UNIVERSITÉ DE MONTRÉAL

ON THE INFLUENCE OF REPRESENTATION TYPE AND GENDER ON RECOGNITION TASKS OF PROGRAM COMPREHENSION

ZOHREH SHARAFI TAFRESHI MOGHADDAM
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(GÉNIE INFORMATIQUE)
APRIL 2015

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ON THE INFLUENCE OF REPRESENTATION TYPE AND GENDER
ON RECOGNITION TASKS OF PROGRAM COMPREHENSION

présentée par : SHARAFI TAFRESHI MOGHADDAM Zohreh
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

Mme NICOLESCU Gabriela, Ph.D., présidente
M. ANTONIOL Giuliano, Ph.D., membre et directeur de recherche
M. GUÉHÉNEUC Yann-Gaël, Ph.D., membre et codirecteur de recherche
M. ROBILLARD Pierre, Ph.D., membre
Mme STOREY Margaret-Anne, Ph.D., membre externe

To my mom and dad . . .

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Dr. Giuliano Antoniol for the continuous support of my Ph.D study and research, for his patience, motivation, and immense knowledge. Thank you Giulio for believing in me and challenging me, I could not have imagined having a better advisor and mentor for my Ph.D study.

I would like to gratefully and sincerely thank my co-supervisor, Dr. Yann-Gaël Guéhéneuc for his understanding, and enthusiasm during my study. Without his guidance and persistent help this dissertation would not have been possible. Thanks a lot for pushing me farther that I thought I could go.

During my Ph.D., I also had the chance to work with great researchers from whom I learned a lot, Dr. Alessandro Marchetto and Dr. Angelo Susi. Thanks a lot for giving me the opportunity to collaborate and share great ideas.

The members of the SoccerLab, Ptidej, MCIS and SWAT groups have contributed exceptionally to my personal and professional time at École Polytechnique de Montréal. These groups have been the source of friendships as well as good advice and collaboration. My special thanks to my colleagues, Zéphyrin, Laleh, and Venera for all the helps, discussions, and feedbacks.

I also warmly thank my best friend and best teammate, Parisa, for all the support, camaraderie, and encouragement. Cheers to all the projects, the deadlines we met a minute before midnight, and all the passionate discussions.

This dissertation is also dedicated to Ali - my husband, soul-mate, partner in life who gave me home in Canada while I'm living far away from my homeland. Thanks a lot for unfailing love, for sharing this entire amazing journey with me, and for believing in me when I didn't believe in myself. I love you dearly.

Last, but certainly not least, I owe my loving and sincere thanks to the most influential people in my life : my mother, Dr. Mahnaz Jafaripour for constant love, hope, and perspective and my father, Dr. Aliakbar Sharafi on whose shoulders I stand. Thanks for your endless support, enduring love, constant guidance, motivation, and encouragement. Your strong and kind-hearted personalities taught me to be dedicated and never bend to difficulty.

My twin sister, Azadeh, for being always beside me, helping me, giving me love in every situation, listening to me, and finding me the light, whenever it was far away. I'm so proud and fortunate beyond measure in having you by my side all these years.

RÉSUMÉ

L'objectif de la maintenance logicielle est d'améliorer les logiciels existants en préservant leur intégrité. La maintenance peut représenter jusqu'à 60% du budget d'un logiciel. Ainsi, améliorer la maintenabilité des logiciels est bénéfique aussi bien pour les fournisseurs que les utilisateurs de logiciels. Les développeurs de logiciels consacrent un effort considérable à la compréhension des programmes, qui est une étape primordiale à la maintenance logicielle. Nous faisons l'hypothèque que le genre des développeurs de logiciels et le type de représentation peut affecter leur effort et leur efficacité. Ces facteurs doivent être considérés et minutieusement analysés dans la mesure où ils peuvent cacher certains effets significatifs pouvant être identifiés en analysant le processus de compréhension.

Dans cette thèse, nous nous inspirons de l'utlisation de l'occulomètre pour l'étude du processus cognitif lors de la résolution des problèmes. Nous avons effectué une étude fonctionnelle pour évaluer tous les travaux de recherche faisant usage de l'occulomètre en génie logiciel. Les résultats obtenus nous ont motivé à utiliser l'occulomètre pour effectuer un ensemble d'études afin analyser l'effet de deux facteurs importants sur la compréhension des programmes : le type de représentation (textuelle ou graphique) et le genre du développeur. Afin de comprendre comment les différents types de représentations et le genre influencent les stratégies de visualisation, nous avons étudié la différence de strategie entre développeurs.

Les résultats obtenus montrent que, comparé à une représentation graphique, la représentation sous forme de texte structuré aide mieux le développeur dans son processus cognitif lors de la compréhension des programmes de petite taille. Ainsi, la représentation textuelle requiert moins de temps et d'effort aux participants. Par contre, la représentation graphique est celle préférée par les développeurs. Nos résultats montrent que la structure topologique de la représentation graphique aide les développeurs à mémoriser l'emplacement des éléments et à retrouver plus rapidement les éléments pertinents comparé à la représentation textuelle. En plus, la structure hiérarchique de la représentation graphique guide les développeurs à suivre une stratégie de visualisation spécifique.

Nous avons observé que les femmes et les hommes ont des stratégies de visualisation différentes lors de la lecture du code ou de la mémoririsation des noms des identificateurs. Les femmes ont tendance à inspecter minutieusement toutes les options afin de procéder à l'élimination de la mauvaise réponse. Au contraire, les hommes ont tendance à inspecter brièvement certaines réponses. Pendant que les femmes consacrent plus de temps à analyser chaque type d'entité l'un après l'autre, les hommes alternent leur attention entre différents type d'entité.

ABSTRACT

The purpose of software maintenance is to correct and enhance an existing software system while preserving its integrity. Software maintenance can cost more than 60% of the budget of a software system, thus improving the maintainability of software is important for both the software industry and its customers. Program comprehension is the initial step of software maintenance that requires the major amount of maintenance's time and effort. We conjuncture that developers' gender and the type of representations that developers utilize to perform program comprehension impact their efficiency and effectiveness. These factors must be considered and carefully studied, because they may hide some significant effects to be found by analyzing the comprehension process.

In this dissertation, inspired by the literature on the usefulness of eye-trackers to study the cognitive process involved in problem solving activities, we perform a mapping study and evaluate all research relevant to the use of eye-tracking technique in software engineering. The results motivate us to perform a set of eye-tracking studies to analyze the impact of two important factors on program comprehension: representation type (textual vs. graphical) and developers' gender. Moreover, we investigate and compare viewing strategies variability amongst developers to understand how the representation type and gender differences influence viewing strategies.

Overall, our results indicate that structured text provides more cognitive support for developers while performing program comprehension with small systems compared to a graphical representation. Developers spend less time and effort working with textual representations. However, developers mostly preferred to use graphical representations and our results confirm that the topological structure of graphical representations helps developers to memorize the location of the elements and to find the relevant ones faster in comparison with textual representation. Moreover, the hierarchical structure of the representation guides developers to follow specific viewing strategies while working with representations.

Regarding the impact of gender, our results emphasize that male and female developers exploit different viewing strategies while reading source code or recalling the names of identifiers. Female developers seem to carefully weigh all options and rule out wrong answers, while male developers seem to quickly set their minds on some answers and move forward. Moreover, female developers spend more time on each source code entity and analyze it before going to the next one. In contrast, male developers utilize several attention switching strategies between different source code entities.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

ACM ICMI	ACM International Conference on Multimodel Interaction
AFD	Average Fixation Duration
AOI	Area of Interest
BPMN	Business Process Model and Notation
BRM	Journal of Behavior research methods
CAiSE	Conference on Advanced Information Systems Engineering
CASCON	Centre for Advanced Studies Conference
CC	Camel Case
CiB	Class name in Bold
CSCL	Computer supported Collaborative Learning Conference
CSCW	Computer Supported Cooperative Work
DV	Dependent Variable
EBSE	Evidence Based Software Engineering
EMSE	Empirical Software Engineering Journal
ERD	Entity Relationship Diagram
ETRA	Eye-tracking Research and Application
ESEM	Symposium on Software Engineering and Measurement
FC	Fixation Count
FT	Fixation Time
FR	Fixation Rate
HICSS	Hawaii International Conference on System Sciences
HCI	Human Computer Interaction
HUM-COMPUT	Journal of Human Computer Studies
IEEE Computer	IEEE Computer Journal
ICPC	International Conference on Program Comprehension
ICSM	International Conference on Software Maintenance
IV	Independent Variable
JSS	Journal of Systems and Software
LNCS	Lecture Notes in Computer Science
LOC	Line Of Code
MiB	Method name in Bold
MV	Mitigating Variable
MVC	Model View Controller

PFT	Proportional Fixation Time
PPIG	Workshop of Psychology of Programming interest group
RFV	Restricted Focus Viewer
ROAFT	Ratio of “On_targer:All_target” Fixation Time
RQ	Research Question
RTA	Retrospective Think-aloud
SCE	Source Code Entity
SCP	Science of Computer Programming Journal
SETP	Conference on Software Engineering Theory and Practice
SLR	Systematic Literature Review
SPAM	Sequential PAttern Mining
SD	Spatial Density
SPP	Scan-path Precision
SPR	Scan-path Recall
SPF	Scan-path F-measure
TEFSE	Workshop on Traceability in Emerging Forms of Software Engineering
TM	Transitional Matrix
US	Under Score
UML	Unified Modeling Language
VISSOFT	Working Conference on Software Visualization

CHAPTER 1 INTRODUCTION

“Programming is far more complex than usual human mental activities studied by psychologists” (Weinberg and Schulman, 1974).

1.1 Research Context and Problem Statement

As a software system evolves over time, the quality of the system’s usefulness declines and its complexity increases (Lehman, 1996). Therefore, software maintenance activities are required to preserve the usefulness of the system. It is reported that 50 % to 75% of the overall cost of a software system is dedicated to its maintenance (Lientz and Swanson, 1980; Davis, 1995; Pressman, 2001) and this cost is rising drastically (TIOBE Software, 2014) (see Figure 1.1).

Program comprehension is the foremost step for developers to perform any maintenance task and plays a vital role in software maintenance (Littman et al., 1987; Roehm et al., 2012a). During maintenance, developers spend at least half their time reading software artifacts to understand systems (Corbi, 1989). Any advancement in program comprehension can have real subsequent effects in improving education, training, and the design and evaluation of tools and development environments (Siegmund et al., 2014).

Program comprehension is based on a developer’s cognitive model of the program. A cognitive model describes the cognitive processes along with different types of information that helps to create this model (Navarro-Prieto and Cañas, 2001; Storey, 2005; LaToza et al., 2006). Therefore, to ease program comprehension, we must first understand the cognitive model underlying comprehension activity.

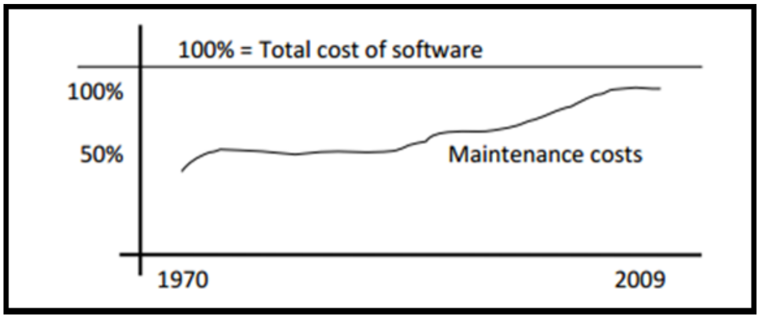


Figure 1.1 Increase of software maintenance costs as percentage of total cost (adapted from TIOBE Software (2014)).

The think-aloud method (verbal protocol) and observational studies (video-taping) along with other methods, such as questionnaires, surveys, and interviews, have been used as the main data collection methods for program comprehension studies (Fan, 2010; Bednarik, 2012; Siegmund et al., 2014). These methods, when skilfully used, can provide valuable insights regarding program comprehension. However, they have limitations (Siegmund et al., 2014). These methods rely on self-reporting and suffer more or less from the Hawthorn (Observer) effect (Eteläpelto, 1993; Fan, 2010) in which participants tend to work harder and to be under pressure to deliver an understandable response to study conductors. These methods also require considerable manual transcription and analysis to obtain results.

With the emergence of modern eye-trackers, researchers in the software engineering community have started to perform eye-tracking studies to study the cognitive processes underlying program comprehension. Compared to the aforementioned traditional methods, eye-trackers are cost-effective way of collecting data at a fine level of details with less intrusion in a developer's cognitive processes (Just and Carpenter, 1980; Bednarik and Tukiainen, 2006). An eye-tracker also provides information that are not available from the above-mentioned traditional methods, including the developers' extent and patterns of visual attention (Duchowski, 2007; Calitz et al., 2009; Kara Pernice, 2009; Bednarik, 2012). Analyzing developers' visual attention helps us to better understand the cognitive process behind program comprehension (Rayner, 1978; Duchowski, 2007) by explaining on which parts of stimulus material do participants focus and how they go back and forth between these parts.

Consequently, to examine the evidence and identify the advantages and problems associated with eye-trackers, we perform a mapping study and evaluate all research relevant to the usage of eye-tracking technique in software engineering. We identify 36 relevant publications out of which 22 focused on program comprehension (either code or model comprehension). The results of this mapping study emphasize that the eye-movement data provided by an eye-tracker is exploited as a proxy of visual attention by different studies and it has been used to provide additional insights on how developers perform different tasks. However, we identified the following gaps that we aim to address in this dissertation:

1. To perform program comprehension, in addition to source code, developers benefit from other artifacts (if any are available) to better understand the system. These artifacts can impact the cognitive process by providing different levels of cognitive support (Walenstein, 2002, 2003). Cognitive support refers to the mental assistance provided by different tools or artifacts to ease the cognitive process of developers performing a comprehension task (Walenstein, 2002, 2003).

   Software artifacts take the forms of graphical (diagrams) and textual representations.

Graphical representations (*e.g.,* UML diagrams) play an important role in presenting software artifacts and they are effective tools (Moody, 2009) (1) to provide a quick understanding of data, (2) to enhance data comparison and processing, and (3) to improve and facilitate the communication process between end-users and developers. Conventional wisdom assumes that graphical representations carry information more effectively to non-technical people than textual ones (Moody, 2009).

Thus, a huge amount of work has been dedicated to propose different graphical representations under the assumption that developers prefer to work with graphical representations rather than text to understand systems. Only a handful of studies investigated the effectiveness of graphical vs. textual representations or the developers' preferences (*i.e.,* textual vs. graphical representations) in program understanding tasks (Somervell et al., 2002; Snook and Harrison, 2007; Heijstek et al., 2011; Ottensooser et al., 2012).

2. Developers' individual characteristics influence the way a developer performs program comprehension (Brooks, 1980; Storey, 2005). Differences in the performance of novices and experts are well studied and studying these differences is a well-recognized, accepted way to study program comprehension (Ormerod, 1990).

   However, there is still a great divergence between developers' abilities that cannot be detected and measured only by their level of expertise (Brooks, 1980; Storey, 2005). This observation by Storey (2005) along with the work of Burnett and her colleagues in the Gender HCI project[1] inspire us to take into account gender differences while studying program comprehension using eye-tracking technique.

   Although a significant amount of research has been dedicated to the impact of education and society on the success of women in technology, the possibility of gender issues in software engineering has only started to receive attention recently (Grigoreanu et al., 2009; Burnett et al., 2011). Moreover, no previous studies take into account gender differences as a factor that can impact program comprehension.

3. Developers explore software system using different strategies to obtain the relevant information. Strategy refers to a reasoned plan or method for achieving a specific goal (Grigoreanu et al., 2009). As the eye-tracker allows us to identify the areas of interest in which developers spent more time, it also helps to highlight different cognitive strategies by exhibiting the patterns employed by developers to explore these areas (*i.e.,* viewing strategies). Therefore, several eye-tracking studies used eye-movement data to identify the viewing strategies deployed by developers to perform the tasks (Crosby and Stelovsky, 1990; Crosby et al., 2002; Hejmady and Narayanan, 2012; De Smet et al.,

[1] http://eusesconsortium.org/gender/

2012). However, no study investigated the impact of gender and representation type on developers' viewing strategies during program comprehension.

1.2 Thesis Statement

Representation type (graphical vs. textual) and gender impact the cognitive process underlying comprehension activity. Therefore, they influence developers' efficiency and effectiveness. Representation type and gender are also proxy for developers' viewing strategies, which can be inferred partly from the developers' eye-movements while developers perform program comprehension tasks.

1.3 Contributions

While researching to validate our thesis statement, we perform the following contributions:

1.3.1 Mapping Study on the Usage of Eye-tracking Technique in Software Engineering

We perform a mapping study to provide a comprehensive view of the usage and usefulness of eye-tracking technique in software engineering. Our mapping study covers eye-tracking studies in software engineering published from 1990 up to the end of 2013. To search all recognized resources, instead of applying manual search, we perform an extensive automated search using Engineering Village [1]. We identify 36 relevant publications including nine journal articles, two workshop articles, and 25 conference articles. Our results show that the software engineering community started to use eye-trackers in 1990 and that have become increasingly recognized 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*, *traceability*, and *usability*.

Moreover, we find that the studies use different metrics based on eye-movement data to collect quantitative measurements. We classify these metrics in three groups: (1) visual effort and efficiency metrics to measure developers' visual efficiency; (2) visualize gaze behaviour metrics to visualize and display developers' eye-movements; and (3) scan-path metrics to identify, compare, and display developers' viewing strategies.

In addition, we discuss in details threats to the validity of experiments using eye-trackers along with suggestions on how to mitigates these threats. However, acknowledging these

[1] http://www.ei.org/engineering-village

threats and setting limits on what is realistically possible, we conclude that the advent of new eye-trackers makes the use of these tools easier and unobtrusive and the software engineering community should use them more to perform empirical studies.

Furthermore, we identify gaps in current research that inspires us to perform a set of empirical studies, presented in this dissertation. Using eye-tracking data, we measure and quantify the cognitive effort involved in program comprehension and correlate it with gender, representations types, and developers' viewing strategies.

Our results are currently under review in the Journal of Information and Software Technology:

– Zohreh Sharafi, Zéphyrin Soh, Yann-Gaël Guéhéneuc. *Systematic Literature Review on the Usage of Eye-tracking Technique in Software Engineering.* Submitted to The Journal of Information and Software Technology (IST), December 2014, Under revision as of February, 12^{th}, 2015.

1.3.2 Impact of Representation Type on Program Comprehension

We undertake an empirical study to measure the amount of cognitive supports provided by different representation types (graphical vs. textual) for comprehension tasks along with the viewing strategies deployed by developers to perform the tasks using these representations. To assess the amount of cognitive support, we measure developers' effectiveness by the percentages of correct answers and their efficiency based on the time and the effort that developers spend to perform the tasks. We also consider effort as the amount of visual attention that developers must spend to perform their tasks: less attention and less time means less effort. We detect and compare viewing strategies based on the exact location and the duration of eye-movement data using a novel quantitative approach.

Our results confirms the superiority of structured text over a graphical representation for comprehension tasks of a small system regarding developers' spent time and effort. However, we do not observe a significance difference between the accuracy obtained when the graphical representation was used vs. the textual representation.

Surprisingly, the developers' preference is largely in favor of the graphical representation, even though it requires a higher effort. Moreover, our participants spent significantly less time and less effort while working with a mixed representation in which both graphical and textual information are available at the same time compared to working with only one representation type. This finding confirms that the formalism of a graphical representation must be learnt by users and that training is required before the benefits of a graphical representation can materialise. Additionally, our results imply that a structured text can be more effective than a graphical representation while dealing with comprehension of small systems. Our findings help developers and maintainers to choose the right type of artifact for a given problem.

Moreover, we use a quantitative method to identify the viewing strategies of developers while using graphical representations. Using this method, we detect developers' viewing strategies and compare them with each other. We find that the hierarchical structure of the representation lead developers to deploy different viewing strategies, either top-down or bottom-up. This observation emphasizes the importance of the the layout of representation in comprehension.

This finding encourages us to take one step back and look at the very first phase of program comprehension, which is source code reading, and analyze source code as a type of structured textual representation. Moreover, because our developers' preference is not consistent with our finding regarding the effectiveness of textual over graphical representations, we also investigate whether gender differences play a role in program comprehension or not.

Our study have resulted in the following conference publication:

– Zohreh Sharafi, Alessandro Marchetto, Angelo Susi, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. *An Empirical Study on the Efficiency of Graphical vs. Textual Representations in Requirements Comprehension.* Proceedings of the 21^{st} International Conference on Program Comprehension (ICPC), May 2013, IEEE Computer Society Press.

1.3.3 Impact of Gender on Program Comprehension

Thus, we conduct a set of empirical studies with eye-tracker to analyze the impact of gender on developers' efficiency and effectiveness while performing program comprehension tasks. We analyze the impact of the location of identifiers (*i.e.,* Source Code Entities (SCE) including: class name vs. method name vs. variable names along with comments) and the identifier style (camel case vs. underscore) on male and female developers' efficiency and effectiveness while performing program comprehension.

We find no difference between camel case and underscore regarding developers' effectiveness, efficiency, and viewing strategies. In addition, our results bring evidence that no difference exists between male and female developers regarding accuracy and efficiency for a given task. Our results also show that developers, both males and females, spend more visual attention on method names, comments, variable names, and class names, respectively.

We also investigate the different viewing strategies used by male and female developers to read the source code and answer comprehension questions. We observe that female developers follow different viewing strategies compared to male developers. Female developers are more careful and spend more visual effort (attention) on selecting and ruling out wrong identifiers, although, the average time to complete the task and accuracy of female developers are not statistically different to that of the male developers. In addition, with regards to viewing strategies deployed during source code reading, our results emphasize that male and female developers use different strategies. Female developers spend more time on different SCEs to carefully analyze them, then go to the next one. However, male developers scan the source code rapidly, then, go back and forth between different SCEs and spend time on the SCEs that they find useful.

Our studies shed light on the viewing strategies that male and female developers perform during program comprehension, in addition to the areas of code on which they focus. Yet, we cannot generalize our findings because our population is not large enough. Our findings at the moment should be considered more as an additional call with respect to previous work (Storey, 2005; Grigoreanu et al., 2009; Burnett et al., 2011), for further studies concerning the role of gender in software engineering in general and program comprehension in particular.

Our results have been published in Empirical Software Engineering Journal (EMSE) and also resulted in two conference publications:

– Nasir Ali, Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol, *An empirical study on the importance of source code entities for requirements traceability*, Empirical software engineering Journal (EMSE), July 2014, Springer.

– Zohreh Sharafi, Zéphyrin Soh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. *Women & Men: Different but Equal: A Study on the Impact of Identifiers on Source Code Understanding.* Proceedings of the 20th International Conference on Program Comprehension (ICPC), June 2012, IEEE Computer Society Press.

– Nasir Ali, Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. *An Empirical Study on Requirements Traceability Using Eye-Tracking.* Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), September 2012, IEEE.

1.4 Organization of the Dissertation

Figure 1.2 provides an overview of the chapters of this dissertation along with the main questions that drove this research and our major conclusions. The reminder of this dissertation is organized as follows:

Chapter 2: We provide the necessary background for this dissertation. This chapters starts with background on eye-tracking technique followed by discussion on different eye-trackers and metrics. It also summarizes the experiment process in general and planning and analysis steps in particular.

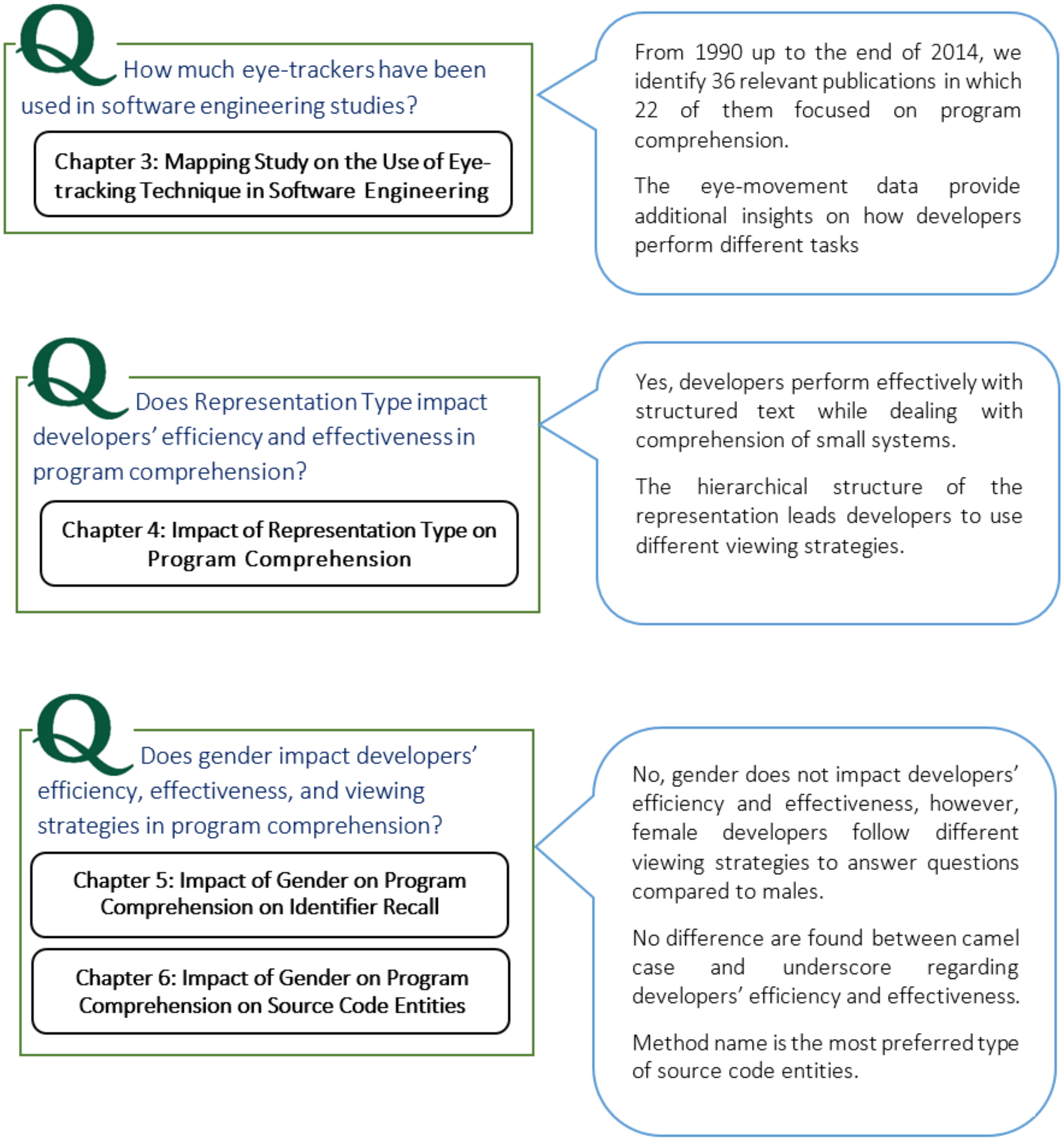
Chapter 3: We present the mapping study that we undertook to investigate the usefulness of eye-trackers in software engineering.

Chapter 4: We analyze the impact of representation type (graphical vs. textual) on developers' efficiency and effectiveness while performing model comprehension tasks.

Chapter 5: We provide necessary background and presents our motivation and provides prelude about our work analyzing the impact of gender in program comprehension. In this chapter, we also report the results of a study investigates the effect of gender in developers' ability to recall identifiers in program comprehension.

Chapter 6: We present the results of a study contains two experiments that report the differences between male and female developers' viewing strategies while performing comprehension tasks. We also identify the most important source code entities, including class name, method name, variable name, or comment to which male and female developers pay more visual attention while reading source code to perform these tasks.

Chapter 7: We conclude the dissertation by summarizing the contributions of our work and outlining future directions.



Q How much eye-trackers have been used in software engineering studies?

Chapter 3: Mapping Study on the Use of Eye-tracking Technique in Software Engineering

From 1990 up to the end of 2014, we identify 36 relevant publications in which 22 of them focused on program comprehension.

The eye-movement data provide additional insights on how developers perform different tasks

Q Does Representation Type impact developers' efficiency and effectiveness in program comprehension?

Chapter 4: Impact of Representation Type on Program Comprehension

Yes, developers perform effectively with structured text while dealing with comprehension of small systems.

The hierarchical structure of the representation leads developers to use different viewing strategies.

Q Does gender impact developers' efficiency, effectiveness, and viewing strategies in program comprehension?

Chapter 5: Impact of Gender on Program Comprehension on Identifier Recall

Chapter 6: Impact of Gender on Program Comprehension on Source Code Entities

No, gender does not impact developers' efficiency and effectiveness, however, female developers follow different viewing strategies to answer questions compared to males.

No difference are found between camel case and underscore regarding developers' efficiency and effectiveness.

Method name is the most preferred type of source code entities.

Figure 1.2 Overview of the dissertation chapters.

CHAPTER 2 BACKGROUND

In this chapter, we discuss the necessary background to this dissertation, starting with eye-tracking in Section 2.1 followed by experiment process in Section 2.2.

2.1 Eye-tracking Technique

Eye-trackers are designed to collect a participant's visual attention by recording eye-movement data (Rayner, 1978; Duchowski, 2007). Visual attention can trigger the mental processes required for comprehension and solving a given task. Therefore, eye-trackers are useful to study the cognitive behavior and effort involved in solving the task (Duchowski, 2007).

Eye-gaze data is studied with respect to certain areas on the stimuli (*e.g.,* diagram or source code) that are called Areas of Interest (AOIs). An AOI can be relevant to the participant while answering a question or can be irrelevant. For example, if we consider a class diagram as a stimuli, an irrelevant AOI can be any class or any notation while a relevant AOI could be a specific class that is relevant to the given task.

Eye-gaze data are classified based on the following significant indicators of ocular behavior (Rayner, 1978; Duchowski, 2007), namely:

- Fixation: it is described as a spatially stable eye-gaze that lasts for approximately 200-300 ms (on average, three eye fixations happen per second during active looking). During a fixation, the visual attention is focused on a specific area of display. Researchers in psychology claim that most of the information acquisition and processing occur during fixations. In addition, only a small set of fixations is enough for human brain to acquire and process a complex visual input (Privitera and Stark, 2000). Interpreting the meaning of fixations are completely context-dependent. Higher fixation rate on specific AOI indicates great interest in encoding tasks such as browsing a web-pages. However, a cluster of fixations indicate uncertainty for a search task (Poole and Ball, 2005).
- Saccade: it is a continuous and rapid eye-gaze movement that occurs between fixations. Saccadic eye-movements are extremely rapid (within 40-50 ms). Therefore, the cognitive process that is happening during saccades is very limited.
- Pupil dilation: it is dilation of the pupil of the eye. Larger pupils may indicate more cognitive effort (Poole and Ball, 2005).
- Scan-path: it is a series of fixations (fixation-based) or visited AOIs (AOI-based) in chronological order. An AOI is visited if there is at least one fixation in it. However, the fixation-

based scan-paths become very long and complex due to the large quantity of fixation data. Therefore, analysing the fixation-based scan-paths is laborious and complex to reason about (Lorigo et al., 2006) and, also most of the time, it is impossible to recognize the path that is visualised.

Thus, several researchers calculate and visualise AOI-based scan-paths to examine how participants scan the stimulus and find relevant AOIs (Lorigo et al., 2006; De Smet et al., 2012; Sharafi et al., 2013). Using AOI-based scanpaths, the large quantities of data can be significantly reduced without losing meaningful information and make the analysis of the data more efficient. In addition, because an AOI-based scan-path shows how participants switch their focus of attention from one AOI to another, it helps to reason about the trend in which different AOIs are visited and focused.

2.1.1 Assumptions

The relation between eye-movements and comprehension is defined based on two assumptions: the immediacy assumption and the eye-mind assumption (Just and Carpenter, 1980). The immediacy assumption is that as soon as a reader sees the word, she tries to interpret it. The eye-mind assumption is that reader fixated her attention on the word until she comprehends the word (Just and Carpenter, 1980).

Our dissertation along with other previous works (Crosby and Stelovsky, 1990; Crosby et al., 2002; Fan, 2010) support these assumptions and use them to interpret fixations that are recorded during program comprehension. Obviously, we also assume that a participant is not day dreaming during given tasks.

2.1.2 Devices

With the emergence of eye-trackers, researchers in different disciplines start analysing the relationship between eye movements and cognitive processes. Eye-trackers are divided into two main categories:

- Intrusive eye-trackers: these devices consist of three miniature cameras that are mounted on a padded headband (a heavy goggle and wires as shown in Figure 2.1). Two infrared eye cameras are used to track the eyes' movements, while an optical head-tracking camera that is integrated into the headband allows an accurate tracking of the participant's head. Instability is the major problem of these devices in which even a light head movement of participants may result in an inaccurate detection of eye movement. Therefore, the head of participants must be fixed when wearing the padded headband. The *Eye-link II* from SR Research Ltd (2006) is an intrusive eye-tracker that has the resolution (noise limited

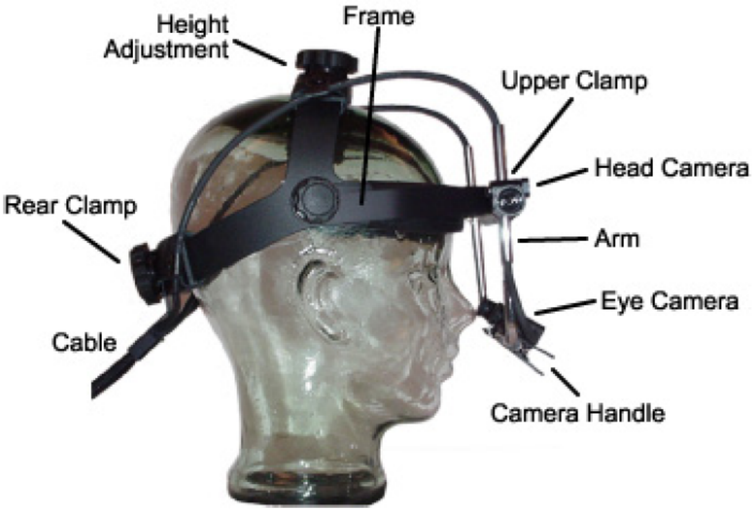


Figure 2.1 Example of goggle used in intrusive eye-trackers (adapted from (Duchowski, 2007)).

at < 0:01 degree) and data rate (500 samples per second).

- Non-intrusive Eye-tracker: these devices consist of one computer (either a desktop PC or a laptop) and two miniature cameras and one infra-red pad as shown in Figure 2.2. By tracking the position of the participant's head using her eye-brows, nose, and lips, the device tracks the participant's gaze. To avoid capturing head movement instead of eye-gaze data, these systems use two features, namely corneal reflection (of an infra-red beam) and pupil center to distinguish head movement from eye-rotation.

In this dissertation, we use *FaceLAB* from Seeing Machine (2010), which is a non-intrusive, video-based eye-tracker to capture eye-gaze data.

2.1.3 Tools

An eye-tracker captures the eye-gaze data and records them in the files. These files mainly contains the start and the end time of the fixations along with the duration of fixations and the position of eye pupils. These data must be processed and sometimes filtered or corrected. Most of the eye-trackers are accompanied by analysis tools to visualise, modify, and analyse the data recorded by eye-trackers. In the following, we present two commercial analysis tools along with an open-source tool, called *Taupe*, which is designed and developed in our research group (Ptidej Team)[1].

- *Eye-link II Windows Developer Kit*: *Eye-link II* eye-tracker has been accompanied by this development kit, which provides sample display programs, C source code, and instructions

[1] http://www.ptidej.net/research/taupe

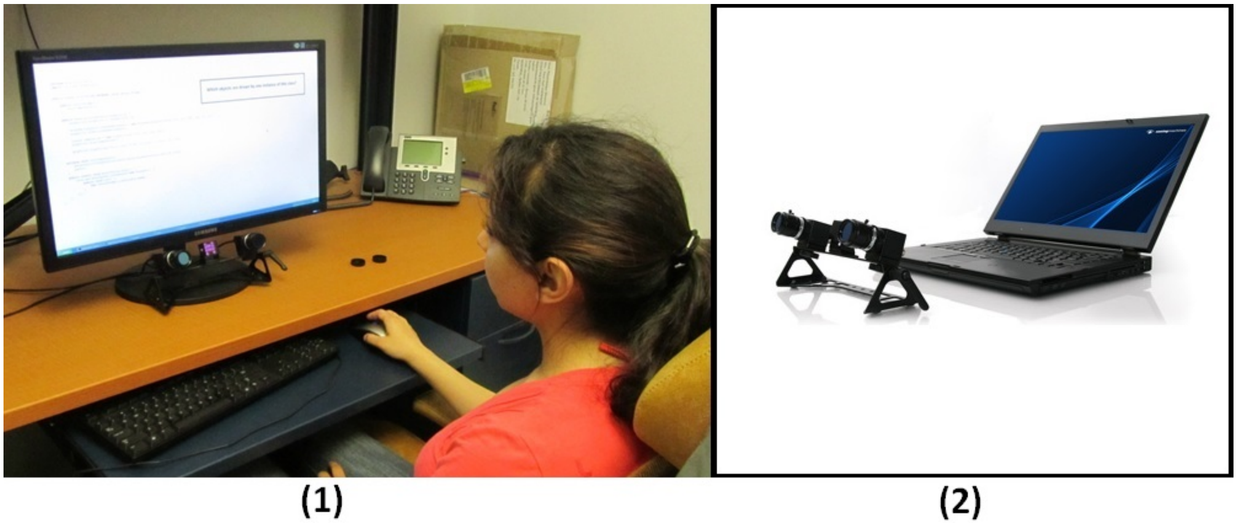


Figure 2.2 1) FaceLAB is an example of video-based eye-tracker that we used in this work (adapted from FaceLAB manual (Seeing Machine, 2010)). 2) Participant sits in front of the computer and a video-based eye-tracker captures the eye-gaze data.

for creating experimental programs. To display eye fixations and scan-paths on top of the stimulus, the kit uses Data Viewer 3[2] and also stored files containing eye-gaze data in the Host PC. One output file contains eye fixations, saccades and the information related to the AOIs.

