, 1990, Cepeda and Guéhéneuc, 2010, Sharif and Maletic, 2010], [21], [Soh et al., 2012, Binkley et al., 2013, Sharafi et al., 2013] |
| Ratio of *ON_target All_target* Fixation Time (ROAFT) | $\mathrm{ROAFT} = \frac{\sum_{i=1}^{n}(ET(F_i) - ST(F_i)) \text{ in AOI}}{\sum_{j=1}^{n}(ET(F_j) - ST(F_j)) \text{ in AOG}}$ | (6) | [Bednarik and Tukiainen, 2006, Cepeda and Guéhéneuc, 2010, Bednarik, 2012, Sharif et al., 2012, Binkley et al., 2013] |
| Fixation Time (FT) | FT = Total duration of all fixations in AOI | (7) | [Crosby and Stelovsky, 1990, Crosby et al., 2002, Uwano et al., 2006, Bednarik, 2012, Ali et al., 2012, Petrusel and Mendling, 2012, Busjahn et al., 2014, Rodeghero et al., 2014] |
| Average Duration of Relevant Fixations (ADRF) | $\mathrm{ADRF} = \frac{\text{Fixations Duration of Relevant AOIs}}{\text{Total Number of Relevant AOIs}}$ | (8) | [Jeanmart et al., 2009, De Smet et al., 2014] |
| Normalised Rate of Relevant Fixations (NRRF) | $\mathrm{NRRF} = \frac{\mathrm{ADRF}}{\frac{\text{Fixation Duration of All AOIs}}{\text{Number of All AOIs}}}$ | (9) | [Jeanmart et al., 2009, De Smet et al., 2014, Soh et al., 2012] |

**Table 12**
Metrics for visual effort calculation based on saccades.

| Visual effort and efficiency metrics<br>Saccades | | | |
|---|---|---|---|
| Names | Formulas | | Papers |
| Number of saccades | Total number of saccades | (10) | [Fritz et al., 2014] |
| Saccade duration | Total duration of saccade | (11) | [Fritz et al., 2014] |

[Rodeghero et al., 2014] shows that participants spend more visual attention on method signatures than method invocations and more visual attention on and invocations than control flow.

### 5.3.3. Debugging

[Uwano et al., 2006, Sharif et al., 2012] report that participants who spent sufficient time scanning source code tend to find defects more efficiently. Longer scanning means that participants carefully read the code and find suspicious candidate code lines. [Sharif et al., 2012] observes two different debugging strategies to find bugs: (1) by finding something odd in a file or (2) by comparing information provided through different representations. [Uwano et al., 2006] reports repetitive patterns of going back and forth between code and a graphical representations for novices. The experts change their strategies and focus on the output at later stages while novices mostly switch between the two. [Uwano et al., 2006] finds some patterns of retracing (looking back at the declarations of the variables). However, [Sharif et al., 2012] did not notice these patterns.

[Bednarik and Tukiainen, 2008] partially replicates [Bednarik and Tukiainen, 2007] and reports that the amount of effort that participants spend on defective lines and their defect detection times are highly correlated with scanning. [Bednarik and Tukiainen, 2007] compares the data reported by both a RFV approach and an eye-tracker. The results show that there is a difference between the reported amount of time for the same task. Moreover, the blurring of the RFV approach interferes with the experts' strategies. [Bednarik and Tukiainen, 2008] reanalyses the data of [Bednarik and Tukiainen, 2007] and reports that eye-movement patterns during debugging change in time. At later stages of debugging, experts change focus their attention mostly on the output of the programs rather than on source code.

[Romero et al., 2002, Romero et al., 2003, Bednarik, 2012, Hejmady and Narayanan, 2012] provide both source code and graphical representations of source code to participants. [Bednarik, 2012] reports that participants with higher performance mainly use the graphical representation although they frequently switch between the two. [Hejmady and Narayanan, 2012] reports that participants with lower performance switch more their attention, even at the end of the debugging session, compared to more successful ones.

[Sharif et al., 2013] reports that SeeIT 3D leads to higher performance for participants who perform overview tasks. However, working with SeeIT 3D takes significantly longer during bug fixing tasks.

### 5.3.4. Collaborative interaction

The results of [Stein, 2004] support the usefulness of eye-movements as cues for productive collaboration in problem solving. They show that participants in a second group find bugs more quickly after watching the eye-movements of successful participants in a first group.

[Jermann and Nüssli, 2012] reports that participant pairs who are actively working with each other share a high level of

**Table 13**
Metrics for visual effort calculation based on scan-paths.

| Visual effort and efficiency metrics<br>Metrics based on scan-paths | | | |
|---|---|---|---|
| Names | Formulas | | Papers |
| Transitional Matrix | $\mathrm{TM} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} c_{i,j}}{n \cdot n}$<br>$n$: number of fixations in the specific area (for one cell). $c_i$: equal to 1 if the area number $i$ visited, otherwise 0 | (12) | [De Smet et al., 2014] |
| Attention switching frequency | The number of switches between AOIs | (13) | [Bednarik and Tukiainen, 2006, Bednarik and Tukiainen, 2008] |
| Scan-path precision | $\frac{\mathrm{SP} \cap \mathrm{RR}}{\mathrm{SP}}$<br>SP: number of AOIs that are visited (fixated).<br>RR: number of relevant AOIs that are visited. | (14) | [Petrusel and Mendling, 2012] |
| Scan-path recall | $\frac{\mathrm{SP} \cap \mathrm{RR}}{\mathrm{RR}}$<br>SP: number of AOIs that are visited (fixated).<br>RR: number of relevant AOIs that are visited. | (15) | [Petrusel and Mendling, 2012] |
| Scan-path *F*-measure | $2 * \frac{\mathrm{SPP} * \mathrm{SPR}}{\mathrm{SSP} + \mathrm{SPR}}$<br>SP: number of AOIs that are visited (fixated).<br>RR: number of relevant AOIs that are visited. | (16) | [Petrusel and Mendling, 2012] |

cross-recurrences of eye-movements. Moreover, [Sharma et al., 2013] observes that the pairs who spend more time "focused together" have higher levels of program understanding compared to the rest.

#### 5.3.5. Traceability

[Ali et al., 2012] reports that participants have different preferences for different source code entities and prefer method names and comments. It also proposes a new weighting scheme enhanced by adding the results of an eye-tracking study and reports that this scheme statistically improves the accuracy of a IR-based technique.

[Walters et al., 2014] reports that the SimpleGraph Gaze-Link algorithm provides traceability links automatically with high recall. The use of iTrace to automatically apply the proposed algorithm and generate the traceability links is also promising.

### 5.4. How have researchers used eye-trackers to collect and visualize quantitative measurements?

Different studies propose and–or use several metrics based on eye-movement data provided by eye-trackers to measure and calculate the amount of visual effort required to perform the task. These metrics divide in (1) metrics based on the numbers of fixations, (2) metrics based on the durations of fixations, (3) metrics based on saccades, and (4) metrics based on scan-paths.

In addition, various visualizations techniques have been used to display eye-movements. Visualizing eye-movements helps performing qualitative analysis to better understand participants' behavior. A same metric or visualization technique may be used in different studies with a different name. In the following, we use the most common name to refer to a metric and provide its other names in parentheses along with references to the related papers.

#### 5.4.1. Visual effort and efficiency metrics

Tables 10–13 summarize all metrics used in the selected papers to measure the amount of visual effort and efficiency along with equations and a related papers.

*Metrics based on the number of fixations*

- **Fixation Count (FC)** is measured by counting the number of fixations on specific AOIs or the whole stimulus. Higher number of fixations for the whole stimulus indicates less efficient search for finding relevant information [46].
- **Fixation Rate (FR)** is defined as the number of fixations on specific AOIs divided by the total number of fixations on the Area of Glance (AOG), which can be the whole stimulus or a set of AOIs. Goldberg et al. [46] proposed this metric in 1999 and called it the Ratio of On_target:All_target Fixations (ROAF). Interpreting fixation rate is dependent on the task being performed [48]. For browsing/encoding tasks, a higher fixation rate for a specific AOI indicates that the participants show a great interest in that AOI [28]. Yet, it could indicate that the this area is difficult to encode [28]. For search task, smaller rates indicate lower efficiency because the participants spend more time and effort to find the relevant areas required to perform their task [48].
- **Fixation Spatial Density** was proposed by Goldberg et al. [46] and used in several studies. If we divide a stimulus into a grid, the spatial density index is equal to the number of cells containing at least one fixation, divided by the total number of cells. Fixations that are concentrated in a small area indicate an

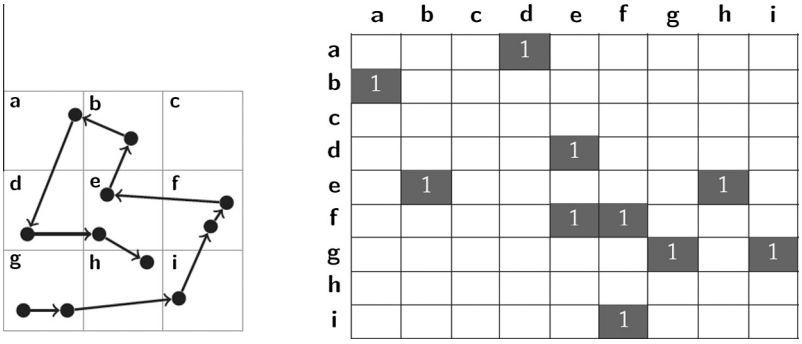


**Fig. 8.** Example of scan-path and corresponding transition matrix [De Smet et al., 2014].

efficient search, which is highly focused. [De Smet et al., 2014] explains how to use their tool, Taupe, to compute this metric along with some examples.