– *GazeTracker*: it receives eye-gaze data from *FaceLAB* device via network. This tool provides a list of visualizations and analysis for the eye-gaze data.

– *Taupe* (De Smet et al., 2012): *Taupe* can import and analyse eye-gaze data provided by different eye-trackers including *Eye-link II* and *FaceLAB*. Taupe is enhanced with a user-friendly GUI that allows researchers to analyse and edit eye-gaze data separately based on different criteria, *e.g.,* participants' expertise or gender. It supports execution of a set of algorithms to analyse and visualise data. In addition, *Taupe* enhances with offset correction feature that allows users to correct the locations of fixation. Three types of offset have been reported in the literature including: static offsets, non-static offsets, and chaotic offsets. The offset correction feature of *Taupe* allows us to apply a constant transformation function and change the coordinates of all fixations in the case of static offsets. For non-static and chaotic offsets, user can select a group of fixations or one fixation at a time and apply transformation respectively. In addition, a set of visualisation techniques including heatmap, convex hull and scan-path visualizations are also supported in *Taupe*.

In contrast with *Data Viewer* and *GazeTracker*, *Taupe* is free and open-source. It is

[2] http://www.sr-research.com/accessories_EL1000_dv.html

designed to be extensible and therefore it can be extended with new parsers and analyses for different eye-trackers. In this dissertation, we use *Taupe* to analyse and visualise our eye-tracking data.

2.1.4 Metrics

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 visualisations techniques have been used to display eye-movements. Visualising eye-movements helps performing qualitative analysis to better understand participants' behaviour. A same metric or visualisation 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.

Visual Effort and Efficiency Metrics

Tables 2.1, 2.2, 2.3, and 2.4 summarise metrics used in previous eye-tracking studies to measure the amount of visual effort and efficiency.

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 (Goldberg and Kotval, 1999).
- **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 and Kotval (1999) 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 (Poole and Ball, 2005). For browsing/encoding tasks, a higher fixation rate for a specific AOI indicates that the participants show a great interest in that AOI (Jacob and Karn, 2003). Yet, it could indicate that the this area is difficult to encode (Jacob and Karn, 2003). 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 (Poole and Ball, 2005).

Table 2.1 Metrics for visual effort calculation based on the number of fixations.

		Names	Formulas
Visual Effort and Efficiency Metrics	Number of Fixation	Fixation Count (FC)	$FC = \text{Total number of fixations in AOI} \quad (2.1)$
		Fixation Rate (FR)	$FR = \frac{\text{Total Number of Fixations in AOI}}{\text{Total Number of Fixations in AOG}} \quad (2.2)$
		Spatial Density	$SD = \frac{\sum_{i=1}^{n} c_i}{n} \quad (2.3)$ n: number of fixations in the specific area (for one cell). c_i: equal to 1 if the area number i visited, otherwise 0
		Convex hull Area	Area of the smallest convex set of fixations (2.4)

Table 2.2 Metrics for visual effort calculation based on the duration of fixations.

		Names	Formulas
Visual Effort and Efficiency Metrics	Duration of Fixations	Average Fixation Duration (AFD)	$AFD(AOI) = \frac{\sum_{i=1}^{n}(ET(F_i) - ST(F_i))\text{in AOI}}{n}$ (2.5) $ET(F_i)$ and $ST(F_i)$: the end time and start time for fixation F_i n: the total number of fixations in a given AOI
		Ratio of $ON\_targetAll\_target$ Fixation Time (ROAFT)	$ROAFT = \frac{\sum_{i=1}^{n}(ET(F_i) - ST(F_i))inAOI}{\sum_{j=1}^{n}(ET(F_j) - ST(F_j))\text{in AOG}}$ (2.6)
		Fixation Time (FT)	$FT = \text{Total duration of all fixations in AOI}$ (2.7)
		Average Duration of Relevant Fixations (ADRF)	$ADRF = \frac{\text{Fixations Duration of Relevant AOIs}}{\text{Total Number of Relevant AOIs}}$ (2.8)
		Normalised Rate of Relevant Fixations (NRRF)	$NRRF = \frac{ADRF}{\frac{\text{Fixation Duration of All AOIs}}{\text{Number of All AOIs}}}$ (2.9)

- **Fixation Spatial Density** was proposed by Goldberg *et al.* (Goldberg and Kotval, 1999) 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 efficient search, which is highly focused. De Smet et al. (2012) 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 (Goldberg and Kotval, 1999).

  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 (Duchowski, 2007; Goldberg and Kotval, 1999). This metric is computed as shown in Equation 2.5, Table 2.2. Longer fixations show that participants spend more time analysing 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 and Kotval (1999) 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 and Carpenter (1976), 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 the whole stimulus (Poole and Ball, 2005).
- **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. (2009) to compare two or more diagrams with each other regarding the impact of the Visitor design pattern.

Table 2.3 Metrics for visual effort calculation based on saccades.

		Names	Formulas
Visual Effort and Efficiency Metrics	Saccades	Number of saccades	Total number of saccades (2.10)
		Saccade duration	Total duration of saccade (2.11)

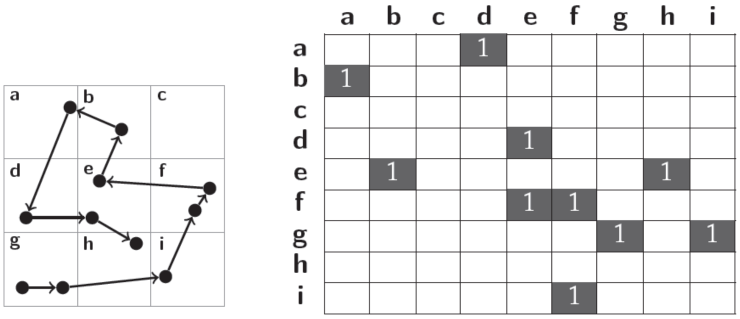
Metrics based on saccade

– **Number of saccades** represents the total number of saccades in a stimulus. A higher number of saccades indicates more searching (Poole and Ball, 2005).

– **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 (Poole and Ball, 2005) to compute a visual effort. Saccade amplitude indicates meaningful load cues. The higher the saccade amplitude, the lower the mental effort (Poole and Ball, 2005). Saccades are usually rightward (progressive). However, sometimes they may be backward (leftward in text reading) or regressive, which indicates difficulties in understanding some text (Sibert et al., 2000) or the presence of less meaningful cues in the stimulus (Poole and Ball, 2005).

Metrics based on scan-paths

– **Transitional matrix** is a tabular representation that shows the frequency of transitions



	a	b	c	d	e	f	g	h	i
a				1					
b	1								
c									
d					1				
e		1						1	
f					1	1			
g							1		1
h									
i						1			

Figure 2.3 Example of scan-path and corresponding transition matrix (De Smet et al., 2012).

Table 2.4 Metrics for visual effort calculation based on scan-paths.

Visual Effort and Efficiency Metrics	Metrics Based on Scan-paths	Names	Formulas
		Transitional Matrix	$TM = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} c_{i,j}}{n.n}$ (2.12) n: number of fixations in the specific area (for one cell). c_i: equal to 1 if the area number i visited, otherwise 0
		Attention Switching Frequency	The number of switches between AOIs (2.13)
		Scan-path Precision	$\frac{SP \cap RR}{SP}$ (2.14) SP: number of AOIs that are visited (fixated). RR: number of relevant AOIs that are visited.
		Scan-path Recall	$\frac{SP \cap RR}{RR}$ (2.15) SP: number of AOIs that are visited (fixated). RR: number of relevant AOIs that are visited.
		Scan-path F-measure	$2 * \frac{SPP * SPR}{SSP + SPR}$ (2.16) SP: number of AOIs that are visited (fixated). RR: number of relevant AOIs that are visited.

between defined AOIs (Ponsoda et al., 1995). Equation 2.12 in Table 2.3 presents this metric. In addition to the search area, this metric also considers the temporal order of the search by detecting movements over time (Goldberg and Kotval, 1999). 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., 2012) describes how to compute and use a transitional matrix. Figure 3.5 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 (Goldberg and Kotval, 1999).

- **Attention switching frequency** is the number of switches between two specific AOIs. This metric is used in (Bednarik and Tukiainen, 2006, 2008).
- **Scan-path precision (SPP)** was proposed by (Petrusel and Mendling, 2013) as the percentage of relevant AOIs visited from all defined AOIs in the stimulus.
- **Scan-path recall (SPR)** was proposed by (Petrusel and Mendling, 2013) 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, 2013) 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 (Goldberg and Kotval, 1999). 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 (Goldberg et al., 2002).

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

- **Edit distance** is based on the Levenshtein algorithm (Levenshtein, 1966), 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. (2012) 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 analysing eye-tracking data (Henderson and Pierce, 2008). Consequently, new techniques have been proposed to consider also the duration of fixations.

– **Sequential PAttern Mining (SPAM)** was proposed by Ayres et al. (2002) 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 categorising fixations as short (less than 500 milliseconds) and long (higher than 500 milliseconds). The threshold (500 milliseconds) 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 visualisation more often than those who were less familiar with the IDE.

– **ScanMatch** was defined by Cristino et al. (2010), 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.

Visual Gaze Behaviour

Eye-tracking studies in software engineering have used so far three types of visualisation 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. Figure 3.6 shows heat-maps that are presented and discussed in (Sharif and Maletic, 2010a; Busjahn et al., 2011) while Figure 3.7 shows heat-maps presented in (Ali et al., 2012; Sharafi et al., 2012). (Sharif and Maletic, 2010a) investigates the impact of different layouts on the comprehension of design patterns. It uses two heat-maps to visualise 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 visualise 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 visualise 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, 2010a) uses a gaze plot to compare experts and novices performing design pattern comprehension tasks. Experts' gaze plots (as presented in Figure 3.8) 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 Figure 3.8) 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 Figure 3.9 to show different parts of source code that attracts different levels of attention and consequently time.

2.2 Experiment Process

Wohlin et al. (2000) define five steps to carry out an experiment that we recall here, because we use them to conduct eye-tracking experiments presented in this dissertation.

2.2.1 Definition

Experimenter defines the problem, the goals, the objectives of the experiment, and the experiment's hypotheses.

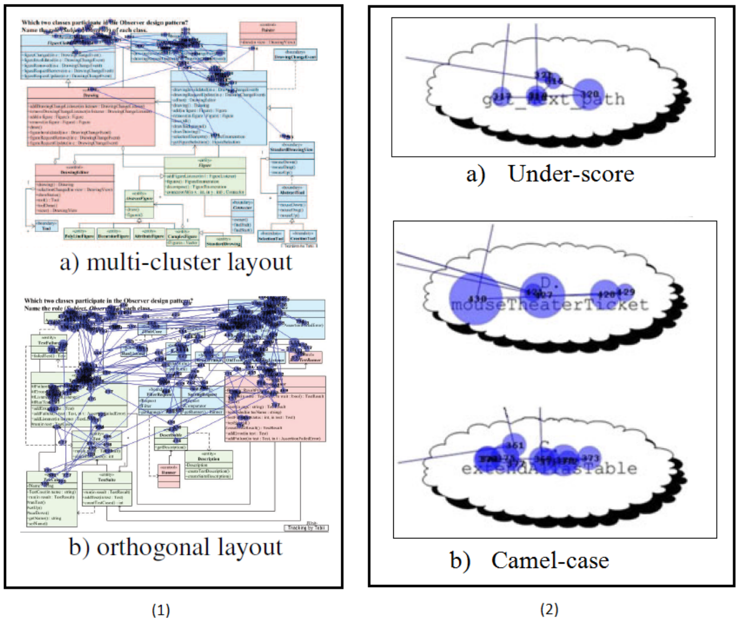
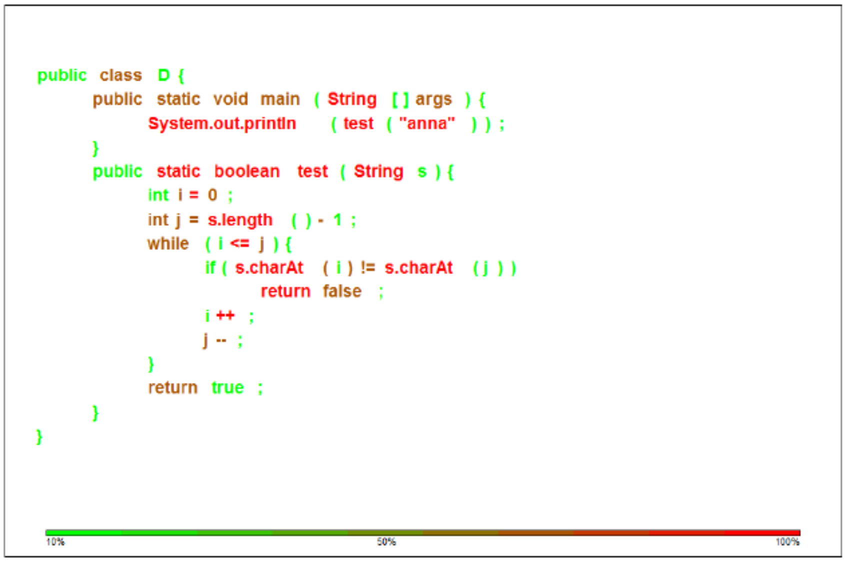


Figure 2.4 (1) shows the gaze plots of an expert for the Observer pattern for orthogonal and multi-cluster layouts (Sharif and Maletic, 2010a). (2) shows gaze plots for underscore and two camel case 3-word code identifiers (Binkley et al., 2013).



```
public class D {
        public static void main ( String [ ] args ) {
                System.out.println      ( test ( "anna" ) ) ;
        }
        public static boolean test ( String s ) {
                int i = 0 ;
                int j = s.length ( ) - 1 ;
                while ( i <= j ) {
                        if ( s.charAt ( i ) != s.charAt ( j ) )
                                return false ;
                        i ++ ;
                        j -- ;
                }
                return true ;
        }
}
```

Figure 2.5 Color-coded attention allocation map based on the number of fixations per word (Busjahn et al., 2011).

2.2.2 Planning

Experimenter designs the experiment by: 1) stating the hypotheses formally and 2) determining the study variables.

Hypothesis Statement

The definition of an experiment is formally expressed through a hypothesis statement. Two hypotheses must be formally stated and tested:

1. Null hypothesis (H_0) states that there is no difference in the trends of two populations and any differences in findings results from a coincidence. It is an interest of the experimenter to reject the null hypothesis.
2. Alternative hypothesis (H_1) states that there is an observed effect for the experiment. The experimenter tries to demonstrate and to accept the alternative hypothesis in an indirect way.

Different types of risks are involved during hypothesis testing that are referred to as:

1. *Type-I-Error (α):* the error of rejecting (H_0) when it is true.
2. *Type-II-Error (β):* the error of failing to reject (H_0) when it is false.

To determine whether the result is statistically significant and to reject the null hypothesis (H_0), an experimenter must calculate the p-value that is the estimated probability of rejecting the H_0. Typically in software engineering (this dissertation included), the minimum confidence required is 95%, thus an accepting maximum of *Type-I-Error i.e.,* (α) is 5%. If a p-value is smaller than α, H_0 is rejected and the H_1 is accepted.

Study Variables

There exist three types of variables:

1. Independent variables (IV): there are input variables that an experimenter can control and change to observe their impacts on dependent variables.
2. Dependent variables (DV): these response variables are those that the experimenter is interested to study their variants in the experiment when the independent variables change.
3. Mitigating variables (MV): these confounding variables change systematically when one or more independent variables change. They propose an alternative explanation of the observations and how dependent and independent variables are related.

The experimenter also determines the values that these variables can actually take and consequently, she chooses the measurement scale based on these values. Moreover, the participants must be selected.

Threats to Validity

The experimenter must consider the threats to the validity of the experiment. There are four major types of these threats to the validity that we discussed here:

1. Internal: this type is concerned with the extent to which the casual conclusion is derived based on the relation between the independent and dependent variables. The experimenter must verify whether the observation can be attributed to the dependent variables rather than other possible causes. In this dissertation, we consider three threats to the internal validity: maturation, instrumentation, and diffusion of the treatments. We mitigate the impact of maturation by assigning the different pieces of source code or representations randomly to our participants. This random ordering prevents fatigue effect concerning the stimulus given at the end. The instrumentation threat is related to the equipment that we used in our experiments. We used a video-based eye-tracker that does not involve any heavy goggle. Participants can move their head easily without changing the calibration of camera. We also used a dentist chair to support participants' neck to make them comfortable and avoid fatigue. To mitigate the possible diffusion of the treatments, we asked our participants not to talk about the experiment with the other participants.
2. External: this type is related to the generalization of the results. The selection of participants and–or objects that are not representative of the studied population is an example of threats to external validity. In this dissertation, we used students as participants in our experiments and we did not distinguish novices and experts. Kitchenham et al. (2002) mentioned that "using students as participants is not a major issue as long as you are interested in evaluating the use of a technique by novice or non-expert software engineers. Students are the next generation of software professionals and, so, are relatively close to the population of interest." Moreover, the number of participants also impacts the results. Previous eye-tracking studies (Bednarik, 2012; Sharif et al., 2012a; Cagiltay et al., 2013; Walters et al., 2014; Busjahn et al., 2014; Rodeghero et al., 2014a) mainly had around 15 to 20 participants and they mentioned that eye-tracking studies usually have about the same number of participants. In this dissertation, we have around 25 participants for each experiment, which is much more than some previous eye-tracking studies.

Hawthorne effect (also referred to as the observer effect) is one of the threats to external validity. While performing an experiment, an experimenter is responsible for providing guidance to the participants throughout the experiment. In case of conducting an eye-tracking experiment, the experimenter must calibrate the camera, and constantly checks the recording to ensure that the eye-tracker does not stop tracking the eyes. The experimenter's presence can bias the results due to the Hawthorne (observer) effect, because participants feel that they are being watched. In this dissertation, we explain to participants that eye-tracker does not provide any video and we provided the minimum interaction between the participants and the experimenter during the experiments.

3. Construct: if we define construct as an ability, or skill that happens or forms in the human brain (people's mind) such as overall English language proficiency, construct validity refers to whether an experiment measures the construct adequately or not. In other words, this type evaluates ”whether the theoretical constructs are interpreted and measured correctly” (Easterbrook, 2007). Hypothesis guessing is one of the threats to this validity that happens when participants try to guess the intent of the experiment and consciously, or subconsciously, modify their behavior.

   In this dissertation, we did not informed the participants about the precise goal of the experiment to avoid hypothesis guessing. We explained them the process of performing the experiment, the number of session, and questions that they must answer. We did not set a time limit, we asked participants to answer the questions as soon as they can.

4. Conclusion: this type or statistical conclusion validity holds when the conclusions of an experiment are constructed based on an adequate analysis of the data using adequate statistical tests. Low statistical power is an example of threats to this validity, which increases the likelihood of making *Type-II-Error (β).* In this dissertation, we do not consider any assumption regarding the distribution and normality of our data, therefore, we use non-parametric, non-paired statistical tests to determine significant differences. Moreover, we choose well-documented metrics from the previous works as presented in Section 2.1.4 to ensure the reliability of our measures. Finally, we make sure that the eye-tracker is well calibrated for every participant before collecting data.

2.2.3 Operation

There are three main steps to conduct an experiment:

1. Preparing the materials as well the participants: the experimenter must contact the participants and have their consent.

2. Executing the experiment and collecting data: the experimenter must install the experiment and and properly record and save the data.
3. Validating the data: the experimenter must check to confirm that the collected data is correct.

2.2.4 Analysis and Interpretation

This step usually starts with applying descriptive statistics to better understand the data and to check if any pre-processing (*e.g.,* noise removal) is required before starting to analyze and to interpret the data.

The next step is hypothesis testing where the experimenter chooses the statistical tests based on the measurement scales and the values of the results. In this dissertation, we use the following statistical tests for measuring the dependency and effect size.

- Wilcoxon signed-rank test (Wohlin et al., 2000): it is used to decide whether two data distributions are identical without assuming that they follow a normal distribution.
- Logistic regression models (Hosmer Jr and Lemeshow, 2004; Venables and Ripley, 2002): it models the relationship between a dependent variable and one or more independent variables. This model helps to check the fit and the significance of the relationships between dependent and independent variables. We use logistic regression analysis as a proxy for correlation when the dependent variable is a dichotomous variable, *e.g.,* the answer to a question is either correct (1) or wrong (0).
- Contingency table: it is a table in the format of a matrix (two-way) for displaying the frequency distribution of the variables (Venables and Ripley, 2002). It is the categorical equivalent of the scatter plot and it is a useful tool to record and analyze the relationships between categorical variables.

Finally, the experimenter examines and interprets the results to determine whether to accept or reject the hypotheses.

2.2.5 Presentation and Packaging

This step includes providing thorough documentation explaining the goals and the results of the experiments completely. Moreover, a replication package must be prepared to further assist the replication.

CHAPTER 3 MAPPING STUDY ON THE USE OF EYE-TRACKING TECHNIQUES IN SOFTWARE ENGINEERING

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. On the one hand, 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. On the other hand, mapping studies are also useful in the context of EBSE by bringing together all existing evidence of research and results about a particular topic.

This mapping study 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.

We perform a mapping study 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.

The software engineering community started using eye-trackers in the 1990's 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.

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.

3.1 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" (Sackett et al., 2000).

Kitchenham et al. (2004) 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 a systematic literature review (SLR) is recognized as an approach to realize EBSE with the goal of bringing together all existing evidence on a topic and providing evidence-based guidelines to push forward that topic (Kitchenham et al., 2009). Mapping studies are a form of SLR that strives for identifying and categorising the available research on a specific topic (Kitchenham et al., 2010). The main difference between an SLR and a mapping study is that conducting an SLR is steered by one specific research question, followed by an empirical study to answer this question. However, a mapping study targets a list of high-level questions and mainly investigates the sub-topics that have been addressed and the methods that have been used. A mapping study also identifies sub-topics that are supported by sufficient empirical studies for performing a more precise SLR (Kitchenham et al., 2009).

In cognitive psychology, researchers have traditionally used eye-trackers to study information processing tasks (Rayner, 1998). 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) (Duchowski, 2007). 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 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) (Blackwell et al., 2000) and its modified, enhanced version to study debugging strategies (Romero et al., 2002a,b, 2003). RFV tracks the movements of a computer mouse over a stimulus and their coordinates and durations (Blackwell et al., 2000; Jansen et al., 2003). Yet, since 2006, software engineering researchers started to leverage the availability of unobtrusive eye-trackers to perform different studies.

This paper presents a mapping study 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 mapping study 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 mapping study 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 mapping study 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 summarise all the metrics and tools available for the analysis of eye-tracking data to study developers' cognitive processes.
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 mapping study also concludes with the needs for new metrics and tools to analyse eye-tracking data as well as advices to overcome or limit threats to the validity of eye-tracking studies.

The remainder of this chapter is organized as follows: Section 3.2 contains an overview of related work. In Section 3.3, we discuss the methodology that we follow in conducting the mapping study. We present the summary of evidence along with the complete answers to our research questions in Section 3.4. Threats to validity are discussed in Section 3.5 and we discuss and conclude this chapter in Section 3.6.

3.2 Related Work

There is neither a mapping study nor an SLR with regards to the usage of eye-tracking technique in software engineering. We can identify only two relevant articles. In the first article, Busjahn et al. (2011) provided useful comparisons between several eye-tracking studies that are focused on source code reading and comprehension. This comparison has been done based on seven criteria including:

- Number of participants.
- Experience levels (expert, intermediate, and novice).
- Methods (eye-tracking and eye-tracking with Retrospective Think-aloud (RTA)).
- Stimuli programs (algorithms, number of programs, and number of bugs, LOC).
- Programming language.
- Comparison with natural text.

They concluded that despite some methodical shortcomings, the usage of eye-tracking technique is useful for studying program comprehension.

The second article analysed various ways in which eye movements can be systematically measured (Poole and Ball, 2005). Poole and Ball (2005) provided a list of eye-tracking metrics for usability evaluation based on the number and the duration of fixations, saccades, and scan-paths. It also discusses technical issues related to eye-tracking technique and some propositions for improvement.

3.3 Study Design

We carry out this mapping study following Kitchenham's guidelines (Kitchenham, 2004).

3.3.1 Research Questions

This mapping study answers the following general, related to research trend research questions:

- RQ3.1: How many articles have reported on the use of eye-tracking techniques in software engineering research since 1990?
- RQ3.2: Which research topics have been explored and examined in eye-tracking studies?
- RQ3.3: How much have eye-tracking studies contributed to software engineering?
- RQ3.4: How have researchers used eye-trackers to collect quantitative measurement?
- RQ3.5: What are the limitations of current research?
- RQ3.6: Which eye-trackers are most frequently used in software engineering research?

To address RQ3.1, 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 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 RQ3.2, 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 RQ3.3, we summarise the outcomes of the selected studies. With respect to RQ3.4, we provide a list of all metrics based on eye-tracking data as well as their definitions and applicability. To answer RQ3.5, 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 RQ3.6.

3.3.2 Search Process

We use Engineering Village[1] 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[2] and IEEE Xplore[3] 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:

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 "representation*". We define this set of keywords to find papers focusing on the different types of stimuli used in software engineering studies.

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

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 preliminary 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[4] and Compendex[5] databases. We sorted the results of the preliminary query by relevance. Second, we first 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 obtained the final search query shown in Figure 3.1.

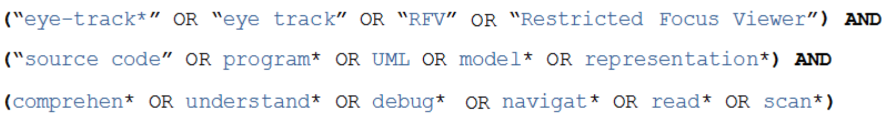
3.3.3 Selection Process

The process that we adopted to select the relevant papers is presented in Figure 3.2. 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 executed our query in Engineering Village. The search engine searched 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 used 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 performed this step to check whether the resulting papers are relevant or not to our goal. We presented and explained these criteria in details in the next section.
4. Analyse selected papers for duplicate removal: for each selected paper, we performed its full analysis. Using title, keywords, abstract, and body, we checked whether the paper is

[4] http://www.theiet.org/resources/inspec/

[5] http://adat.crl.edu/databases/about/compendex



```
("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*)
```

Figure 3.1 Search query.

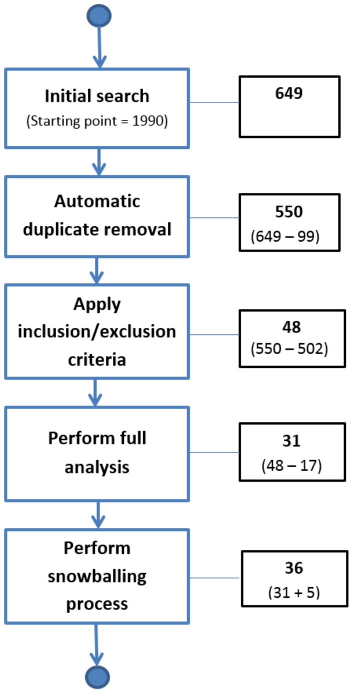


Figure 3.2 Selection process adapted to choose appropriate related papers. The numbers on the left are the numbers of papers remaining after each step has been performed.

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, we went through the list of all its references, check each reference starting from its title, the conference proceeding and journal names, and then check the full paper if necessary to decide whether to include the paper in the study or not.

3.3.4 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.
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 Budgen et al. (2011). (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 mapping study 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.

3.3.5 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.

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 3.1.

3.4 Study Results

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

3.4.1 RQ3.1: How Many Articles Have Reported on the Use of Eye-tracking Techniques in Software Engineering Research Since 1990?

Overall, we find 36 relevant papers, summarised in Table 3.4. All of these papers report at least one study with some eye-tracker. Figure 3.3 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.

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 Figure 3.2:

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 analyse 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 (Romero et al., 2002b) because (Romero et al., 2002a) is its extended journal paper version.

   We also remove (Sharif and Kagdi, 2011; Walters et al., 2013) because they do not contain any eye-tracking study. We also remove six duplicated papers. Out of these

Table 3.1 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

six papers, one removed paper is a replication of another eye-tracking study (Yusuf et al., 2007) using a new questionnaire. Another (Guéhéneuc et al., 2009) is not a study and is only used in a working session of a conference. Another (Sharif, 2011) 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, 2010a). Regarding the second eye-tracking study, we remove the original study (Sharif and Maletic, 2010b) and use a more complete version published in 2013 (Binkley et al., 2013). We replace (Bednarik and Tukiainen, 2005, 2004) 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., 2014a).

   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

Table 3.2 Journals, conference, and workshop proceedings of the selected papers.

Source (Acronym)	Total
ACM Computer Supported Cooperative Work (CSCW)	1
Conference of the Center for Advanced Studies on Collaborative Research (CASCON)	1
Computer Supported Collaborative Learning Conference (CSCL)	1
Conference on Advanced Information Systems Engineering (CAiSE)	1
Conference on Computing Education Research (Koli Calling)	1
Conference on Software Maintenance (ICSM)	2
Conference on Program Comprehension (ICPC)	5
Empirical Software Engineering Journal (EMSE)	2
Eye-tracking Research and Application (ETRA)	7
Hawaii International Conference on System Sciences (HICSS)	1
IEEE Computer Journal (IEEE Computer)	1
International Conference on Software Engineering (ICSE)	2
Journal of Behavior research methods (BRM)	1
Science of Computer Programming Journal (SCP)	1
Journal of Human Computer Studies (HUMCOMPUT)	1
International Journal of Human Computer Interaction (IJHCI)	1
International Conference on Multimodel Interaction (ICMI)	1
Journal of Systems and Software (JSS)	1
Lecture Notes in Computer Science (LNCS)	1
Symposium on Software Engineering and Measurement (ESEM)	1
Working Conference on Software Visualization (VISSOFT)	1
Workshop of Psychology of Programming Interest (PPIG)	2

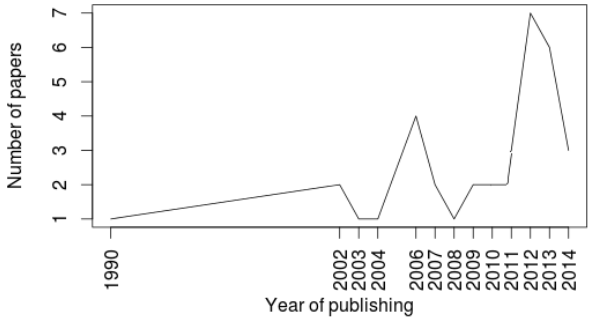


Figure 3.3 Number of published papers per year.

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 to 2014. Table 3.2 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.

3.4.2 RQ3.2: Which Research Topics Have Been Explored and Examined in Eye-tracking Studies?

We categorise the selected papers in five groups, shown in Table 3.7. 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, we finalise 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.

Table 3.3 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. (2002a)	To track attention switching of developers while performing debugging tasks with multiple representations
Romero et al. (2003)	To characterise 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. (2006a)	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' debugging behaviour
Jeanmart et al. (2009)	To analyze the impact of visitor pattern on program comprehension and maintenance
Cepeda Porras and Guéhéneuc (2010a)	To analyze the impact of representation on design pattern comprehension
Sharif and Maletic (2010a)	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

Table 3.4 List of selected papers - continued.

Papers	Goals
De Smet et al. (2012)	To study the impact of Composite, Observer, and MVC pattern on maintenance tasks
Hejmady and Narayanan (2012)	To study 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. (2012a)	To study the impact of scan Time in detecting source code defects
Soh et al. (2012)	To analyze the impact of expertise on developers' efficiency
Binkley et al. (2013)	To analyze the impact of identifier style on 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 (2013)	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
Walters et al. (2014)	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. (2014a)	To study the code summarization task using eye-tracking technique
Turner et al. (2014)	To study differences between individuals' gaze behaviour and reading patterns

Table 3.5 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. (2002a)	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. (2006a)	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 Porras and Guéhéneuc (2010a)	Model comprehension	UML	24 S	EMSE	EyeLink II
Sharif and Maletic (2010a)	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	HUM-COMPUT	Tobii 1750

Table 3.6 Topic domains, artifacts, participants (S = Students, FM = Faculty members, P = Professionals, and NM = Not mentioned), sources, and the eye-tracker for the selected papers - continued.

Papers	Topic domains	Artifacts	Participants	Sources	Eye-trackers
De Smet et al. (2012)	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. (2012a)	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 (2013)	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
Walters et al. (2014)	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. (2014a)	Code comprehension	Java	10 P	ICSE	Tobii TX300
Turner et al. (2014)	Code comprehension	C++, Python	38 S	ETRA	NM

Table 3.7 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 Porras and Guéhéneuc (2010a), Sharif and Maletic (2010a), De Smet et al. (2012) Soh et al. (2012), Cagiltay et al. (2013) Petrusel and Mendling (2013), 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. (2014a)
Debugging	9	Romero et al. (2002a), Romero et al. (2003), Uwano et al. (2006a) Bednarik and Tukiainen (2007), Bednarik and Tukiainen (2008), Bednarik (2012), Hejmady and Narayanan (2012) Sharif et al. (2012a), 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)

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 3.4.4. 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 behaviours but rather to understand their *uses* of some artifacts and tools when performing of tasks to help them in their work.

Model Comprehension

We find ten papers pertaining to model comprehension. All of the participants participating in the studies mentioned in selected papers for this category perform comprehension tasks.

Guéhéneuc (2006) uses eye-trackers to study the comprehension of UML class diagrams. It introduces a visualisation technique to aggregate and display eye-tracking data (fixations and saccades). This visualisation 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[6] as artifacts.

[6] www.slac.stanford.edu/grp/ek/hippodraw

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[7], JRefactory[8], and PADL[9] 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 Porras and Guéhéneuc (2010a) 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 (Schauer and Keller, 1998), Gamma (Gamma, 1996), and Dong (Jing et al., 2007)) and tasks (participation, composition, and role).
	MV	JHotdraw and design pattern knowledge

Sharif and Maletic (2010a) analyses the impacts of orthogonal and multi-clustered layouts on the comprehension of design patterns. It uses UML class diagrams of JHotDraw, JUnit[10], and Qt[11] as artifacts.

[7] http://www.jhotdraw.org/

[8] http://jrefactory.sourceforge.net/

[9] http://wiki.ptidej.net/doku.php?id=padl

[10] http://junit.org/

[11] http://qt-project.org/

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 behaviour
	MV	

De Smet et al. (2012) 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[12], and ArgoUML[13] 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.

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 (2013) focuses on the understanding of business-process models (BPMN[14] diagrams) and investigates the factors that influence their comprehension.

[12] http://sourceforge.net/projects/quj/

[13] http://argouml.tigris.org/

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

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 (Bresciani et al., 2004) notation in modelling 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

Code Comprehension

We identify 12 papers pertaining to code comprehension. All of the participants participating in the studies mentioned in selected 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 visualisation technique of Java source code and studies how developers use source code and the visualisation 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

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 visualisation 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 visualisation 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 (Hart and Staveland, 1988) 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. (2014a) conducts an eye-tracking study and use its findings to build a code summarisation 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

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. (2002a) 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) characterises 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. (2006a) focuses on source code reviews by developers to find defects.

Artifacts		Five C source codes (12 to 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. (2012a) partially replicates (Uwano et al., 2006a) 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) and 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 visualisation tool, called SeeIT, for performing code overview and bug-fixing tasks.

Artifacts	Java source code of an open source system: Gantt Project [15]	
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.

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 Nüssli (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.

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

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

Discussions

Table 3.8 summarises the types of artifacts that have been studied for performing the eye-tracking studies in the selected papers: 16 studies out of 22, which used pieces of source code as stimuli, used 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 used Pascal because it was published in 1990. Two other studies used C. One study compares C++ with Python regarding code comprehension.

UML diagrams were 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, 1,022 participants took part in the studies reported in the selected papers, as summarised in Figure 3.4. 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).

3.4.3 RQ3.3: 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 3.9 and 3.10 summarise the results for model comprehension while Tables 3.11 and 3.12 summarise those for code comprehension and debugging, respectively. Finally, we present the summary of the results for collaborative interactions and traceability in Tables 3.13 and 3.14, respectively.

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.

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

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

Table 3.8 Types of artifacts used in eye-tracking studies.

Table 3.9 Summary of the results for model comprehension.

Model Comprehension
Experts vs. novices: 1. Experts use extra information, *e.g.,* color, layout, and stereotypes, more efficiently to browse UML diagrams (Yusuf et al., 2007). 2. Expertise impacts the speed and accuracy more than experience. Also, practitioners are more accurate than students (Soh et al., 2012).
Design patterns: 1. The canonical representation of the Visitor design pattern has a negative impact (Jeanmart et al., 2009). 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 Porras and Guéhéneuc, 2010a). 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., 2012).
UML representations: 1. Using similar visual notations enhances diagram comprehension and reduces effort (Yusuf et al., 2007). 2. Using multi-cluster layout leads to a significant improvement in the accuracy, time, and effort compared to an orthogonal layout (Sharif and Maletic, 2010a).

Table 3.10 Summary of the results for model comprehension (continued).

Model Comprehension
Navigation strategies: 1. Vertical scanning is more efficient than horizontal scanning for ERD comprehension tasks (Cagiltay et al., 2013). 2. Scan-paths and relevant areas are connected with the participants' strategies and correctness when understanding BPMN models (Petrusel and Mendling, 2013). 3. The structure of representations leads participants to use different navigation strategies (Sharafi et al., 2013).
Others: 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). 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). 3. We expect practitioners to be more efficient and accurate than students, yet students are faster for comprehension tasks (Yusuf et al., 2007). 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). 5. Participants spend more time and effort working with graphical representations compared to textual ones (Sharafi et al., 2013).

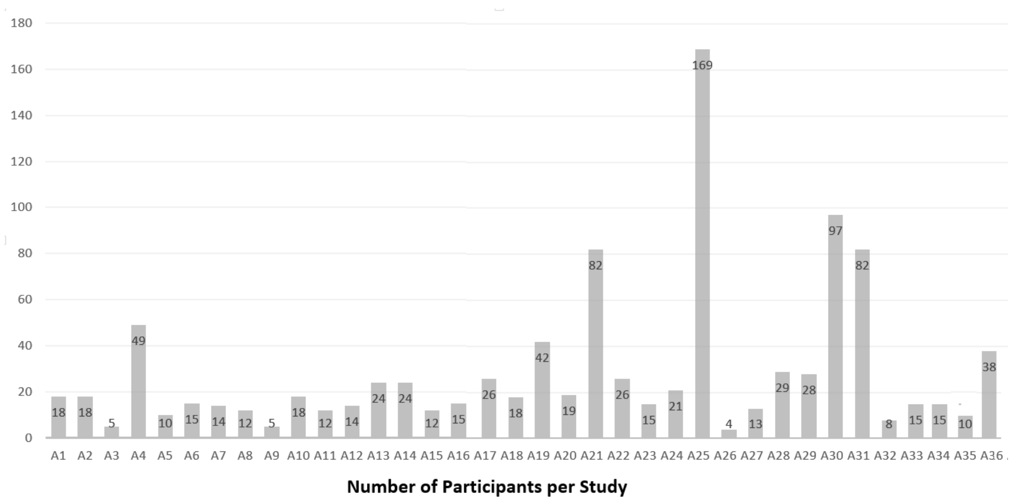


Figure 3.4 Total number of participants for the selected papers.

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. (2012) could not find any impact for the Observer and Composite design patterns although, by analysing 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* (Erich Gamma and Vlissides, 1995), 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, 2010a) reports a significant improvement in the accuracy, time, and visual effort for the multi-cluster layout.

Cepeda Porras and Guéhéneuc (2010a) compares and reports that Dong's representation (Jing et al., 2007) is more efficient for Composition and Role Tasks while Gamma's (Gamma, 1996) and Schauer's (Schauer and Keller, 1998) are more efficient for the Participation Task.

Petrusel and Mendling (2013) 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.

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.

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' behaviour during code reading. It suggests that any area of code that exhibits very long fixations, above 1,000 milliseconds, 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

Table 3.11 Summary of the results for code comprehension.

Code comprehension

Experts vs. novices:

1. Novices spend more visual attention on comments than experts (Crosby and Stelovsky, 1990).
2. Novices do not start by reading source code and look at the graphical representation first (Bednarik and Tukiainen, 2006).
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).
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:

1. Participants have higher fixation times and regression rates when reading source code than when reading natural text (Busjahn et al., 2011).
2. Source code reading and comprehension are fundamentally different from natural text reading and comprehension (Binkley et al., 2013).

Identifier styles:

1. Expertise affects the impact of identifier styles on reading and comprehension (Binkley et al., 2013).
2. No difference exists between camel-case and underscore identifier styles regarding accuracy, time, and effort for comprehension tasks (Sharafi et al., 2012).
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:

1. Source code visualisation techniques provide more important information for participants at early stages of comprehension and less at later ones (Bednarik and Tukiainen, 2006).
2. Visualisation techniques improve participants' performance and help them to follow more systematics strategies during program comprehension (Duru et al., 2013).
3. Male and female participants follow different strategies when answering identifier recall questions (Sharafi et al., 2012).

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 analysing 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 visualisation 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.

Rodeghero et al. (2014a) shows that participants spend more visual attention on method signatures than method invocations and more visual attention on and invocations than control flow.

Debugging

Uwano et al. (2006a); Sharif et al. (2012a) report that participants who spent more time scanning source code (compared to others) tend to find defects more efficiently. Longer scanning means that participants carefully read the code and find suspicious candidate code lines. Sharif et al. (2012a) 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., 2006a) 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. (2006a) finds some patterns of retracing (looking back at the declarations of the variables). However, (Sharif et al., 2012a) 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. (2002a, 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.

Table 3.12 Summary of the results for debugging.

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

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.

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 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.

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.

Table 3.13 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). 2. Participants find bugs faster after watching the eye-movements of the developers in another group (Stein, 2004). 3. Pairs of participants who spend more focused time together are more accurate for comprehension tasks (Jermann and Nüssli, 2012).

Table 3.14 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). 2. The SimpleGraph Gaze-Link algorithm provides traceability links with high recall, automatically (Walters et al., 2014).

3.4.4 RQ3.4: How Have Researchers Used Eye-trackers to Collect Quantitative Measurement?

Different studies propose several metrics based on eye-movement data provided by eye-trackers. The formulas and explanations and the goal of these metrics are presented in Chapter 2, Section 2.1.4. In this section, we explain how these metrics have been used in selected studies.

Visual Effort and Efficiency Metrics

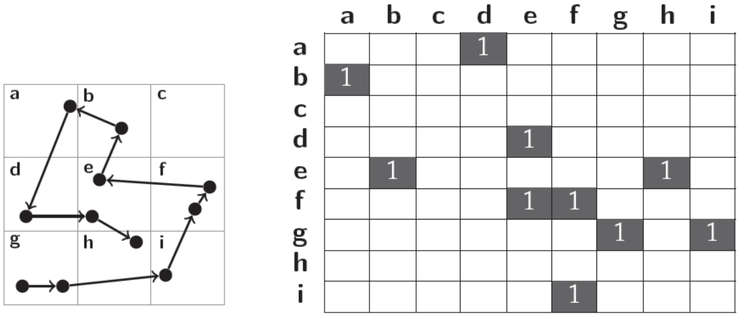
Tables 3.15, 3.16, 3.17, and 3.18 summarise all metrics used in the selected papers to measure the amount of visual effort and efficiency along with related papers.

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

Scan-path metrics:

these metrics are used to measure and display participants' scan-paths.

– Edit distance:De Smet et al. (2012) uses the Levenshtein distance and reports that the average distance among novices' scan-paths is lower than that of experts. However, the



	a	b	c	d	e	f	g	h	i
a				1					
b	1								
c									
d					1				
e		1						1	
f					1	1			
g							1		1
h									
i						1			

Figure 3.5 Example of scan-path and corresponding transition matrix (De Smet et al., 2012).

Table 3.15 List of metrics for visual effort calculation based on fixation number along with selected studies - Part 1.

		Names	Papers
Visual Effort and Efficiency Metrics	Number of Fixation	Fixation Count (FC)	(Crosby and Stelovsky, 1990), (Crosby et al., 2002), (Uwano et al., 2006a), (Yusuf et al., 2007), (Sharif and Maletic, 2010a), (Sharafi et al., 2012), (Sharif et al., 2012a), (Sharif et al., 2013), (Turner et al., 2014)
		Fixation Rate (FR)	(Cepeda Porras and Guéhéneuc, 2010a), (Sharif and Maletic, 2010a), (De Smet et al., 2012), (Sharafi et al., 2012), (Sharif et al., 2012a), (Binkley et al., 2013), (Turner et al., 2014)
		Spatial Density	(De Smet et al., 2012), (Soh et al., 2012)
		Convex hull Area	(Sharafi et al., 2012), (Soh et al., 2012), (Sharafi et al., 2013)

Table 3.16 List of metrics for visual effort calculation based on fixation duration along with selected studies - Part 2.

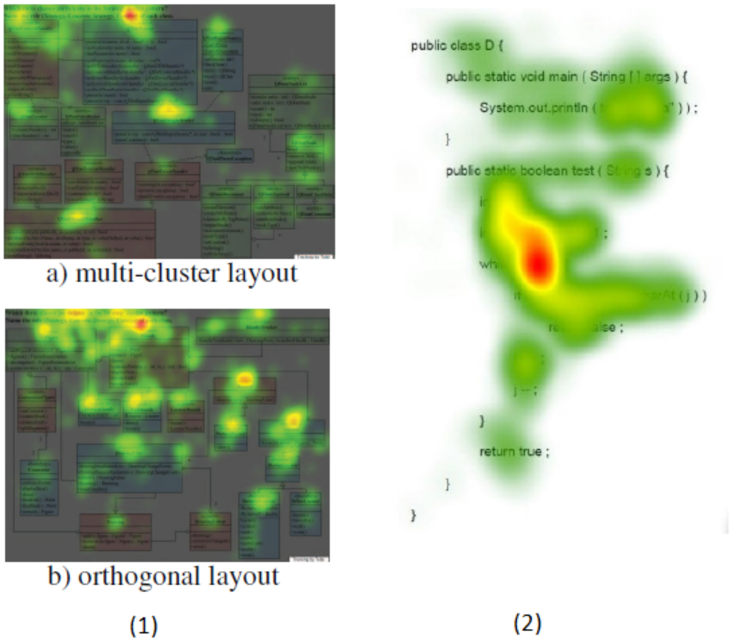
		Names	Papers
Visual Effort and Efficiency Metrics	Duration of Fixations	Average Fixation Duration (AFD)	(Crosby and Stelovsky, 1990), (Cepeda Porras and Guéhéneuc, 2010a), (Sharif and Maletic, 2010a), (Sharif and Kagdi, 2011), (Soh et al., 2012), (Binkley et al., 2013), (Sharafi et al., 2013)
		Ratio of ON_target All_target Fixation Time (ROAFT)	(Bednarik and Tukiainen, 2006), (Cepeda Porras and Guéhéneuc, 2010a), (Bednarik, 2012), (Sharif et al., 2012a), (Binkley et al., 2013)
		Fixation Time (FT)	(Crosby and Stelovsky, 1990), (Crosby et al., 2002), (Uwano et al., 2006a), (Bednarik, 2012), (Ali et al., 2012), (Petrusel and Mendling, 2013), (Busjahn et al., 2014), (Rodeghero et al., 2014a)
		Average Duration of Relevant Fixations (ADRF)	(Jeanmart et al., 2009; De Smet et al., 2012)
		Normalised Rate of Relevant Fixations (NRRF)	(Jeanmart et al., 2009), (De Smet et al., 2012), (Soh et al., 2012)

Table 3.17 Metrics for visual effort calculation based on saccades.

		Names	Papers
Visual Effort and Efficiency Metrics	Saccades	Number of saccades	Fritz et al. (2014)
		Saccade duration	Fritz et al. (2014)

Table 3.18 Metrics for visual effort calculation based on scan-paths along with selected studies.

		Names	Papers
Visual Effort and Efficiency Metrics	Scan-paths	Transitional Matrix	(De Smet et al., 2012)
		Attention Switching Frequency	(Bednarik and Tukiainen, 2006), (Bednarik and Tukiainen, 2008)
		Scan-path Precision	(Petrusel and Mendling, 2013)
		Scan-path Recall	(Petrusel and Mendling, 2013)
		Scan-path F-measure	(Petrusel and Mendling, 2013)



a) multi-cluster layout

b) orthogonal layout

```
public class D {
    public static void main ( String [ ] args ) {
        System.out.println ( [illegible]" ) ) ;
    }
    public static boolean test ( String s ) {
        i[illegible]
        i[illegible] ;
        whi[illegible]
            i[illegible]arAt ( j ) )
                re[illegible]alse ;
            [illegible] ;
            j -- ;
        }
        return true ;
    }
}
```

Figure 3.6 (1) shows the heat-map of a participant working with (a) multi-cluster and (b) orthogonal layouts (Sharif and Maletic, 2010a). (2) presents areas of source code that attracts higher interests (Busjahn et al., 2011).

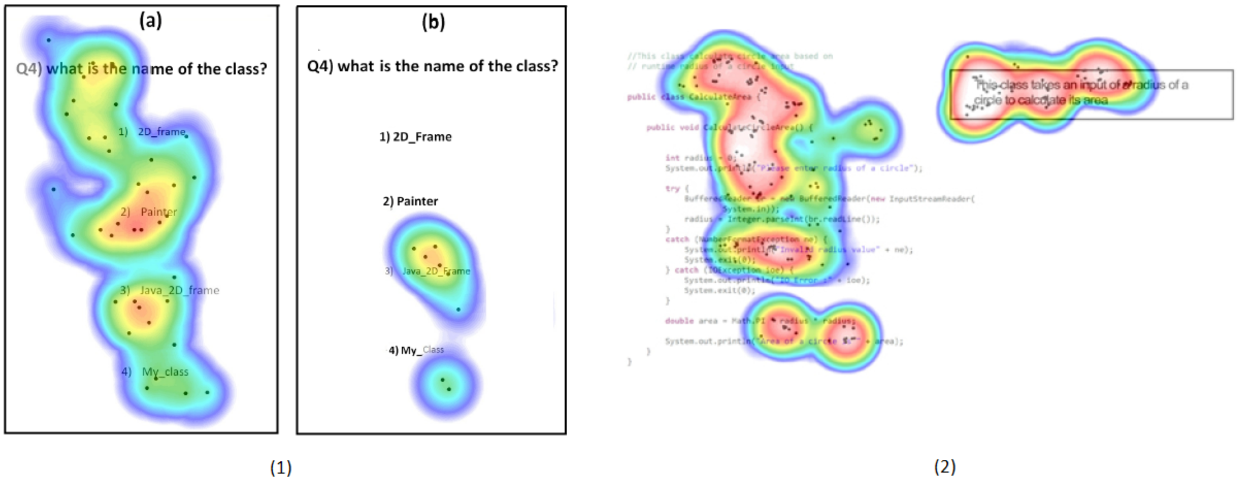


Figure 3.7 (1) shows a heat-map of (a) a female participant and (b) a male participant (Sharafi et al., 2012). (2) shows a heat-map that presents the cumulative fixations of a participant on different source code entities, including class name, method name, variables, and comment (Ali et al., 2012).

edit distance does not take into account the duration of different fixations and treats all fixations equally. Fixation duration plays an important role in analysing eye-tracking data (Henderson and Pierce, 2008). Consequently, new techniques have been proposed to consider also the duration of fixations.

- Sequential Pattern Mining (SPAM): Hejmady and Narayanan (2012) uses this technique and takes into account fixation durations by categorising fixations as short (less than 500 milliseconds) and long (higher than 500 milliseconds). The threshold (500 milliseconds) 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 visualisation more often than those who were less familiar with the IDE.
- ScanMatch: Sharafi et al. (2013) uses ScanMatch and calculates the similarity of participants' scan-paths while working with graphical representations.

Visual gaze behaviour:

these metrics provide visualisations to display eye-movements; they are classified as follows:

- Heatmap: Sharif and Maletic (2010a) investigates the impact of different layouts on the comprehension of design patterns. It uses two heat-maps to visualise 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 visualise 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: Sharif and Maletic (2010a) uses a gaze plot to compare experts and novices performing design pattern comprehension tasks. Experts' gaze plots (as presented in Figure 3.8) 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 Figure 3.8) 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.
– Color coded attention allocation map: Busjahn et al. (2011) uses such a map based on the number of fixations per word as depicted in Figure 3.9 to show different parts of source code that attracts different levels of attention and consequently time.

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 summarise 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 (Duchowski et al., 2010; Dewhurst et al., 2012), 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

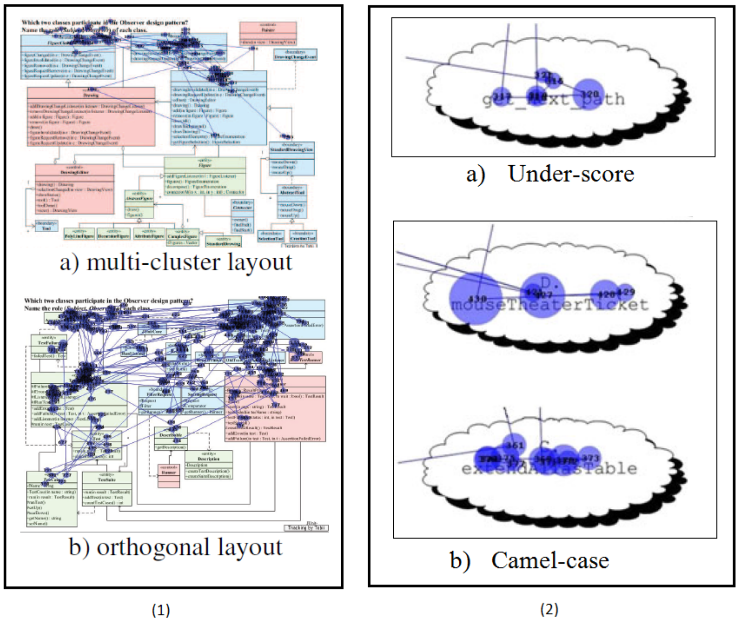
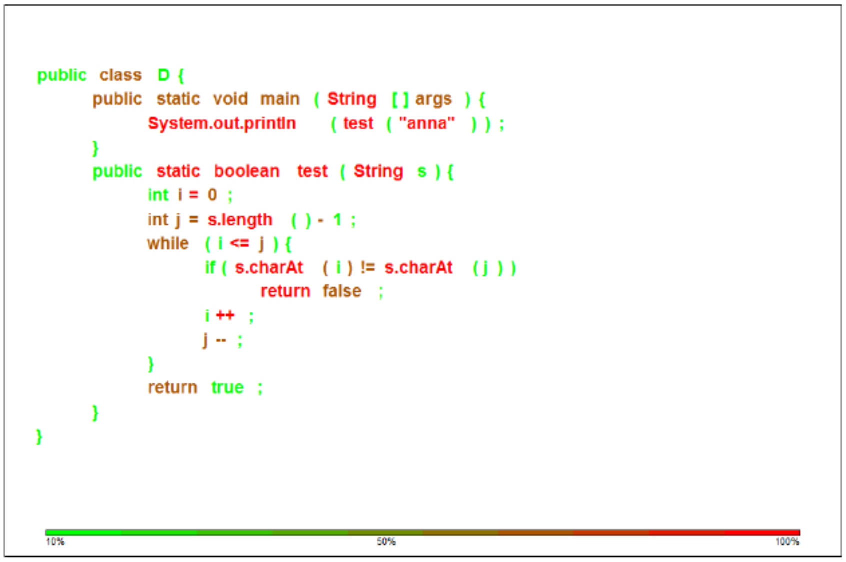


Figure 3.8 (1) shows the gaze plots of an expert for the Observer pattern for orthogonal and multi-cluster layouts (Sharif and Maletic, 2010a). (2) shows gaze plots for underscore and two camel case 3-word code identifiers (Binkley et al., 2013).



```
public class D {
        public static void main ( String [ ] args ) {
                System.out.println      ( test ( "anna" ) ) ;
        }
        public static boolean test ( String s ) {
                int i = 0 ;
                int j = s.length   ( ) - 1 ;
                while  ( i <= j ) {
                        if ( s.charAt  ( i ) != s.charAt  ( j ) )
                                return false  ;
                        i ++  ;
                        j --  ;
                }
                return true  ;
        }
}
```

Figure 3.9 Color-coded attention allocation map based on the number of fixations per word (Busjahn et al., 2011).

comparison (Dewhurst et al., 2012).

However, the results of applying available scan-path comparison techniques as shown in few selected studies (De Smet et al., 2012; 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 visualisations of source code together. After comparing scan-paths using SPAM (Ayres et al., 2002) quantitatively, (Hejmady and Narayanan, 2012) reports a prominent attention switching pattern (short attention switching between code and dynamic visualisation) 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 (Cristino et al., 2010), it categorises scan-paths into two different groups. By visualising these groups, it shows that they represent two different viewing strategies: top-down or bottom-up.

3.4.5 RQ3.5: What Are the Limitations of Current Research?

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.

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 Poole and Ball (2005). 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 Goldberg and Helfman (2010).

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 Sajaniemi (2004). 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 Sajaniemi (2004). 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' behaviour 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.

Data Analysis

Eye-trackers produce huge amount of data, whose analyses are complex. Using a set of tools to automatically filter and analyse the data is necessary to save time and prevent human errors caused by manual analyses. Several tool have been used by researchers to analyse high volumes of data generated by eye-trackers. For example, Taupe De Smet et al. (2012) and

OGAMA[17] are open-source software systems designed for analysing 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 visualiser Fehringer (2014), 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 analysing eye-tracking data. Yet, currently, there are no detailed best-practices for defining AOIs. Goldberg *et al.* Goldberg and Helfman (2010) 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.

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 Just and Carpenter (1976); Poole and Ball (2005). Task definition also identifies the type of artifacts of interest in the the study. Researchers must eliminate any visual distractions (*e.g.,* colourful or moving objects) to avoid contaminating eye-movement data Goldberg and Wichansky (2003); Poole and Ball (2005).

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 Lankford (2000) 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 Walters et al. (2013, 2014) also supports horizontal and vertical scrolling while recording the correct locations of eye-movements. It also is integrated in Eclipse[18] IDE.

[17] http://www.ogama.net/

[18] https://eclipse.org/

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.

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.

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, 2013), researchers mention the choices of the environment and participants as a limitation. However, according to Kitchenham *et al.* Kitchenham et al. (2002), “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.* Goldberg and Wichansky (2003), it is prudent to use a within-participants design for eye-tracking studies, to reduce the impact of participants’ individual characteristics on the results.

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.

3.4.6 RQ3.6: Which Eye-trackers Are Most Frequently Used in Software Engineering Research?

Table 3.19 provides a list of the eye-trackers used in the selected papers. Two papers used the RFV approach to perform studies Romero et al. (2002a, 2003). Four studies did not specify the used eye-trackers Crosby and Stelovsky (1990); Stein (2004); Petrusel and Mendling (2013); 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.

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 degree. If a participant is seated 50cm away from a stimulus and the eye-tracker has 1 degree of accuracy, the eye-movement positions could be measured anywhere within a radius of 1 cm of the actual positions Sajaniemi (2004). The reported accuracy values for the eye-trackers used in the selected paper are 0.5 degree 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

Table 3.19 List of eye-trackers used in the selected papers with their accuracy values and sampling rates.

Eye-tracker	Manufacturer	Accuracy (degree)	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, 2010a), (Bednarik, 2012), (Jermann and Nüssli, 2012), (Sharif et al., 2012a), (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),(Walters et al., 2013)
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., 2014a)
FaceLab	Seeing machine	0.5	60	3	(Ali et al., 2012),(Sharafi et al., 2012),(Sharafi et al., 2013)
Eye-Link II	SR Research	0.25-0.5	500 pupil only	5	(Guéhéneuc, 2006; Jeanmart et al., 2009),(Cepeda Porras and Guéhéneuc, 2010a),(De Smet et al., 2012),(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	0.3	30	1	(Uwano et al., 2006a)

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 10Hz to 2,000Hz. As shown in Table 3.19, the lowest sampling rate for the eye-trackers used in the selected papers is observed for EMR-NC (30Hz) while the highest is reported for Eye-Link II (500Hz). Poole *et al.* Poole and Ball (2005) reported that a sampling rate of 60Hz is adequate for usability studies but that it is not good enough for reading studies, which require sampling rates of 500Hz or higher. The majority of previous studies use eye-trackers with sampling rates of 60Hz 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 re-calibrations.
- 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 visualisations 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.

3.5 Threats to Validity

There is an evident threat to validity of a mapping study 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, 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 mapping study 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" (Zhang et al., 2011) string evaluation approach (Lavallée et al., 2014). 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 (Brereton et al., 2007) for data extraction in which one researcher extracted the data and another researcher checked the data extraction. (Turner et al., 2008) 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.

3.6 Discussions and Conclusion

We performed a mapping study 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 mapping study 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 analysed in different eye-tracking studies and we divided them in five categories: *model comprehension*, *debugging*, *code comprehension*, *collaborative interactions*, and *traceability*. This categorisation 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 needs standard guidelines to design experiments for software engineering tasks and to use eye-trackers while reducing the risks of failure and mitigating threats to validity. As emphasised and explained by Sjøberg *et al.* Sjøberg et al. (2005), using a uniform way of reporting an experiment benefits the software engineering community, because it provides valuable information on how to review, replicate, and analyse the data. There are guidelines for performing controlled experiments in software engineering Kitchenham (2007); Jedlitschka and Pfahl (2005). 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) Do et al. (2005) [19]). Sharing data online in an organised repository would allow researchers to receive feedback and improve their artifacts and to aggregate their findings and single out missing artifacts Do et al. (2005).

We recommend newcomers who start in the field (1) to familiarise 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 Just and Carpenter (1976); Duchowski (2007); Rayner (1998); Goldberg and Kotval (1999); Poole and Ball (2005). (2) To consider the limitations associated with

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

this technology, as summarised in Section 3.4.6, 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 3.4.2 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 study in 2020 to keep track of the progress and results of new eye-tracking studies in software engineering.

CHAPTER 4 IMPACT OF REPRESENTATION TYPE ON PROGRAM COMPREHENSION

In this chapter, we present the results of a study investigating and quantifying the effect of graphical vs. structured texts in program comprehension. First, the effectiveness of structured texts vs. graphical representations is explored by means of an eye-tracking experiment. Second, we investigate the developer's characteristics that impact representation understanding. Third, we assess the impacts of representations on comprehension strategies for obtaining information from textual or graphical representations.

4.1 Introduction

Graphical representations (*e.g.,* UML diagrams) play a vital role in presenting software artifacts and they are effective tools to (Moody, 2009): 1) provide a quick understanding of data, 2) enhance data comparison and processing, and 3) improve and facilitate the communication process between end-users and developers.

Conventional wisdom assumes that graphical representations carry information more effectively to non-technical people than textual one (Moody, 2009). A huge amount of work has been dedicated to propose different graphical representations under the assumption that developers prefer to work with graphical representations rather than text to understand the system. However, a purely graphical representation is not as expressive as the textual representation and there are some aspects of system properties that can not be specified completely using diagrams (Petre, 1995; Ottensooser et al., 2012; Snook and Harrison, 2007). Therefore, graphical representations usually accompanied by textual representation to improve visualization while achieving the full expressiveness and precision (Snook and Harrison, 2007; Ottensooser et al., 2012).

Yet, to the best of our knowledge, only a handful of studies investigated the effectiveness of graphical vs. textual representations or the developers' preferences (*i.e.,* textual vs. graphical representations) in program understanding tasks (Yusuf et al., 2007; Heijstek et al., 2011; Ottensooser et al., 2012). Some works have been arguing for the superiority of graphical representations, and some favour textual ones for some aspects (Ottensooser et al., 2012) or even some report no differences (Somervell et al., 2002; Snook and Harrison, 2007; Ottensooser et al., 2012).

In this dissertation, we do not intend to examine the quality or the accuracy of a software artifact (presented by either graphical or textual representation) itself but rather to quantify the effect of the representation type (graphical vs. structured textual) on program comprehension. We report the results of an eye-tracking experiment conducted to compare graphical vs. textual representations regarding developers' accuracy, task time, and effort.

Section 4.2 contains an overview of related work. In section 4.3, the experimental design is presented and section 4.4 outlines an overview of the results. The threats to validity are addressed in Section 4.5. We provide the discussion and conclusion in Section 4.6.

4.2 Related Work

In recent years, graphical representations have received increasing attention. However, only a few works addressed the comparison of textual vs. graphical representations.

Ottensooser et al. (2012) reported significant improvement in understanding of business processes when participants work with textual representations. They also underlined that participants must learn how to understand and use the specific graphical notations before starting to use them.

Somervell et al. (2002) were interested to investigate which combination of graphical and textual representations is more efficient when participants working on dual-tasks by sharing their attention between different tasks. They considered three different criteria including: 1) facilitation of information monitoring, 2) awareness of information, and 3) introduction of distraction. As a result, they provided a list of guidelines on the use of combinations of textual and graphical representations to improve participants' efficiency.

Snook and Harrison (2007) compared UML-based graphical formal specification vs. a purely textual, formal specification in understanding a software specification. They reported that a combination of semi-formal and formal notations improves the participants' accuracy in comprehension tasks.

Heijstek et al. (2011) reported that neither textual nor graphical representations were significantly effective for understanding a software architecture. They also reported that the more experienced participants, mostly, preferred textual ones.

This chapter stems from the belief that there is a need to investigate the difference (if any) between graphical and textual requirement representations. The work presented in this chapter is complementary to previous work, because we investigate the impact of textual and graphical representations not only on participants' effectiveness but also on the strategies that they use to read and understand the requirements' representations. Moreover, in contrast with some of the previous works, we use a textual representation which is text without any graphical notation but it has a meaningful structure thus easing information access. This structure makes our textual representation different from a purely textual, formal specification.

4.3 Study Design

The goal of our study is to investigate the relations between the type of requirement representations (graphical vs. textual) and participants' visual effort, task time, as well as accuracy in understanding software artifacts. The *quality focus* is model comprehension effort, which may depend on representation type. The *perspective* is that of developers, who perform development or maintenance activities and need to understand a system. The researchers could also use our findings to design methods, techniques, and tools to better benefits from different representation types. The *context* of this study consists of three model comprehension tasks involving 28 participants (twelve females) and two variants of models. The experiment is conducted as between participants design. An overview of our experiments is outlined in Table 4.1.

4.3.1 Research Questions

In this chapter, we are interested in the following research questions:

RQ4.1: Does the type of representations (graphical vs. textual) impact developers' effort, task time, and accuracy in comprehension tasks?

RQ4.2: Does the structure of the representations lead developers to use specific viewing strategies (top-down vs. bottom-up) during comprehension tasks?

RQ4.3: Given graphical and textual representations, is there any preferred representation by the participants?

4.3.2 Research Hypotheses

We formulate RQ4.1 null hypotheses as follows:

- Hα_{11}: There is no significant difference in the average accuracy of the participants' answers when performing the requirement understanding tasks with graphical and textual representations.
- Hα_{12}: There is no significant difference in the average task time when performing the requirement understanding tasks with graphical and textual representations.
- Hα_{13}: There is no significant difference in the average visual effort when performing the requirement understanding tasks with graphical and textual representations.

Table 4.1 Overview of the eye-tracking experiment.

	Experiment
Goal	Study the impact of representations type (graphical vs. textual) on model comprehension.
Independent variables	Representation types: Textual (T) or Graphical (G).
Dependent variables	Accuracy, Time, and Visual effort.
Mitigating variables	Study level, Experience, UML knowledge, and English language proficiency.

Research question RQ4.2 deals with the participants' viewing strategies; we formulate the following null hypothesis:

- Hα_{21}: Despite the structure of the representation, participants will not use specific viewing strategies while working with the representation to answer the comprehension questions.

Finally, for research question RQ4.3, we formulate the following null hypothesis:

- Hα_{31}: Between graphical and textual representations, there is no preferred representation by participants while comprehension reading tasks.

4.3.3 Data Collection

We use two methods to collect data during the experiment:

1. We use two questionnaire (pre-experiment and post-experiment) to obtain relevant information. The pre-experiment questionnaire is used to collect information about participants' level of study and their self assessment of UML and modeling knowledge, and English proficiency. The post-experiment contains eleven questions about participants' preferred representations. We ask participants to rate the extent to which they found and used two representations effectively.

   In addition, we ask participants to give us their opinion about the experiment in general, (are the tasks and questions clear and easy to understand? and do they feel safe while sitting in front of eye-tracker or not?).

2. We perform a controlled experiment using FaceLAB (See Section 2.1.2) to capture eye-movements data.

4.3.4 Task

To design our experiment, we assign a set of comprehension tasks to developers to perform. This is because we can not directly observe how a reader understands a representation, unless we assign a list of tasks to the reader that can only be solved correctly when the representation is fully understood (Krogstie et al., 2006; Petrusel and Mendling, 2013).

We randomly assign our participants to the four groups presented in Table 4.2. Each participant works on three different sessions. In each session, each participant works with one treatment: graphical representation, textual representation, or a mixed model, including both graphical and textual representations.

4.3.5 Material

Stimuli

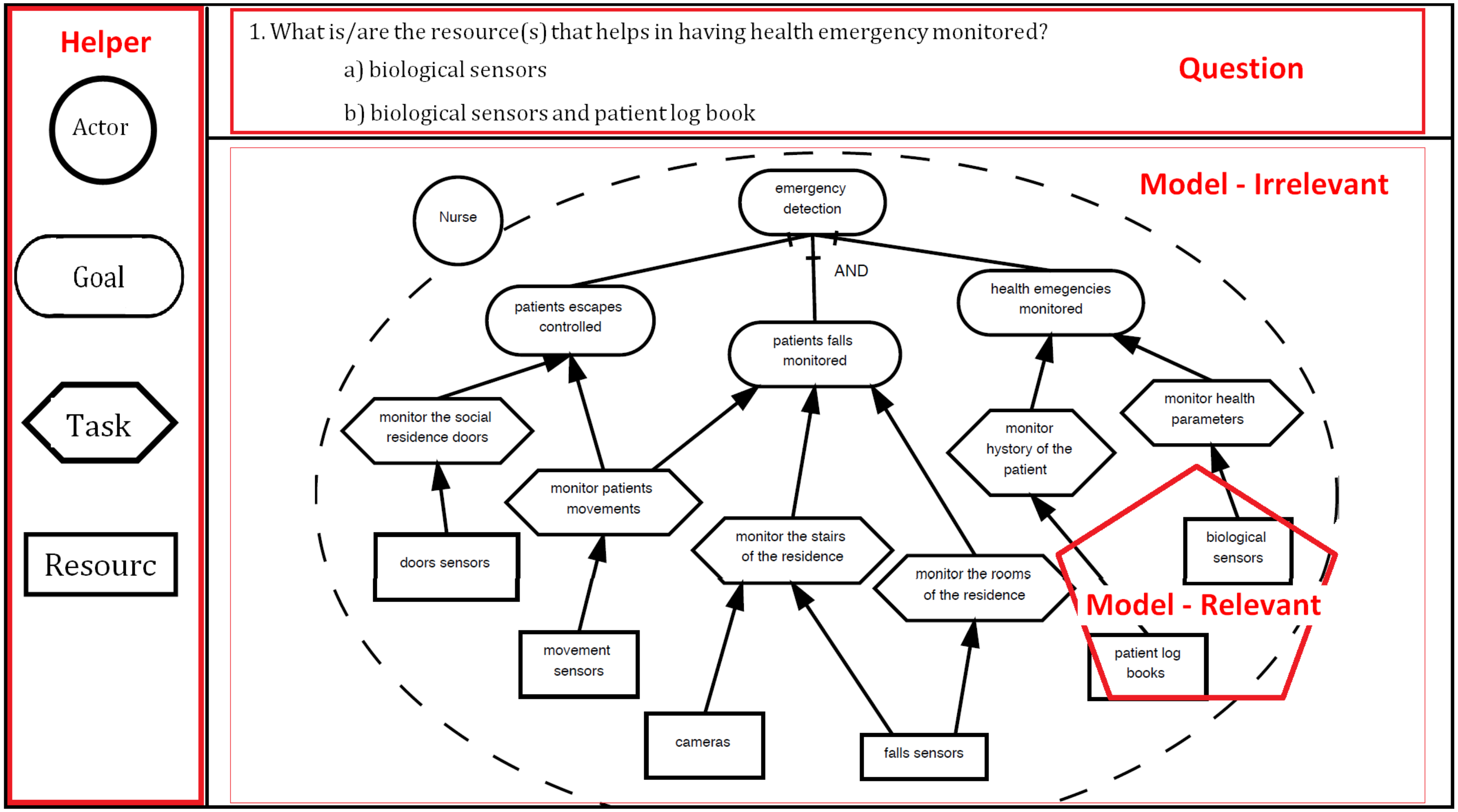
Stimuli are presented using the TROPOS (Bresciani et al., 2004) graphical and textual representations. There exist several textual and graphical representations to describe requirements, including GLR [1] and many others (Laurent et al., 2010). We choose TROPOS because it proposes both a graphical representation and a structured text. We are interested to investigate the impact of this structure on developers' efficiency. Moreover, the objects that use as stimuli are extracted from a real industrial project, documenting a hospital information system. These materials have been verified to confirm that not only no error exits but also the content of graphical representation conforms with textual one. In another word, we ensure that these two representations present the same content only in two different representations.

Three objects are used in each stimulus: Model A, Model B, and Model C. These objects Model A and B are presented in either graphical or textual forms for each participant while model C is presented in both graphical and textual forms at the same time. Therefore, our participants can choose between the graphical or textual representations or use both of them while working with model C. Figure 4.1 and Figure 4.2 show examples of our graphical and textual representations along with different AOIs.

For Model A and Model B, our graphical representations contain 17 and 18 elements while the number of lines for the textual representations are 19 and 22 for Model A and Model B respectively. The graphical representation of Model C contains 13 elements and its textual representation contains 15 lines of text (font size = 16).