- **Convex hull Area** represents the smallest convex set of fixations that contains all the participants' fixations [46]. A smaller value indicates that the fixations are closed together and that the participants spend less effort to find relevant areas.

*Metrics based on the duration of fixations*

- **Average Fixation Duration (AFD)** is correlated with cognitive processes [39,46]. This metric is computed as shown in Eq. (5), Table 11. Longer fixations show that participants spend more time analyzing and interpreting the content of the AOIs while working on their tasks. Therefore, they are spending more mental effort to solve their tasks. This metric can be computed for either a whole stimulus or each AOI separately.
- **Ratio of "On_targer:All_target" Fixation Time (ROAFT)** or Proportional Fixation Time (PFT) is computed as the ratio of the fixation duration on an AOI to the overall fixation duration on a stimulus. Similar to the FR metric, Goldberg et al. [46] proposed this metric in 1999 and explained that a smaller ratio indicates a lower efficiency, because the participants spend a lot of time searching the stimulus to find a relevant area. In addition, according to Just et al. [37], the duration of the fixations on a specific area can have two different meanings: (1) it is hard for participants to extract information or (2) participants are more engaged by the content of the area.
- **Fixation time (FT)** is computed by calculating the total time of all fixations for a specific AOI or the whole stimulus. It is also referred to as gaze or fixation cluster. It can be used to compare the amount of attention on different AOIs or stimuli [48].
- **Average Duration of Relevant Fixations (ADRF)** is the total duration of the fixations for relevant AOIs. The same measure has been proposed for non-relevant AOIs and is called "Average Duration of Non-Relevant Fixations (ADNRF)".
- **Normalised Rate of Relevant Fixations (NRRF)** is proposed by Jeanmart et al. [Jeanmart et al., 2009] to compare two or more diagrams with each other regarding the impact of the Visitor design pattern.

*Metrics based on saccade*

- **Number of saccades** represents the total number of saccades in a stimulus. A higher number of saccades indicates more searching [48].
- **Saccade duration** represents the total duration of all saccades for one or a set of AOIs.

[Fritz et al., 2014] explain that the number and the duration of saccades are related to the mental workload and can provide insight into the influence of the artifacts on the participants' cognitive processes. Previous studies in usability also used saccades amplitude and the number of regressions [48] to compute a visual effort. Saccade amplitude indicates meaningful load cues. The higher the saccade amplitude, the lower the mental effort [48]. Saccades are usually rightward (progressive). However, sometimes they may be backward (leftward in text reading) or regressive, which indicates difficulties in understanding some text [27] or the presence of less meaningful cues in the stimulus [48].

*Metrics based on scan-paths*

- **Transitional matrix** is a tabular representation that shows the frequency of transitions between defined AOIs [23]. Eq. (12) in Table 12 presents this metric. In addition to the search area, this metric also considers the temporal order of the search by detecting movements over time [46]. The density of a transition matrix is computed as the number of non-zero matrix cells divided by total number of cells. [De Smet et al., 2014] describes how to compute and use a transitional matrix. Fig. 8 shows an example of a scan-path on the display grid and its transition matrix with a spatial density of 12% (10 cells out of 81 are filled). Frequent transitions, which produce a dense transition matrix (with most cells filled with at least one transition), indicate extensive search with inefficient scanning on a stimulus. A sparse matrix points out more efficient and directed search [46].
- **Attention switching frequency** is the number of switches between two specific AOIs. This metric is used in [Bednarik and Tukiainen, 2006, Bednarik and Tukiainen, 2008].
- **Scan-path precision (SPP)** was proposed by [Petrusel and Mendling, 2012] as the percentage of relevant AOIs visited from all defined AOIs in the stimulus.
- **Scan-path recall (SPR)** was proposed by [Petrusel and Mendling, 2012] as the percentage of relevant retrieved AOIs from all relevant AOIs in the stimulus.
- **Scan-path f-measure (SPF)** was proposed by [Petrusel and Mendling, 2012] as the harmonic mean of SPP and SPR. It is the percentage of relevant retrieved AOIs from all relevant AOIs in the stimulus.

Previous studies in usability research also use the scan-path duration and scan-path length, which are proxy of the search efficiency. Longer-lasting scan-paths indicate that the participants spend more time on each AOI before going to the next one, which means a less efficient scanning [46]. A longer scan-path indicates that the participant performed more attention switching between different AOIs. It also indicates that the participant explored the stimulus more, which means a less efficient searching [45].

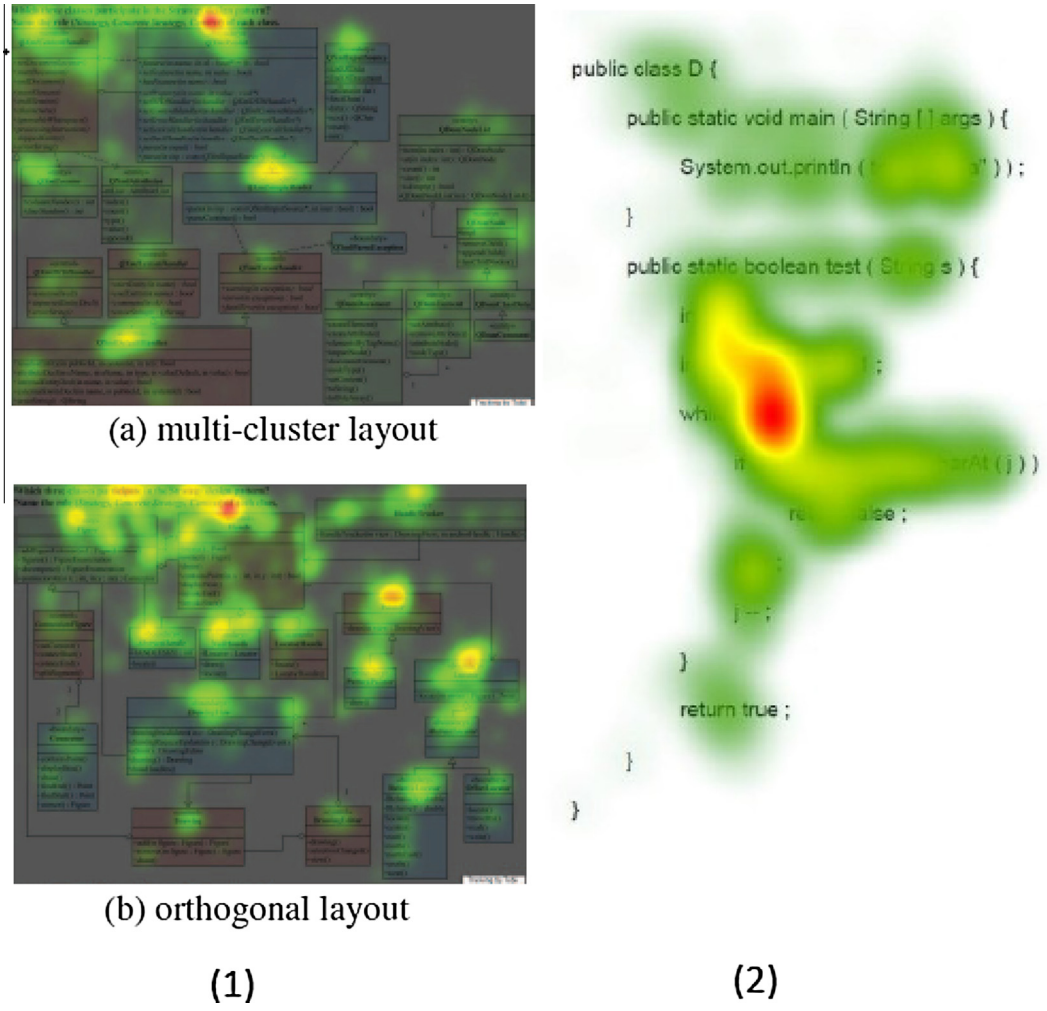


**Fig. 9.** (1) shows the heat-map of a participant working with (a) multi-cluster and (b) orthogonal layouts [Sharif and Maletic, 2010]. (2) presents areas of source code that attracts higher interests [Busjahn et al., 2011].

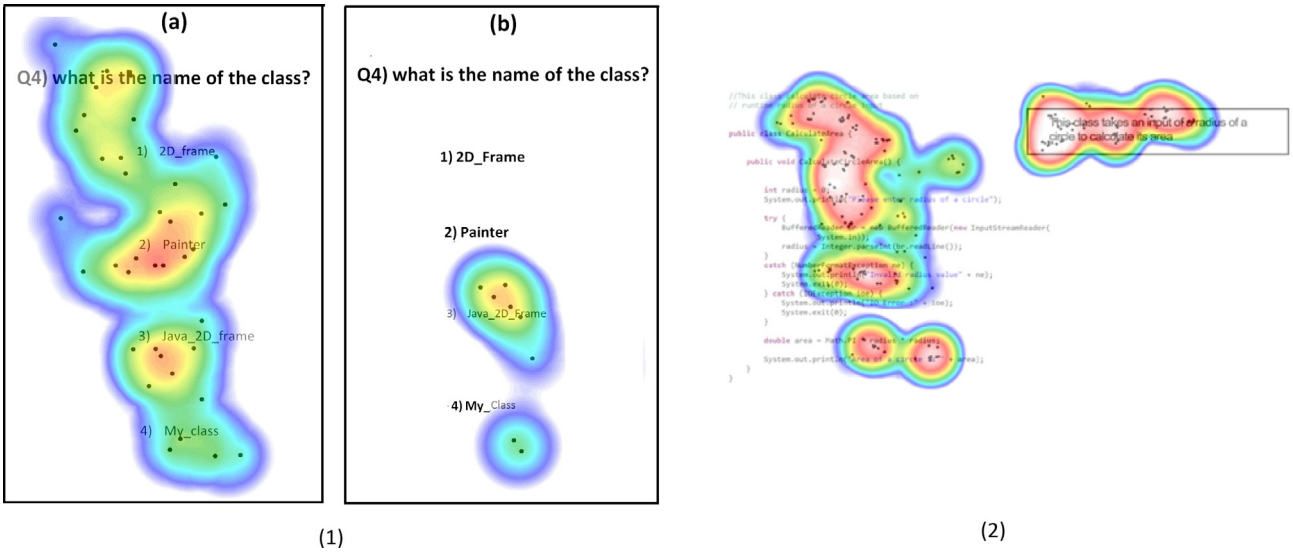


**Fig. 10.** (1) Shows a heat-map of (a) a female participant and (b) a male participant [Sharafi et al., 2012]. (2) Shows a heat-map that presents the cumulative fixations of a participant on different source code entities, including class name, method name, variables, and comment [Ali et al., 2012].