TROPOS is a goal-based oriented modeling approach to visualise requirement through actor

[1] http://www.cs.toronto.edu/km/GRL/



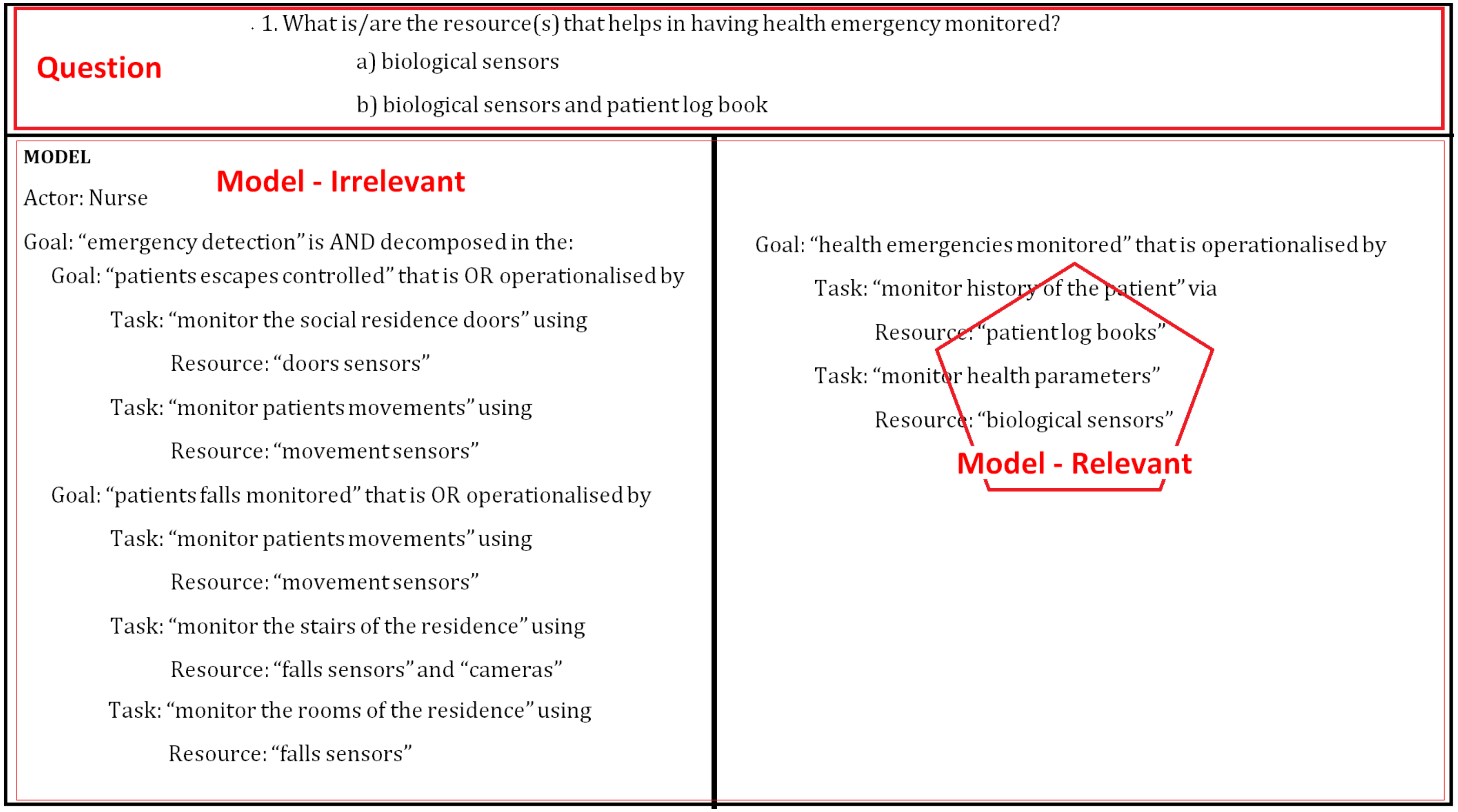
1. What is/are the resource(s) that helps in having health emergency monitored?

a) biological sensors

b) biological sensors and patient log book

Helper
Actor
Goal
Task
Resourc
Question
Model - Irrelevant
Model - Relevant
Nurse
emergency detection
AND
patients escapes controlled
patients falls monitored
health emegencies monitored
monitor the social residence doors
monitor patients movements
monitor the stairs of the residence
monitor the rooms of the residence
monitor hystory of the patient
monitor health parameters
doors sensors
movement sensors
cameras
falls sensors
biological sensors
patient log books

Figure 4.1 Portion of the graphical stimulus that contains five AOIs: 1) model area contains: 2) model relevant, 3) model irrelevant; 4) question area, and 5) help area.



Question
Model - Irrelevant
Model - Relevant

1. What is/are the resource(s) that helps in having health emergency monitored?

a) biological sensors

b) biological sensors and patient log book

MODEL

Actor: Nurse

Goal: "emergency detection" is AND decomposed in the:

Goal: "patients escapes controlled" that is OR operationalised by

Task: "monitor the social residence doors" using

Resource: "doors sensors"

Task: "monitor patients movements" using

Resource: "movement sensors"

Goal: "patients falls monitored" that is OR operationalised by

Task: "monitor patients movements" using

Resource: "movement sensors"

Task: "monitor the stairs of the residence" using

Resource: "falls sensors" and "cameras"

Task: "monitor the rooms of the residence" using

Resource: "falls sensors"

Goal: "health emergencies monitored" that is operationalised by

Task: "monitor history of the patient" via

Resource: "patient log books"

Task: "monitor health parameters"

Resource: "biological sensors"

Figure 4.2 Portion of the textual stimulus that contains four AOIs: 1) model area contains 2) model relevant and 3) model irrelevant areas; 4) question area.

Table 4.2 Design groups for assigning models to different participants.

Group 1	Session 1: Model A in graphical representation. Session 2: Model B in textual representation. Session 3: Model C in both graphical and textual representation.
Group 2	Session 1: Model A in textual representation. Session 2: Model B in graphical representation. Session 3: Model C in both graphical and textual representation.
Group 3	Session 1: Model B in graphical representation. Session 2: Model A in textual representation. Session 3: Model C in both graphical and textual representation.
Group 4	Session 1: Model B in textual representation. Session 2: Model A in graphical representation. Session 3: Model C in both graphical and textual representation.

and goal diagrams (Bresciani et al., 2004). It is based on the i* modelling framework and proposes goal-oriented modelling techniques. TROPOS defines five basic concepts including:

- Actor: it represents a position (role) or an agent that can be a human stakeholder or an artificial agent (software and hardware system).
- Goal: it is an interest of an actor that can be composed of several sub-goals. For each actor, there should be at least one goal.
- Task: it is a particular course of actions that can be executed to satisfy a goal.
- Resource: it is a physical (*e.g.,* printer) or informational (*e.g.,* notes, web-sites) entity that could be used/produced by actors through different tasks to realise goals.
- Social dependency (between two actors): one actor depends on another actor to achieve a goal, execute a task, or deliver a resource.

TROPOS models are visualised through actor and goal diagrams. An actor diagram is a graph whose nodes represent actors while the vertices show the dependencies between pairs of actors. A goal diagram is dedicated to an individual actor and represents the actor's main goals (their decomposition into sub-goals), the tasks and the resources to achieve the goals.

In addition, a TROPOS model can be represented using graphical or textual formats. In graphical format, each concept is shown using a unique element *e.g.,* the actor, goal, task, and resource are represented by a circle, ellipse, hexagon, and rectangle respectively. Moreover, all elements are shown based on three levels of abstraction including the goal level (high level) at the top, the task level in the middle, and the resource level at the bottom that contains goal and task and resource elements respectively. In textual format, each sentences starts with a word mentioning the name of the concepts that is explained (*e.g.,* "Goal: health emergency management.”). In this dissertation, we use the goal diagram in both graphical and textual formats.

Questions

In each session, the participants answer six comprehension questions. These questions were divided into two main categories, bottom-up and top-down as shown in Table 4.3. In top-down questions, we ask our participants to find the resources or the tasks to fulfill a goal. For these questions, our participants must first find the top-level goal (high-level of abstraction) and then refine the abstraction to find the required tasks or resources (lower-level of abstraction). In bottom-up questions, we ask our participants to find the goals that use specific resources. For these questions, our participants must first find the resources then, going up in the representation, identify the goal.

4.3.6 Study Variables

The variables under study are designed as follows:

Independent variables

The type of representation (graphical vs. textual) is the independent variable, *i.e.,* treatment, for RQ4.1 and RQ4.2. For RQ4.3, we focus the mixed model in which we present both treatments to understand participants’ preferences.

Dependent variables

The dependent variables are chosen as follows:

Accuracy: we quantify and measure this variable by the percentage of correct answers given by a participant in the multiple choice questions.

Task Time: we measure this variable as the amount of time that each participant spends on each model stimulus. Our eye-tracker provides this variable for each participant per stimulus.

Effort: mental effort is the amount of brain activity required to complete a task. Eye-movement processing helps us to provide an approximate value for the mental effort by measuring the amount of visual attention that participants spend on each stimulus (Duchowski, 2007). Based on this analogy, we can assume that less attention and less time mean less effort.

Table 4.3 Two groups for comprehension questions.

Top-down Group	Bottom-up Group
1) What are the resources that used in the realisation of the Goal X? 2) What are the tasks that used in the realisation of the Goal X?	1) What is the usage of resource Y? 2) Resource Y is used in task Z, is it correct or not?

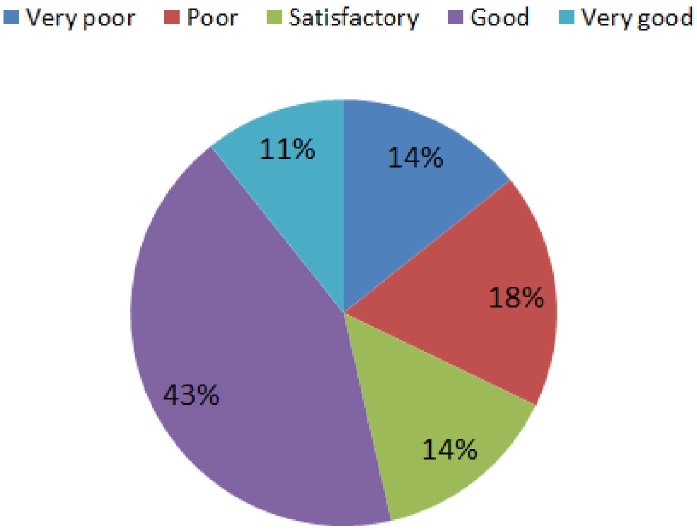


Figure 4.3 Participants' self assessment of object-oriented modeling experience.

In our experiment, we have two treatments/stimuli: the graphical representation and the textual representation. We collect data about fixations on the set of areas of interest (AOI) in each stimulus to compute the participants' effort. We establish five AOIs for graphical stimulus and four AOIs for textual stimulus as illustrated in Figure 4.1 and Figure 4.2, respectively. Some areas cannot be clearly identified in the figures as areas may overlap/intersect, *e.g.,* the model area and the question area. In addition, some areas are parts of other areas, *e.g.,* Model area contains Model relevant and irrelevant areas.

Thus, we defined:

– Model area (M): it includes all model elements.

– Model relevant area (Re): it includes all model elements that are related to the correct answer.

– Model irrelevant area (Ir): it includes all model elements that are *not* related to the correct answer.

– Question area (Q): it includes the question and multiple choices that appear at the top of

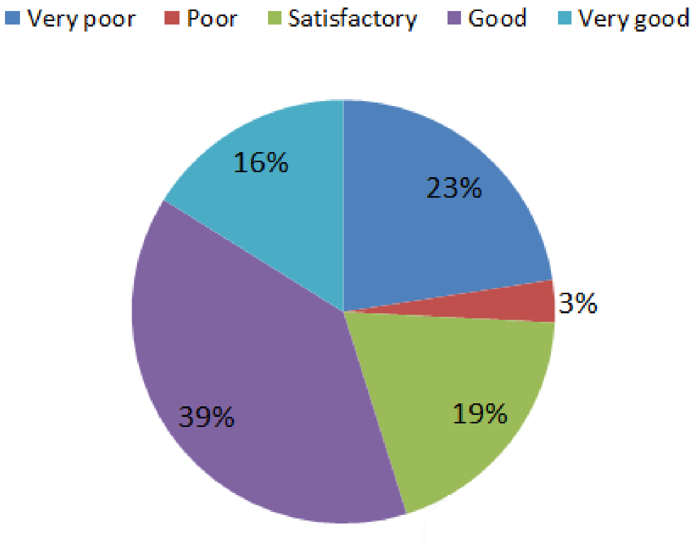


Figure 4.4 Participants' self assessment of UML and modeling knowledge.

the screen.

– Help area (H): it includes help items for the graphical model displaying TROPOS graphical elements.

We use two metrics for calculating the visual effort:

1. The average fixation duration (AFD) (See Section 3.4.4): we compute AFD for five AOIs of graphical stimulus and four AOIs of textual one separately and compare them. Using AFD, we identify on which representation, our participants spent shorter fixations, thus they perform more efficiently.

   Representations that require shorter fixations are more efficient than the ones with longer fixations (Cepeda Porras and Guéhéneuc, 2010b).

2. The surface of convex hull (See Section 3.4.4): for each participant, we compute convex hull for graphical and textual stimuli separately. By comparing the value of the convex hull of graphical with textual, we identify the representation in which fixations are close from one another, hence, participants spent less effort navigating the stimulus to understand it and answer the question.

Mitigating variables

In this experiment, we use a questionnaire to collect data about the following mitigating variables:

1. Level of study: values for this variable are B.Sc., M.Sc., and Ph.D.

2. Level of experience in object-oriented modelling. We ask our participants to provide a self-assessment (very poor, poor, satisfactory, good, very good) of their level of expertise of object-oriented modelling, in addition to, the years of modeling experience (less than 1 year, 1 to 3 years, 3 to 5 years, and more than 5 years).
3. Level of UML knowledge. We ask our participants to provide a self-assessment (very poor, poor, satisfactory, good, very good) of their UML knowledge.
4. English language proficiency and linguistic distance. Regarding English language proficiency, we ask our participants to provide a self-assessment (very poor, poor, satisfactory, good, very good) of their proficiency.

   Chiswick and Miller (2005) proposed a measure, called Linguistic distance, to find out how difficult for someone who knows language A to learn language B. They assigned each language a score (Hart-Gonzalez and Lindemann, 1993) and stated that "if it is more difficult to learn language B1, than it is to learn language B2, it can be said that language B1 is more 'distant' from A than language B2.".

   Heijstek et al. (2011) adopted this measure to investigate whether the difference between their participants' native language and English can impact their efficiency and accuracy while understanding a model. In this study, we are also interested to investigate if the distance of our participants' native language from English can impact our findings.

4.3.7 Participants' Demographics

The study participants are 28 volunteers, 12 (46%) female participants and 16 (54%) male participants. The participants are two B.Sc., 11 M.Sc., and 15 Ph.D. students from computer and software engineering and computer science departments of the Montreal area.

Figure 4.3 displays Participants' self reported level of experience in object-oriented modeling. 67% of our participants have satisfactory, good, or very good level of experience in object-oriented modeling. Regarding the knowledge of UML, 75% of our participants have satisfactory, good, or very good UML knowledge as depicted in Figure 4.4.

All participants had normal vision; some wore contact or corrective lenses. The participants were not aware of the experiment's hypotheses.

Before the experiment, we inform our participants that the experiment has three sessions and that each session was allotted about 10 minutes and that they are free to leave at any time without incurring any penalty. Collected information is anonymous. We validate the response forms to make sure that participants correctly followed the experiment procedure.

The participant demographics are presented in Table 4.4.

Table 4.4 Participants' demographics

Participants' Demographics				
Academic Background			Gender	
Ph.D.	M.Sc.	B.Sc	Male	Female
15	11	2	16	12

4.3.8 Procedure

We use a quiet room to perform the experiment. A 27" LCD screen is used to display the stimuli while the participants are seated approximately 70 cm away from the screen in a comfortable chair with arms and head rests. Before running the experiment, we give a tutorial to explain TROPOS modelling concepts and its elements during about 20 minutes. Then, we briefly explain how the eye-tracker works and what information is gathered by the tool. For each participant, we first calibrate the eye-tracking system.

Then, we start the first session by presenting the first screen which describes to participants how to perform the tasks and complete the experiment. When participants begin a task, we start collecting data. We do not give any time limit to the participants. We display a representation and a question at the same time; therefore, participants always have access to the representation to answer the questions. When participants finish analysing representations and find an answer, they press the "space" key to go to the next blank screen, and write down their answer to the question, *i.e.,* choose one of the two alternatives. Once a task is finished and the answer given, participants press the "space" key to go to the next question. When participants complete the three tasks, we ask them to answer the post-experiment questionnaire.

4.4 Study Results

In this section, we report hypotheses testing and discuss the results of our experiment. Figure 4.5 presents the workflow of our data analysis.

1. We use *FaceLAB* to capture the eye-movement data of the participants and export them to text files (.out files). Table 4.5 summarises the collected data.
2. We use *Taupe* (De Smet et al., 2012) to parse .out files, calculate the data, and generate the following outputs:
   (a) CVS files that contain the following information for each participant per stimulus and per AOI: 1) the duration of spent time, 2) the number and the duration of fixations, 3) the amount of visual effort (AFD), and 4) the area of convex-hull.

   (b) Visualization of heatmaps, convex hulls, and scan-paths for each participant per stimulus.

3. We provide four different files (CVS files of time and effort, Excel files of accuracy, and user's opinion) to *R* (Team et al., 2010) to perform statistical analysis. In the analysis of our data, we made no assumption and applied non-parametric, non-paired tests to determine significance differences.

4. We use *MATLAB* to perform scan-path analysis and compare the scan-paths of participants and visualize them.

4.4.1 RQ4.1: Does the Type of Representations Impact Developers' Effort, Task Time, and Accuracy in Comprehension Tasks?

Accuracy

Each participant, when answering a question, choose either a correct or a wrong answer. Table 4.6 is the contingency table reporting the number and the percentages of correct and wrong answers for textual, graphical, and mixed representations. We test our hypotheses, $H\alpha_{11}$, to find any potential advantage of graphical vs. textual representations. After applying two-tailed Wilcoxon with ($\alpha = 0.05$), the p-value reports that there is no significant difference between textual and graphical representations regarding accuracy. We cannot reject the null hypothesis $H\alpha_{11}$. Our results concur with the findings of (Heijstek et al., 2011), who reported that neither graphical nor textual representation had a significant effect on correct answers.

Task Time

We perform three different analyses. First, we calculate the amount of time that each participant spends on each stimulus to perform the task (Task time), then we examine the impact of representation type. Second, we calculate the amount of spent time on each AOI for graphical and textual stimuli and present the results. Finally, we calculate the start time of the first fixation that appears on the relevant AOI for textual and graphical stimuli separately and compare the results.

– The effect of representation type on task time: we investigate the second hypothesis, $H\alpha_{12}$, which examines the effect of representation type on the time that participants spend on each stimulus to read a question, analyse the model, and answer the question.
  Figure 4.6 presents the distribution of the task time dependent variable for graphical and textual representations while Table 4.7 and Table4.8 show the average amount of time that our participants spent on graphical and textual representations for Model A, B, and C,

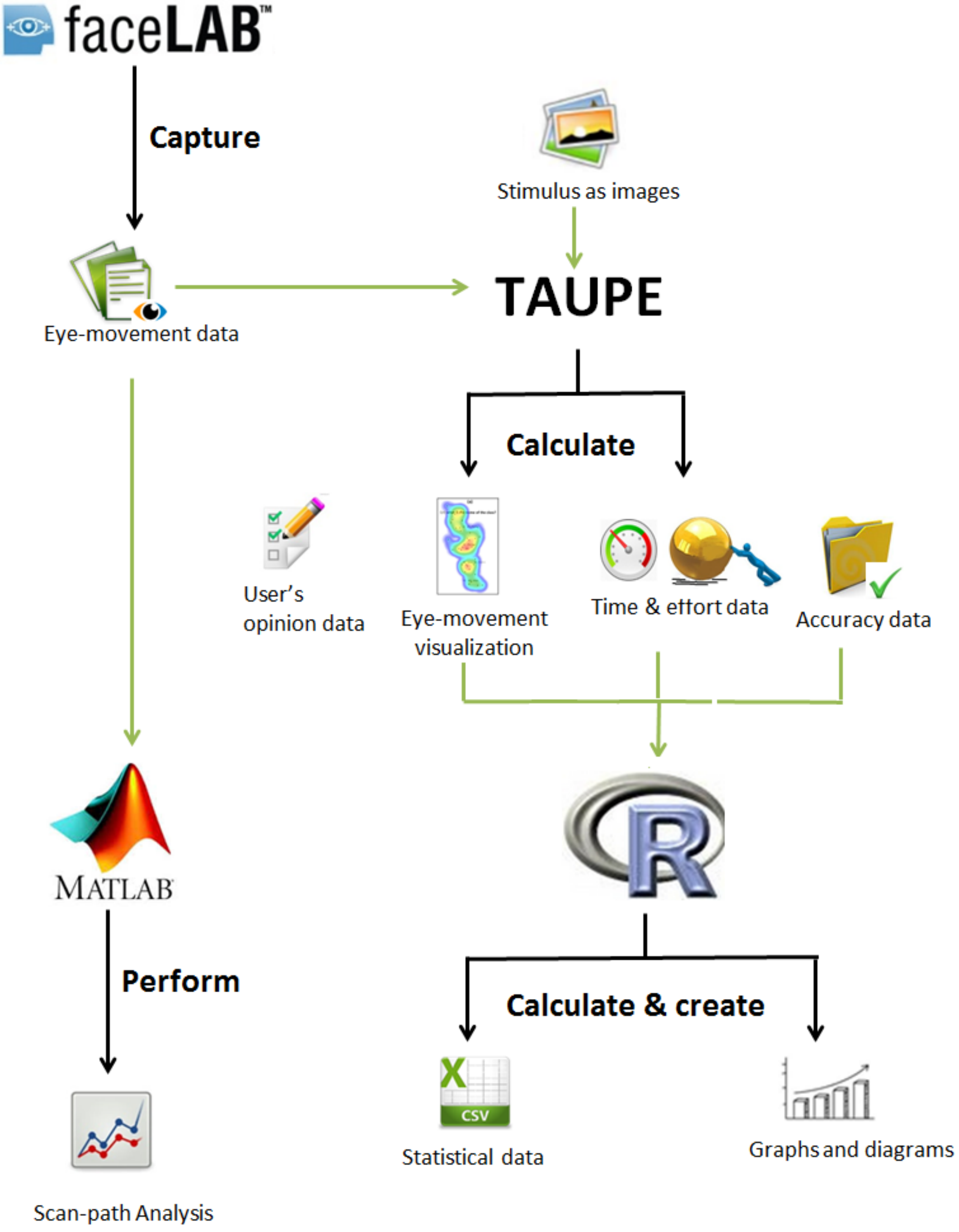


Figure 4.5 Workflow of data analysis.

Table 4.5 Main features of the experiment.

Collected data	
Number of participants (#)	28
Total number of questions	504
Number of Text-related questions	168
Number of Graphical-related questions	168
Number of Mixed questions	168
Total time of eye-tracking (hours)	2.85
Total number of fixations (#)	50,652

Table 4.6 Participants' contingency table for textual, graphical, and mixed representations.

Answers					
Textual		Graphical		Mixed	
Correct	Wrong	Correct	Wrong	Correct	Wrong
164	4	165	3	162	6
97%	3%	98%	2%	96%	4%

separately. On average while considering Model A and Model B, our participants spent 47% more time (10,498 ms) on graphical representation than textual representation. There is a significant difference between graphical and textual representations (p-value = 1.487e-05, Cohen-d: 0.54 (medium effect)) although this extra time does not affect the accuracy. We can reject the null hypothesis $H\alpha_{12}$.

– Time spent on each AOI: an eye-tracker gives us the ability to compute the time that is spent on different parts of a representation (AOI) separately. Therefore, by considering a set of AOIs that are presented in Section 4.3.6, we separately compute the percentage of time that our participants spent on different AOIs of textual and graphical representations. Figure 4.7 shows the percentages of time that our participants spent on different parts of representations. When we compare the amount of time on different parts, the results shows that our participants spend more time on both, relevant ($p\text{-value} = 0.0016 < 0.05$, Cohen-d: 1.0 (large effect)) and irrelevant ($p\text{-value} = 0.0022 < 0.05$, Cohen-d: 1.0 (large effect)) parts of the presentation when working with the graphical presentation compared to the textual model. The time spent on the helper part was negligible.
– Start time of the first fixation on relevant AOI: to calculate this variable, we find the first fixation that is displayed on the relevant AOI and extract its start time. This variable shows how fast the participant finds relevant elements or texts in a representation. Figure 4.8 shows the amount of start time of the first fixation for all participants for different questions of Model A for graphical and textual representations. It shows that for the first question,

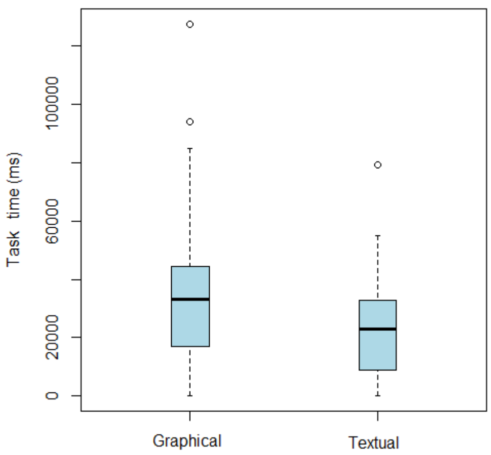


Figure 4.6 Descriptive statistics for task time for graphical and textual representations.

Table 4.7 Average time and effort spent by participants on Models A, B while performing the requirements understanding tasks with graphical (G) or textual (T) representations.

		Model A	Model B
Average Time (ms) (SD)	G	36,925.5 (23,258.8)	32,856.3 (18,471.60)
	T	23,346.5 (14,570.77)	22,340.05 (15,406.64)
Average Effort (SD)	G	72.96 (29.05)	65.89 (27.56)
	T	33.9 (37.88)	35.7 (48.26)

it takes much more time to find the relevant AOI. But as participants continue answering more questions, they become familiar with the model and they spend less and less time to find the relevant AOI and answer the question.

Surprisingly, this observation can not be made for the textual representation. It shows that participants can not memorize the location of the elements as they spend more time on the questions. Although, in average participants work faster with textual ones.

Visual Effort

We analyse the third hypothesis, $H\alpha_{13}$, which examines the effect of a representation type on the effort that participants spend to perform model comprehension tasks. We compare the value of our participants' AFDs for graphical and textual representations while considering different AOIs including Model area (AFD(M)), Question area (AFD(Q)), Model relevant (AFD(Re)) and Model Irrelevant (AFD(Ir)) areas. Figure 4.9 presents the distribution of the AFD dependent variable for graphical and textual representations. The results of applying Wilcoxon test as presented in Table 4.9 show that there is a significant difference between

Table 4.8 Average time and effort spent by participants on Model C while performing the requirements understanding tasks with graphical (G) or textual (T) representations.

		Model C
Average Time (ms) (SD)	G	12,195.47 (1276.39)
	T	4,445.18 (751.80)
Average Effort (SD)	G	36.30 (26.29)
	T	16.61 (15.39)

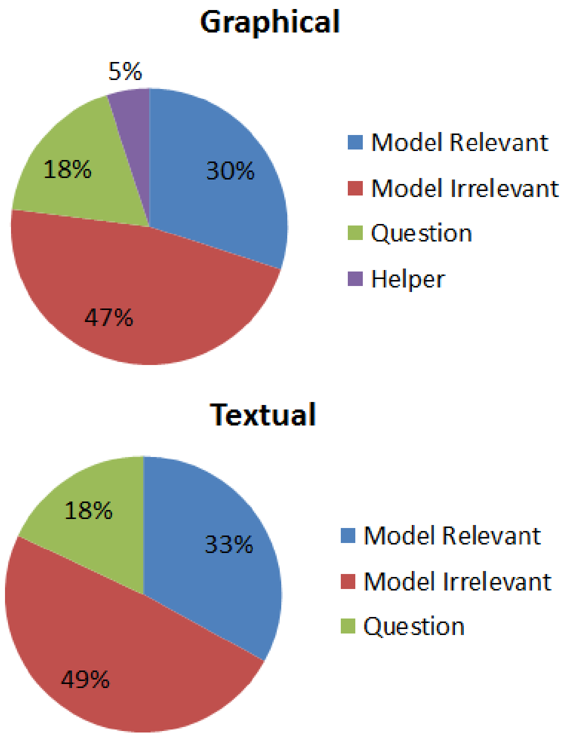


Figure 4.7 Percentage of time that our participants spend on different part of graphical and textual representations.

graphical and textual representation while comparing AFD for Model area (AFD(M)) and Model relevant (AFD(Re)) and Model Irrelevant (AFD(Ir)) areas. These values show that our participants spent more visual effort while looking and analysing the irrelevant parts of the model to find the relevant elements and relevant parts to find the correct answer with the graphical representations.

Our results show that our participants have longer fixations while working with the graphical representations, which means they are spending more effort analysing and interpreting the elements of graphical representations. As expected, there is no significant differences between

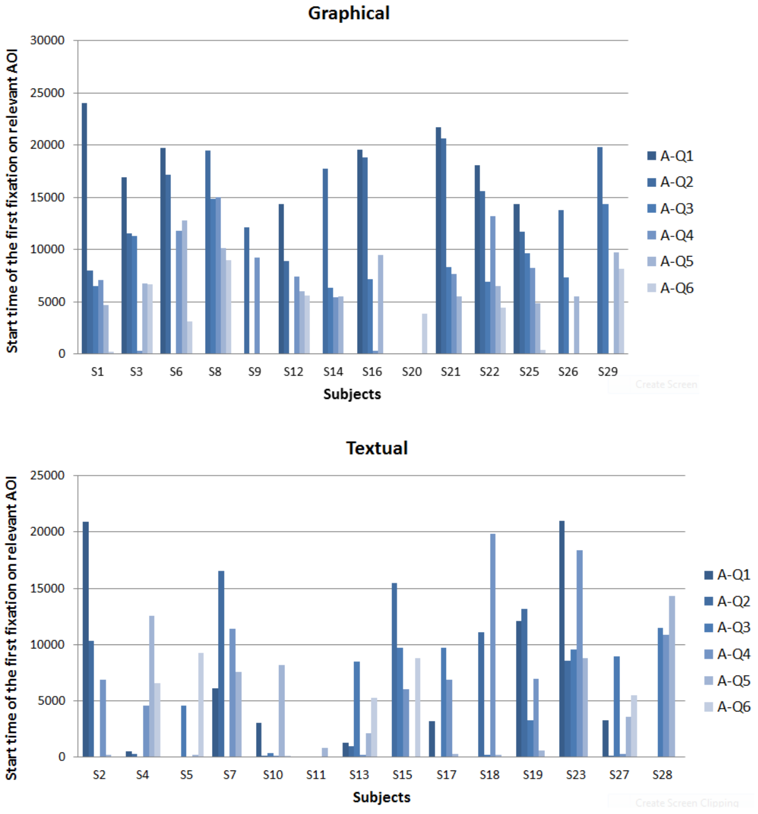


Figure 4.8 Start time of the first fixation on relevant AOIs for all participants answering six questions of Model A presented in graphical (top) and textual (bottom) representation.

the amount of visual effort for Question part (AFD(Q)) because the question part is the same for both representations.

Moreover, we use the non-parametric Wilcoxon test to compare the surface of the convex hull of our participants' fixations on Model AOI for textual and graphical representation. We use Taupe (De Smet et al., 2012) to compute the surface of convex hulls. As expected, using the textual representation significantly decreases the value of convex hull ($\alpha = 0.05$, p-value $=$ 0.01). This result shows that the fixations are close from one another when our participants work with textual representation, thus, our participants put less effort to explore the whole model to find the relevant parts of the stimulus to answer the question.

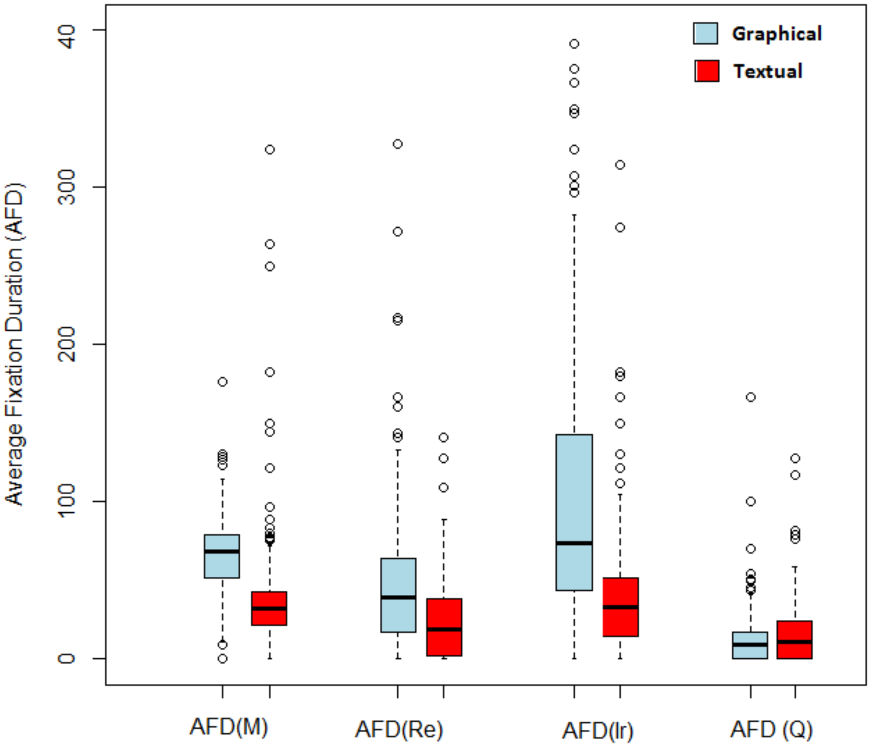


Figure 4.9 Descriptive statistics of AFD for graphical and textual models.

4.4.2 RQ4.2: Does the Structure of the Representations Lead Developers to Use Specific Task-solving Strategies (Top-down vs. Bottom-up) During Comprehension Tasks?

We perform the following steps to answer RQ4.2: first, we consider a new set of AOIs including goal, task, and resource areas representing different levels of abstraction of TROPOS representations and also the Question and Help areas, as shown in Figure 4.10.

Second, we use ScanMatch (Cristino et al., 2010) to compute and show different scan-paths for each participant working on all questions. ScanMatch assigns a character to represent each AOI. The question AOI, goal AOI, task AOI, resource AOI and helper AOI are represented by B, C, D, E, and F respectively. To display the scan-path, a small letter is attached to the capital letter to make it easy to read the sequence. Therefore, if a participant goes from goal AOI to task AOI and then to resource AOI, the sequence for the scan-path is cCdDeE.

Third, using ScanMatch, we compare the participants' scan-paths when working with six different questions of the Model A and Model B, presented in graphical representations. ScanMatch calculates the similarity values to show the similarity of two scan-paths temporally

Table 4.9 Two-tailed Wilcoxon p-value ($\alpha = 0.05$) and Cohen-d for the Average Fixation Duration (AFD) metrics of different AOIs.

Variables	p-value	Cohen-d
AFD(M)	$< 2.2e\text{-}16$	0.64
AFD(Re)	$< 9.328e\text{-}07$	0.50
AFD(Ir)	$< 6.787e\text{-}11$	0.83
AFD(Q)	0.07163	–

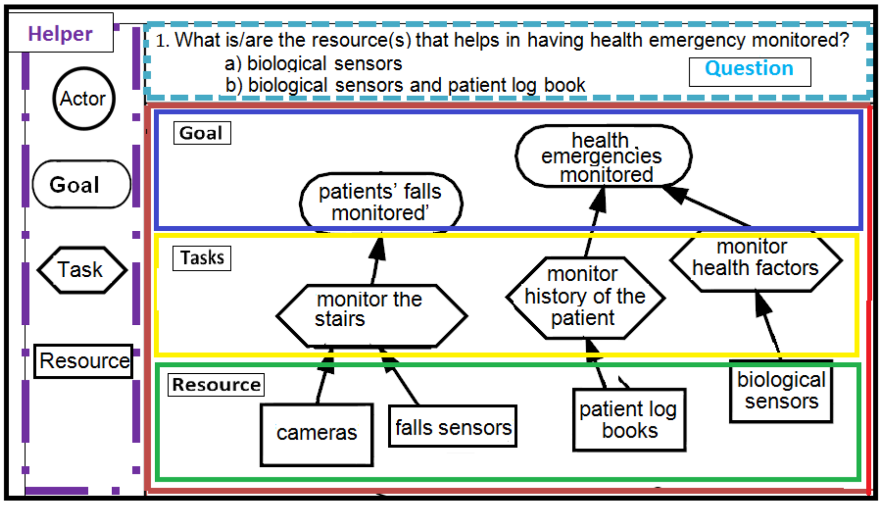

Helper
Actor
Goal
Task
Resource
1. What is/are the resource(s) that helps in having health emergency monitored?
a) biological sensors
b) biological sensors and patient log book
Question
Goal
health emergencies monitored
patients' falls monitored'
Tasks
monitor health factors
monitor history of the patient
monitor the stairs
Resource
cameras
falls sensors
patient log books
biological sensors

Figure 4.10 Set of five AOIs including: 1) goal level, 2) task level, 3) resource level, 4) question, and 5) helper areas for TROPOS graphical representation.

and spatially. If two scan-paths are identical, the score is 1. If they do not have any relationship, the value will be 0. We calculate the similarities for all participants pair-wise. For example, if participants 1, 2, and 3 are working on Q1 of Model A, we compare and compute the scan-paths of participants 1, 2, and 3 then, we calculate the similarity value of participant 1 vs. participant 2, participant 1 vs. participant 3, and participant 2 vs. participant 3. We perform this procedure for all six questions for both models A and B and obtain, for each question, a list of similarity values for each pair of participants.