In addition, several techniques also exist to describe, compare, and analyze scan-paths:

- **Edit distance** is based on the Levenshtein algorithm [36], which calculates the editing cost of transforming one string into another using three basic operations (insertion, deletion, and substitution). If we consider a cost of 1 for each operation, the Levenshtein distance metric is the minimum editing cost. [De Smet et al., 2014] uses the Levenshtein distance and reports that the average distance among novices' scan-paths is lower than that of experts. However, the edit distance does not take into account the duration of different fixations and treats all fixations equally. Fixation duration plays an important role in analyzing eye-tracking data [35]. Consequently, new techniques have been proposed to consider also the duration of fixations.
- **Sequential PAttern Mining (SPAM)** was proposed by Ayres et al. [49] and uses a depth-first search strategy for mining scan-paths. [Hejmady and Narayanan, 2012] uses this technique and takes into account fixation durations by categorizing fixations as short (less than 500 ms) and long (higher than 500 ms). The threshold (500 ms) was chosen based on a review of all the participants' eye-movements. Its results confirm that experts look at the program output more frequently than novices. Moreover, participants who were familiar with the IDE switched their attention between the source code and the graphical visualization more often than those who were less familiar with the IDE.
- **ScanMatch** was defined by Cristino et al. [50], who propose the ScanMatch algorithm to compare scan-paths based on the Needleman–Wunsch algorithm used in bio-informatics to compare DNA sequences. ScanMatch assigns a character to represent each AOI and uses the temporal binning to repeat the letters corresponding to the AOIs in a way that their quantities are proportional to the fixation durations. This technique also computes a matching score to show exactly how much two scan-paths are similar. [Sharafi et al., 2013] uses ScanMatch and calculates the similarity of participants' scan-paths while working with graphical representations.

*5.4.2. Visual gaze behavior*

Eye-tracking studies in software engineering have used so far three types of visualization techniques: heat-map, gaze plot, and color-coded attention allocation map.

- **Heat-map** is a color spectrum that represents the intensity (number and duration) of fixations. The colors red, orange, green, and blue indicate the number of fixations from highest to lowest. A heat-map is superimposed on top of a stimulus and highlights areas where participants have been looking. Fig. 9 shows heat-maps that are presented and discussed in [Sharif and Maletic, 2010, Busjahn et al., 2011] while Fig. 10 shows heat-maps presented in [Sharif and Maletic, 2010, Busjahn et al., 2011]. [Sharif and Maletic, 2010] investigates

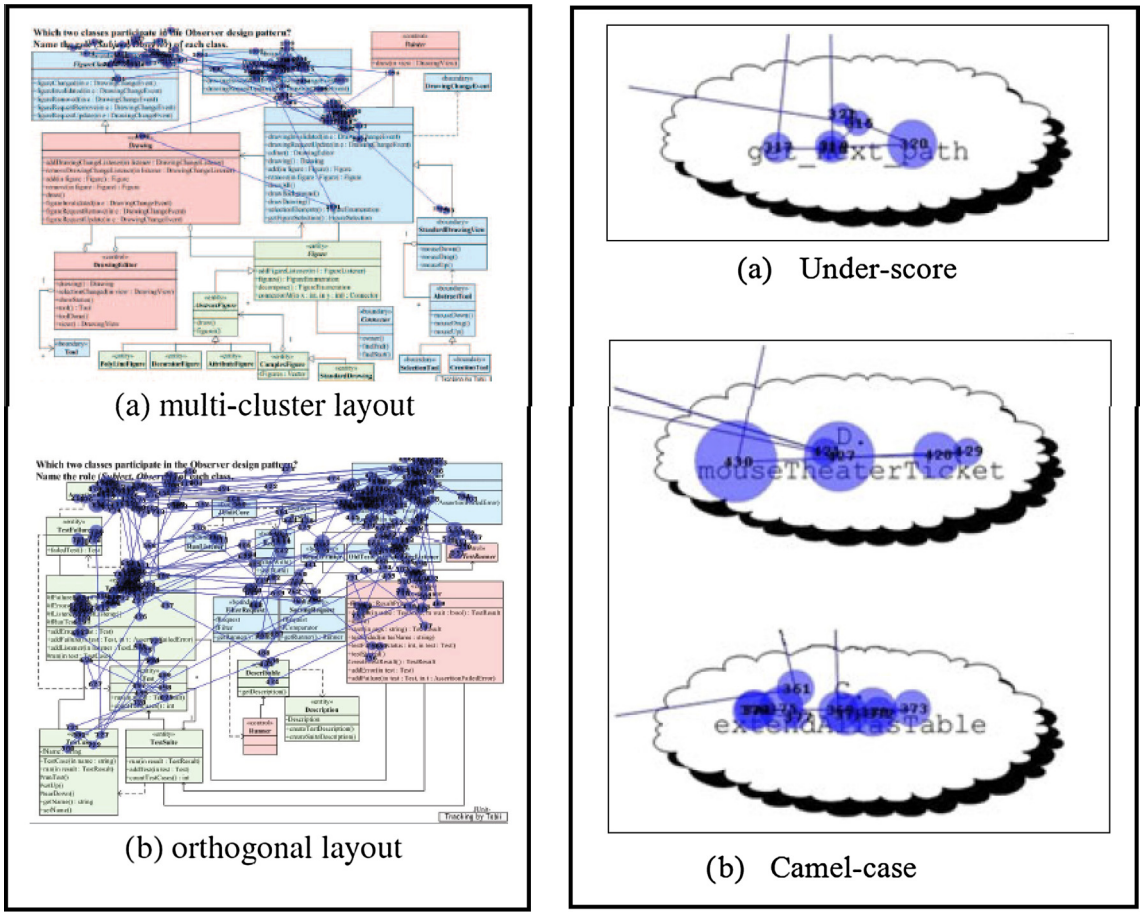


**Fig. 11.** (1) Shows the gaze plots of an expert for the Observer pattern for orthogonal and multi-cluster layouts [Sharif and Maletic, 2010]. (2) Shows gaze plots for underscore and two camel-case 3-word code identifiers [Binkley et al., 2013].

the impact of different layouts on the comprehension of design patterns. It uses two heat-maps to visualize experts' eye-movements while working with two layouts (multi-cluster vs. orthogonal), separately. The heat-maps show that experts spent more time on the classes participating in the design patterns while working with a multi-cluster layout comparing to the orthogonal layout. [Busjahn et al., 2011] uses heat-maps to visualize the areas in the source code that attracts higher interest (higher fixation duration). [Ali et al., 2012] creates heat-maps for different participants for code comprehension tasks. The heat-maps show that large numbers of fixations are found on the method names, comments, variable names, and class names in decreasing order of importance. [Sharafi et al., 2012] uses heat-maps to show the differences among the men and women participating in code comprehension and recall tasks. The heat-maps show that men and women use different strategies for answering the multiple-choice questions. The heat-maps for women show fixations scattered through all choices while, for men, the fixations are mainly focused on one choice.

- **Gaze plot** provides a static view of the eye-gaze data. It is also useful to visualize scan-paths. Using a gaze plot, a scan-path is shown as a directed sequence of fixations, where a fixation is illustrated using a circle whose radius represents the durations of the fixation. [Sharif and Maletic, 2010] uses a gaze plot to compare experts and novices performing design pattern comprehension tasks. Experts' gaze plots (as presented in Fig. 11) show that they are looking at attributes and methods to find answers, while novices mainly focus on class names. [Binkley et al., 2013] compares camel-case and underscore identifer styles for code comprehension. The gaze plots (as presented in Fig. 11) show that developers put a larger number and longer fixations for the camel-case style. [Jermann and Nüssli, 2012] uses an enhanced version of a gaze plot called a gaze cross-recurrence plot.
  Using this plot, it determines if two participants who are working on the same stimulus look at the same area at roughly the same time and in the same sequence.
- **Color coded attention allocation map** is based on either the numbers of fixations or the total durations of all fixations for some set of words. This map allocates a color to each word separately from a color spectrum that starts from light green (lowest attention level) going through dark green and dark red while finishing with light red (highest level of attention). [Busjahn et al., 2011] uses such a map based on the number of fixations per word as depicted in Fig. 12 to show different parts of source code that attracts different levels of attention and consequently time.

#### 5.4.3. Discussions

Because cognitive processes happen during fixations, the majority of eye-tracking studies use metrics that are calculated using either the numbers or durations of fixations. Only five studies use saccade and scan-path metrics. Four and three studies use heat-maps and gaze-plots, respectively, and only one study uses color-coded attention allocation maps. Heat-maps are more popular than color-coded maps because they summarize fixations on top of the stimuli and make it easy to see the locations and intensities of the fixations. Moreover, the color spectrum of heat-map shows how participants scan the stimuli and what are their most preferred areas in the stimuli.

Although several approaches have been proposed for the quantitative comparison of scan-paths [56,57], only 10% of selected studies use these quantitative metrics to measure and compare scan-paths, possibly because scan-paths are inherently complex and there are major computational challenges in scan-path modeling and comparison [57].

However, the results of applying available scan-path comparison techniques as shown in few selected studies [De Smet et al., 2014, Hejmady and Narayanan, 2012, Sharafi et al., 2013] are promising. These selected studies compares participants based on their scan-paths. [Hejmady and Narayanan, 2012] uses an environment that displays source code and the dynamic and the static visualizations of source code together. After comparing scan-paths using SPAM [49] quantitatively, [Hejmady and Narayanan, 2012] reports a prominent attention switching pattern (short attention switching between code and dynamic visualization) for all participants.

[Sharafi et al., 2013] compare participants' viewing strategies while working with Tropos graphical representation. Based on the similarity scores provided by ScanMatch [50], it categorizes scan-paths into two different groups. By visualizing these groups, it shows that they represent two different viewing strategies: top-down or bottom-up.

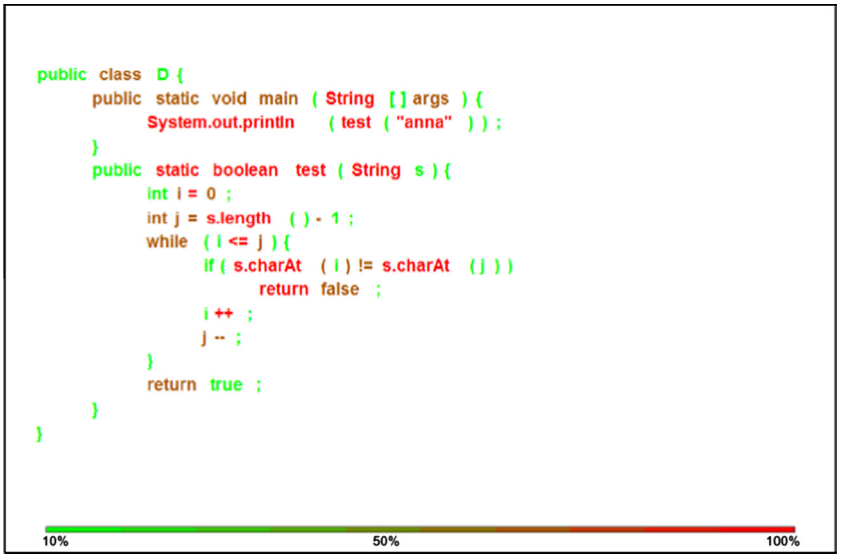


**Fig. 12.** Color-coded attention allocation map based on the number of fixations per word [Busjahn et al., 2011].

### 5.5. What are the limitations of current eye-tracking studies?

Researchers wanting to perform eye-tracking studies must consider limitations in the uses of eye-trackers in software engineering and the impact of these limitations on the collected data and possible analyses and interpretations.

#### 5.5.1. Eye-tracking technology

The eye-tracking technology comes with the following intrinsic limitations.

**Precision and accuracy.** The accuracy values reported in eye-trackers manuals are measured in ideal situations in which (1) cameras have been calibrated just before the measurement and (2) participants do not move their heads or have eye-wears. Sometimes, manufacturers even use a set of artificial eyes to avoid any head movements.

Also, manufacturers eliminate any obstacle that could interrupt the normal path of (infrared) light, in particular eye-wears, including hard contact lenses and eye-glasses.

Yet, researchers may face difficulties with participants with large pupils or "crazy eyes", i.e., whose eyelids partially hide the pupils and make them difficult to detect [48]. 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' eye-movements. Yet, eye-trackers mostly perform calibration based on participants' both eyes and use the average display location to improve accuracy. If AOIs are located towards the edges of the calibrated area, the calibration error is considerable [25].

To mitigate the impact of the limitations pertaining to precision and accuracy, researchers could define AOIs large enough to capture all relevant fixations. Some previous studies also use larger font sizes to present text and models to compensate for lack of precision. Researchers must also calibrate the eye-trackers on a regular basis, in addition to calibrating for every participant just before starting to perform the tasks.

Drift is also a limitation of eye-tracking technology. Drift is the gradual decrease in time of the accuracy of the eye-tracking data, when compared to the true coordinates of the eye-movements, which indicates the deterioration of calibration over time [30]. Changes in the physiology of the eye in time, e.g., changes in wetness, cause drift. To reduce the impact of drift, the light conditions of the experiment environment must remain stable and there must be equal light intensity between calibration and experiment stimuli [30]. Also, the tasks should have reasonable time durations to avoid fatigue. The calibration procedure must be repeated regularly to maintain the quality of the results.

**Hawthorne effect.** While performing an eye-tracking study, a researcher is responsible for providing guidance to the participants, calibrating the eye-tracker, and checking its recording to ensure that the eye-tracker does not stop tracking the eyes. The researcher's presence may bias the results due to the Hawthorne effect, also called the observer's effect, because participants feel that they are being watched. The selected papers mitigate the impact of this effect by sitting researchers away from the participants. The papers also report minimal interaction or face-to-face contact between researchers and participants. They state that researchers should not help or steer participants in any way and that researchers should does not check the participants' behavior during the study. They should only focus on the quality of the recorded data. Finally, at the beginning of a study, researchers must explain to the participants that the eye-tracker records eye-movement data anonymously and that no video is recorded.

#### 5.5.2. Data analysis

Eye-trackers produce huge amount of data, whose analyses are complex. Using a set of tools to automatically filter and analyze the data is necessary to save time and prevent human errors caused by manual analyses. Several tool have been used by researchers to analyze high volumes of data generated by eye-trackers. For example, Taupe [De Smet et al., 2014] and OGAMA[17] are open-source software systems designed for analyzing eye-tracking data. They support many commercial eye-trackers, including FaceLAB and Tobii eye-trackers. OGAMA also supports mouse movements collected from slide-shows.

Another tool is the eye-tracking gaze visualizer [59], which displays eye-movements on top of the video captured from the screen during the tasks and which provides many features, including object detection and repetitive pattern exploration.

Defining a set of AOIs is usually a required step before analyzing eye-tracking data. Yet, currently, there are no detailed best-practices for defining AOIs. Goldberg et al. [25] propose a set of general guidelines for defining AOIs and report that the padding around AOIs is dependent on the task and artifacts. For example, for text, because fixations are usually located closely to each other, less padding is required; for a graphical notation, more padding could be considered so that the AOI captures all relevant fixations.

#### 5.5.3. Task and material selection

To carry out eye-tracking study, researchers must ask participants to perform a set of well-defined tasks so that the recorded eye-movements are correctly associated with their cognitive processes [37,48]. Task definition also identifies the type of artifacts of interest in the study. Researchers must eliminate any visual distractions (e.g., colorful or moving objects) to avoid contaminating eye-movement data [44,48].

The selected papers use small and easy-to-read pieces of source code, texts, or models that can fit on one screen. By fitting a stimulus in a single screen, researchers avoid the scrolling and–or traversing between different pages so that eye-movement data can be accurately and unambiguously match to positions on the screen and, therefore, well-defined parts of the stimuli to measure visual effort. Therefore, due to this single-screen limitation, there is no previous study with realistic, interactive tasks.

Some researchers tackle the single-screen limitation by proposing new recording techniques that support scrolling. Lankford [26] presents a tool, called GazeTracker, that records keystrokes and mouse clicks and movements and saves and displays the correct location of eye-movements even if scrolling happens. The authors presents the applicability of the tool for Web-page viewing analysis. iTrace [58, Walters et al., 2014] also supports horizontal and vertical scrolling while recording the correct locations of eye-movements. It also is integrated in Eclipse[18] IDE.

Yet, for software engineering tasks, in addition to scrolling, developers usually also use features in integrated/interactive development environments (IDEs) that change the screen content without scrolling (e.g., click on package controller content in Eclipse and select another file to read). Even, resizing the window will be problematic and must be considered. These issue still impose some limits on choosing realistic materials and tasks for eye-tracking experiments in software engineering that we ask the experimenters to bear in mind before considering eye-tracking technology.

[17] http://www.ogama.net/.
[18] https://eclipse.org/.

**Table 14**
List of eye-trackers used in the selected papers with their accuracy values and sampling rates.

| Eye-tracker | Manufacturer | Accuracy (°) | Recording rate (Hz) | Number of studies | Papers |
|---|---|---|---|---|---|
| Tobii 1750 | Tobii Technology | 0.5 | 30 or 60 | 11 | [Bednarik and Tukiainen, 2006, Bednarik and Tukiainen, 2007, Yusuf et al., 2007, Bednarik and Tukiainen, 2008, Sharif and Maletic, 2010, Bednarik, 2012, Jermann and Nüssli, 2012, Sharif et al., 2012, Binkley et al., 2013, Sharma et al., 2013, Duru et al., 2013] |
| Tobii X60 | Tobii Technology | 0.5 | 60 | 3 | [Hejmady and Narayanan, 2012, Sharif et al., 2013], [58] |
| Tobii T120 | Tobii Technology | 0.4 | 120 | 2 | [Busjahn et al., 2011, Busjahn et al., 2014] |
| Tobii TX300 | Tobii Technology | 0.5 | 300 (set to 120) | 2 | [Fritz et al., 2014, Rodeghero et al., 2014] |
| FaceLab | Seeing Machines | 0.5 | 60 | 3 | [Ali et al., 2012, Sharafi et al., 2012, Sharafi et al., 2013] |
| Eye-Link II | SR Research[a] | 0.25–0.5 | 500 pupil only | 5 | [Guéhéneuc, 2006, Jeanmart et al., 2009, Cepeda and Guéhéneuc, 2010, De Smet et al., 2014, Soh et al., 2012, Cepeda and Guéhéneuc, 2010, De Smet et al., 2014, Soh et al., 2012] |
| ASL | Applied Science Laboratories | 0.5 | 50 or 60 | 2 | [Crosby et al., 2002, Aschwanden and Crosby, 2006] |
| EMR-NC | NAC Image Technology Inc.[b] | 0.3 | 30 | 1 | [Uwano et al., 2006] |

[a] http://www.eyelinkinfo.com/.
[b] http://www.nacinc.jp/.