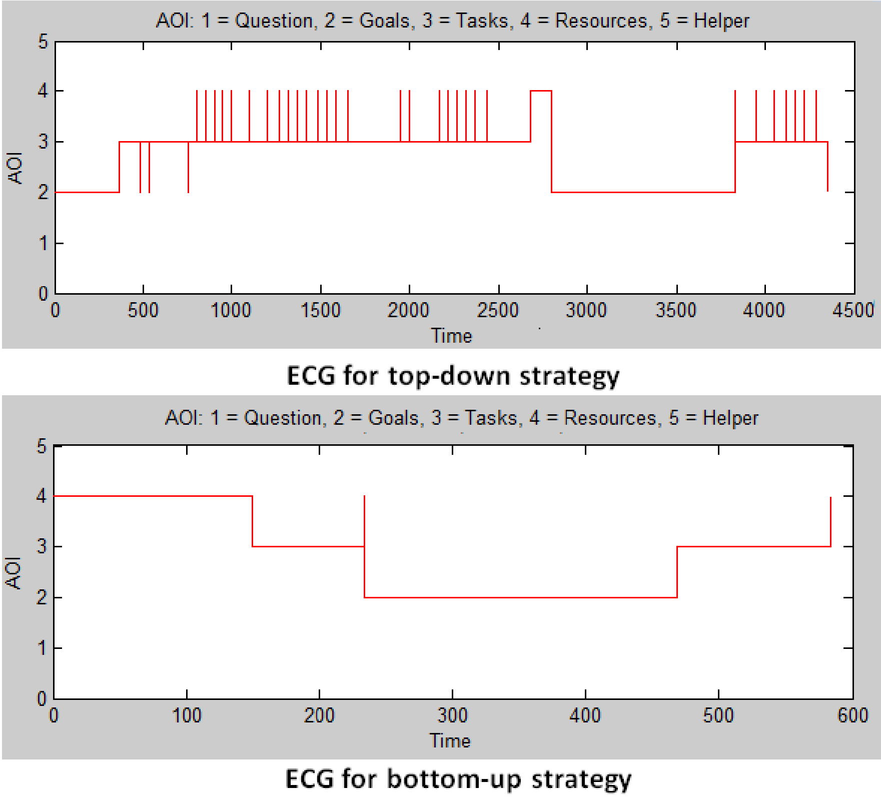
Fourth, based on our top-down and bottom-up questions presented in Table 4.3, we expect to detect two different strategies. Our participants answer six different questions for each model. For each question, we have a list of similarity values for the scan-paths of all pairs of participants. We perform the two-tailed version of the unpaired Wilcoxon test, using the Bonferroni correction, on these lists. Then, we compute Cliff's delta to indicate the right categorisation with the highest effect size. Based on this result, our questions are divided into two groups: first group consists of Q1, Q2, Q4, and Q5 while the second group consists of Q3 and Q6. The result of the statistical test shows that our participants use different strategies to answer questions presented in the first and second group with the first group containing top-down questions and the second group consisting of bottom-up questions.

Finally, to visually show the different task-solving strategies used by our participants, we draw the structure transition graphs (STG) proposed by Sim et al. (2009) for different participants working on two different groups of questions: top-down and bottom-up. Sim et al. (2009) use the STG to visualise the progress and the rhythm of program comprehension while asking participants to modify a system. They depict STG to show that developers attempts to go back and forth within files that are in neighboring layers. They conclude that the STGs show patterns in developers' behaviour. We adopt this method to show how our participants navigate through different AOIs (different levels of abstractions) to answer the questions for graphical representations.

As shown in Figure 4.11, the x-axis in the STG is time while the y-axis shows the different AOIs. A STG shows the navigation sequence between AOIs for participants while looking and analysing the graphical model to answer two different types of questions: bottom-up and top-down. As shown in the image at the top, the participants started from goal part (level 2) going through task part (level 3) to reach the resource part (level 4) to answer a question of top-down group. The second image at the bottom shows the bottom-up navigation that started from resource part (level 4), going through task part (level 3), and finally reaches goal part (level 2) to answer a question of group B.

Sometimes our participants read the question from the paper in hand. Therefore, they do not visit the question part (level 1). We believe that fast traversing between two different AOIs, which appears as a straight vertical line in the STG, can be considered as a saccade and can be removed. The vertical line between AOI 1 and AOI 2 means that the participant was looking at AOI 1 then suddenly looked at AOI 2 for less that 0.01 ms and looked back at AOI 1. Because the participants do not spent enough time on AOI 2, we can not consider the vertical lines as a complete transition between two AOIs.

Our findings reject $H\alpha_{21}$ and confirm that the hierarchical structure of the TROPOS graphical representation leads our participants to follow a specific strategy, either top-down or bottom-up, to answer the question.



ECG for top-down strategy

ECG for bottom-up strategy

Figure 4.11 STG graph which shows the navigation sequence of AOIs for a participant using top-down and bottom-up strategies.

4.4.3 RQ4.3: Given a Graphical and Textual Representations of a Comprehension Task, Is There any Preferred Representation by the Participants?

In our post-questionnaire, we ask our participants their preferences for answering comprehension question using graphical vs. textual representations. 82% of our participants prefer to work with a graphical representation as shown in Figure 4.12. Moreover, in the third session, when we provide participants with a mixed model (Model C), consisting of both graphical and textual representations, for 96% of the questions, our participants started with the graphical representation to find the answer.

Yet, our participants spend significantly more time (p-value < 0.05, Cohen-d: 1.0) and effort (p-value < 0.05, Cohen-d: 1.02) on graphical representations (see Table 4.7, Table 4.8 and Table 4.9). Therefore, we reject $H\alpha_{31}$ and answer RQ4.3 as follows: our participants' preferences is largely in favor of graphical representation. We conclude that participants' preference is not related to the real effort and time spent.

We apply Kruskal-Wallis rank sum test on three sets of time and visual effort to see if our participants spend equal amount of time and effort on the three models. The result of the

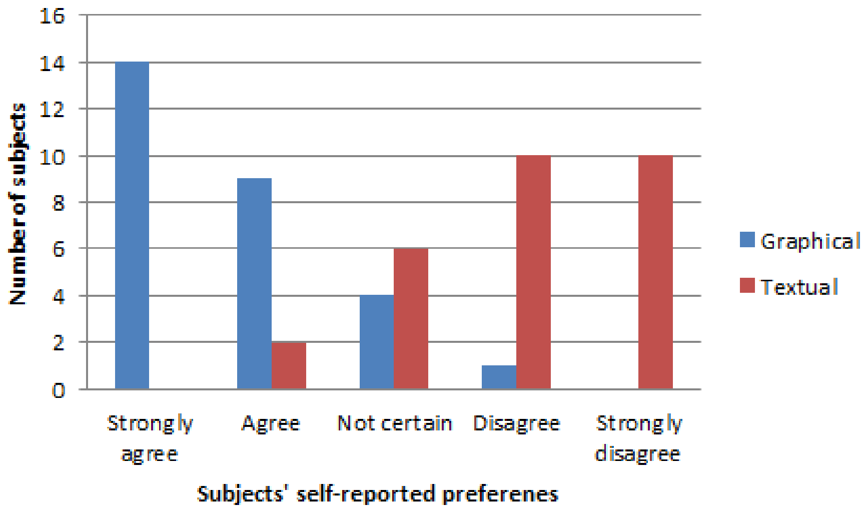


Figure 4.12 Participants' self reported preferences about working with graphical vs. textual representations.

test (p-value < 0.05) for both time and effort confirm that our participants spend more time and effort on Model A and B compared to Model C. This result confirms that, although none of our participants is familiar or use TROPOS before, they learn TROPOS through performing the experiment and this can significantly improve their efficiency. This finding emphasise that (in agreement with previous works (Siau, 2004)) developers must learn how to use a graphical representation before using it.

4.4.4 Impact of the Mitigating Variables

English language proficiency: 26% of our participants evaluate their English language proficiency as satisfactory while 40% and 34% evaluate it as good and very good respectively. However, there is no interaction between participants' English proficiency self-assessment and accuracy, time, and effort.

Linguistic distance: in Table 4.10, we provide a list of the languages that are encountered in the experiment and their associated linguistic distances. we find that the language distance is significantly correlated with time (p-value < 0.05). This result shows that the further a participant's native language is from English, the more time she spends to find the answer. This result is independent from the type of representation, which implies that the graphical representation could not help our non-native participants to overcome the language barrier. This result is in agreement with the work of Heijstek et al. (2011).

Experience: to evaluate the impact of experience, we consider study level, level of UML and of object-oriented modelling, and the number of years of modelling experience. None of

Table 4.10 Language grouping and distance. Language is self-reported by our participants, the score is reported in (Hart-Gonzalez and Lindemann, 1993) while the distance is 1/*score* as reported in (Chiswick and Miller, 2005)

Language	n	Score	Family	Distance
English	0	-	Indo-European	0.00
French	8	2.5	Indo-European	0.40
Farsi	14	2.00	Indo-European	0.50
Bulgarian	1	2	Indo-European	0.50
Bengali	2	1.75	Indo-European	0.57
Mandarin	2	1.50	Sino-Tibetan	0.67
Arabic	1	1.5	Afroasiatic	0.67

these values significantly interact (after applying one-way ANOVA) with representation types to have an effect on accuracy, time, and effort. Our participants are students so their prior modelling knowledge is rather homogeneous. We also design our tasks such that solving them does not require industrial experience. Our results show that the impact of representation types and the task cognitive complexity is not hidden by the participants' experience.

4.5 Threats to Validity

In the following, in addition to those factors that are discussed in Section 2.2.2, we discuss other factors that may have influenced our results.

The number of elements and lines is one limitation of the experiment using eye-tracking systems. There is a need to accurately and unambiguously quantify the visual effort by precisely identifying AOIs and thus precisely locating elements of the presented requirements. Yet, the number of lines for the textual representations in our experiment is similar to the source code size of previous eye-tracking studies (Sharif et al., 2012b; Uwano et al., 2006b). Moreover, the number of elements that are presented in our graphical representations are in the range of recommended number of elements for effective program comprehension (Ambler, 2005).

In addition, as explained in Section 2.2.2, the number of participants and their expertise are the most important factors regarding this threat to validity which also concern the generalizability of the results. We use 28 students as participants, our participants are graduate, Masters, and Ph.D students (except for two bachelor student who are currently enrolled in their 4^{th} year) with good knowledge of object-oriented modeling.

4.6 Discussions and Conclusion

We designed and performed an eye-tracking experiment to investigate the impact of textual vs. graphical representations on developers' efficiency while performing requirements comprehension tasks. We also examined the effect of graphical representation structure on developers' strategies.

We found no statistically-significant differences between representation types when considering accuracy. Remarkably, our developers spent more time and effort while working with the TROPOS graphical representations compared to textual ones. Moreover, the developers who performed more efficiently with both graphical and textual representations had a native language close to English, which implies that the graphical representation could not help non-native English speakers to improve their efficiency. In summary, these findings questions the role of graphical representations in requirement understanding task.

However, although graphical representations were not superior regarding time and effort, we observed that they still played an important role. Developers mostly preferred to use the graphical representation and they felt that it was more efficient than textual ones. Also, they looked more at the graphical representation while working with the mixed model in which both representations were available. In addition, our results showed that the structure of TROPOS graphical representation helped and guided developers to follow specific strategies, either top-down or bottom-up, to traverse different levels of abstraction. Also, while comparing the start time of the first fixation on the relevant AOI, we observed that developers found the relevant parts faster as they continued answering questions about the same model, which means that the topological structure of the graphical representation helped developers to memorize the location of the elements and find the one that they were looking for faster in comparison with textual ones.

In addition, developers performed significantly better regarding time and effort while working with the mixed model after they worked with the two first models and learnt TROPOS. Our results imply that the formalism of a graphical representation must be learnt by developers and that training is required before the benefits of a graphical representation can materialise. Although, the TROPOS graphical notation that we used is not complex and we gave a tutorial to familiarize developers with TROPOS notation, the developers spent 5% of their time looking at the Helper part.

We reported that structured text is more effective to work with compared to a graphical representation, which emphasizes the important role of layout for comprehension. This implication encourages us to take one step back and look at the very first phase of program comprehension, which is source code reading, and analyze source code as a type of structured textual representation. Moreover, because the results of our study did not confirm developers' preference regarding the usefulness of graphical vs. textual representations, we also consider gender as a factor that may impact program comprehension. We thus perform a set of empirical studies investigating the impact of gender on program comprehension, presented in following chapters.

CHAPTER 5 IMPACT OF GENDER ON PROGRAM COMPREHENSION ON IDENTIFIER RECALL

Although significant amount of work has been dedicated to gender differences in education and practice, there is little evidence about the role of gender in problem solving activities in software engineering (Burnett et al., 2010).

There are several reasons for analyzing gender differences in software engineering tasks. First, if such a difference exist, it confirms that male and female developers work differently with tools and–or use different strategies to solve the problem. Therefore, to consider this new understanding, current technology must be adapted to support both male and female developers.

Second, taking into account the difference and adopt current technology may benefit both genders because research confirmed that studying the need of one sub-population can benefit both sides (Ljungblad and Holmquist, 2007). For example, Czerwinski et al. (2002) conducted a study to investigate the impact of screen parameters on the efficiency of women navigating through 3D environments. The results show that a female gamer needs a wider field of view to navigate as effectively as a male gamer. Moreover, providing a wider range of view does not jeopardize male gamers' performance and even could enhance it.

Previous works, as presented in Section 5.2, report that gender differences are likely in problem-solving activities including programming. It has been reported that men and women use different problem solving (see Section 5.2.1) and–or information processing strategies (see Section 5.2.1) while tackling the task in hand. In software engineering, previous works analyzed the impact of gender on feature acceptance and attitudes toward new features (Beckwith and Burnett, 2004; Beckwith et al., 2005, 2006a,b, 2007; Grigoreanu et al., 2008). In addition, a handful of studies analyzed the role of gender in developers' debugging strategies (Subrahmaniyan et al., 2008; Grigoreanu et al., 2009).

However, the existence of gender differences during source code reading as a very first stage of program comprehension has received almost no attention. Moreover, little is known about gender differences that may exist in developers' program comprehension strategies while reading source code to understand it. Without this type of information, designers and developers of development environments cannot enhance these systems to support both male and female developers.

We hypothesize that gender has impacts on viewing strategies used by developers to read the source code and perform comprehension tasks. This dissertation provides an initial foundation for the following open question: Does gender play an important role in source code reading and consequently program comprehension?

In this dissertation, we present the results of a study investigating the effect of gender on developers' ability to recall identifiers during source code reading and comprehension. To some extent, we can look at source code as structured text in which some identifiers (class name, method name) bring not only semantic but layout and structure to source code. First, as identifiers are mainly presented in either Camel Case (CC) or Underscore (US) styles by literate developers, we seek to verify if, given our developer population, there was a difference in identifier style preference between male and female developers. Second, we investigate the relations between gender and developers' visual effort, task time, as well as ability to recall identifiers (accuracy). In addition, we identify the most important source code entities (SCEs) including class name, method name, variable name, or comment to which developers pay more attention while reading source code and performing their task. Finally, we investigate the viewing strategies used by male and female developers to read source code and answer questions. These strategies characterize the ways developers switch their focus of attention from one SCE to another.

5.1 Introduction

Several previous works investigated the role of identifiers in program comprehension (Deissenbock and Pizka, 2005; Binkley et al., 2009; Sharif and Maletic, 2010c). These works agreed on the important role of identifiers in program comprehension, in code readability, and, more generally, in software evolution (Caprile and Tonella, 2000; Deissenbock and Pizka, 2005; Binkley et al., 2009).

Developers often compose identifiers by concatenating terms, abbreviations, acronyms, and complete words. For example, Java developers often use English and the camel case (CC) convention to create identifiers. The CC convention is the practice of creating identifiers by concatenating terms with their first letter capitalized, giving the identifiers a camel-like look with flats and humps, *e.g., absolutePath.* The CC convention is the de facto standard for Java. However, it is not common in other programming languages, such as in C, where developers use different conventions, in particular the use of underscore (US), *e.g., absolute_path.*

The underlying conjecture in these works is that identifier style, *i.e.,* CC vs. US, affects memorability. Memorability (Binkley et al., 2009; Sharif and Maletic, 2010c) is an important factor that impact the comprehension of identifiers. It includes being able to recall an identifier by providing some semantic information associated with it (Jones, 2003). Lacking memorability may impair identifiers recall, code readability, and, ultimately, program comprehension.

Previous work, such as (Binkley et al., 2009) and (Sharif and Maletic, 2010c), reported contradictory findings on identifier styles. In terms of accuracy, measured as the percentage of correctly recalled identifiers and thus memorability, Binkley et al. (2009) reported that CC is more accurate than US while Sharif and Maletic (2010c) concluded that there is no difference regarding task time and accuracy.

In this chapter, we report the results of an empirical study conducted to analyse the impact of gender on recalling identifiers.

Section 5.2 contains an overview of related work. In section 5.3, the experimental design is presented and section 5.4 outlines an overview of the results. The threats to validity are addressed in Section 5.5. We provide discussion and a conclusion in Section 5.6.

5.2 Related Work

The approaches that are relevant to our work presented in this chapter focus on identifier names and general background about gender difference in problem solving activities.

5.2.1 Gender Differences in Problem Solving Activities

Gender differences in problem solving activities has been investigated in different research and different domains including marketing, psychology, and mathematics. They reported the role and the importance of different factors that are observed while considering gender differences including confidence, problem-solving style, and information-processing style (Burnett et al., 2010).

In this chapter, we present related works in three groups: self-efficacy, problem solving strategies, and electivity Hypothesis.

Self-efficacy

Self-efficacy is one's belief in her/his own ability to deal with and complete the task. Hartzel (2003) reported that females do not feel confident if they do not have task-specific experience but males do not hesitate to apply general knowledge for specific tasks.

Previous studies explained that females has less self confidence towards their abilities to success than males for the following tasks: word processing, spreadsheet and file manipulation, software management, and programming (Beckwith and Burnett, 2004; Torkzadeh and Koufteros, 1994; Busch, 1995; Margolis and Fisher, 2003; McDowell et al., 2003).

Beckwith et al., performed a set of studies (Beckwith and Burnett, 2004; Beckwith et al., 2005, 2006a,b, 2007) to analyze the impact of gender on feature acceptance and attitudes toward new features regarding end-user programmers. First, they investigated the gender differences related to end-user programmers from five different domains including computer science, computer gaming, education, phycology, and marketing. Moreover, they also provided a taxonomy of these literature (Beckwith and Burnett, 2004). Second, they investigated gender differences for end users while performing debugging tasks on spreadsheets. The results show that female participants approached the familiar features provided by the debugging environment earlier than their male counterparts. Moreover, male participants tended to tinker an unfamiliar environment and approached the new, unknown features much earlier than female participants.

They also performed another study (Beckwith et al., 2006b) to examine whether encouraging tinkering would help female participants perform more successfully or not. Their results showed that, although female participants tinkered less, they performed more effectively than males. Therefore, female developers must be supported and encouraged to tinker more. Finally, they performed the fourth study (Beckwith et al., 2007) in a commercial environment, which also confirmed that self-efficacy impacts the performance of male and female end-user programmers.

In addition, Grigoreanu et al. (2008) designed two new features to support participants with lower self-efficacy and analyzed the results to validate the impact of these new features. In the regular version of the experiment, participants answered multiple-choice questions with two checkmarks: “it’s wrong” and “it’s correct”. In the enhanced version, they added two more extra checkmarks including “seems right maybe” and “seems wrong maybe”. Moreover, the strategy explanations were also provided in two formats: video snippet and hypertext. The results showed that it is possible to help participants with lower self-efficacy (usually female participants) to work more effectively. The new items encouraged female developers to use new features more often and boost their level of self-efficacy. Moreover, these new features benefited female developers without an accompanying detriment to male developers’ performance.

McDowell et al. (2003) observed 24% increase in confidence when female developers work in pair. For male developers, pair programming leads to 15% confidence boost.

Problem Solving Strategies

Different studies investigated the problem solving strategies deployed by male and female participants to solve a problem.

Grigoreanu et al. (2009) performed a think-aloud study to investigate scripting strategies and explain how male and female participants differently used these strategies. Male participants used testing for all stages of debugging, including finding and fixing the bug and evaluating their fix. However, female participants used testing only for finding bugs. In addition, offering code inspection strategies in a testing environment improved female developers' success. For male developers, however, environments offering data flow and incremental testing strategies separately increased males' success in debugging.

Subrahmaniyan et al. (2008) investigated the debugging strategies used by end-user programmers. They identified eight different strategies and reported that dataflow and testing are useful strategies for male testers while code inspection and specification checking worked well for female testers.

Fisher et al. (2006) conducted a study to compare male and female participants' spatial cognition and source code navigation. Their results indicated that there is no differences between male and female participants with respect to mental rotation, object and location memory. They explained that male and female participants improve their skills and adapt them to be professional developers. Therefore, they are equivalent in these skills while working as a software developer. However, they concluded that female developers mostly used a bottom-up approach while male developers preferred to adopt a top-down approach.

Moreover, gender differences have been reported in problem solving strategies deployed for solving mathematical problems (Fennema et al., 1998; Gallagher and De Lisi, 1994), spatial way finding strategies (Lawton, 1994), and financial decision making strategies (Powell and Ansic, 1997).

Fennema et al. (1998) performed a study of gender differences in problem solving strategies used by young boys and girls in grades 1 to 3. They reported that girls tend to use more concrete strategies, such as modelling and counting, while boys use more abstract strategies. In addition, they also reported that, when flexibility is required to solve a problem, boys are more successful.

Selectivity Hypothesis (Information Processing Strategies)

Information processing style is related to finding and processing new information to tackle the problem. The research in selectivity hypothesis (Meyers-Levy, 1989; O'Donnell and Johnson,

2001) proposed that females pay more attention to details and use disparate, multiple cues for processing information in both simple and complex tasks. However, males tend to find the first cue and follow it to solve a problem. Males also use complex, comprehensive strategies only for complex problems.

O'Donnell and Johnson (2001) showed that male and female auditors use different strategies completing a planning analytical procedures task.

5.2.2 Impact of Identifiers on Program Comprehension

Lawrie et al. (2007) performed an experiment to investigate the impact of identifier length on source code understanding. They reported that full-word identifiers lead to better understanding in comparison with short identifiers.

Binkley et al. (2009) analyzed the impact of identifier style on the speed and accuracy of source code modification tasks. Results implied that CC identifiers lead to higher accuracy but they take more time in comparison with US.

Sharif and Maletic (2010c) carried on an eye-tracking experiment to analyze the effect of identifier style. Their results showed that there is no significant difference between CC and US styles with respect to the participants' accuracy. Moreover, using US style led to have higher efficiency and lower visual effort in comparison with CC. They used eye-tracking data to calculate the visual effort.

Binkley et al. (2013) presented a family of studies investigating the impact of identifer styles (CC vs. US) on developers' speed and accuracy while conducting comprehension task. First study focused on readability of identifiers while the rest addressed semantic implications of identifier styles. Their results showed that while working with source code, identifier style does not have any impacts on speed and accuracy of experts. However, they found interaction between novices' accuracy and time, and style.

All of these previous studies presented contradictory results. Binkley et al. (2013) also reported that the results of their four studies are not in agreement with each other and emphasized that understanding the comprehension process is a very complex task. These findings encouraged us to consider gender as a factor that can impact the comprehension process and perform a set of empirical studies to analyze the impact of gender on program comprehension.

5.3 Study Design

The *goal* of our study is to investigate the relations between gender and participants' visual effort, task time, as well as ability to recall identifiers in source code reading. The *quality focus* is identifiers memorability and thus program comprehension effort, which may depend on gender. The *perspective* is that of developers, who perform development or maintenance activities and need to understand a code fragment. It is also that of researchers to possibly find systematic bias that can be considered in the future empirical studies involving male and female participants. The researchers could also use our findings to design methods, techniques, and tools better adapted to different developers or support different code reading and program understanding strategies. The *context* of this study consists of three program comprehension tasks involving 25 participants (nine female participants) and two variants of three Java code fragments where identifiers have been coded following the CC style (first treatment) and the US style (second treatment). The experiment is conducted as not within-participants design. An overview of our experiments is outlined in Table 5.1.

5.3.1 Research Questions

In this Chapter, we are interested in the following research questions:

RQ5.1: Does the identifier style impacts the effort, the task time, and as well as their ability to recall identifiers (accuracy) in source code reading?

RQ5.2: Does developers' gender impact the effort, the task time, and as well as their ability to recall identifiers (accuracy) in source code reading?

5.3.2 Research Hypotheses

We formulate **RQ5.1** null hypotheses as follows:

– Hα_{11}: The amount of effort is the same regardless of identifier styles for recall tasks.
– Hα_{12}: The time is the same regardless of identifier styles for recall tasks.
– Hα_{13}: The accuracy is the same regardless of identifier styles for recall tasks.

Table 5.1 Overview of the eye-tracking experiment.

	Experiment
Goal	Study the impact of the gender, CC and US styles on source code reading.
Independent variables	Gender: male or female and Identifier style: CC or US)
Dependent variables	Accuracy, Task time (Speed), and Visual Effort.
Mitigating variables	Study level and Style preference of participants.

The null hypotheses for **RQ5.2** are presented as follows:

– Hα_{21}: There is no significant difference in the average accuracy between male and female participants while reading the source code to recall identifiers.

– Hα_{22}: There is no significant difference in the amount of task time (speed) between male and female participants while reading the source code to recall identifiers.

– Hα_{23}: There is no significant difference in the average visual effort between male and female participants while reading the source code to recall identifiers.

5.3.3 Data Collection

We use two methods to collect data during the experiment:

1. We use two questionnaire (pre-experiment and post-experiment) to obtain relevant information. The pre-experiment questionnaire is used to collect information about participants' level of study and their self assessment of Java and Eclipse knowledge, and English proficiency. Participants are also asked to write down their preferred style if that have one. In the post experiment questionnaire, we ask our participants for their comments regarding the experiment.
2. We perform a controlled experiment using a video-based, non-intrusive eye-tracker, FaceLAB (See Section 2.1.2) to capture eye-movements data.

5.3.4 Task

To design our experiments, we assign a set of three comprehension and recall tasks to our participants to perform. We assign randomly each task to one of the two treatments (CC or US) in a way to avoid learning and hopefully maximize the possibility to observe a gender related difference. We randomly assign our participants to six groups presented in Table 5.2. Each participant works on four different sessions. In each session, the participant reads one source code that is presented by either camel case or underscore styles.

5.3.5 Material

Stimuli

We have two stimuli: the source code stimulus and the question stimulus. Source code stimulus contains a Java program. In overall, we use three small Java programs: a `2D graphical frame`, a `Database tester`, and a `Prime number calculator`. We find these small programs in Java Source Code Example webpage[1]. We adapt to the two identifier styles: CC and US, the two treatments. The lengths of the programs (in lines of code) are 30, 36, and 44, respectively.

We must use small Java programs to accurately and unambiguously quantify the visual effort. However, the source code size is similar to previous eye-tracking studies (Sharif et al., 2012b; Uwano et al., 2006b). Indeed, we can display a small Java program on a single screen and thus, for reading source codes, the participants do not need to scroll down or traverse different pages. If the participants had to scroll or traverse pages, it would have been difficult and error-prone to analyze the eye-trackers' data, especially if the participants go back and forth between different pages.

Our experiment focuses on identifiers. We remove any comments from the source code. Thus, all the information about the source code is captured by its identifiers. We use two variants of the three programs: one in which all identifiers are written in CC style and another in which all identifiers follow the US style. We choose each identifier to contain only two or three terms. Table 5.3 shows the list of all identifiers used in our experiment for the CC program variants. Figure 5.1 shows an example of source code and question.

Questions

5.3.6 Study Variables

Independent Variables

The participants' gender (male or female) is the main independent variable along with the style of the identifiers (CC or US) as summarized by Table 5.1.

[1] http://www.javadb.com/

Table 5.2 Design groups for assigning source codes to participants.

Group 1	Session 1: Code 1 in camel case Session 2: Code 2 in camel case Session 3: Code 3 in underscore
Group 2	Session 1: Code 1 in camel case Session 2: Code 2 in underscore Session 3: Code 3 in camel case
Group 3	Session 1: Code 1 in camel case Session 2: Code 2 in underscore Session 3: Code 4 in underscore
Group 4	Session 1: Code 1 in underscore Session 2: Code 3 in camel case Session 3: Code 4 in camel case
Group 5	Session 1: Code 1 in underscore Session 2: Code 2 in camel case Session 3: Code 4 in underscore
Group 6	Session 1: Code 1 in underscore Session 2: Code 2 in underscore Session 3: Code 3 in camel case

Dependent Variables

The dependent variables are chosen as follows:

Accuracy: we quantify and measure this variable by the percentage of correct answers given by a participant in the multiple choice questions.

Task Time: we measure this variable as the amount of time that each participant spends on the source code and question stimuli. We measure this variable using the eye-tracker with each participant.

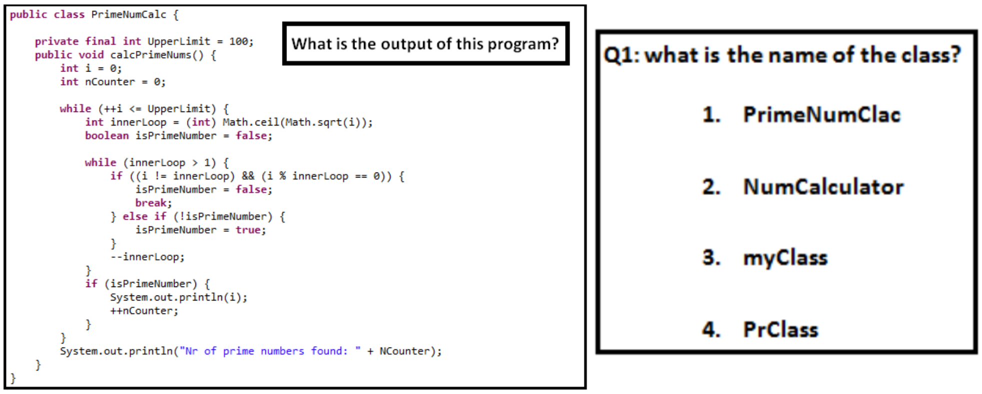
Effort: we measure effort using the eye-tracker data. We consider effort as the amount of visual attention that participants must spend to answer the question: less attention and less time means less effort.

In our experiment, we have two stimuli: the source code stimulus and the question stimulus that we differentiate. Thus, we use two different sets of metrics for effort calculation.

1. Source code stimulus: we calculate the convex hull of the fixations to compute the visual effort. A convex hull represents the smallest convex sets of fixations that contains all of a participant's fixations. In Figure 5.2, we show the source code stimulus with the convex hull shown by red lines and the fixations shown by black dots.

Table 5.3 List of identifiers used in the three Java programs.

	Code 1	Code 2	Code 3
Class name	Java2DFrame	DBTest	PrimeNumCalc
Method names	paint2DObjects drawString drawLine initComponents	executeQuery closeConnection	calcPrimeNums
Variables' names	graphics2D roundRectangle sampleLine	dBConnection dBStatement dBResultSet dBDriver dBQuery	upperLimit nCounter innerLoop isPrimeNum



```
public class PrimeNumCalc {

    private final int UpperLimit = 100;
    public void calcPrimeNums() {
        int i = 0;
        int nCounter = 0;

        while (++i <= UpperLimit) {
            int innerLoop = (int) Math.ceil(Math.sqrt(i));
            boolean isPrimeNumber = false;

            while (innerLoop > 1) {
                if ((i != innerLoop) && (i % innerLoop == 0)) {
                    isPrimeNumber = false;
                    break;
                } else if (!isPrimeNumber) {
                    isPrimeNumber = true;
                }
                --innerLoop;
            }
            if (isPrimeNumber) {
                System.out.println(i);
                ++nCounter;
            }
        }
        System.out.println("Nr of prime numbers found: " + NCounter);
    }
}
```

What is the output of this program?

Q1: what is the name of the class?

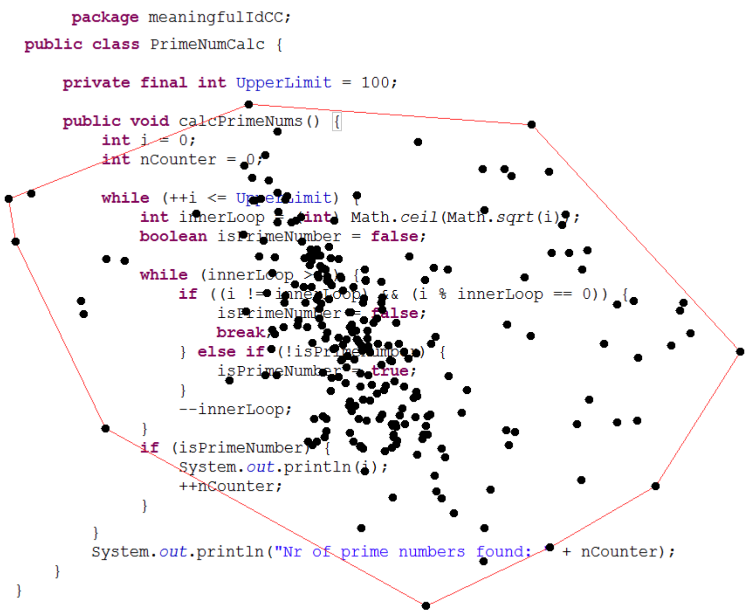
1. PrimeNumClac
2. NumCalculator
3. myClass
4. PrClass

Figure 5.1 (Left) source code stimulus; (right) question stimulus.

2. Question stimulus: each question stimulus contains a multiple choice question. We collect data about fixations on the set of areas of interest (AOI) in the screen to compute the participants' effort. An area of interest is a relevant element of the stimulus. We establish four AOI for the question stimulus as illustrated in Figure 5.3.
   - Entire stimulus: the question description and the four multiple choices.
   - Question: the question displayed at the top of the stimulus.
   - Correct answer: the choice that represents the correct answer.
   - Distractors: the three other incorrect choices corresponding to the irrelevant AOIs.

We compute the visual effort using Fixation Count (FC) and Fixation Rate (FR) as explained in Section 2.1.4. Higher number of fixations indicates more effort to answer a question.

- $FC(Q)$: the total number of fixations on question AOI.
- $FR(correct)$: the total number of fixations on the correct answer with respect to all four



```
     package meaningfulIdCC;
public class PrimeNumCalc {

    private final int UpperLimit = 100;

    public void calcPrimeNums() {
        int i = 0;
        int nCounter = 0;

        while (++i <= UpperLimit) {
            int innerLoop = (int) Math.ceil(Math.sqrt(i));
            boolean isPrimeNumber = false;

            while (innerLoop > 1) {
                if ((i != innerLoop) && (i % innerLoop == 0)) {
                    isPrimeNumber = false;
                    break;
                } else if (!isPrimeNumber) {
                    isPrimeNumber = true;
                }
                --innerLoop;
            }
            if (isPrimeNumber) {
                System.out.println(i);
                ++nCounter;
            }
        }
        System.out.println("Nr of prime numbers found: " + nCounter);
    }
}
```

Figure 5.2 Source code stimulus that contains a convex hull (See Section 2.1.4). The convex hull is shown by red lines with the fixations represented by black dots.

choices on the question stimulus.

- $FR(distractors)$: the total number of fixations on the incorrect choices with respect to all four choices on the question stimulus.

Mitigating Variables

In this experiment, we used a questionnaire to collect our two mitigating variables:

- Level of Study: values for this variable are B.Sc., M.Sc., and Ph.D.
- Style preferences: the values for this variable can be CC, US or None.
- Java knowledge: the value for this variable can be very good, good, satisfactory, and poor.

5.3.7 Participants' Demographics

The study participants are 24 volunteers, nine (46%) female participants and 15 (54%) male participants. The participants are two B.Sc., 11 M.Sc., and 15 Ph.D. students from computer and software engineering and Computer science departments of the Montreal area. The participant demographics are presented in Table 5.4.

Figure 5.4 displays participants' self reported level of experience in Java programming in general. 92% of our participants have satisfactory, good, or very good level of experience in

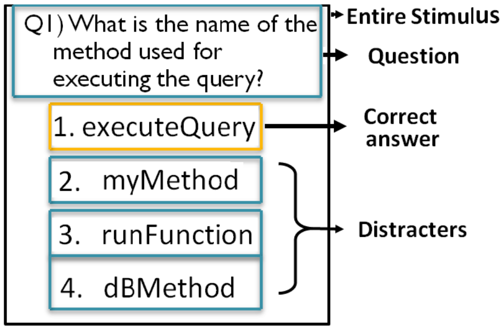


Figure 5.3 Question stimulus that contains four areas of interest: 1) entire stimulus, 2) question, 3) correct answer, 4) distractors.

Java programming.

We asked participants about their style preference. The 29% has no preferences (3 female participants and 4 male participants) while the other 71% preferred CC. We asked participants about their style preference. The 29% has no preferences (3 female participants and 4 male participants)while the other 71% preferred CC.