#### 5.5.4. Experimental setting

Settings and the environment in which the participants perform their tasks are also important. All of the previous studies have been performed in quiet laboratories to avoid distractions but also because it is difficult to reach practitioners for eye-tracking studies. Practitioners have little spare time, may have concerns about intellectual property. Also, few eye-trackers are really portable.

Different studies use different settings for the eye-trackers and do not always explain their choices precisely in the published papers. Thus, eye-tracking studies are difficult to replicate. We encourage researchers to present precisely all the details related to their setting to allow replicating and comparison between studies.

Most previous papers recommended to perform several pilots studies to identify and fix any problems before collecting eye-tracking data.

#### 5.5.5. Participant selection

Most of eye-tracking studies have been performed in research laboratories because researchers do not have easy access to practitioners to perform their studies. For those studies in which no comparison between experts and novices has been provided, such as [Ali et al., 2012, Sharafi et al., 2012, Sharafi et al., 2013], researchers mention the choices of the environment and participants as a limitation. However, according to Kitchenham et al. [12], "using students as participants is not a major issue as long as [researchers] are interested in evaluating the use of a technique by novice or non-expert software engineers. Students are the next generation of software professionals so, are relatively close to the population of interest."

Whether experts or novices, there are large differences in participants' eye-movements for identical tasks. These differences are due to participants' individual characteristics. Therefore, as recommended by Goldberg et al. [44], it is prudent to use a within-participants design for eye-tracking studies, to reduce the impact of participants' individual characteristics on the results.

#### 5.5.6. Device vendors

Regarding the settings of eye-trackers, device vendors should ease the choice of the settings of their devices and help in the effort towards having a uniform approach to report, import, and share settings. Although the single-screen limitation mostly exists only in the context of software engineering research, this limitation is important because it precludes many interesting usability studies. For example, studies should be carried to understand how participants would scroll (or not) to find relevant links in the results of a search engine or how they would read code in an IDE. Therefore, device vendors should also start tackling this limitation and offer analyses that can relate eye-movements positions with moving artifacts on screens.

### 5.6. What eye-trackers are most frequently used in eye-tracking studies?

Table 14 provides a list of the eye-trackers used in the selected papers. Two papers used the RFV approach to perform studies [Romero et al., 2002, Romero et al., 2003]. Four studies did not specify the used eye-trackers [Crosby and Stelovsky, 1990, Stein, 2004, Petrusel and Mendling, 2012, Turner et al., 2014].

The most frequently used eye-trackers are Tobii eye-trackers, which are used in about 47% of the papers (17 out of 36). In addition, the Tobii 1750 is the most frequently used eye-trackers among Tobii models. It has been used in 11 studies. There is one paper [Cagiltay et al., 2013] that did not specify the used model of Tobii eye-tracker (see Table 14).

There is an extreme variability in the costs of eye-trackers. Costs vary by tens of thousands of dollars. Thus, when considering the use of eye-trackers, researchers must consider a tradeoff between the cost and the quality of the eye-trackers. In the following, we provide a list of factors that should be considered while comparing eye-trackers.

- Accuracy values represent the differences between the eye-movements positions recorded by an eye-tracker and the actual fixations positions. Accuracy is measured in degrees of visual angle and, usually, ranges from 0.5° to 1°. If a participant is seated 50 cm away from a stimulus and the eye-tracker has 1° of accuracy, the eye-movement positions could be measured anywhere within a radius of 1 cm of the actual positions [30]. The reported accuracy values for the eye-trackers used in the

**Table 15**
List of selected papers.

| Papers | Goals |
|---|---|
| [Crosby and Stelovsky, 1990] | To explore the impact of developers' viewing strategies and experience on program comprehension |
| [Crosby et al., 2002] | To study how experts and novices use source code beacons for comprehension |
| [Romero et al., 2002] | To track attention switching of developers while performing debugging tasks with multiple representations |
| [Romero et al., 2003] | To characterize the debugging strategies using multiple representations |
| [Stein, 2004] | To analyze the impact of one developer's focus of attention on another developer doing the same task |
| [Aschwanden and Crosby, 2006] | To study the impact of source code beacons on developers' viewing strategies during program comprehension |
| [Bednarik and Tukiainen, 2006] | To discover the role of different representations (source code vs. code visualization) on program comprehension |
| [Guéhéneuc, 2006] | To study how developers gain information about the program under study and use this information to perform different tasks |
| [Uwano et al., 2006] | To investigate the individual performance of source code review |
| [Bednarik and Tukiainen, 2007] | To study the effects of RFV's display blurring and expertise on visual attention during debugging |
| [Yusuf et al., 2007] | To investigate the comprehension of UML class diagrams |
| [Bednarik and Tukiainen, 2008] | To reanalyze the data of [Bednarik and Tukiainen, 2007] to achieve more detailed information about developers' behavior during debugging |
| [Jeanmart et al., 2009] | To analyze the impact of visitor pattern on program comprehension and maintenance |
| [Cepeda and Guéhéneuc, 2010] | To analyze the impact of representation on design pattern comprehension |
| [Sharif and Maletic, 2010] | To analyze the impact of various layouts on design pattern comprehension |
| [Busjahn et al., 2011] | To compare the process of source code and natural text reading |
| [Ali et al., 2012] | To identify the most important source code entities |
| [Bednarik, 2012] | To investigate the impact of expertise on defect finding using a multi-representational IDE |
| [De Smet et al., 2014] | To study the impact of Composite, Observer, and MVC pattern on maintenance tasks |
| [Hejmady and Narayanan, 2012] | To understand the impact of multiple representations on debugging |
| [Jermann and Nüssli, 2012] | To study the impact of sharing selection among collaborators in a pair-programming task |
| [Sharafi et al., 2012] | To investigate the impact of gender on developers' effectiveness for source code reading and identifier recalling |
| [Sharif et al., 2012] | To study the impact of scan Time in detecting source code defects |
| [Soh et al., 2012] | To analyze the impact of expertise on developers' speed and accuracy |
| [Binkley et al., 2013] | To analyze the impact of identifier style on program comprehension |
| [Cagiltay et al., 2013] | To propose measures to enhance understanding of ERD diagrams |
| [Duru et al., 2013] | To study the impact of software visualization in program comprehension |
| [Petrusel and Mendling, 2012] | To understand the impact of multiple representations in debugging |
| [Sharafi et al., 2013] | To study the relations between the type of requirement representations (graphical vs. textual) and developers efficiency |
| [Sharif et al., 2013] | To analyze the impact of SeeIT 3D on software engineering tasks |
| [Sharma et al., 2013] | To study the interaction of developers in a pair programming task |
| [58] | To recover traceability links from eye-movements data |
| [Busjahn et al., 2014] | To present a method to recover traceability links from eye-gaze data |
| [Fritz et al., 2014] | To use psycho-physiological measures to calculate task difficulty |
| [Rodeghero et al., 2014] | To study the code summarization task using eye-tracking technique |
| [Turner et al., 2014] | To study differences between individuals' gaze behavior and reading patterns |

selected paper are 0.5° or less. However, the accuracy values reported in eye-tracker manuals are measured under ideal conditions: the measurement was done immediately after calibrating the device and for participants with not corrective eye-wear.

- Sampling rates indicate the numbers of eye-movement positions that can be recorded per second. The typical sampling rate of eye-trackers ranges from 10 Hz to 2000 Hz. As shown in Table 14, the lowest sampling rate for the eye-trackers used in the selected papers is observed for EMR-NC (30 Hz) while the highest is reported for Eye-Link II (500 Hz). Poole et al. [48] reported that a sampling rate of 60 Hz is adequate for usability studies but that it is not good enough for reading studies, which require sampling rates of 500 Hz or higher. The majority of previous studies use eye-trackers with sampling rates of 60 Hz for text and code reading tasks.
- Customer support is important to consider because eye-trackers are complex devices. Customer support vary between device manufacturers. It is necessary that the eye-tracker be accompanied by a user manual that is easy to use. The availability of on-site training, online support, and demo projects are important and may be considered as well.
- Time needed for setting up a study usually includes participants' setup, stimulus adjustment, and calibration. Some eye-trackers require participants to keep their heads perfectly still during calibration, which makes this process error-prone and time consuming because of the necessary recalibrations.
- Software features of the eye-trackers driver and analysis software systems are important. Some eye-trackers come with systems that can perform on-the-fly analyses while others provide different visualizations techniques.

Only one paper compared three different eye-trackers (Tobii 1750 and ASL 504 Pan/Tilt Optics and ASL 501 Head Mounted Optics, both from Applied Science Laboratories) in terms of accuracy and ease of use. The authors of that paper measured the accuracy of the eye-trackers by calculating the mean distances between recorded points of gaze and requested points of gaze. They asked participants to work on a short piece of source code using an animator. Their results show that the ASL 501, which requires mounting the camera on top of participants' heads, needs twice as much time to setup the study and that it is also the least accurate eye-tracker in the set.

## 6. Threats to validity of this study

There is an evident threat to validity of a SLR such as this one, whether the major articles in the literature have been found adequately or not. We have restricted our search to Engineering villages that finds relevant articles based on our research query.

However, the search engine of Engineering village uses the most trusted and well-recognized engineering literature repositories,

**Table 16**
Topic domains, artifacts, participants (S = Students, FM = Faculty members, P = Professionals, and NM = Not mentioned), sources, and the eye-tracker for the selected papers.

| Papers | Topic domains | Artifacts | Participants | Sources | Eye-trackers |
|---|---|---|---|---|---|
| [Crosby and Stelovsky, 1990] | Code comprehension | Pascal | 18 S & FM | IEEE Computer | NM |
| [Crosby et al., 2002] | Code comprehension | Pascal | 18 S | PPIG | ASL |
| [Romero et al., 2002] | Debugging | Java | 5 S & P | LNCS | RFV |
| [Romero et al., 2003] | Debugging | Java | 49 S | PPIG | RFV |
| [Stein, 2004] | Collaborative interactions | Java | 10 S & P | ACM ICMI | NM |
| [Aschwanden and Crosby, 2006] | Code comprehension | Java | 15 NM | HICSS | ASL |
| [Bednarik and Tukiainen, 2006] | Code comprehension | Java | 14 S | ETRA | Tobii 1750 |
| [Guéhéneuc, 2006] | Model comprehension | UML | 12 S | CASCON | EyeLink II |
| [Uwano et al., 2006] | Debugging | C | 5 S | ETRA | EMR-NC |
| [Bednarik and Tukiainen, 2007] | Debugging | Java | 18 S & FM | BRM | Tobii 1750 |
| [Yusuf et al., 2007] | Model comprehension | UML | 12 S & FM | ICPC | Tobii 1750 |
| [Bednarik and Tukiainen, 2008] | Debugging | Java | 14 S | ETRA | Tobii 1750 |
| [Jeanmart et al., 2009] | Model comprehension | UML | 24 S | ESEM | EyeLink II |
| [Cepeda and Guéhéneuc, 2010] | Model comprehension | UML | 24 S | EMSE | EyeLink II |
| [Sharif and Maletic, 2010] | Model comprehension | UML | 12 S & FM | ICSM | Tobii 1750 |
| [Busjahn et al., 2011] | Code comprehension | Java | 15 NM | Koli Calling | Tobii T120 |
| [Ali et al., 2012] | Traceability | Java | 26 S | ICSM | FaceLAB |
| [Bednarik, 2012] | Debugging | Java | 18 S & P | HUMCOMPUT | Tobii 1750 |
| [De Smet et al., 2014] | Model comprehension | UML | 42 S & FM | SCP | EyeLink II & FaceLAB |
| [Hejmady and Narayanan, 2012] | Debugging | Java | 19 S | ETRA | Tobii X60 |
| [Jermann and Nüssli, 2012] | Collaborative interactions | Java | 82 S | CSCW | Tobii 1750 |
| [Sharafi et al., 2012] | Code Comprehension | Java | 26 S | ICPC | FaceLAB |
| [Sharif et al., 2012] | Debugging | C | 15 S & FM | ETRA | Tobii 1750 |
| [Soh et al., 2012] | Model comprehension | UML | 21 S & FM | ICPC | EyeLink II |
| [Binkley et al., 2013] | Code comprehension | English | 169 S | EMSE | Tobii 1750 |
| [Cagiltay et al., 2013] | Model comprehension | ER diagram | 4 P | JSS | Tobii |
| [Duru et al., 2013] | Code comprehension | C♯ | 13 P | IJHCI | Tobii 1750 |
| [Petrusel and Mendling, 2012] | Model Comprehension | BPMN | 29 P | CAiSE | NM |
| [Sharafi et al., 2013] | Model comprehension | Tropos | 28 S | ICPC | FaceLAB |
| [Sharif et al., 2013] | Usability evaluation | Java | 97 S | VISSOFT | Tobii X60 |
| [Sharma et al., 2013] | Collaborative interactions | Java | 82 S | CSCL | Tobii 1750 |
| [58] | Traceability | Java | 8 S & FM | ETRA | Tobii X60 |
| [Busjahn et al., 2014] | Code comprehension | Java | 15 P | ETRA | Tobii T120 |
| [Fritz et al., 2014] | Code comprehension | C♯ | 15 P | ICSE | Tobii TX300 |
| [Rodeghero et al., 2014] | Code comprehension | Java | 10 P | ICSE | Tobii TX300 |
| [Turner et al., 2014] | Code comprehension | C++, Python | 38 S | ETRA | NM |

**Table 17**
Data extraction form.

| Aspect | Attribute |
|---|---|
| Title | |
| Authors and affiliation | First author |
| | Second author |
| | Third author |
| | Forth author |
| Year and source | |
| Eye-tracker | Device |
| | Distance between the participant and the screen |
| | Screen resolutions |
| | Frame rate |
| Research questions | |
| Study variables | Dependent |
| | Independent |
| | Mitigating |
| Experiment design | |
| Participants demography | |
| Materials | |
| Tasks | |
| Metrics | Number of fixations |
| | Duration of fixations |
| | Saccade analysis |
| | Scan-path analysis |
| | Others |

including ACM and IEEE. In addition, we try to evaluate the quality of our proposed query and modify it to find all relevant papers. Because no previous SLR exists with regards to the usage of eye-tracking technique in software engineering, we cannot evaluate the quality of the search string using "quasi-gold standard" [4] string evaluation approach [6]. However, to reduce the possibility of missing a relevant paper, we performed full analysis and apply snow-balling to detect missing papers.

Still, we may have missed some articles published in national journals and conferences. Thus, our results must be qualified as considering articles that are published in the major international journals, conferences, and workshops in computer science and software engineering.

Moreover, we use the method proposed by [18] for data extraction in which one researcher extracted the data and another researcher checked the data extraction. [5] show that extractor/checker method would be problematic while dealing with a large number of primary studies or the data is complex. However, in our case, the number of primary studies are not very large and our data extraction method is relatively objective which reduces the data extraction errors. In addition, the extracted data have been checked by third author to reduce the likelihood of erroneous results.

## 7. Conclusion

We performed a systematic literature review to investigate the uses of eye-tracking technology in software engineering. Instead of performing manual searches in international journals and conferences, we undertook a broad automated search using Engineering village that helps us to visit all recognized resources from 1990 to 2014. Out of 649 publications found, we identified 35 relevant papers relevant to the uses of eye-tracking technology in software engineering.

A major finding of this SLR is that the software engineering community benefits from the uses of eye-trackers despite their limitations. The results of eye-tracking studies add to the existing body of knowledge on how participants perform different software engineering tasks and how they use different models and representations along with source code to understand software systems. Furthermore, several metrics have been proposed to quantitatively assess and measure participants' visual efforts and display how they scan the stimuli while performing their tasks.

Different topics have been considered and analyzed in different eye-tracking studies and we divided them in five categories: *model comprehension*, *debugging*, *code comprehension*, *collaborative interactions*, and *traceability*. This categorization shows that the spread of topics is fairly even and important software engineering tasks, including comprehension, maintenance, and debugging, have been studied.

We identified a list of limitations of the uses of eye-trackers that lead to threats to the validity of the selected studies. These limitations come from the participants that perform the tasks, the tasks that they performed, the artifacts that they used, and the environments in which the experiments were done. We discussed in details how to mitigate these limitations and improve the quality of eye-tracking experiments.

We also provided two general recommendations for the software engineering community and a list of suggestions for researchers who are newcomers and want to perform an eye-tracking study. Our two general recommendations are:

- There is a lack of consistent terminology, metrics names, and methods. The community must adopt the standard guidelines and terminology while conducting and reporting an eye-tracking experiment. Using standard guidelines to design experiments for software engineering tasks could reduce the risks of failure and also mitigates threats to validity. As emphasized and explained by Sjøberg et al. [11], using a uniform way of reporting an experiment benefits the software engineering community, because it provides valuable information on how to review, replicate, and analyze the data. There are guidelines for performing controlled experiments in software engineering [2,3]. However, conducting an eye-tracking experiment requires further, more specific recommendations with regards to the limitations associated with this technology.
- The majority of the studies prepared their own artifacts to design and preform experiments, which are not available online. Therefore, it is not possible to re-use them for replicating the studies or performing new experiments. Thus, researchers should make their data accessible to other researchers who are interested in replicating their experiments while accurately reporting the different criteria and factors that they considered while designing and performing the experiments. In addition, the software engineering community must promote online software artifacts repositories (e.g., Software-artifact Infrastructure Repository (SIR) [20][19]). Sharing data online in an organized repository would allow researchers to receive feedback and improve their artifacts and to aggregate their findings and single out missing artifacts [20].

We recommend newcomers who start in the field (1) to familiarize themselves with eye-tracking basics including how eye-trackers work, what are the eye-gaze data (e.g., fixations and saccades), and how to measure and interpret these data in general by reading valuable literature including [37,39,40,46,48]. (2) To consider the limitations associated with this technology, as summarized in Section 5.5, while thinking about using eye-trackers and designing the experiments. (3) To begin with less demanding tasks (e.g., searching and locating specific elements in a simple graphical representation or text) to become acquainted with eye-movement data recording and analysis before working on complex tasks with fine-grained artifacts (e.g., source code). We also encourage researchers not to perform one single experiment to study a broad topic that contains several subtopics (e.g., maintenance), because it complicates data analysis and interpretation. (4) Finally, to check previous studies using the annotated bibliography in Section 5.2 when having a topic in mind and to benefit from previous studies, especially regarding the metrics used and their interpretation.