5.3.8 Procedure

We conduct the experiment in a quiet, small room where the eye-tracker is installed. We use a 27” LCD screen to show the stimuli while the participants were seated approximately 70 cm away from the screen in a comfortable chair with arms and head rests. Before running the experiment, we briefly give a tutorial to explain the procedure of the experiment and the eye-tracker (*e.g.,* how it works and what information is gathered by the eye-tracker).

We also explain that the experiment consists of three pieces of source code and that participants must answer five questions for each piece. We provide no explanation to participants on the particular goal of the experiment. We also ask the participants to fill a pre-experiment questionnaire to gather their basic information, such as gender and the level of study.

We ask each participant to read three different pieces of source code and recall the name of identifiers. We assign randomly each task to one of the two treatments (CC or US) in a way to avoid learning and hopefully maximize the possibility to observe a gender related difference. For each participant, we first calibrate the eye-tracker. Then, we present the first screen, which describes to the participant how to perform the tasks and complete the experiment. When participants begin a task, we start collecting data. No time limit is set but we asked participants to answer the questions as soon as possible. The source code of each Java program is displayed. When participants finish reading the source code, they press

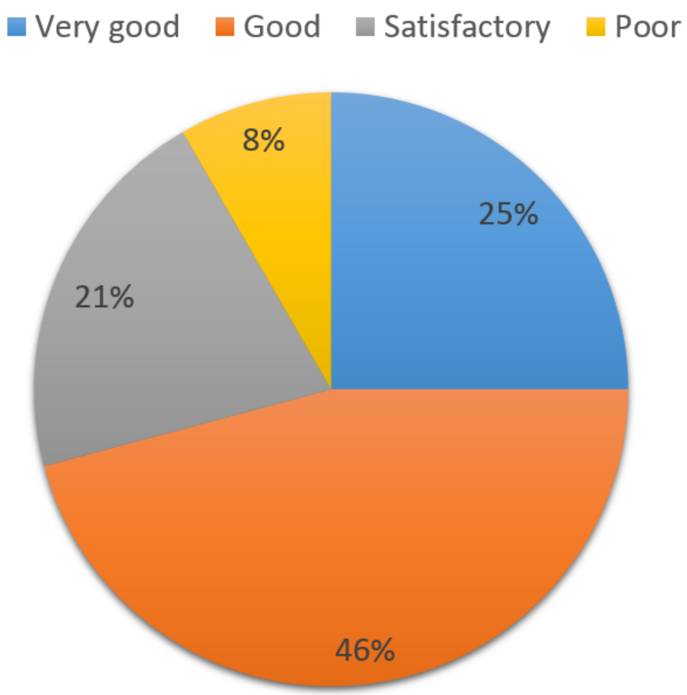


Figure 5.4 Participants' self assessment of Java knowledge.

Table 5.4 Participants' demographics

Participants' Demographics				
Academic Background			Gender	
Ph.D.	M.Sc.	B.Sc	Male	Female
11	10	3	15	9

the "space" key to go to the next screen which contains one question and four choices, press space again, which displays a blank screen, and write down their answer to the question. To answer the question, *i.e.,* choose one of the four alternatives, participants must recall the correct name of the identifier that performs a specific task in the program. Once a task finished and the answer given, participants press the "space" key to go to the next Java program.

When participants complete the three tasks, we ask them to answer the post-experiment questionnaire. The eye-tracking experiment took 25 minutes in average to complete.

5.4 Study Results

In this section, we report hypothesis testing and discuss the results of our experiment. Figure 5.5 presents the workflow of our data analysis.

1. We use *FaceLAB* to capture the eye-movement data of the participants and export them to text files (.out files). Table 5.5 summarises the collected data.
2. We use *Taupe* (De Smet et al., 2012) to parse .out files, to calculate the data, and to generate the following outputs:
   (a) CVS files that contain the following information for each participant per stimulus and per AOI: 1) the amount of task time, 2) the number and the duration of fixations, 3) the amount of visual effort (AFD), and 4) the area of convex-hull.
   (b) Visualization of heatmaps, convex hulls, and scan-paths for each participant per stimulus.
3. We provide four different files (CVS files of time and effort, Excel files of accuracy, and user's opinion) to *R* (Team et al., 2010) to perform statistical analysis. In the analysis of our data, we made no assumption and applied non-parametric, non-paired tests to determine significance differences.

5.4.1 RQ5.1 Does the Identifier Style Impact the Effort, the Time, and the Accuracy in Source Code Reading?

Indeed, when we consider all participants as part of the same population, we cannot reject the null hypothesis that CC and US correct answers have the same proportion. In addition, we found no significant differences between the amount of time and effort that our participants put forth while comparing camel case with underscore.

5.4.2 RQ5.2: Does Developers' gender Impact the Effort, the Time, and Accuracy in Source Code Reading?

Table 5.5 summarizes the collected data. Each participant, when answering a question, gave either a correct or a wrong answer. Table 5.6 is the contingency table reporting for the two populations the numbers of correct and wrong answers for both CC and US identifiers. We use this data to perform a proportion test, to test whether the proportion of correct (wrong) answers in the overall population as well as in the two sub-populations (males and females) for the two treatments, CC and US, are the same.

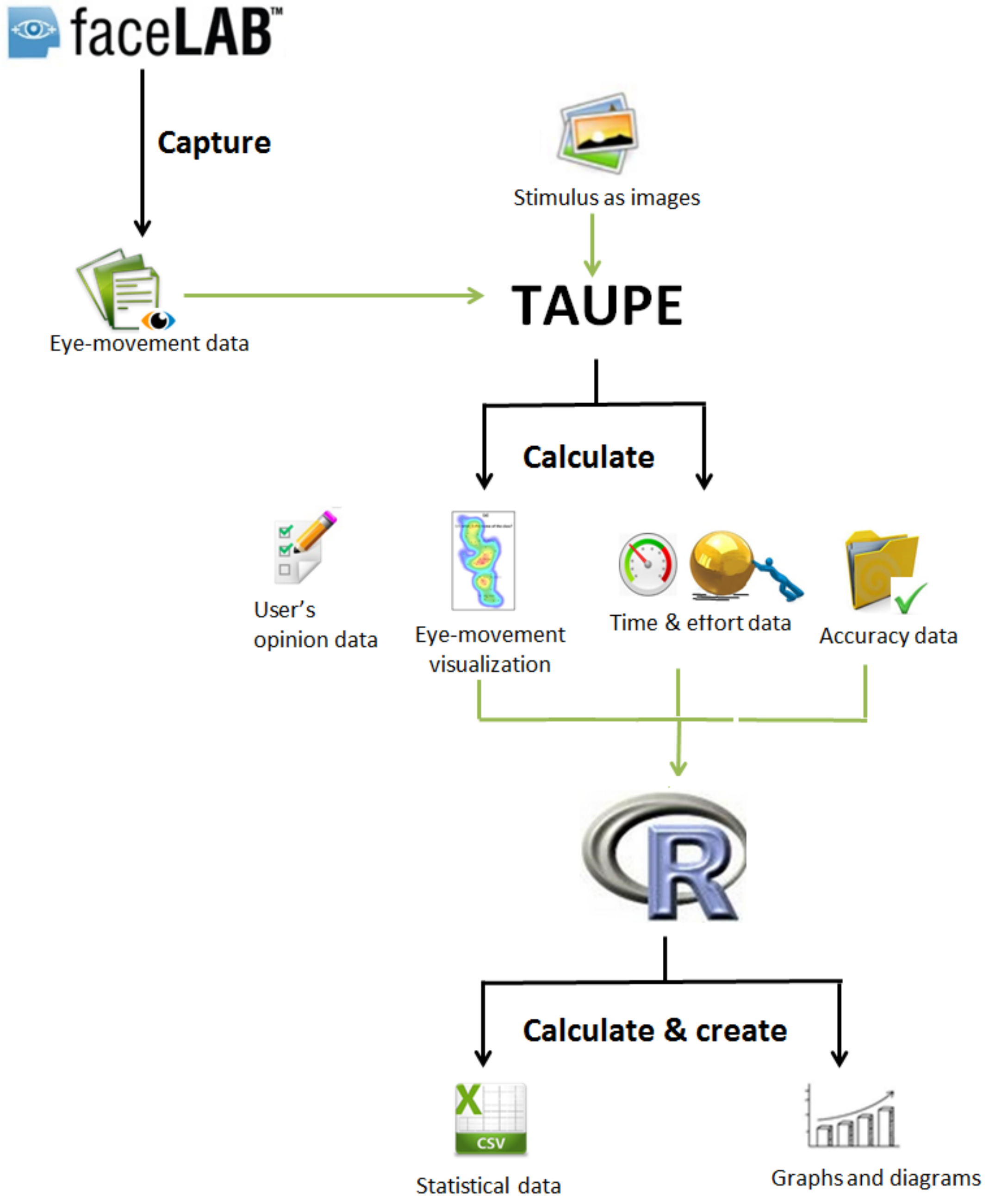


Figure 5.5 Workflow of data analysis.

Table 5.5 Main features of the experiment.

Number of participants (#)	24
Number of female participants	9
Number of male participants	15
Number of CC-related questions	147
Number of US-related questions	200
Total time of eye-tracking (hours)	11
Total number of fixations (#)	28,881

Table 5.6 Participants' contingency table for CC and US identifiers.

	Answers			
	Female		Male	
	Correct	Wrong	Correct	Wrong
CC	52	13	75	15
US	58	10	91	41

If we limit ourselves to the male population, we clearly see that male participants prefer CC style vs. US style identifiers as they (roughly) gave three time more wrong answers when the code was written with US identifiers. This observation points to a lacking of training on US style coding style, *i.e.,* though male and female participants have about the same training, male participants seem less at ease with US identifiers.

However, a more thorough analysis based on a logistic regression sheds a different light. We use logistic regression analysis as a proxy for correlation when the response is a dichotomous variable as it is in the case of correct or wrong answers. Tables 5.7 and 5.8 report the variables retained by the logistic regression models built on the whole population of participants along with the p-values and the AIC criterion. We observe that gender plays a marginal role in both models and that AIC value are close. We computed a logistic regression using R; the gender was coded as a factor with two levels "GenderMale" and "GenderFemale"; R just reports one of the two levels meaning that the model for a female is obtained by removing the "GenderMale" factor. In other words, in both models, the male coefficient is negative, *i.e.,* GenderMale lowers the probability and thus it impacts negatively the results of the models. Considering the best model using the AIC criterion in Table 5.7, although the time required to answer a question plays a role and as a single coefficient is statistically relevant, this time actually only marginally affects the probability as an increase in a unit of time would only marginally affect the probability because the coefficient is of the order of one over

Table 5.7 Best logistic regression on the experiment population (AIC 370.01)

	Estimate	Std. Error	z value	Pr(> $\|z\|$)
(Intercept)	1.2884	0.2458	5.24	0.0000
Time	8.698e-06	3.459e-06	2.51	0.0119
GenderMale	-0.4631	0.2793	-1.66	0.0973

Table 5.8 Alternative logistic regression on the experiment population (AIC 373.59)

	Estimate	Std. Error	z value	Pr(> $\|z\|$)
(Intercept)	1.3597	0.2433	5.59	0.0000
FR(Correct)	0.7987	0.3551	2.25	0.0245
GenderMale	-0.6086	0.2845	-2.14	0.0324

one million. If we consider the slightly worse model in Table 5.8, we see that the effort spent on the correct answer, $FR(correct)$, increases the probability of a correct answer.

We do not make any claim on the validity of these models but we found their coefficient puzzling and, thus, we decided to study the two different sub-populations of male and female participants independently, with the same type of analysis. We report the results of this analysis in Tables 5.9 and 5.10 for female and male participants, respectively. For both sub-populations, AIC criterion improves (more for the female sub-population than for the male one), supporting the heterogeneity of the population and further justifying the presence of gender in the models in Tables 5.7 and 5.8.

We can consider the two models in Tables 5.9 and 5.10 as derived froms the models in Tables 5.7 and 5.8. Yet, while for female participants, their efforts on the correct answers help to explain the data (see Table 5.9); for the male participants, only time plays a role and its effect is less strong than the coding style. Indeed, US-style has a much stronger effect in the model than time. Table 5.6 reports that male participants gave $75+91 = 166$ correct answers out of $75 + 91 + 15 + 41 = 222$, which means 75% correct answers. Thus, male participants performed better than choosing a random choice because a purely random choice would have given a percentage of correct answers close to 25%.

However, male participants seem not to have invested too much time in pondering their answers. It is possible that they did not really-seriously performed their tasks, as even the US-style coefficient is smaller than the intercept. Yet, we cannot explain how it was possible for male participants to achieve a percentage of correct answers of 75% on US-style identifiers without spending time to consider all the possible answers. We believe that other experiments must investigate this observation and attempt to bring some answers backed with evidences.

Table 5.9 Logistic regression on the female sub-population (AIC 116.83)

	Estimate	Std. Error	z value	Pr(> $\|z\|$)
(Intercept)	0.9707	0.2884	3.37	0.0008
FR(Correct)	3.1580	1.3328	2.37	0.0178

Table 5.10 Logistic regression on the male sub-population (AIC 244.25)

	Estimate	Std. Error	z value	Pr(>|z|)
(Intercept)	1.3637	0.2984	4.57	0.0000
US-style	-0.8522	0.3437	-2.48	0.0131
Time	8.996e-06	4.146e-06	2.17	0.0300

Table 5.6 reports that female participants gave $52 + 58 = 110$ correct answers out of $52 + 58 + 13 + 10 = 133$, which means 83% correct answers. Their errors on US-style identifiers was much lower than that of male participants although male and female participants have comparable curricula. When considering time in Tables 5.7 and 5.10, time plays no role for both male and female participants.

We now model the participants' accuracy, *i.e.,* precision (Baeza-Yates and Ribeiro-Neto, 1999): the numbers of correct answers divided by the numbers of questions answered. In agreement with Table 5.6 and the considerations in the previous subsection, the values reported in Table 5.11 show that female participants performed 8% more accurately than male participants but that they spent 18% more time than male participants.

However, Table 5.12 reports that there is no significant differences between male and female participants for their accuracy and the required time (speed) performing the tasks. Therefore, we cannot reject the null hypotheses $H\alpha_{01}$ and $H\alpha_{02}$.

The values in Table 5.11 also mean that gender, for the current tasks and population, does neither significantly affect the participants' accuracy nor required time by the participants performing the tasks. These figures are also supported by the male and female participants' Required time distributions not reported here for lack of space and because these distributions do not bring any additional insight.

As expected from the previous sub-section, male participants' accuracy are not the same when comparing CC and US identifiers. The Wilcoxon test rejects the null hypothesis that the two sets have the same mean distribution with a confidence level of 0.09 (thus not a 95%-confidence level but a 90% one). We make a similar observation for time: the times spend by male participants with CC and US identifiers are close (but at a 90%-confidence

Table 5.11 Values for percentages of correct answers and required time to compare male and female participants' accuracy and speed.

	Accuracy %	Task time (min)
Male Participants	0.745	5.94
Female Participants	0.827	7.18

Table 5.12 Wilcoxon p-values ($\alpha = 0.05$) for accuracy and required time.

Task Time		Accuracy
Source Code Stimulus	Question Stimulus	
0.2107	0.3472	0.3217

level, p-value: 0.06128). In conclusion, male participants spent more time on US identifiers.

The male and female participants' accuracy is the same with CC style identifiers but not with US style identifiers. The Wilcoxon test rejects the null hypothesis that the two sub-populations performed with the same accuracy with a p-value of 0.05376. Thus, strictly speaking, we cannot rule out the hypothesis at a 95%-confidence level but are quite close.

When considering the tradeoff between speed and accuracy, even if the overall difference is not significant, we observe a trend in the male sub-population as shown in Figure 5.6 while such a trend is not present for female participants. We computed Figure 5.6 as follows: for each participant, we calculate the average total time to perform the experiment and the percentage of their correct answers (accuracy) and we draw a scatter-plot with a super-imposed trend line. In Figure 5.6, the horizontal axis represents the time and the vertical axis represents the accuracy. The trend for male participants is just apparent because for both sub-populations (male and female participants), the variable selection of the linear models only retain the intercepts. Thus, there is no correlation between time (or other measured variables) and accuracy. In conclusion, for male participants, the data shows the apparent trend that they are more accurate if they spend more time, which is in agreement with the previous results of the following studies. Uwano et al. (2006b) observed that the longer a participant reads the code, the more efficiently s/he finds the defects in the source code. Moreover, Sharif et al. (2012b) reported that there is a correlation between scan time and defect detection time. However,our results are not statistically supported by either male or female participants.

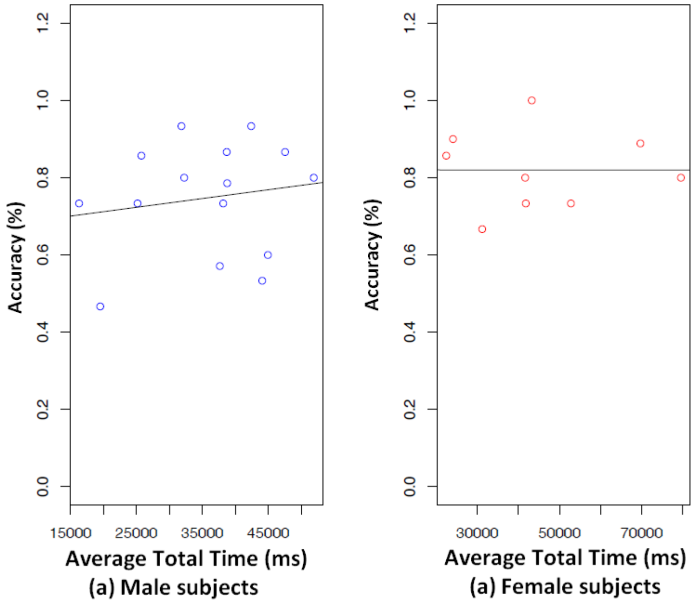


Figure 5.6 Speed-Accuracy tradeoff for male and female participants.

Visual Effort

When considering the entire population, no significative linear model can explain accuracy with respect to visual effort but we can gain insight by considering single variable and sub-populations. Table 5.13 reports the p-values for Wilcoxon tests comparing male versus female participants on the different effort variables. We cannot observe statistical differences for the overall effort spent, $FC(Q)$, and the effort spent on the correct answers, $FR(correct)$. Interestingly, there is a statistically-significant difference for the effort spent on the wrong answers, the distractors, $FR(distractors)$.

Moreover, in Table 5.13, we observe that female participants put more visual attention (effort) on distractors (irrelevant areas of interest) than male participants. We can reject the null hypothesis H_{03}. Our rejection of the null hypothesis is also supported by Figure 5.7, which shows the box-plots of the three independent variables related to effort for male and female participants. Figure 5.8 shows the heatmaps for a male and a female participant for one question. Heatmap is a color spectrum that represents the intensity of fixations. The heatmap for our female participants shows that her fixations are scattered through all choices while for our male participants, his fixations are focused on the correct choice.

We gain interesting insight again as in previous sub-sections when considering the male and female sub-populations independently. While we cannot model the accuracy of the male sub-population with a linear model and the dependent variables, we obtain a model with

Table 5.13 Wilcoxon p-values ($\alpha = 0.05$) for each visual effort measure

Visual Effort Metrics			
FC(Q)	FR(correct)	FR(distractor)	Convex hull
0.7884	0.1737	0.006	0.7685

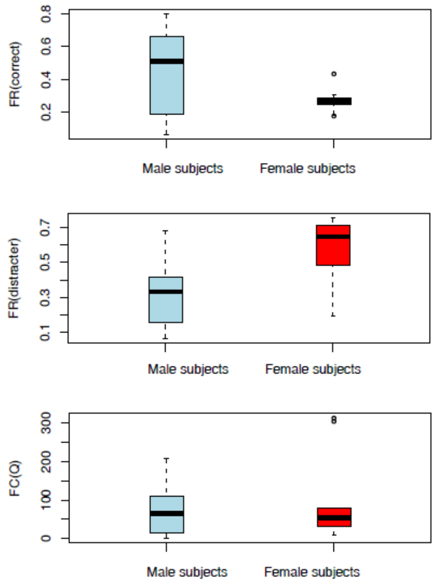


Figure 5.7 Data distribution for visual effort of male and female participants.

interacting variables for the female sub-population, shown in Table 5.15. The model is highly significant, it explains 84% of variability and support the conjecture that it is the complex interplay between the visual effort on correct answers and wrong answers that helps the female participants to obtain a high accuracy. We will investigate the rationale for this observation (and lack thereof for male participants) in future work. In particular, we have only nine female participants and the model is likely over-fitted. Table 5.14 reports a simplified model—not modelling interaction. This model explains less the variance as the adjusted R^2 is 45% compared to the 84% of the model in Table 5.15. Yet, the coefficients of the model in Table 5.14 are expected: the more the female participants concentrate on distractors, the lower their accuracy while the more they concentrate on the correct answers, the better their accuracy.

5.4.3 Impact of the Mitigating Variables

Style preferences: for this variable, we consider three treatments including CC, US, and None. We do not find any interaction between style preference and participants' time, ac-

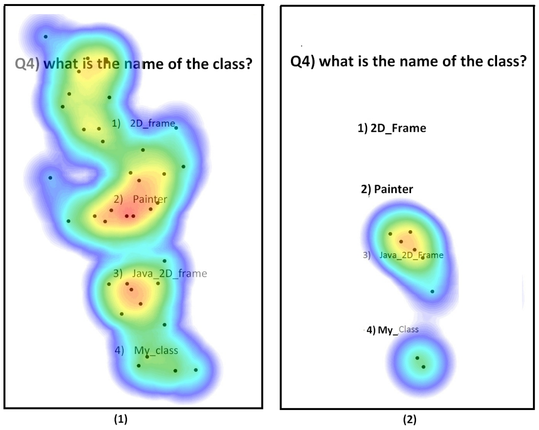


Figure 5.8 Heatmap illustration of 1) a female participant, and 2) a male participant.

Table 5.14 Effort–accuracy model for the female sub-population with no interaction (adjusted R^2 of 0.45)

	Estimate	Std. Error	t value	$\Pr(>\|t\|)$
(Intercept)	0.7970	0.1470	5.42	0.0016
FR(distractor)	-0.5545	0.2124	-2.61	0.0401
FR(correct)	1.1711	0.5174	2.26	0.0642

curacy, and effort. In addition, after applying two-way ANOVA, we have not found any statistically significant interaction on participants' accuracy, time, and effort.

Experience: to evaluate the impact of experience, we consider Java knowledge and level of study. After applying the one-way ANOVA separately for Java knowledge and the level of study, no statistically significant interaction was found. Our participants are students so their Java knowledge is rather homogeneous. We also design our tasks such that solving them does not require industrial experience.

5.5 Threats to Validity

In the following, in addition to those factors that are discussed in Section 2.2.2, we discuss other factors that may have influenced our results.

One threat of validity is the fact that our experiment was designed by a woman and this may bias the experiment towards the female population. We believe this risk is somehow mitigated by the measured variables and the constraint on the code size that must fit into one screen. The risk is also intrinsic to any experiment of the past as we are not aware of experiments designed by mixed teams.

Table 5.15 Effort–accuracy model for the female sub-population (adjusted R^2 of 0.84)

	Estimate	Std. Error	t value	Pr($> \|t\|$)
(Intercept)	1.9259	0.2984	6.45	0.0013
FR(distractor)	-2.8541	0.5969	-4.78	0.0050
FR(correct)	-4.3654	1.4377	-3.04	0.0289
FR(distractor):FR(correct)	10.5043	2.6753	3.93	0.0111

5.6 Discussions and Conclusion

We designed and performed an eye-tracking experiment to investigate the impact of gender on the performance of developers during code reading and program understanding activities. We also examined the effect of identifier style: camel case (CC) vs. underscore (US).

Our findings support the belief that CC and US styles do not impact the developers' effort and program comprehension. We found no significant differences between identifier styles when considering accuracy, effort, and gender. However, a more fine-grain analysis of the developers' visual effort on correct and wrong answers revealed unexpected details. Indeed, the variability of this visual effort is significantly different between male and female developers. While the time spent by male and female developers is not significantly different, male and female developers focused differently on the alternative answers that we proposed to them. Female developers spent more effort on the wrong answers than male developers, which can explain the female developers' higher accuracy. Thus, female developers seem to carefully weight all options and rule out wrong answers while male developers seem to quickly set their minds on some answers, possibly the wrong ones.

First, we found a statistically strong interaction between accuracy, effort spent on distracters, and correct answers when modelling the female developers' accuracy. Second, no correlation exists between visual effort on correct answers or distracters and the male developers' accuracy. Consequently, for female developers, it is not how much visual effort they spend on distracter or on correct answers but rather a mix of the two, *i.e.,* the complex pondering of correct and wrong answers, that describe best their accuracy.

Yet, we must be careful because only nine female developers participated in our study and, although this number is much higher than that of any previously reported studies, we cannot consider the population large enough to generalize. Our findings should be considered more as a hint to suggest further studies concerning the role of gender in software engineering rather than a general truth.

Different studies (see Section 5.2.1) report that females are usually less confident than males while using different programming environments. They report that self-efficacy—a person's confidence about her/his own competence in performing a specific task successfully—impacts the amount of effort that is spent and the strategies that are used to solve a problem. The root cause of differences in males' and females' behaviors and possibly their different abilities are likely to go back to their different roles in society and society evolution (Harris, 1991; Fisher, 1994). Gathering seeds or harvesting require different sets of skills; accuracy may not be so relevant to harvest but picking seeds or identifying a poisonous or non-edible plant has always been critical for pre-industrial societies (Fisher, 1994). Over thousands of years of evolution, social training shaped different skills and abilities in males and females (Harris, 1991; Fisher, 1994).

In our experiment, female developers put more effort reading and analyzing distracters. Consequently, we could explain this observation by the lower level of self-efficacy perceived by the female developers when compared to that of male developers. Female developers wanted to make sure that they find the correct answers precisely so they investigated all choices before making their decisions. In addition, there is no significant difference between the time that female or male developers spent to perform the comprehension tasks. Thus, even though female developers devote visual attention not only to the correct answers but also to distracters, they do not spend more time than male developers. Nevertheless, they are more effective as they obtained a better accuracy and a better precision.

In the following chapter, we study the impact of the location of identifiers (*i.e.,* Source Code Entities (SCE) including: class names, method names, variable names, and comments) on program comprehension. We are interested to analyze how much time male and female developers spend on each SCE and what type of viewing strategies they deploy to read and analyze these SCEs. The results could shed some light on the influence of gender on source code reading and comprehension.

CHAPTER 6 IMPACT OF GENDER ON PROGRAM COMPREHENSION ON SOURCE CODE ENTITIES

In this chapter, we present the results of an empirical study to identify the Source Code Entities (SCEs), including class names, method names, variable names, or comments to which developers pay more attention while reading source code and performing a task. First, we use an eye-tracker to capture developers' eye movements while they read source code to understand it. We analyse the obtained data to identify and rank developers' preferred types of SCEs. Second, we perform another eye-tracking experiment to confirm that it is the semantic content of the developers' preferred types of SCEs and not their locations that attract developers' attention and help them in their task. In addition, we identify and compare the viewing strategies used by male and female developers for program comprehension.

6.1 Introduction

Program comprehension studies consistently report that developers seem to avoid understanding the entire system (Roehm et al., 2012b; Rodeghero et al., 2014b) and would prefer to focus on small sections of code during comprehension (Lakhotia, 1993; LaToza et al., 2006; Rodeghero et al., 2014b). Therefore, to save more time, developers tend to skim source code and focus only on some parts to quickly gain a better understanding of the software system. Busjahn et al. (2011) define keywords, identifiers, numbers, and operators as four parts of interest and show that the major fraction of total reading time is dedicated to identifiers.

In this thesis, we also focus on source code and consider four SCEs including class name, method name, variable name, or comment and report the first analysis of developers' visual attention on SCEs with the focus on gender differences. The purpose of this study is to identify the most preferred type of SCEs from male and female developers' point of view for the purpose of better understanding the program comprehension process in the context of software development.

We perform two eye-tracking experiments. The results of the first eye-tracking experiment show that developers prefer some types of SCEs over others while reading source code to understand it. Our results show that method name is the most preferred SCE while class name is the least preferred one. However, developers could prefer method names and comments because they have been taught in their programming classes that method names are important. Hence, it is possible that their preferences are due only to the presence of these types of SCEs, not to their semantic content. This situation is similar to the situation in which participants trust a piece of information by virtue if it is being shown rather than by its intrinsic value, as it has been shown by Hembrooke et al. (2005). Thus, we perform the second experiment to observe whether facilitating the detection of the most preferred types of SCEs (method names) and the least preferred type of SCEs (class names) impact the participant' efficiency or not.

We also analyse the ways participants switch their focus of attention from one SCE to another to compare viewing strategies with respect to participants gender. We identify AOI based scan-path (See Section 2.1) to reason about male and female developers viewing strategies during comprehension.

Section 6.2 contains an overview of related work. In section 6.3, the experimental design is presented and section 6.4 outlines an overview of the results. The threats to validity are addressed in Section 6.5. We provide discussions and a conclusion in Section 6.6.

6.2 Related Work

Some researchers (Kowalski, 2010; Erol et al., 2006; Sun et al., 2004) observed that if a term appears in different zones of a document, then its importance changes. This idea that a term has different importance to a reader depending on where it appears has been investigated in the domain of information retrieval (Kowalski, 2010).

Search engines, such as Google, assign higher ranks to the Web pages that contain the searched terms in specific parts of the pages, *e.g.,* their titles.

Erol et al. (2006) used a questionnaire to ask participants which parts of some documents are more important for performing tasks. They concluded that title, figure, and abstract are the most important parts for both searching and understanding documents while figure caption is only important for understanding.

Busjahn et al. (2011) performed an experiment to investigate the differences between source code reading and natural text reading. They reported that developers spend more fixation time and have higher regression rate (backward directed eye movements) when reading source code compared to natural text. Moreover, developers spent significantly more time reading identifiers in the source code compared to keywords, numbers, and operands.

Thus, we conclude that physically dividing documents into zones, *e.g.,* title and abstract, has been already investigated in the field of information retrieval. However, we report the first analysis of developers' visual attention on SCEs using eye-tracking with regards to gender differences. In particular, we believe that people's preferences for documents may differ from developers' preferences for source code.

6.3 Study Design

Our goal is to observe precisely which types of SCEs receive more developers' visual attention during program comprehension. The *quality focus* is the importance of SCEs and thus program comprehension effort, which may depend on gender. The *perspective* is that of developers, who perform development or maintenance activities and need to understand a code fragment. It is also that of researchers to possibly find systematic bias that can be considered in the future empirical studies involving male and female participants. The researchers could also use our findings to design methods, techniques, and tools better adapted to different developers or support different code reading and program understanding strategies. The *context* of this study consists of six program comprehension tasks involving 24 participants (seven female participants) reading source codes to understand it. The experiment is conducted as not within-participants design. An overview of our experiment is outlined in Table 6.1.

6.3.1 Research Questions

We are interested in the following research questions:

RQ6.1: What are the important source code entities (SCEs) to which developers pay more visual attention when reading source code?

RQ6.2: Does developers' gender impact the viewing strategies of developers while reading source code?

RQ6.3: Are some types of SCEs preferred by developers because of their semantic content or because of their very presence in source code?

6.3.2 Research Hypotheses

We formulate **RQ6.1** null hypotheses as follows:

- Hα_{11}: All SCEs have equal importance for developers.
- Hα_{12}: There is no significant difference in the average time spend on SCEs when performing program comprehension task.

RQ6.2 null hypotheses are defined as follows:
- Hα_{21}: There is no difference between viewing strategies of male and female developers.

We formulate **RQ6.3** null hypotheses as follows:

Table 6.1 An overview of the eye-tracking experiment.

	Experiment
Goal	To observe which types of SCEs receive more developers' visual attention during program comprehension.
Independent variables	Type of SCE: Class name, Method name, Variable name, and Comment.
Dependent variables	Accuracy, and Average fixation time.
Mitigating variables	Study level and Programming experience.

– Hα_{31}: There is no significant differences between the accuracy of developers answering comprehension questions with source code in which a method name is easier to detect compared to the rest of code.

– Hα_{32}: There is no significant differences between the time spent by developers answering comprehension questions with source code in which a method name is easier to detect compared to the rest of code.

– Hα_{33}: There is no significant differences between the effort of developers answering comprehension questions with source code in which a method name is easier to detect compared to the rest of code.

We have similar null hypotheses pertaining to class names, the least preferred type of SCEs.

6.3.3 Data Collection

We use two methods to collect data during the experiment:

1. We use two questionnaire (pre-experiment and post-experiment) to obtain relevant information. The pre-experiment questionnaire is used to collect information about participants' level of study, number of years of programming in general and Java programming in particular, and their self assessment English proficiency. Participants are also asked to write down their native language. In the post experiment questionnaire, we ask our participants to rank the SCEs based on their importance. Moreover, we collect their opinion about the readability and understandability of source codes.

2. We perform an experiment using a video-based, non-intrusive eye-tracker, FaceLAB (See Section 2.1.2) to capture eye-movements data.

Table 6.2 List of questions answered by participants to perform program comprehension task.

	Questions
1	Does this class take as an input the radius of a circle and calculates its area?
2	Does this class use a JDBC driver to connect to a database and reads data from myTable?
3	Does this class draw a circle in using JFrame?
4	Does this class use the Java 2D API to draw lines and one rectangle using JFrame?
5	Does this class compare two strings?
6	Does this class use a JDBC driver to connect to a database and inserts data in myTable?

6.3.4 Task

To design our experiment, we assign a set of six comprehension tasks to our participants to perform. All participants work on all 6 source codes by reading them and answer a comprehension question. The order of presenting these codes are the same for all of the participants.

6.3.5 Material

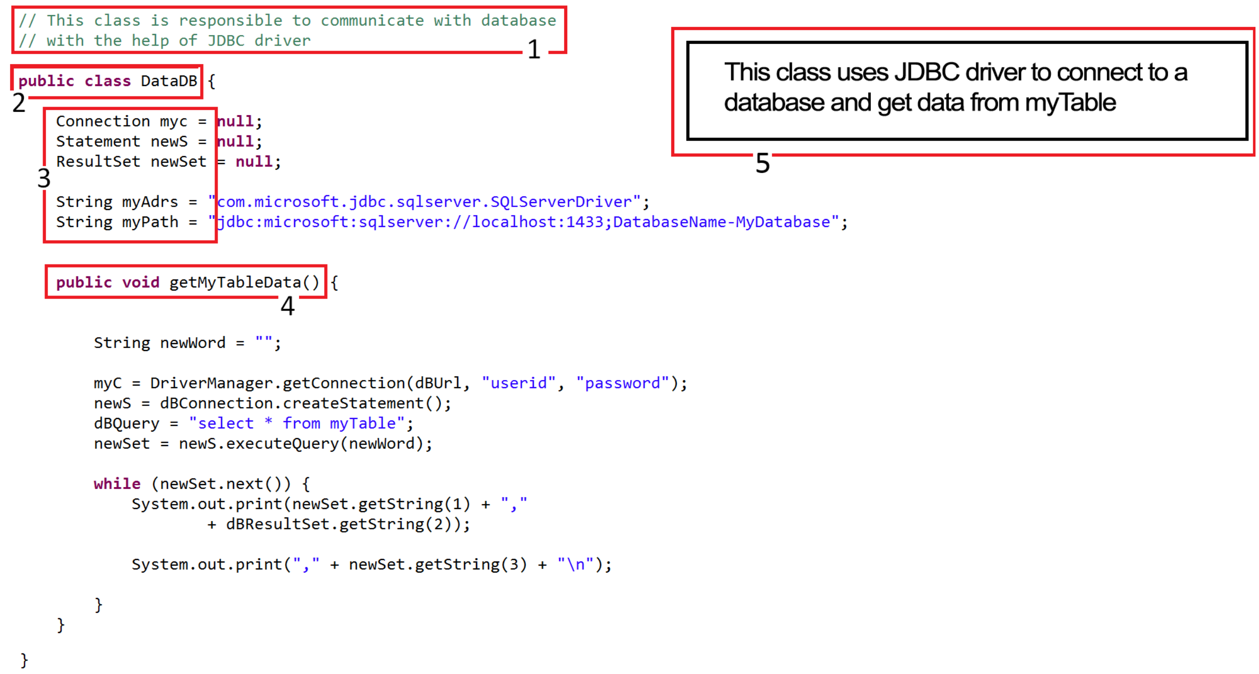
We have one stimulus which contains the source code and the questions. Source code is a Java program. In overall, we use six small Java programs: a `2D graphical frame - Draw circle`, a `2D graphical frame - Draw rectangle`, `2D graphical frame - Draw line and rectabgle`, a `Database tester`, and a `String comparator`. We find these small programs in the Java Source Code Example Web page [1]. The six pieces of source code have 19, 18, 19, 18, 24, and 28 lines of code. Participants answered 6 comprehension questions, one for each source code. Table 6.2 shows the list of questions.

We must use small Java programs to accurately and unambiguously quantify the visual effort. We use the Java programming language to perform our experiment because it is well known by our participants. In addition, Java is one of the (many) object-oriented programming languages that contains several different types of SCEs, *i.e.,* class names, method names, variable names, and comments.

The font size is 20. We modify the code to remove any automatically generated source code comments and empty lines are used to distinguish between two types of SCEs.

The stimulus along with the five SCEs (AOIs) is presented in Figure 6.1.