We plan to perform this SLR in 2020 to keep track of the progress and results of new eye-tracking studies in software engineering.

[19] http://sir.unl.edu/portal/index.php.

## Acknowledgments

We thank the editor and anonymous reviewers for their constructive comments, which helped us to improve the manuscript. This work has been partly funded by the Canada Research Chairs on Software Patterns and Patterns of Software and on Software Change and Evolution and by Fonds de Recherche du Québec Nature et Technologies (FQRNT).

## Appendix A. SLR articles

[Ali et al., 2012]
Ali, N., Sharafi, Z., Guéhéneuc, Y.-G., and Antoniol, G. (2012). An empirical study on requirements traceability using eye-tracking. In *Proceedings of the 28th IEEE International Conference on Software Maintenance*, ICSM '12, pages 191–200. IEEE Computer Society.

[Aschwanden and Crosby, 2006]
Aschwanden, C. and Crosby, M. (2006). Code scanning patterns in program comprehension. In *Proceedings of the 39th Hawaii International Conference on System Sciences*, HICSS '06.

[Bednarik, 2012]
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.

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

[Bednarik and Tukiainen, 2007]
Bednarik, R. and Tukiainen, M. (2007). Validating the restricted focus viewer: A study using eye-movement tracking. *Behavior Research Methods*, 39(2):274–282.

[Bednarik and Tukiainen, 2008]
Bednarik, R. and Tukiainen, M. (2008). Temporal eye-tracking data: Evolution of debugging strategies with multiple representations. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08, pages 99–102, New York, NY, USA. ACM.

[Binkley et al., 2013]
Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., and Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 18(2):219–276.

[Busjahn et al., 2014]
Busjahn, T., Bednarik, R., and Schulte, C. (2014). What influences dwell time during source code reading?: Analysis of element type and frequency as factors. In *Proceedings of the Symposium on Eye Tracking Research & Applications*, ETRA '14, pages 335–338, New York, NY, USA. ACM.

[Busjahn et al., 2011]
Busjahn, T., Schulte, C., and 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*, Koli Calling '11, pages 1–9, New York, NY, USA. ACM.

[Cagiltay et al., 2013]
Cagiltay, N. E., Tokdemir, G., Kilic, O., and Topalli, D. (2013). Performing and analyzing non-formal inspections of entity relationship diagram (erd). *Journal of Systems and Software*, 86(8):2184–2195.

[Cepeda and Guéhéneuc, 2010]
Cepeda, G. and Guéhéneuc, Y.-G. (2010). An empirical study on the efficiency of different design pattern representations in uml class diagrams. *Empirical Software Engineering*, 15(5):493–522.

[Crosby et al., 2002]
Crosby, M. E., Scholtz, J., and 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*, pages 18–21.

[Crosby and Stelovsky, 1990]
Crosby, M. E. and Stelovsky, J. (1990). How do we read algorithms? a case study. *Computer*, 23(1):24–35.

[De Smet et al., 2014]
De Smet, B., Lempereur, L., Sharafi, Z., Guéhéneuc, Y.-G., Antoniol, G., and Habra, N. (2014). Taupe: Visualizing and analyzing eye-tracking data. *Science of Computer Programming*, 79:260–278.

[Duru et al., 2013]
Duru, H. A., Çakır, M. P., and İşler, V. (2013). How does software visualization contribute to software comprehension? a grounded theory approach. *International Journal of Human–Computer Interaction*, 29(11):743–763.

[Fritz et al., 2014]
Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., and Züger, M. (2014). Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE '14, pages 402–413, New York, NY, USA. ACM.

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

[Hejmady and Narayanan, 2012]
Hejmady, P. and Narayanan, N. H. (2012). Visual attention patterns during program debugging with an IDE. In *Proceedings of the 2012 Symposium on Eye Tracking Research & Applications*, ETRA '12, pages 197–200, New York, NY, USA. ACM.

[Jeanmart et al., 2009]
Jeanmart, S., Guéhéneuc, Y.-G., Sahraoui, H. A., and 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*, pages 69–78.

[Jermann and Nüssli, 2012]
Jermann, P. and Nüssli, M.-A. (2012). Effects of sharing text selections on gaze cross-recurrence and interaction quality in a pair programming task. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 1125–1134, New York, NY, USA. ACM.

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

[Rodeghero et al., 2014]
Rodeghero, P., McMillan, C., McBurney, P. W., Bosch, N., and 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*, ICSE '14, pages 390–401, New York, NY, USA. ACM.

[Romero et al., 2002]
Romero, P., Cox, R., du Boulay, B., and Lutz, R. (2002). Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. In *Diagrammatic Representation and Inference*, pages 221–235. Springer.

[Romero et al., 2003]
Romero, P., du Boulay, B., Cox, R., and Lutz, R. (2003). Java debugging strategies in multi-representational environments. In *Proceedings of the 15th Workshop of Psychology of Programming Interest Group*, pages 421–434.

[Sharafi et al., 2013]
Sharafi, Z., Marchetto, A., Susi, A., Antoniol, G., and Guéhéneuc, Y.-G. (2013). An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. In *Proceeding of 21st International Conference on Program Comprehension*, ICPC '13, pages 33–42.

[Sharafi et al., 2012]
Sharafi, Z., Soh, Z., Guéhéneuc, Y.-G., and 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, pages 27–36.

[Sharif et al., 2012]
Sharif, B., Falcone, M., and Maletic, J. I. (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*, ETRA '12, pages 381–384, New York, NY, USA. ACM.

[Sharif et al., 2013]
Sharif, B., Jetty, G., Aponte, J., and Parra, E. (2013). An empirical study assessing the effect of seeit 3D on comprehension. In *Proceeding of 1st IEEE Working Conference on Software Visualization*, VISSOFT '13, pages 1–10. IEEE.

[Sharif and Maletic, 2010]
Sharif, B. and Maletic, J. I. (2010). 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*, pages 1–10. IEEE Computer Society.

[Sharma et al., 2013]
Sharma, K., Jermann, P., Nüssli, M.-A., and Dillenbourg, P. (2013). Understanding collaborative program comprehension: Interlacing gaze and dialogs. In *Proceedings of the 10th Internation Conference on Computer Supported Collaborative Learning*, CSCL '13.

[Soh et al., 2012]
Soh, Z., Sharafi, Z., den Plas, B. V., Porras, G. C., Guéhéneuc, Y.-G., and 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, pages 163–172.

[Stein, 2004]
Stein, R. (2004). Another persons eye gaze as a cue in solving programming problems. In *Proceedings of the 6th International Conference on Multimodal Interface*, pages 9–15. ACM Press.

[Turner et al., 2014]
Turner, R., Falcone, M., Sharif, B., and Lazar, A. (2014). An eye-tracking study assessing the comprehension of C++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research & Applications*, ETRA '14, pages 231–234, New York, NY, USA. ACM.

[Uwano et al., 2006]
Uwano, H., Nakamura, M., Monden, A., and Matsumoto, K.-i. (2006). Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the 2006 symposium on Eye tracking research & applications*, ETRA '06, pages 133–140. ACM.

[Walters et al., 2014]
Walters, B., Shaffer, T., Sharif, B., and Kagdi, H. (2014). Capturing software traceability links from developers' eye gazes. In *Proceedings of the 22nd International Conference on Program Comprehension*, ICPC '14, pages 201–204, New York, NY, USA. ACM.

[Yusuf et al., 2007]
Yusuf, S., Kagdi, H. H., and Maletic, J. I. (2007). Assessing the comprehension of UML class diagrams via eye tracking. In *Proceeding of 15th IEEE International Conference on Program Comprehension*, ICPC '07, pages 113–122. IEEE Computer Society.