[1] http://www.javadb.com/



```
// This class is responsible to communicate with database
// with the help of JDBC driver

public class DataDB {

    Connection myc = null;
    Statement newS = null;
    ResultSet newSet = null;

    String myAdrs = "com.microsoft.jdbc.sqlserver.SQLServerDriver";
    String myPath = "jdbc:microsoft:sqlserver://localhost:1433;DatabaseName-MyDatabase";

    public void getMyTableData() {

        String newWord = "";

        myC = DriverManager.getConnection(dBUrl, "userid", "password");
        newS = dBConnection.createStatement();
        dBQuery = "select * from myTable";
        newSet = newS.executeQuery(newWord);

        while (newSet.next()) {
            System.out.print(newSet.getString(1) + ","
                    + dBResultSet.getString(2));

            System.out.print("," + newSet.getString(3) + "\n");

        }
    }
}
```

This class uses JDBC driver to connect to a database and get data from myTable

Figure 6.1 Stimulus that contains five AOIs: 1) comment, 2) class name, 3) variables, 4) method name, and 5) question.

6.3.6 Study Variables

We have one independent variable: the SCEs (class name, method name, variable name, and comment). The dependent variables are chosen as follows:

– Accuracy: we quantify and measure this variable as the percentage of correct answers given by a participant while reading source code and answer comprehension questions.

– Average fixation time: we measure this variable as the amount of fixation time that each participant spends on each type of SCEs. To compute the average fixation time, we collect data about fixations on each AOI for each piece of source code shown to the participants and calculate time by adding the duration of all fixations for that AOI.

– Mitigating variables may impact the effect of the independent variables on the dependent variables. In this first experiment, we use a questionnaire to collect data about about the following mitigating variables: (1) level of study (B.Sc., M.Sc., or Ph.D.); (2) number of years of programming experience in Java in academia; and, (3) number of years of programming experience in Java in industry.

6.3.7 Participants' Demographics

Out of 26 participants, there were four M.Sc., and 22 Ph.D students who enrolled in graduate programs in the departments of computer science and software engineering at École Polytechnique de Montréal and Université de Montréal. Participants' years of general and Java programming experience are presented in Table 6.3.

There was only one participant for whom we could not have eye-tracking data because of the participant's eyeglasses. Thus, we excluded that participant. We also excluded the participant from a pilot study to validate that the requirements used in the experiment were clear and simple and the piece of source code on the screen was easy to read and understand. We analysed the data of a total of 24 participants.

The participants were volunteers and they have guaranteed anonymity and all data have been gathered anonymously. The participants could leave the experiment at any time, for any reason, without any kind of penalty. The participant demographics are presented in Table 6.4.

6.3.8 Procedure

The procedure that we followed in our study has mainly four steps:

1. We provide a single-page guideline to the participants to perform the experiment.
2. We ask participants to answer the pre-experiment questionnaire to collect data related to the mitigating variables. Before running the experiment, we briefly give a tutorial to explain the procedure of the experiment and the eye-tracker (e.g., how it works and what type of information is gathered by an eye-tracker).

   The participants are seated approximately 70cm away from the screen in a comfortable chair with arm and head rests and the eye-tracker is calibrated. This process takes less than five minutes to be completed.
3. We ask participants to read the source code and answer the questions. We instruct participants that they can press space bar key when they find the answer. Pressing the space bar key takes the participants to a blank screen so that they can write down their answer on the answer sheet.

   For each question, we ask participants to spend adequate time to explore the code while we capture participants' eye movements during traceability verification process. The order of the questions was the same for all the participants.
4. We ask participants to provide feedback about source code readability and understand-

Table 6.3 Participants' years of general and Java programming experience.

	Average	Max	Min
Programming Industry	2.3	5	1
Programming Academy	3.9	7	1
Java Industry	2	4	1
Java Academy	3	7	1

Table 6.4 Participants' demographics.

Participants' Demographics				
Academic Background			Gender	
Ph.D.	M.Sc.	B.Sc	Male	Female
20	4	0	17	7

ability. We also ask participants about SCEs that they prefer to analyze during comprehension task. We have asked this question to analyse whether or not participants followed the same pattern in the eye-tracking experiment.

6.4 Analysis and Results

In this section, we report hypothesis testing and discuss the results of our experiment. The workflow of our data analysis is presented as follows:

1. We use *FaceLAB* to capture the eye-movement data of the participants and export them to text files (.out files).
2. We use *Taupe* (De Smet et al., 2012) to parse .out files, calculate the data, and generate the following outputs:
   (a) CVS files that contain the following information for each participant per stimulus and per AOI: 1) the amount of spent time, 2) the number and the duration of fixations for each AOI (SCEs for this experiment).
   (b) Visualization of heatmaps and scan-paths for each participant per stimulus.
3. We provide four different files (CVS files of total time and time on each SCE, and Excel files of accuracy) to *R* (Team et al., 2010) to perform statistical analysis. In the analysis of our data, we made no assumption and applied non-parametric, non-paired tests to determine significance differences.

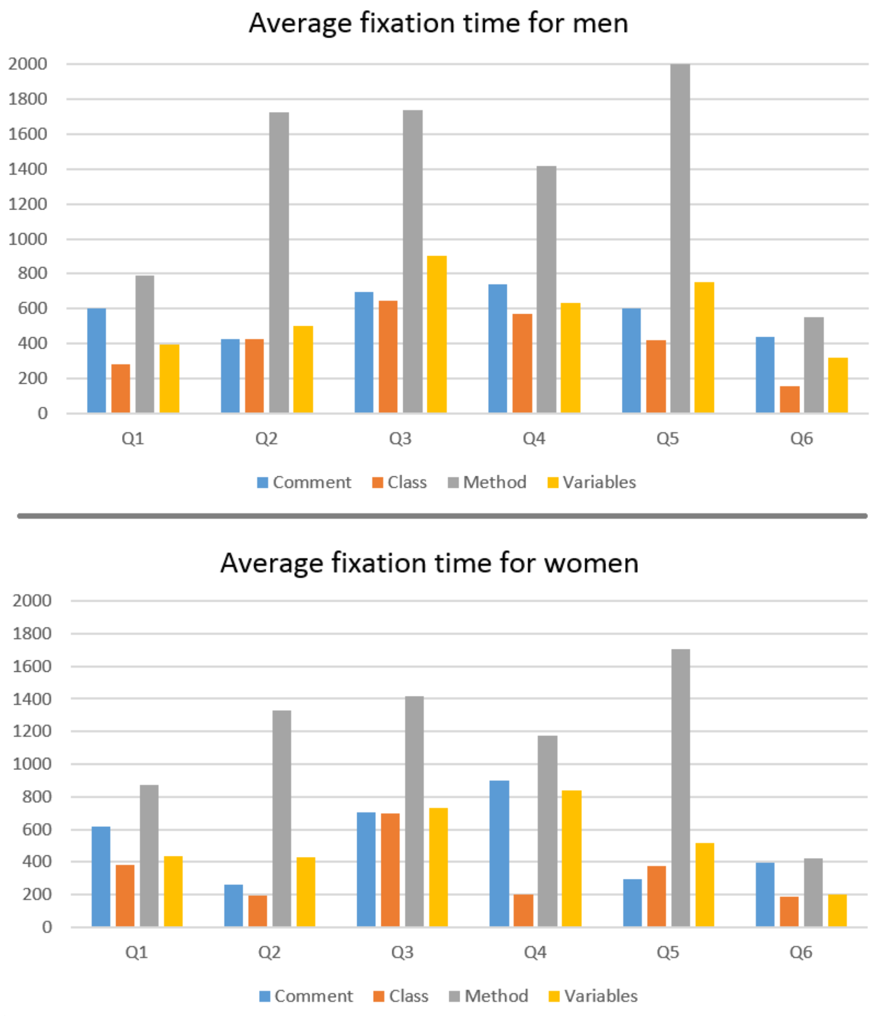


Figure 6.2 Distribution of average fixation time for SCEs.

6.4.1 RQ6.1: What Are the Important Source Code Entities (SCEs) to Which Developers Pay More Visual Attention When Reading Source Code?

It took on average 20 minutes to perform the experiment including setting up the eye-tracker. No participant mentioned any difficulties regarding the source code readability and understandability after performing the task.

We perform the following analysis to answer RQ6.1 and try to reject our null hypothesis. For each participant, we calculate the average fixation time that have been spent on different types of SCEs, e.g., class or method name, in milliseconds. We consider the time for both wrong and correct answers. Figure 6.2 shows the distribution of average fixation time for each type of SCEs for male and female developers. The x-axis shows the questions and its SCEs while the y-axis shows the average time spent on each type of SCEs.

Table 6.5 Average fixation time spent on each type of SCEs. Rankings of each type of SCEs are based on the average fixation time (1 means the most important).

		Class name	Method name	Variables	Comment	p-value
Female Participants	time (ms)	339.21	1153.88	523.90	529.63	<0.05
	Ranking	4	1	3	2	
Male Participants	time (ms)	416.83	1407.5	583.80	584.86	<0.05
	Ranking	4	1	3	2	

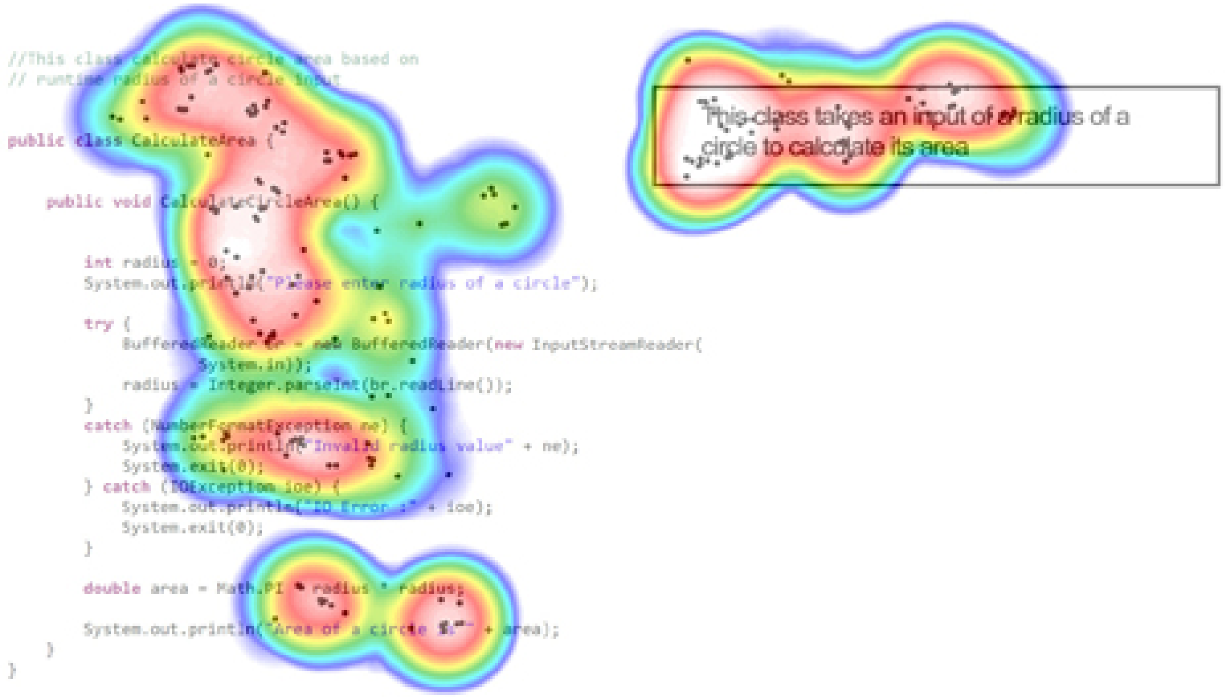
We also apply Kruskal-Wallis rank sum test (See Section 2.2.4) on the average fixation time that our participants spent on four types of SCEs to analyse statistically the participant's preferences for class names, method names, variable names, and comments. Table 6.5 reports that the p-value of the Kruskal-Wallis test for all the SCEs which is below the standard significant value $\alpha = 0.05$. Table 6.5 and Figure 6.2 show that participants focused more on method names and class names are the least preferred SCEs.

We also check the heatmaps (See Section 2.1.4) consisting of the cumulative fixations of all the participants on each stimulus. Figure 3 shows the heatmap for one of our participants working on one of our source codes. It shows a large number of fixations on the method name, comments, and question.

At the end of the eye-tracking study, we ask participants to answer a post questionnaire about the most important SCEs to compare with our eye-tracking results. The post-experiment questionnaire about participants' personal ranking for SCEs to comprehend the source code confirms the results of eye-tracking data.

The selection of source code snippets with only one SCE per type (one class name, one method name, one paragraph of comments, and one group of variables) could be a direct threat to the validity of our results. It is quite possible that developers may have different SCEs preference on larger source code snippets with multiple SCEs. Because we are interested to find the most important SCE based on the time that our participants spent, we used classes containing only one SCE of each type. Adding more samples of each SCE makes the analysis of the results more complicated because other factors must be considered, such as the relationship between the SCEs. For example, for a particular task, the developers may prefer variable names in large methods than method names. To avoid, such a problem, we chose to have a simple version of each class.

Thus, we reject $H\alpha_{11}$ and $H\alpha_{12}$ and answer RQ6.1 as follows: developers pay different amount of visual attention on different types of SCEs to perform program comprehension.



```
//This class calculate circle area based on
// runtime radius of a circle input

public class CalculateArea {

    public void CalculateCircleArea() {

        int radius = 0;
        System.out.println("Please enter radius of a circle");

        try {
            BufferedReader br = new BufferedReader(new InputStreamReader(
                    System.in));
            radius = Integer.parseInt(br.readLine());
        }
        catch (NumberFormatException ne) {
            System.out.println("Invalid radius value" + ne);
            System.exit(0);
        } catch (IOException ioe) {
            System.out.println("IO Error :" + ioe);
            System.exit(0);
        }

        double area = Math.PI * radius * radius;

        System.out.println("Area of a circle is " + area);
    }
}
```

This class takes an input of a radius of a circle to calculate its area

Figure 6.3 Heatmap showing the cumulative fixations of participants. The colors red, orange, green, and blue indicate the decrease in the number and duration of fixations from highest to lowest.

6.4.2 RQ6.2: Does Developers' Gender Impact the Viewing Strategies of Developers While Reading Source Code?

We perform the following steps to answer RQ6.2: first, we consider our set of AOIs including SCEs , as shown in Figure 6.1. Second, we use *Taupe* to compute and visualise different scan-paths for each participant working on all questions. Taupe assigns a character to represent each AOI. The comment AOI, class name AOI, variables AOI, method name AOI, and Question Area are represented by B, C, D, E, and F respectively. Similar to ScanMatch representation, to display the scan-path, we attach a small letter to the capital letter to make it easy to read the sequence. Therefore, if a participant goes from comment AOI to class name AOI and then to method name AOI, the sequence for her scan-path is bDcCeE.

Finally, we calculate the length of the scan-path for male and female participants separately for each question and compare the results as presented in Table 6.6. The length of a scan-path is equal to the number of attention switching between SCEs as represented by the number of characters in the path. For example, the length of the scan-path, “bBcCeEbBeE”, is five.

Table 6.6 Average length and standard deviation of the scan-paths for male and female participants, working on six questions. Two-tailed Wilcoxon p-value (α = 0.05) for the average path length are presented while comparing male and female participants.

	Average path length (Standard deviation (SD))		p-value
	Male participants	Female participants	
Q1	4 (SD = 7.7)	9.2 (SD = 4.2)	0.07
Q2	31 (SD =12.7)	13.5 (SD = 10.8)	<0.05
Q3	27 (SD = 18.6)	14 (SD = 9.8)	<0.05
Q4	33 (SD = 23)	16 (SD = 9.2)	<0.05
Q5	29 (SD = 10.6)	16 (SD = 9.2)	<0.05
Q6	21 (SD = 11.1)	11.5 (SD = 2.9)	<0.05

Interestingly, there is a statistically-significant difference between the length of scan-paths for male and female participants working on all questions except question 1. Shorter scan-paths implies less attention switching between SCEs and the question.

Thus, we reject $H\alpha_{21}$ and answer RQ6.2 as follows: female developers use different viewing strategies compared to their male counterparts while reading source code to understand it. They switch their attention between different SCEs less often and analyse each SCE more carefully before going to the next one. Our results are in agreement with previous work (Beckwith and Burnett, 2004; Beckwith et al., 2005, 2006a,b, 2007), in which they show that male developers are inclined to tinker an unfamiliar environment and approach the new, unknown features much earlier and more often than female developers. Our results also confirm that male developers try to explore the stimulus more than female developers.

However, using different viewing strategies does not lead to the superiority of any gender group regarding accuracy or time. No significant difference has been found between the accuracy and the time of male and female developers. Table 6.7 reports that male participants gave 79 correct answers out of 79 + 17 = 96, which means 82% correct answers while female participants gave 33 correct answers out of 33 + 9 = 42. which means 79% correct answers. After applying two tailed Wilcoxon with (α = 0.05), the p-value reports that there is no significant difference between male and female participants regarding accuracy.

Table 6.8 presents the average of time spent on each stimulus for male and female participants separately. The p-value implies that there is no significant differences between male and female participants regarding time spent (speed).

Table 6.7 Participants' contingency table for answers.

Answers			
Female participants		Male participants	
Correct	Wrong	Correct	Wrong
33	9	79	17

Table 6.8 Values for the amount of time that spent by male and female participants on each stimulus separately. The wilcoxon p-value ($\alpha = 0.05$) is presented after comparing male and female participants' accuracy.

	Average Time (min)						
	Q1	Q2	Q3	Q4	Q5	Q6	p-value
Female participants	5.51	7.05	5.85	5.7	7.73	4.06	0.6
Male participants	5.11	6.65	5.04	6.97	9.84	5.15	

6.4.3 RQ6.3: Are Some Types of SCEs Preferred by Developers Because of Their Semantic Content or Because of Their Very Presence in Source Code?

To answer this question, we perform another eye-tracking experiment. We use two methods to collect data that are similar to the first experiment. The type of SCEs (class name or method name) is the independent variable. We select two different sets of dependent variables to compare and complement the results of the first study. Similar to the first eye-tracking study, the first set of dependent variables contains the participants' average fixation time on each SCE. For the second set of dependent variables, because we are interested to investigate the impact of the presence of specific type of SCEs on participants' efficiency and effectiveness, we choose the following variables:

- Accuracy: we quantify and measure this variable as the percentage of correct answers given by a participant while reading source code and answer comprehension questions.
- Total fixation time: we measure this variable as the amount of time that each participant spends on the source code stimuli. We use the data provided by the eye-tracker to measure this variable for each stimulus separately.
- Effort: we use the Average Fixation Duration (AFD) metric for calculating the participants' visual effort.

We use the same criteria and the same programming languages as in the previous experiment to choose the pieces of code that make up the stimuli. The lengths of the pieces of code (in lines of code) are 28, 29, 19, 27, 33, and 35, respectively. In this experiment, for each piece of source code, there is one instance of each type of SCEs. Thus, each piece of source code contains one class name, one method name, one comment, and one set of variables. Figure 6.4 shows an example of a source code stimulus CiB.

Participants answered six comprehension questions, one for each source code. Table 6.9 shows the list of questions.

Table 6.9 List of questions answered by participants to perform program comprehension task.

	Questions
1	This class finds and prints the name of files of certain type (cvs).
2	This class finds and prints the number of prime numbers between 1 to 10000.
3	This class checks if two input files are recently updated and prints the last modified time.
4	This class accepts the path of an existing file as an input and adds one sentence to the end of file.
5	This class extracts a zip directory and prints the name of files inside that directory.
6	This class connects to the database and prints all the records in table called myTable.

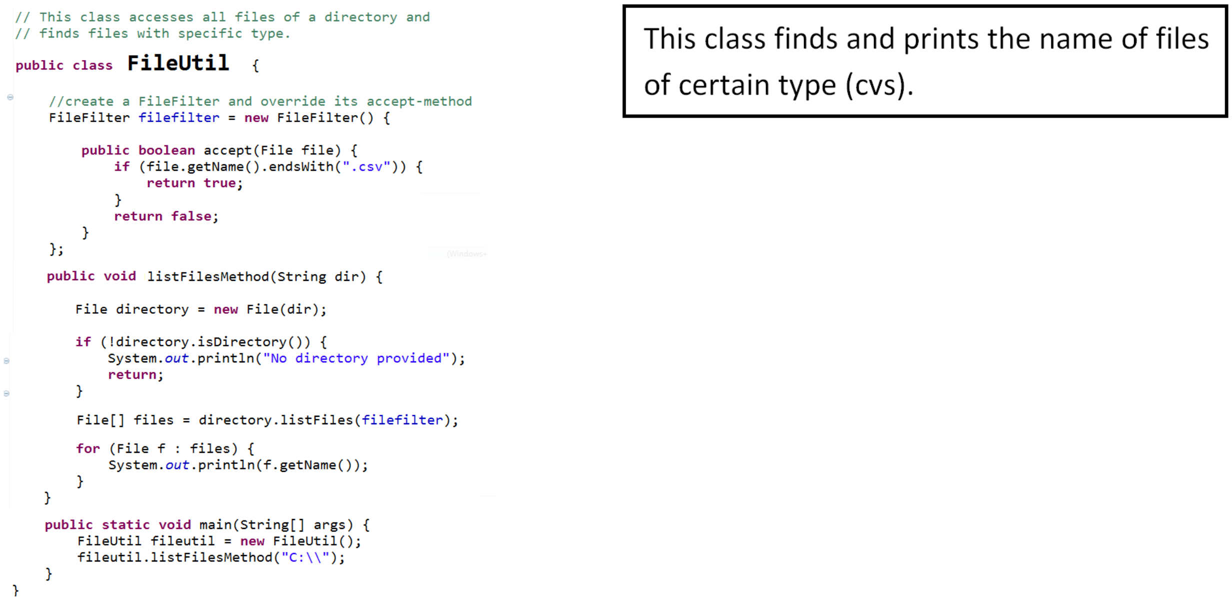
Our 14 participants are two B.Sc., three M.Sc., and nine Ph.D. students in the Department of Computer and Software Engineering at École Polytechnique de Montréal. All of these participants have, on average, 2 years of Java programming experience in academia and 0.5 years in industry.

As with the previous experiment, these participants were all volunteers. Some participants are common to both eye-tracking studies. However, for the second eye-tracking study, participants do not know the goal of the study and the questions and code snippets are different. The participants' demographics are presented in Table 6.10.

Here, we present the results of the second experiment to answer RQ6.3.3. By comparing Tables 6.11 and Table 6.12 with Table 6.5, we observe that if method names are highlighted, the result is the same as previous study in which method name is the most preferred SCE while class name is the least. If class names are highlighted, still participants spent more time on method names, but class name is not the least preferred one anymore.

Moreover, for MiB, the p-value shows that there is statistically significant difference between amount of time that participants spent on each SCE.

Table 6.13 shows the difference of participants' accuracies, total fixation times, and efforts on MiB and CiB code variants. We test our hypotheses to find any potential advantage of CiB over MiB. After applying the non-parametric unpaired Wilcoxon test with $\alpha = 0.05$, the p-values report that there is no significant difference between the two code variants for



```
// This class accesses all files of a directory and
// finds files with specific type.
public class FileUtil  {

    //create a FileFilter and override its accept-method
    FileFilter filefilter = new FileFilter() {

        public boolean accept(File file) {
            if (file.getName().endsWith(".csv")) {
                return true;
            }
            return false;
        }
    };

    public void listFilesMethod(String dir) {

        File directory = new File(dir);

        if (!directory.isDirectory()) {
            System.out.println("No directory provided");
            return;
        }

        File[] files = directory.listFiles(filefilter);

        for (File f : files) {
            System.out.println(f.getName());
        }
    }

    public static void main(String[] args) {
        FileUtil fileutil = new FileUtil();
        fileutil.listFilesMethod("C:\\");
    }
}
```

This class finds and prints the name of files of certain type (cvs).

Figure 6.4 Example of a source code stimulus in which the class name is written in bold using a larger font size.

Table 6.10 Participants' demographics.

Participants' Demographics				
Academic Background			Gender	
Ph.D.	M.Sc.	B.Sc	Male	Female
9	3	2	7	7

accuracy, total fixation time, and effort. We thus cannot reject any of our null hypotheses, H_{31}, H_{32}, or H_{33}. We use non-parametric unpaired Wilcoxon statistical test to determine significance because our data is not normally distributed.

We consider and compute the total fixation time that our participants spent to perform the whole task and effort that are required for reading the code to understand it. These results and observations emphasize that one types of SCEs does not lead and help our participants to read the code and find the answers more effectively.

In conclusion, this second experiment confirms the results of the previous one: method names are the most preferred SCEs. It also shows that whether method names or class names are highlighted, participants spend more time on method names, with equivalent accuracy, total time, and effort. It is indeed the content of the method names, and not just their presence, that participants seek to perform comprehension tasks.

Table 6.11 Average fixation times spent on each type of SCEs for CiB source code. Rankings of each type of SCEs are based on the average fixation times (1 means the most important).

		Class name	Method name	Variables	Comment	p-value
Female participants	Average time (ms)	237	428	174	50	0.06
	Ranking	2	1	3	4	
Male participants	Average time (ms)	331	391	171	148	0.06
	Ranking	2	3	2	4	

Table 6.12 Average fixation times spent on each type of SCEs for MiB source code. Rankings of each type of SCEs are based on the average fixation times (1 means the most important).

		Class name	Method name	Variables	Comment	p-value
Female participants	Average time (ms)	109	563	242	300	<0.05
	Ranking	4	1	3	2	
Male participants	Average time (ms)	107	824	322	154	<0.05
	Ranking	4	1	2	3	

We also compute and compare the average path length of male and female participants as shown in Table 6.14. The results confirm our finding from previous experiment and show that frequency of attention switching for female participants are less compared to males.

6.4.4 Impact of the Mitigating Variables

Experience: to evaluate the impact of experience, we consider level of study and the number of years of programming experience in Java in academia and industry. No statistically significant interaction was found, after applying one-way ANOVA.

6.4.5 Further Remarks

We use the results of two experiments and perform more analyses to evaluate the usefulness of these findings in Information Retrieval (IR) techniques. We conjecture that understanding how developers verify Requirements Traceability (RT) links could help improve the accuracy of IR-based RT techniques to create RT links. We propose an improved term weighting scheme, *i.e.,* Developers Preferred Term Frequency/Inverse Document Frequency

Table 6.13 Accuracy, total fixation time, and the amount of effort put forth by participants while working with CiB and MiB stimuli.

	MiB		CiB	
Accuracy (SD)	0.78	(SD = 0.41)	0.85	(SD = 0.34)
Total fixation time (ms) (SD)	20,134.38	(SD = 9664.29)	21,164.31	(SD = 8136.51)
Effort (SD)	78.99	(SD = 6.63)	76.95	(SD = 6.81)

Table 6.14 The average length and standard deviation of scan-paths for male and female participants, working with CiB and MiB source codes. Two-tailed Wilcoxon p-value ($\alpha = 0.05$) for the average path length is shown for each questions while comparing male and female participants.

	Average path length (Standard Deviation)		p-value
	Female participants	Male participants	
MiB	6.2 (3.3)	9.9 (SD = 2.4)	< 0.05
CiB	4.8 (SD = 3.1)	8.05 (SD = 2.8)	< 0.05

($DPTF/IDF$), that uses the knowledge of the developers' preferred types of SCEs to give more importance to these SCEs into the term weighting scheme. As presented in our EMSE publication (Ali et al., 2014), we integrate this weighting scheme with an IR technique, *i.e.,* Latent Semantic Indexing (LSI), to create a new technique to RT link recovery. Using three systems (iTrust, Lucene, and Pooka), we show that the proposed technique statistically improves the accuracy of the recovered RT links over a technique based on LSI and the usual Term Frequency/Inverse Document Frequency (TF/IDF) weighting scheme. Finally, we compare the newly proposed $DPTF/IDF$ with our original Domain Or Implementation/Inverse Document Frequency (DOI/IDF) weighting scheme.

6.5 Threats to Validity

In the following, in addition to those factors that are discussed in Section 2.2.2, we discuss other factors that may have influenced our results. Selection of participant threat could impact our study due to the natural difference among the participants' abilities. We analyse participants with various experience to mitigate this threat. For our proposed technique, individual participants' Java experience can cause some fluctuation in fixation that may lead to biased results. However, we minimised this threat by asking all participants for their general source code comprehension preference at the end of the experiment. The results of our post-experiment questionnaire are in agreement with the results of the experiments.

In our empirical study, they could be due to measurement errors. We use time spent on each

AOI and percentages of correct answer to measure the participants' performances. These measures are objective, even if small variations due to external factors, such as fatigue, could impact their values. We minimise this factor by using small source code and all the participants finished the whole experiment within 20 minutes.

6.6 Discussion and Conclusion

We designed and performed two eye-tracking experiments to analyse developers' eye movements to identify their preferred SCEs when they read source code to understand it. Results show that developers have different preferences for class names, method names, variable names, and comments.

In addition, we highlighted the developers' preferred types of SCEs to assess whether their preferences are due to the location of the SCEs or due to their semantic contents. Such a preference would be similar to the observation that developers trust a piece of information by virtue if it being shown rather than by its intrinsic value in the results returned by the Google search engine (Pan et al., 2007). We showed that it is indeed the semantic content of the SCEs that developers seek.

Moreover, inspired by developers behaviour, giving more attention to certain type of SCEs, we also devise a new term weighting scheme, called Developers Preference–based Term Frequency and Inverse Document Frequency (DPTF/IDF). Our results confirm that this new technique statistically improves the accuracy of the information retrieval technique as explained in our EMSE paper (Ali et al., 2014).

Moreover, we found no impact of gender on developers' most/least preferred type of SCEs. Both male and female developers spent more time on method names compared to other SCEs. However, we identified that male and female developers use different viewing strategies for reading source code. Women spent more time on each type of SCE and analyze it before going to the next one. In contrast, men showed several attention switching between different types of SCEs. We can thus now conclude our dissertation.

CHAPTER 7 CONCLUSION AND FUTURE DIRECTIONS

Program comprehension is a complex cognitive process that developers perform to build a mental model of the software system being maintained. To better understand the cognitive process underlying program comprehension, researchers exploit modern eye-trackers to collect data about the process of comprehension with minimal intrusion (Just and Carpenter, 1976; Bednarik and Tukiainen, 2006).

In this dissertation, we performed a mapping study to assess the current state of the art regarding the usage of eye-trackers in software engineering and to report on how widely eye-trackers have been used and contributed to empirical studies in software engineering.

The mapping study showed that the software engineering community mostly performed experiments to study developers' cognitive process during program comprehension by examining differences in the performance of novices and experts (Ormerod, 1990; Crosby et al., 2002; Aschwanden and Crosby, 2006; Yusuf et al., 2007; Soh et al., 2012; De Smet et al., 2012). However, analyzing only the level of expertise does not provide enough evidence about differences between developers' abilities (Storey, 2005). Thus, inspired by the previous work on gender differences in HCI (Burnett et al., 2011; Grigoreanu et al., 2009), in this dissertation, we considered developers' gender as one of the main factors influencing program comprehension.

Moreover, because developers usually benefit from the existence of other types of artifacts than source code to perform program comprehension, we examined the difference between graphical and textual representations during program comprehension to add to the existing body of knowledge in empirical research on the strengths and weaknesses of textual vs. graphical representations.

Thus, in this dissertation, we stated the following thesis: Representation type (graphical vs. textual) and gender impact the cognitive process underlying comprehension activity. Therefore, they influence developers' efficiency and effectiveness. Representation type and gender are also proxy for developers' viewing strategies, which can be inferred partly from the developers' eye-movements while developers perform program comprehension tasks.

7.1 Contributions

While researching our thesis, we provide the following main contributions:

1. We performed a mapping study and brought evidence that the advent of new eye-trackers makes the use of these tools easier and unobtrusive and that software engineering community benefits from this technique for empirical studies. Thus, we provided a global view and descriptive statistics about the usage of eye-tracking techniques in software engineering along with the research topics that have been studied using eye-trackers: model comprehension, code comprehension, debugging, collaborative interaction, traceability, and usability.

   We showed researchers and device providers the limitations and advantages of the usage of eye-trackers in software engineering research. Our findings help researchers to: (1) identify unstudied activities and artifacts that may result in new research directions, (2) discover the need for additional metrics and tool for their analysis to better access developers' cognitive process, (3) use a unified way (*e.g.,* guideline) and consistent terminology and metrics names when conducting and reporting eye-trackers studies.

   Moreover,we identified the need for device improvements to capture developers' eye-movements in real software development environment (*e.g.,* scrolling, switching between systems).

2. We performed a study of the identify the differences between graphical and textual representations in program comprehension.

   Our results show the superiority of structured text vs. graphical representations regarding developers' spent time and effort. We reported no differences between these two types of representation regarding accuracy. We observed that choosing between graphical and textual representations for modeling requirements is not trivial. Several factors including the complexity and the purpose of the task could also impact the results. Still, our results showed that for the comprehension of small systems, structured text is more effective than a graphical representation. Moreover, the graphical representation does not seem to bridge the linguistic distance, which also implies that the use of only one representation is not recommended.

   In addition, our results pointed at the value of educating readers of graphical representations with these representation during experiment. Although the TROPOS graphical notation that we used is not complex and we gave a tutorial to familiarise developers with TROPOS notation, the developers spent 5% of their time looking at the Helper part. Although 5% of the time is negligible, it seems profitable to provide more information about the representation that developers must use it.

   Also, as presented by (Heijstek et al., 2011), other approaches such as, annotating a graphical representation with a description of how to read it, may be beneficial. This annotation can be added either in natural text or using a more formal notation such as OCL (OMG Group, 2014).

3. We contributed to the literature on the impact of gender in program comprehension. We performed a set of empirical studies to investigate how well viewing strategies generalised to different population: male and female developers. Our results confirmed that although there is no difference between male and female developers' efficiency and effectiveness during comprehension and identifier recall tasks, they use different viewing strategies to perform the tasks.

   While the time spent by male and female developers is not significantly different, male and female developers focused differently on the alternative answers that we proposed to them. Female developers seem to carefully weight all options and rule out wrong answers while male developers seem to quickly set their minds on some answers. We found a statistically strong interaction between accuracy and effort spent on alternatives by female developers, which confirms that this strategy and spending more time on possible solutions helps female developers to find the correct answer.

   While studying the source code reading process, we observed two distinct strategies deployed by male and female developers. Male developers change their focus of attention regularly and go back and forth between different AOIs, *i.e.,* class names, method names, variables and comments, while, female developers focus and analyze one SCE before changing their focus of attention and go to the others.

   Moreover, our results showed that method names are the most important type of SCEs for both male and female developers. It demonstrators that developers must pay attention to method names and choose precise, understandable names. In addition, we propose a new weighting scheme, $DPTF/IDF$, that considers developers' preferred types of SCEs and conclude that taking into account source code elements is important to improve the accuracy of RT techniques (Ali et al., 2014).

   Furthermore, we provided a set of detailed observations about the viewing strategies of male and females developers to explain (1) how they answer multiple choice questions, (2) how they read the source code, and (3) what are the most important SCEs on which developers spent more visual attention. Our results and observations help to better understand the viewing strategies deployed in program comprehension, which is prerequisite for (1) developing tools (IDE and debugger), (2) choosing a proper representation to document existing software, (3) training that supports program comprehension, and (4) effectively using different artifacts to improve program comprehension.

   However, the number of female developers who participated in our studies are limited, and, although this number is much higher than that of any previously reported studies, we cannot consider the population large enough to generalize. Our findings should be considered suggesting further studies concerning the role of gender in software engineering rather than a general truth.

Yet, our findings raise two significant new open questions: first, are there differences between viewing strategies of male and female developers while performing program comprehension tasks and whether these strategies impact their efficiency and effectiveness or not. Also, if the results from our limited studies can be generalized to the broader population of developers: the second questions is: to which extent do current programming environments support these strategies?

7.2 Limitations

The limitations of the work presented in this dissertation are discussed in details in Section 4.5, Section 5.5, and Section 6.5. But, in general, our experiments suffer from the same limitations as previous eye-tracking studies:

– Number of developers: in our four experiments, we had 28, 24, 26, and 14, developers respectively. The majority of previous eye-tracking studies had less than twenty developers and our number of developers are reasonable but prevent generalization.
– Size of material: to correctly track the developers' visual attention, our stimuli fitted in one screen. Developers could not easily go back and forth between different screens or scroll the screens. Therefore, we used small source code snippets and models the same as previous eye-tracking studies. Moreover, because developers are subjected to fatigue effects in experiments lasting longer than about an hour, we also limited the size and the complexity of the material that we use to ensure that data is of a high quality. Therefore, our artifacts are not representative of all the artifacts usually used in software maintainable.

Therefore, because the current eye-trackers place some limits on preparing the materials, further experiments are required to generalize the results. More advancement in eye-tracking technique will allow us to prepare more realistic materials and conduct our experiments in real environments.

7.3 Future Work

Our work opens several new research directions. In general, further research should focus on improving the generalizability of our results. We plan to replicate our studies with different representations and to extend the experiments to larger numbers of developers. Moreover, we plan to perform additional experiments without eye-trackers but with more realistic tasks and more complex models.

Scan-paths are usually used to understand developers' viewing strategies. Because of limitation associated to analyzing and comparing fixation-based scan-paths, current work uses mostly AOI-based scan-paths. However, there is no method for defining AOIs size and granularity. The AOIs are defined in a top-down approach, based on the experimenter' assumption and experimental goals and conditions. However, Drusch et al. (2014) dynamically defined AOIs using a mean-shift clustering algorithm and visualized developers' scan-paths dynamically during the experiment to understand their visual behaviour while searching Web pages. Their results implied that dynamically defining and analyzing AOIs highlighted some differences in developers' behaviours that are hidden while using static heatmaps. Thus, we plan to further analyse developers' viewing strategies dynamically while performing comprehension tasks. This analysis could show whether developers always use the same strategies for viewing materials or their strategies and relevant AOIs change over time while they work.

Heatmaps clearly show and distinguish areas of higher interest with lowest ones. So far, in software engineering research, heatmaps have been compared qualitatively, providing a visualization for observing the final distribution of fixations' numbers and duration. We plan to further investigate image processing approaches to analyze heatmaps quantitatively.

In the following, we outline other future work related to the factors that have been investigated in this dissertation.

When we compared the developers' scan-paths, we observed that they followed two different viewing strategies: top-down and bottom-up. Our findings suggest further studies concerning the impact of representation structure (layout) on developers' strategies and performance. Previous studies indicate that the layout plays a significant role in the comprehension of UML class diagrams (Sharif and Maletic, 2010a). However, we plan to take one step back and put the focus on how vertical and horizontal lines served as a simple layout and can be used to modularize notations based on different levels of abstraction or relations. Then, we will further study the impact of these layouts on developers' performance (effectiveness and efficiency) using different representations.

While studying developers reading source codes, we did not examine developers' reading ability while working with natural language text. By comparing developers' reading abilities and their source code viewing strategies, we could single out personal characteristics that are not induced by the stimulus material. For example, as explained by Busjahn et al. (2011), for a certain person, having long fixations might be normal because there are not triggered by the stimulus material but by its own cognitive process. Consequently, we plan to study developers' viewing strategies and reading behaviour while reading natural text in addition to their source code viewing strategies.

We encourage future research to study how our finding about gender differences are applicable in daily software engineering tasks and also propose remedies to make these tasks enjoyable and increase the productivity of both male and female developers. We plan to perform more experiments with real programming environments (IDEs) using bigger pieces of source code containing more than one instance of each SCEs to analyze male and female developers' viewing strategies. We aim to perform think-aloud experiments to analyze more the impact of gender in source code reading and comprehension. Moreover, we are interested to analyze whether these strategies are supported by programming environments and programming language constructs to provide further suggestions to better support developers.

REFERENCES

N. Ali, Z. Sharafi, Y.-G. Guéhéneuc, and G. Antoniol, "An empirical study on requirements traceability using eye-tracking." in *Proceedings of the 28th IEEE International Conference on Software Maintenance.* IEEE Computer Society, 2012, pp. 191–200. [Online]. Available: http://dblp.uni-trier.de/db/conf/icsm/icsm2012.html#AliSGA12

——, "An empirical study on the importance of source code entities for requirements traceability," *Empirical Software Engineering*, pp. 1–37, 2014. [Online]. Available: http://dx.doi.org/10.1007/s10664-014-9315-y

S. Ambler, *The Elements of UML (TM) 2.0 Style.* Cambridge University Press, 2005.

C. Aschwanden and M. Crosby, "Code scanning patterns in program comprehension," in *Proceedings of the 39th Hawaii International Conference on System Sciences*, 2006.

J. Ayres, J. Flannick, J. Gehrke, and T. Yiu, "Sequential pattern mining using a bitmap representation," in *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 429–435. [Online]. Available: http://doi.acm.org/10.1145/775047.775109

R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval.* Addison-Wesley, 1999.

L. Beckwith and M. Burnett, "Gender: An important factor in end-user programming environments?" in *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on.* IEEE, 2004, pp. 107–114.

L. Beckwith, M. Burnett, S. Wiedenbeck, C. Cook, S. Sorte, and M. Hastings, "Effectiveness of end-user debugging software features: Are there gender issues?" in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '05. New York, NY, USA: ACM, 2005, pp. 869–878. [Online]. Available: http://doi.acm.org/10.1145/1054972.1055094

L. Beckwith, M. Burnett, V. Grigoreanu, and S. Wiedenbeck, "Gender hci: What about the software?" *Computer*, vol. 39, no. 11, pp. 97–101, 2006.

L. Beckwith, C. Kissinger, M. Burnett, S. Wiedenbeck, J. Lawrance, A. Blackwell, and C. Cook, "Tinkering and gender in end-user programmers' debugging," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '06. New York, NY, USA: ACM, 2006, pp. 231–240. [Online]. Available: http://doi.acm.org/10.1145/1124772.1124808

L. Beckwith, D. Inman, K. Rector, and M. Burnett, “On to the real world: Gender and self-efficacy in excel,” in *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on.* IEEE, 2007, pp. 119–126.

R. Bednarik, “Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations,” *Int. J. Hum.-Comput. Stud.*, vol. 70, no. 2, pp. 143–155, Feb. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.ijhcs.2011.09.003

R. Bednarik and M. Tukiainen, “An eye-tracking methodology for characterizing program comprehension processes,” in *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ser. ETRA ’06. New York, NY, USA: ACM, 2006, pp. 125–132. [Online]. Available: http://doi.acm.org/10.1145/1117309.1117356

——, “Visual attention tracking during program debugging,” in *Proceedings of the Third Nordic Conference on Human-computer Interaction*, ser. NordiCHI ’04. New York, NY, USA: ACM, 2004, pp. 331–334. [Online]. Available: http://doi.acm.org/10.1145/1028014.1028066

——, “Effects of display blurring on the behavior of novices and experts during program debugging,” in *CHI ’05 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA ’05. New York, NY, USA: ACM, 2005, pp. 1204–1207. [Online]. Available: http://doi.acm.org/10.1145/1056808.1056877

——, “Temporal eye-tracking data: Evolution of debugging strategies with multiple representations,” in *Proceedings of the 2008 Symposium on Eye Tracking Research &#38; Applications*, ser. ETRA ’08. New York, NY, USA: ACM, 2008, pp. 99–102. [Online]. Available: http://doi.acm.org/10.1145/1344471.1344497

——, “Validating the restricted focus viewer: A study using eye-movement tracking,” *Behavior research methods*, vol. 39, no. 2, pp. 274–282, 2007.

D. Binkley, M. Davis, D. Lawrie, and C. Morrell, “To camelcase or under_score,” in *IEEE 17th International Conference on Program Comprehension (ICPC).* IEEE, 2009, pp. 158–167.

D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, “The impact of identifier style on effort and comprehension,” *Empirical Softw. Engg.*, vol. 18, no. 2, pp. 219–276, Apr. 2013. [Online]. Available: http://dx.doi.org/10.1007/s10664-012-9201-4

A. F. Blackwell, A. R. Jansen, and K. Marriott, “Restricted focus viewer: A tool for tracking visual attention.” in *Diagrams*, ser. Lecture Notes in Computer Science, M. Anderson, P. C.-H. Cheng, and V. Haarslev, Eds., vol. 1889. Springer, 2000, pp. 162–177. [Online]. Available: http://dblp.uni-trier.de/db/conf/diagrams/diagrams2000.html#BlackwellJM00

P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, “Lessons from applying the systematic literature review process within the software engineering domain,” *Journal of systems and software*, vol. 80, no. 4, pp. 571–583, 2007.

P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, and J. Mylopoulos, “Tropos: An agent-oriented software development methodology,” *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, pp. 203–236, 2004.

R. E. Brooks, “Studying programmer behavior experimentally: The problems of proper methodology,” *Commun. ACM*, vol. 23, no. 4, pp. 207–213, Apr. 1980. [Online]. Available: http://doi.acm.org/10.1145/358841.358847

D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, “Empirical evidence about the uml: A systematic literature review,” *Softw. Pract. Exper.*, vol. 41, no. 4, pp. 363–392, Apr. 2011. [Online]. Available: http://dx.doi.org/10.1002/spe.1009

M. Burnett, S. Fleming, S. Iqbal, G. Venolia, V. Rajaram, U. Farooq, V. Grigoreanu, and M. Czerwinski, “Gender differences and programming environments: Across programming populations,” in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement.* ACM, 2010, pp. 1–10.

M. M. Burnett, L. Beckwith, S. Wiedenbeck, S. D. Fleming, J. Cao, T. H. Park, V. Grigoreanu, and K. Rector, “Gender pluralism in problem-solving software,” *Interact. Comput.*, vol. 23, no. 5, pp. 450–460, Sep. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.intcom.2011.06.004

T. Busch, “Gender differences in self-efficacy and attitudes toward computers,” *Journal of educational computing research*, vol. 12, no. 2, pp. 147–158, 1995.

T. Busjahn, C. Schulte, and A. Busjahn, “Analysis of code reading to gain more insight in program comprehension,” in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling ’11. New York, NY, USA: ACM, 2011, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/2094131.2094133

T. Busjahn, R. Bednarik, and C. Schulte, “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*, ser. ETRA ’14. New York, NY, USA: ACM, 2014, pp. 335–338.

N. E. Cagiltay, G. Tokdemir, O. Kilic, and D. Topalli, “Performing and analyzing non-formal inspections of entity relationship diagram (erd),” *J. Syst. Softw.*, vol. 86, no. 8, pp. 2184–2195, Aug. 2013. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2013.03.106

A. Calitz, M. Pretorius, and D. Greunen, “The evaluation of information visualisation techniques using eye tracking,” 2009.

B. Caprile and P. Tonella, “Restructuring program identifier names,” in *Proc. Int’l Conf. Software Maintenance (ICSM).* IEEE Computer Society Press, 2000.

G. Cepeda Porras and Y. Guéhéneuc, “An empirical study on the efficiency of different design pattern representations in UML class diagrams,” *Empirical Software Engineering*, vol. 15, no. 5, pp. 493–522, 2010.

G. Cepeda Porras and Y.-G. Guéhéneuc, “An empirical study on the efficiency of different design pattern representations in uml class diagrams,” *Empirical Software Engineering*, vol. 15, no. 5, pp. 493–522, Oct. 2010. [Online]. Available: http://dx.doi.org/10.1007/s10664-009-9125-9

B. Chiswick and P. Miller, “Linguistic distance: A quantitative measure of the distance between english and other languages,” *Journal of Multilingual and Multicultural Development*, vol. 26, no. 1, pp. 1–11, 2005.

T. A. Corbi, “Program understanding: challenge for the 1990’s,” *IBM Syst. J.*, vol. 28, pp. 294–306, June 1989. [Online]. Available: http://dx.doi.org/10.1147/sj.282.0294

F. Cristino, S. Mathot, J. Theeuwes, and I. D. Gilchrist, “Scanmatch: A novel method for comparing fixation sequences.” *Behaviour Research Method*, vol. 42, pp. 692–700, 2010.

M. E. Crosby and J. Stelovsky, “How do we read algorithms? a case study,” *Computer*, vol. 23, no. 1, pp. 24–35, Jan. 1990. [Online]. Available: http://dl.acm.org/citation.cfm?id=77577.77580

M. E. Crosby, J. Scholtz, and S. Wiedenbeck, “The roles beacons play in comprehension for novice and expert programmers,” in *Programmers, 14th Workshop of the Psychology of Programming Interest Group, Brunel University*, 2002, pp. 18–21.

M. Czerwinski, D. S. Tan, and G. G. Robertson, “Women take a wider view,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’02. New York, NY, USA: ACM, 2002, pp. 195–202. [Online]. Available: http://doi.acm.org/10.1145/503376.503412

A. Davis, *201 principles of software development.* McGraw-Hill, Inc., 1995.

B. De Smet, L. Lempereur, Z. Sharafi, Y.-G. Guéhéneuc, G. Antoniol, and N. Habra, “Taupe: Visualizing and analyzing eye-tracking data,” *Science of Computer Programming*, 2012.

F. Deissenbock and M. Pizka, “Concise and consistent naming,” in *Proceedings of the International Workshop on Program Comprehension.* IEEE CS Press, 2005, pp. 97 – 106.

R. Dewhurst, M. Nyström, H. Jarodzka, T. Foulsham, R. Johansson, and K. Holmqvist, “It depends on how you look at it: Scanpath comparison in multiple dimensions with multimatch, a vector-based approach," *Behavior research methods*, vol. 44, no. 4, pp. 1079–1100, 2012.

H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empirical Softw. Engg.*, vol. 10, no. 4, pp. 405–435, Oct. 2005. [Online]. Available: http://dx.doi.org/10.1007/s10664-005-3861-2

G. Drusch, J. C. Bastien, and S. Paris, "Analysing eye-tracking data: From scanpaths and heatmaps to the dynamic visualisation of areas of interest," *Advances in Science, Technology, Higher Education and Society in the Conceptual Age: STHESCA*, vol. 20, p. 205, 2014.

A. T. Duchowski, J. Driver, S. Jolaoso, W. Tan, B. N. Ramey, and A. Robbins, "Scanpath comparison revisited," in *Proceedings of the 2010 Symposium on Eye-Tracking Research &#38; Applications*, ser. ETRA '10. New York, NY, USA: ACM, 2010, pp. 219–226. [Online]. Available: http://doi.acm.org/10.1145/1743666.1743719

A. Duchowski, *Eye tracking methodology: Theory and practice.* Springer-Verlag New York Inc, 2007.

H. A. Duru, M. P. Çakır, and V. İşler, "How does software visualization contribute to software comprehension? a grounded theory approach," *International Journal of Human-Computer Interaction*, vol. 29, no. 11, pp. 743–763, 2013.

S. Easterbrook, "Empirical research methods for software engineering," in *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07. New York, NY, USA: ACM, 2007, pp. 574–574. [Online]. Available: http://doi.acm.org/10.1145/1321631.1321749

R. J. Erich Gamma, Richard Helm and J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software.* Addison-Wesley Pub Co, 1995.

B. Erol, K. Berkner, and S. Joshi, "Multimedia thumbnails for documents," in *Proceedings of the 14th annual ACM international conference on Multimedia*, ser. MULTIMEDIA '06. New York, NY, USA: ACM, 2006, pp. 231–240.

A. Eteläpelto, "Metacognition and the expertise of computer program comprehension," *Scandinavian Journal of Educational Research*, vol. 37, no. 3, pp. 243–254, 1993.

Q. Fan, "The effects of beacons, comments, and tasks on program comprehension process in software maintenance," Ph.D. dissertation, Catonsville, MD, USA, 2010, aAI3422807.

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*, ser. ETRA '14. New York, NY, USA: ACM, 2014, pp. 259–262.

E. Fennema, T. P. Carpenter, V. R. Jacobs, M. L. Franke, and L. W. Levi, “A longitudinal study of gender differences in young children’s mathematical thinking,” *Educational researcher*, pp. 6–11, 1998.

H. Fisher, *Anatomy of Love: A Natural History of Mating, Marriage, and Why We Stray.* The random house publishing group, 1994.

M. Fisher, A. Cox, and L. Zhao, “Using sex differences to link spatial cognition and program comprehension,” in *IEEE 22nd International Conference on Software Maintenance.* IEEE Computer Society, 2006, pp. 289–298.

T. Fritz, A. Begel, S. C. Müller, S. Yigit-Elliott, and M. Züger, “Using psycho-physiological measures to assess task difficulty in software development,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE ’14. New York, NY, USA: ACM, 2014, pp. 402–413.

A. M. Gallagher and R. De Lisi, “Gender differences in scholastic aptitude test: Mathematics problem solving among high-ability students.” *Journal of Educational Psychology*, vol. 86, no. 2, p. 204, 1994.

E. Gamma, “Applying design patterns in java,” 1996, pp. 47–53.

J. H. Goldberg and X. P. Kotval, “Computer interface evaluation using eye movements: methods and constructs,” *International Journal of Industrial Ergonomics*, vol. 24, no. 6, pp. 631–645, 1999.

J. H. Goldberg and 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*, ser. BELIV ’10. New York, NY, USA: ACM, 2010, pp. 71–78.

J. H. Goldberg and A. M. Wichansky, “Eye tracking in usability evaluation: A practitioner’s guide,” *To appear in: Hyönä R. Radach, H. Deubel (Eds.), The mind’s eye: Cognitive and applied aspects of eye movement research*, 2003.

J. H. Goldberg, M. J. Stimson, M. Lewenstein, N. Scott, and A. M. Wichansky, “Eye tracking in web search tasks: Design implications,” in *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications*, ser. ETRA ’02. New York, NY, USA: ACM, 2002, pp. 51–58.

V. Grigoreanu, J. Brundage, E. Bahna, M. Burnett, P. ElRif, and J. Snover, “Males and females script debugging strategies,” *End-User Development*, pp. 205–224, 2009.

V. Grigoreanu, J. Cao, T. Kulesza, C. Bogart, K. Rector, M. Burnett, and S. Wiedenbeck, “Can feature design reduce the gender gap in end-user software development environments?”

in *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, ser. VLHCC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 149–156. [Online]. Available: http://dx.doi.org/10.1109/VLHCC.2008.4639077

Y.-G. Guéhéneuc, "Taupe: Towards understanding program comprehension," in *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, ser. CASCON '06. Riverton, NJ, USA: IBM Corp., 2006. [Online]. Available: http://dx.doi.org/10.1145/1188966.1188968

Y.-G. Guéhéneuc, H. Kagdi, and J. Maletic, "Working session: Using eye-tracking to understand program comprehension," in *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, May 2009, pp. 278–279.

M. Harris, *The cannibals and the kings: Origins of Cultures.* Vintage book edition, 1991.

S. Hart and L. Staveland, "Development of nasa-tlx (task load index): Results of empirical and theoretical research," *Human mental workload*, vol. 1, pp. 139–183, 1988.

L. Hart-Gonzalez and S. Lindemann, "Expected achievement in speaking proficiency," School of Language Studies, Foreign Services Institute, Department of State, Tech. Rep., 1993.

K. Hartzel, "How self-efficacy and gender issues affect software adoption and use," *Communications of the ACM*, vol. 46, no. 9, pp. 167–171, 2003.

W. Heijstek, T. Kuhne, and M. R. V. Chaudron, "Experimental analysis of textual and graphical representations for software architecture design," in *Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 167–176. [Online]. Available: http://dx.doi.org/10.1109/ESEM.2011.25

P. Hejmady and N. H. Narayanan, "Visual attention patterns during program debugging with an ide." in *Proceedings of the 2012 Symposium on Eye Tracking Research &Amp; Applications*, C. H. Morimoto, H. O. Istance, S. N. Spencer, J. B. Mulligan, and P. Qvarfordt, Eds. ACM, 2012, pp. 197–200. [Online]. Available: http://dblp.uni-trier.de/db/conf/etra/etra2012.html#HejmadyN12

H. Hembrooke, B. Pan, T. Joachims, G. Gay, and L. Granka, "In google we trust: Users ' decisions on rank, position and relevancy," 2005.

J. M. Henderson and G. L. Pierce, "Eye movements during scene viewing: Evidence for mixed control of fixation durations," *Psychonomic Bulletin & Review*, vol. 15, no. 3, pp. 566–573, 2008.

D. W. Hosmer Jr and S. Lemeshow, *Applied logistic regression.* John Wiley & Sons, 2004.

R. J. Jacob and K. S. Karn, "Eye tracking in human-computer interaction and usability research: Ready to deliver the promises." *The Mind's Eye: Cognitive and Applied Aspects of Eye Movement Research*, vol. 2, no. 3, p. 4, 2003.

A. R. Jansen, A. F. Blackwell, and K. Marriott, "A tool for tracking visual attention: The restricted focus viewer," *Behavior research methods, instruments, & computers*, vol. 35, no. 1, pp. 57–69, 2003.

S. Jeanmart, Y.-G. Guéhéneuc, H. A. Sahraoui, and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance." in *Proceedings of 3rd International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 69–78. [Online]. Available: http://dblp.uni-trier.de/db/conf/esem/esem2009.html#JeanmartGSH09

A. Jedlitschka and D. Pfahl, "Reporting guidelines for controlled experiments in software engineering," in *Proceedings of the 2005 International Symposium on Empirical Software Engineering*, Nov 2005, pp. 10 pp.–.

P. Jermann and M.-A. Nüssli, "Effects of sharing text selections on gaze cross-recurrence and interaction quality in a pair programming task." in *CSCW*, S. E. Poltrock, C. Simone, J. Grudin, G. Mark, and J. Riedl, Eds. ACM, 2012, pp. 1125–1134. [Online]. Available: http://dblp.uni-trier.de/db/conf/cscw/cscw2012c.html#NussliJ12

D. Jing, Y. Sheng, and Z. Kang, "Visualizing design patterns in their applications and compositions," *IEEE Transactions on Software Engineering*, vol. 33, no. 7, pp. 433–453, 2007.

D. M. Jones, 2003, the New C Standard (Identifiers) An Economic and Cultural Commentary.

M. A. Just and P. A. Carpenter, "A theory of reading: from eye fixations to comprehension." *Psychological review*, vol. 87, no. 4, p. 329, 1980.

——, "Eye fixations and cognitive processes," *Cognitive Psychology*, vol. 8, no. 4, pp. 441–480, 1976.

J. N. Kara Pernice, "Eyetracking methodology: How to conduct and evaluate usability studies using eyetracking," `http://www.useit.com/eyetracking/methodology`, August 2009.

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*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 25–32.

——, "Procedures for undertaking systematic reviews," Joint Technical Report, Computer Science Department, Keele University (TR/SE- 0401) and National ICT Australia Ltd, Tech. Rep., 2004.

B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering - a systematic literature review,” *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2008.09.009

B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, “Preliminary guidelines for empirical research in software engineering,” *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002.

B. A. Kitchenham, T. Dyba, and M. Jorgensen, “Evidence-based software engineering,” in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE ’04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 273–281. [Online]. Available: http://dl.acm.org/citation.cfm?id=998675.999432

B. A. Kitchenham, D. Budgen, and P. Brereton, “The value of mapping studies: a participantobserver case study,” in *Proceedings of the 14th international conference on Evaluation and Assessment in Software Engineering.* British Computer Society, 2010, pp. 25–33.

G. Kowalski, *Information retrieval architecture and algorithms.* Springer-Verlag New York Inc, 2010.

J. Krogstie, G. Sindre, and H. Jørgensen, “Process models representing knowledge for action: A revised quality framework,” *Eur. J. Inf. Syst.*, vol. 15, no. 1, pp. 91–102, Feb. 2006. [Online]. Available: http://dx.doi.org/10.1057/palgrave.ejis.3000598

A. Lakhotia, “Understanding someone else’s code: Analysis of experiences,” *J. Syst. Softw.*, vol. 23, no. 3, pp. 269–275, Dec. 1993. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(93)90101-3

C. Lankford, “Gazetracker: Software designed to facilitate eye movement analysis,” in *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ser. ETRA ’00. New York, NY, USA: ACM, 2000, pp. 51–55.

T. D. LaToza, G. Venolia, and R. DeLine, “Maintaining mental models: a study of developer work habits,” in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 492–501.

P. Laurent, P. Mader, J. Cleland-Huang, and A. Steele, “A taxonomy and visual notation for modeling globally distributed requirements engineering projects,” in *Global Software Engineering (ICGSE), 2010 5th IEEE International Conference on.* IEEE, 2010, pp. 35–44.

M. Lavallée, P. N. Robillard, and R. Mirsalari, “Performing systematic literature reviews with novices: An iterative approach,” *IEEE Trans. Education*, vol. 57, no. 3, pp. 175–181, 2014. [Online]. Available: http://dx.doi.org/10.1109/TE.2013.2292570

D. Lawrie, C. Morrell, H. Feild, and D. Binkley, “Effective identifier names for comprehension and memory,” *Innovations in Systems and Software Engineering*, vol. 3, no. 4, pp. 303–318, 2007.

C. A. Lawton, “Gender differences in way-finding strategies: Relationship to spatial ability and spatial anxiety,” *Sex roles*, vol. 30, no. 11-12, pp. 765–779, 1994.

M. M. Lehman, “Laws of software evolution revisited,” in *Software process technology.* Springer, 1996, pp. 108–124.

V. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

B. Lientz and E. Swanson, *Software maintenance management.* Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1980.

D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway, “Mental models and software maintenance,” *J. Syst. Softw.*, vol. 7, no. 4, pp. 341–355, Dec. 1987. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(87)90033-1

S. Ljungblad and L. E. Holmquist, “Transfer scenarios: grounding innovation with marginal practices.” in *CHI*, M. B. Rosson and D. J. Gilmore, Eds. ACM, 2007, pp. 737–746. [Online]. Available: http://dblp.uni-trier.de/db/conf/chi/chi2007.html#LjungbladH07

L. Lorigo, B. Pan, H. Hembrooke, T. Joachims, L. Granka, and G. Gay, “The influence of task and gender on search and evaluation behavior using google,” *Inf. Process. Manage.*, vol. 42, no. 4, pp. 1123–1131, Jul. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.ipm.2005.10.001

J. Margolis and A. Fisher, *Unlocking the clubhouse: Women in computing.* MIT press, 2003.

C. McDowell, L. Werner, H. E. Bullock, and J. Fernald, “The impact of pair programming on student performance, perception and persistence,” in *Proceedings of the 25th International Conference on Software Engineering*, ser. ICSE ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 602–607. [Online]. Available: http://dl.acm.org/citation.cfm?id=776816.776899

J. Meyers-Levy, *Gender Differences in Information Processing: A SelectivityInterpretation.* P. Cafferata and A. Tybout, (Eds) Cognitive and Affective Responses to Advertising. Lexington Books, 1989.

D. L. Moody, “The physics of notations: Toward a scientific basis for constructing visual notations in software engineering,” *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 756–779, 2009.

R. Navarro-Prieto and J. J. Cañas, "Are visual programming languages better? the role of imagery in program comprehension," *International Journal of Human-Computer Studies*, vol. 54, no. 6, pp. 799–829, 2001.

E. O'Donnell and E. Johnson, "The effects of auditor gender and task complexity on information processing efficiency," *International Journal of Auditing*, vol. 5, no. 2, pp. 91–105, 2001.

O. M. G. OMG Group, "Object constraint language," Needham, MA, Tech. Rep., February 2014.

T. Ormerod, "Human cognition and programming," *Hoc et al*, vol. 307, pp. 63–82, 1990.

A. Ottensooser, A. Fekete, H. A. Reijers, J. Mendling, and C. Menictas, "Making sense of business process descriptions: An experimental comparison of graphical and textual notations," *Journal of Systems and Software*, vol. 85, no. 3, pp. 596–606, Mar. 2012.

B. Pan, H. Hembrooke, T. Joachims, L. Lorigo, G. Gay, and L. Granka, "In google we trust Users' decisions on rank, position, and relevance," *Journal of Computer-Mediated Communication*, vol. 12, no. 3, pp. 801–823, 2007.

M. Petre, "Why looking isn't always seeing: Readership skills and graphical programming," *Commun. ACM*, vol. 38, no. 6, pp. 33–44, Jun. 1995. [Online]. Available: http://doi.acm.org/10.1145/203241.203251

R. Petrusel and J. Mendling, "Eye-tracking the factors of process model comprehension tasks." in *CAiSE*, ser. Lecture Notes in Computer Science, C. Salinesi, M. C. Norrie, and O. Pastor, Eds., vol. 7908. Springer, 2013, pp. 224–239. [Online]. Available: http://dblp.uni-trier.de/db/conf/caise/caise2013.html#PetruselM13

V. Ponsoda, D. Scott, and J. M. Findlay, "A probability vector and transition matrix analysis of eye movements during visual search," *Acta psychologica*, vol. 88, no. 2, pp. 167–185, 1995.

A. Poole and 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. Pennsylvania: Idea Group, Inc*, 2005.

M. Powell and D. Ansic, "Gender differences in risk behaviour in financial decision-making: An experimental analysis," *Journal of economic psychology*, vol. 18, no. 6, pp. 605–628, 1997.

R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th ed. McGraw-Hill Higher Education, 2001.

C. M. Privitera and L. W. Stark, “Algorithms for defining visual regions-of-interest: Comparison with eye fixations,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 970–982, 2000.

K. Rayner, “Eye movements in reading and information processing: 20 years of research.” *Psychological bulletin*, vol. 124, no. 3, p. 372, 1998.

——, “Eye movements in reading and information processing,” *Psychological Bulletin*, vol. 85, no. 3, pp. 618–660, 1978.

P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D’Mello, “Improving automated source code summarization via an eye-tracking study of programmers,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 390–401. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568247

——, “Improving automated source code summarization via an eye-tracking study of programmers,” in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE ’14. New York, NY, USA: ACM, 2014, pp. 390–401.

T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, “How do professional developers comprehend software?” in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 255–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337254

——, “How do professional developers comprehend software?” in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 255–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337254

P. Romero, R. Cox, B. du Boulay, and R. Lutz, “Visual attention and representation switching during java program debugging: A study using the restricted focus viewer,” in *Diagrammatic Representation and Inference.* Springer, 2002, pp. 221–235.

P. Romero, R. Lutz, R. Cox, and B. du Boulay, “Co-ordination of multiple external representations during java program debugging,” in *Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on.* IEEE, 2002, pp. 207–214.

P. Romero, B. du Boulay, R. Cox, and R. Lutz, “Java debugging strategies in multi-representational environments,” in *15th Workshop of Psychology of Programming Interest Group*, 2003, pp. 421–434.

D. L. Sackett, S. E. Straus, W. S. Richardson, W. Rosenberg, and R. B. Haynes, *Evidence-Based Medicine: How to Practice and Teach EBM*, 2nd ed. Churchill Livingstone, 2000.

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*, ser. PPIG ’04, 2004, pp. 151–158.

R. Schauer and R. K. Keller, “Pattern visualization for software comprehension,” in *IN 6TH INTERNATIONAL WORKSHOP ON PROGRAM COMPREHENSION*, 1998, pp. 4–12.

Seeing Machine, “Seeing Machine’s website - FaceLAB,” http://www.seeingmachines.com/product/facelab/, 2010, accessed April 16th in 2014.

Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, and G. Antoniol, “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*, D. Beyer, A. van Deursen, and M. W. Godfrey, Eds., 2012, pp. 27–36. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwpc/icpc2012.html#SharafiSGA12

Z. Sharafi, A. Marchetto, A. Susi, G. Antoniol, and Y.-G. Guéhéneuc, “An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension.” in *Proceedings of 20th International Conference on Program Comprehension*, 2013, pp. 33–42. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwpc/icpc2013.html#SharafiMSAG13

B. Sharif and J. Maletic, “An eye tracking study on camelcase and under_score identifier styles,” in *IEEE 18th International Conference on Program Comprehension (ICPC)*. IEEE, 2010, pp. 196–205.

B. Sharif, “Empirical assessment of uml class diagram layouts based on architectural importance,” in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, ser. ICSM ’11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 544–549. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2011.6080828

B. Sharif and 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*, ser. TEFSE ’11. New York, NY, USA: ACM, 2011, pp. 67–70. [Online]. Available: http://doi.acm.org/10.1145/1987856.1987872

B. Sharif and J. I. Maletic, “An eye tracking study on the effects of layout in understanding the role of design patterns.” in *Proceedings of the 26th IEEE International Conference on Software Maintenance.* IEEE Computer Society, 2010, pp. 1–10. [Online]. Available: http://dblp.uni-trier.de/db/conf/icsm/icsm2010.html#SharifM10

——, “An eye tracking study on camelcase and under_score identifier styles.” in *Proceedings of 18th International Conference on Program Comprehension.* IEEE Computer Society, 2010, pp. 196–205. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwpc/icpc2010.html#SharifM10

B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: ACM, 2012, pp. 381–384.

——, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: ACM, 2012, pp. 381–384. [Online]. Available: http://doi.acm.org/10.1145/2168556.2168642

B. Sharif, G. Jetty, J. Aponte, and E. Parra, "An empirical study assessing the effect of seeit 3d on comprehension." in *VISSOFT*, A. Telea, A. Kerren, and A. Marcus, Eds. IEEE, 2013, pp. 1–10. [Online]. Available: http://dblp.uni-trier.de/db/conf/vissoft/vissoft2013.html#SharifJAP13

K. Sharma, P. Jermann, M.-A. Nüssli, and P. Dillenbourg, "Understanding collaborative program comprehension: Interlacing gaze and dialogues," in *Computer Supported Collaborative Learning (CSCL 2013)*, no. EPFL-CONF-184007, 2013.

K. Siau, "Informational and computational equivalence in comparing information modeling methods," *Journal of Database Management (JDM)*, vol. 15, no. 1, pp. 73–86, 2004.

J. L. Sibert, M. Gokturk, and 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*, ser. UIST '00. New York, NY, USA: ACM, 2000, pp. 101–107.

J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 378–389.

S. E. Sim, S. Ratanotayanon, and L. Cotran, "Structure transition graphs: An ecg for program comprehension?" in *ICPC*. IEEE Computer Society, 2009, pp. 303–304.

D. I. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal, "A survey of controlled experiments in software engineering," *Software Engineering, IEEE Transactions on*, vol. 31, no. 9, pp. 733–753, 2005.

C. F. Snook and R. Harrison, "Experimental comparison of the comprehensibility of a uml-based formal specification versus a textual one," in *Proceedings of 11 th International Conference on Evaluation and Assessment in Software Engineering (EASE*, 2007, pp. 955–971.

Z. Soh, Z. Sharafi, B. V. den Plas, G. C. Porras, Y.-G. Guéhéneuc, and G. Antoniol, "Professional status and expertise for uml class diagram comprehension: An empirical study." in *Proceedings of 20th International Conference on Program Comprehension*, D. Beyer, A. van Deursen, and M. W. Godfrey, Eds., 2012, pp. 163–172. [Online]. Available: http://dblp.uni-trier.de/db/conf/iwpc/icpc2012.html#SohSPPGA12

J. Somervell, C. M. Chewar, and D. S. Mccrickard, "Evaluating graphical vs. textual secondary displays for information notification," in *Proceedings of the ACM Southeast Conference, Raleigh NC*, 2002, pp. 153–160.

SR Research Ltd, *EyeLink II User Manual version (07/02/2006)*, SR Research Ltd., Februari 2006.

R. Stein, "Another person's eye gaze as a cue in solving programming problems," in *Proceedings of the 6th International Conference on Multimodal Interface.* ACM Press, 2004, pp. 9–15.

M.-A. Storey, "Theories, methods and tools in program comprehension: Past, present and future," *iwpc*, vol. 00, pp. 181–191, 2005.

N. Subrahmaniyan, L. Beckwith, V. Grigoreanu, M. Burnett, S. Wiedenbeck, V. Narayanan, K. Bucht, R. Drummond, and X. Fern, "Testing vs. code inspection vs. what else?: Male and female end users' debugging strategies," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '08. New York, NY, USA: ACM, 2008, pp. 617–626. [Online]. Available: http://doi.acm.org/10.1145/1357054.1357153

Y.-h. Sun, P.-L. He, and Z.-G. Chen, "An improved term weighting scheme for vector space model," in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 3. IEEE, 2004, pp. 1692–1695.

R. Team *et al.*, "R: A language and environment for statistical computing," *R Foundation for Statistical Computing*, no. 01/19, 2010.

T. I. TIOBE Software, "http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html," 2014.

G. Torkzadeh and X. Koufteros, "Factorial validity of a computer self-efficacy scale and the impact of computer training," *Educational and Psychological Measurement*, vol. 54, no. 3, pp. 813–821, 1994.

M. Turner, B. Kitchenham, D. Budgen, and 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*, ser. EASE'08. Swinton, UK, UK: British Computer Society, 2008, pp. 110–118. [Online]. Available: http://dl.acm.org/citation.cfm?id=2227115.2227127

R. Turner, M. Falcone, B. Sharif, and A. Lazar, “An eye-tracking study assessing the comprehension of C++ and Python source code,” in *Proceedings of the Symposium on Eye Tracking Research & Applications*, ser. ETRA ’14. New York, NY, USA: ACM, 2014, pp. 231–234.

H. Uwano, M. Nakamura, A. Monden, and K. ichi Matsumoto, “Analyzing individual performance of source code review using reviewers’ eye movement.” in *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, K.-J. Räihä and A. T. Duchowski, Eds. ACM, 2006, pp. 133–140. [Online]. Available: http://dblp.uni-trier.de/db/conf/etra/etra2006.html#UwanoNMM06

H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, “Analyzing individual performance of source code review using reviewers’ eye movement,” in *Proceedings of the 2006 symposium on Eye tracking research & applications*, ser. ETRA ’06. New York, NY, USA: ACM, 2006, pp. 133–140.

W. N. Venables and B. D. Ripley, *Modern applied statistics with S-PLUS.* Springer, 2002.

A. Walenstein, “Observing and measuring cognitive support: Steps toward systematic tool evaluation and engineering,” in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, ser. IWPC ’03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 185–. [Online]. Available: http://dl.acm.org/citation.cfm?id=851042.857038

——, “Cognitive Support in Software Engineering Tools: A Distributed Cognition Framework,” Ph.D. dissertation, Simon Fraser University, Canada, 2002. [Online]. Available: http://citeseer.ist.psu.edu/534908.html

B. Walters, M. Falcone, A. Shibble, and 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*, ser. TEFSE ’13, May 2013, pp. 51–54.

B. Walters, T. Shaffer, B. Sharif, and H. Kagdi, “Capturing software traceability links from developers’ eye gazes,” in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC ’14. New York, NY, USA: ACM, 2014, pp. 201–204.

G. M. Weinberg and E. L. Schulman, “Goals and performance in computer programming,” *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 16, no. 1, pp. 70–77, 1974.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering: An Introduction.* Norwell, MA, USA: Kluwer Academic Publishers, 2000.

S. Yusuf, H. Kagdi, and J. I. Maletic, “Assessing the comprehension of uml class diagrams via eye tracking,” in *Proceedings of the 15th IEEE International Conference on Program*

Comprehension, ser. ICPC '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 113–122. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2007.10

H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Information and Software Technology*, vol. 53, no. 6, pp. 625–637, 2011.