## References

[1] S.G. Hart, L.E. Staveland, Development of NASA-TLX (task load index): results of empirical and theoretical research, Adv. Psychol. 52 (1988) 139–183.
[2] B. Kitchenham, Empirical paradigm – the role of experiments, in: Proceedings of the 2006 International Conference on Empirical Software Engineering Issues: Critical Assessment and Future Directions, Springer-Verlag, Berlin, Heidelberg, 2007, pp. 25–32.
[3] A. Jedlitschka, D. Pfahl, Reporting guidelines for controlled experiments in software engineering, in: Proceedings of the 2005 International Symposium on Empirical Software Engineering, 2005, pp. 10 pp.–, http://dx.doi.org/10.1109/ISESE.2005.1541818.
[4] H. Zhang, M.A. Babar, P. Tell, Identifying relevant studies in software engineering, Inf. Softw. Technol. 53 (6) (2011) 625–637.
[5] M. Turner, B. Kitchenham, D. Budgen, P. Brereton, Lessons learnt undertaking a large-scale systematic literature review, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, EASE '08, British Computer Society, Swinton, UK, UK, 2008, pp. 110–118.
[6] M. Lavallée, P.N. Robillard, R. Mirsalari, Performing systematic literature reviews with novices: an iterative approach, IEEE Trans. Educ. 57 (3) (2014) 175–181, http://dx.doi.org/10.1109/TE.2013.2292570.
[7] R.J. Erich Gamma, Richard Helm, J. Vlissides, Design Patterns Elements of Reusable Object-Oriented Software, Addison-Wesley Pub Co., 1995.
[8] R. Schauer, R.K. Keller, Pattern visualization for software comprehension, in: Proceeding of 6th International Workshop on Program Comprehension, IWPC '98, IEEE, 1998, pp. 4–12.
[9] E. Gamma, Applying design patterns in Java, in: D. Deugo (Ed.), Java Gems: Jewels from Java Report, SIGS Reference Library, 1996, pp. 47–53.
[10] D. Jing, Y. Sheng, Z. Kang, Visualizing design patterns in their applications and compositions, IEEE Trans. Softw. Eng. 33 (7) (2007) 433–453, http://dx.doi.org/10.1109/TSE.2007.1012.
[11] D.I. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg, A.C. Rekdal, A survey of controlled experiments in software engineering, IEEE Trans. Softw. Eng. 31 (9) (2005) 733–753.
[12] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones, D.C. Hoaglin, K.E. Emam, J. Rosenberg, Preliminary guidelines for empirical research in software engineering, IEEE Trans. Softw. Eng. 28 (8) (2002) 721–734, http://dx.doi.org/10.1109/TSE.2002.1027796.
[13] B.A. Kitchenham, T. Dyba, M. Jorgensen, Evidence-based software engineering, in: Proceedings of the 26th International Conference on Software Engineering, ICSE '04, IEEE Computer Society, Washington, DC, USA, 2004, pp. 273–281.
[14] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering – a systematic literature review, Inf. Softw. Technol. 51 (1) (2009) 7–15, http://dx.doi.org/10.1016/j.infsof.2008.09.009.
[16] D.L. Sackett, S.E. Straus, W.S. Richardson, W. Rosenberg, R.B. Haynes, Evidence-Based Medicine: How to Practice and Teach EBM, second ed., Churchill Livingstone, 2000.
[17] B. Kitchenham, Procedures for Undertaking Systematic Reviews, Tech. rep., Joint Technical Report, Computer Science Department, Keele University (TR/SE- 0401) and National ICT Australia Ltd., 2004.
[18] P. Brereton, B.A. Kitchenham, D. Budgen, M. Turner, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (4) (2007) 571–583.
[19] D. Budgen, A.J. Burn, O.P. Brereton, B.A. Kitchenham, R. Pretorius, Empirical evidence about the UML: a systematic literature review, Softw.: Pract. Exper. 41 (4) (2011) 363–392, http://dx.doi.org/10.1002/spe.1009.
[20] H. Do, S. Elbaum, G. Rothermel, Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact, Emp. Softw. Eng. 10 (4) (2005) 405–435, http://dx.doi.org/10.1007/s10664-005-3861-2.
[21] B. Sharif, H. Kagdi, On the use of eye tracking in software traceability, in: Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '11, ACM, New York, NY, USA, 2011, pp. 67–70.
[22] M.A. Just, P.A. Carpenter, A theory of reading: from eye fixations to comprehension, Psychol. Rev. 87 (4) (1980) 329.
[23] V. Ponsoda, D. Scott, J.M. Findlay, A probability vector and transition matrix analysis of eye movements during visual search, Acta Psychol. 88 (2) (1995) 167–185.
[25] J.H. Goldberg, J.I. Helfman, Comparing information graphics: a critical look at eye tracking, in: Proceedings of the 3rd Workshop: BEyond Time and Errors: Novel evaLuation Methods for Information Visualization, BELIV '10, ACM, New York, NY, USA, 2010, pp. 71–78.
[26] C. Lankford, Gazetracker: software designed to facilitate eye movement analysis, in: Proceedings of the 2000 Symposium on Eye Tracking Research & Applications, ETRA '00, ACM, New York, NY, USA, 2000, pp. 51–55.
[27] J.L. Sibert, M. Gokturk, R.A. Lavine, The reading assistant: eye gaze triggered auditory prompting for reading remediation, in: Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology, UIST '00, ACM, New York, NY, USA, 2000, pp. 101–107.
[28] R.J. Jacob, K.S. Karn, Eye tracking in human–computer interaction and usability research: Ready to deliver the promises, Mind's Eye: Cognit. Appl. Aspects Eye Move. Res. 2 (3) (2003) 4.
[29] B. Sharif, Empirical assessment of UML class diagram layouts based on architectural importance, in: Proceedings of the 27th IEEE International Conference on Software Maintenance, ICSM '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 544–549.
[30] J. Sajaniemi, 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, 2004, pp. 151–158.
[31] B. Sharif, J.I. Maletic, An eye tracking study on camelcase and under_score identifier styles, in: Proceeding of 18th IEEE International Conference on Program Comprehension, ICPC '10, IEEE Computer Society, 2010, pp. 196–205.
[32] Y. Gueheneuc, H. Kagdi, J. Maletic, Working session: Using eye-tracking to understand program comprehension, in: Proceedings of IEEE 17th

International Conference on Program Comprehension, ICPC '09, 2009, pp. 278–279, http://dx.doi.org/10.1109/ICPC.2009.5090057.
[33] R. Bednarik, M. Tukiainen, Effects of display blurring on the behavior of novices and experts during program debugging, in: Proceeding of Conference on Extended Abstracts on Human Factors in Computing Systems, CHI '05, ACM, New York, NY, USA, 2005, pp. 1204–1207.
[34] R. Bednarik, M. Tukiainen, Visual attention tracking during program debugging, in: Proceedings of the Third Nordic Conference on Human-computer Interaction, NordiCHI '04, ACM, New York, NY, USA, 2004, pp. 331–334.
[35] J.M. Henderson, G.L. Pierce, Eye movements during scene viewing: evidence for mixed control of fixation durations, Psychon. Bullet. Rev. 15 (3) (2008) 566–573.
[36] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Soviet Phys. Doklady 10 (1966) 707.
[37] M.A. Just, P.A. Carpenter, Eye fixations and cognitive processes, Cognit. Psychol. 8 (4) (1976) 441–480.
[39] A. Duchowski, Eye Tracking Methodology: Theory and Practice, Springer-Verlag New York Inc, 2007.
[40] K. Rayner, Eye movements in reading and information processing: 20 years of research, Psychol. Bullet. 124 (3) (1998) 372.
[41] C.M. Privitera, L.W. Stark, Algorithms for defining visual regions-of-interest: comparison with eye fixations, IEEE Trans. Pattern Anal. Mach. Intell. 22 (2000) 970–982.
[42] A.F. Blackwell, A.R. Jansen, K. Marriott, Restricted focus viewer: a tool for tracking visual attention, in: M. Anderson, P.C.-H. Cheng, V. Haarslev (Eds.), Behavior Research Methods, Instruments, & Computers, Lecture Notes in Computer Science, vol. 1889, Springer, 2000, pp. 162–177.
[43] P. Romero, R. Lutz, R. Cox, B. du Boulay, Co-ordination of multiple external representations during java program debugging, in: Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments, IEEE, 2002, pp. 207–214.
[44] J.H. Goldberg, A.M. Wichansky, Eye tracking in usability evaluation: a practitioners guide, in: Hyönä R. Radach, H. Deubel (Eds.), The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research, 2002.
[45] J.H. Goldberg, M.J. Stimson, M. Lewenstein, N. Scott, A.M. Wichansky, Eye tracking in web search tasks: design implications, in: Proceedings of the 20 02 Symposium on Eye Tracking Research & Applications, ETRA '02, ACM, New York, NY, USA, 2002, pp. 51–58. http://dx.doi.org/10.1145/507072.507082.
[46] J.H. Goldberg, X.P. Kotval, Computer interface evaluation using eye movements: methods and constructs, Int. J. Indust. Ergon. 24 (6) (1999) 631–645.
[48] A. Poole, L.J. Ball, 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, Idea Group, Inc., Pennsylvania, 2005, pp. 211–219.
[49] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using a bitmap representation, in: Proceedings of the 8th ACM International Conference on Knowledge Discovery and Data Mining, KDD '02, ACM, New York, NY, USA, 2002, pp. 429–435.
[50] F. Cristino, S. Mathot, J. Theeuwes, I.D. Gilchrist, Scanmatch: a novel method for comparing fixation sequences, Behav. Res. Meth. 42 (2010) 692–700.
[51] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: an agent-oriented software development methodology, Auton. Agents Multi-Agent Syst. 8 (3) (2004) 203–236.
[53] A.R. Jansen, A.F. Blackwell, K. Marriott, A tool for tracking visual attention: the restricted focus viewer, Behav. Res. Meth., Instrum., Comp. 35 (1) (2003) 57–69.
[54] SR Research Ltd, EyeLink II User Manual version (07/02/2006), SR Research Ltd., February 2006.
[55] Seeing Machine, Seeing Machine's website – FaceLAB, 2010 <www.seeingmachines.com/product/facelab/> (accessed 16.04.14).
[56] A.T. Duchowski, J. Driver, S. Jolaoso, W. Tan, B.N. Ramey, A. Robbins, Scanpath comparison revisited, in: Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications, ETRA '10, ACM, New York, NY, USA, 2010, pp. 219–226.
[57] R. Dewhurst, M. Nyström, H. Jarodzka, T. Foulsham, R. Johansson, K. Holmqvist, It depends on how you look at it: scanpath comparison in multiple dimensions with multimatch, a vector-based approach, Behav. Res. Meth. 44 (4) (2012) 1079–1100.
[58] B. Walters, M. Falcone, A. Shibble, B. Sharif, Towards an eye-tracking enabled IDE for software traceability tasks, in: Proceeding of International Workshop on Traceability in Emerging Forms of Software Engineering, TEFSE '13, 2013, pp. 51–54, http://dx.doi.org/10.1109/TEFSE.2013.6620154.
[59] B.C.O.F. Fehringer, Eye tracking gaze visualiser: eye tracker and experimental software independent visualisation of gaze data, in: Proceedings of the Symposium on Eye Tracking Research & Applications, ETRA '14, ACM, New York, NY, USA, 2014, pp. 259–262.