AN EMPIRICAL STUDY INVESTIGATING SOURCE CODE SUMMARIZATION USING MULTIPLE SOURCES OF INFORMATION

by

Sanjana Sama

Submitted in Partial Fulfilment of the Requirements

for the Degree of

Master of Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

May, 2018

Sanjana Sama

Signature:

Sanjana Sama, Student	Date

Approvals:

Bonita Sharif, Thesis Advisor	Date
Abdu Arslanyilmaz, Committee Member	Date
Feng Yu, Committee Member	Date
Salvatore A. Sanders, Dean of Graduate Studies	Date

Abstract

Software developers depend on good source code documentation to understand existing source code to perform various tasks such as fixing bugs and implementing new features. Manual documentation by developers is often missing or outdated. Past research has suggested automatic code summarization tools to remedy this problem. While several works on source code summarization leveraged only source code to generate summaries, very little work exists on using other information sources. A lot of tacit knowledge is exchanged between developers in discussion forums or bug reporting sites that can be very useful for summarization. The novelty of our work is that we conducted an empirical study using eye tracking equipment to investigate the effects of four different types of information sources namely, source code, Stack Overflow, bug reports and their combination on code summarization, to understand how developers perform using these multiple sources of information during code summarization tasks. Each participant is asked to summarize four code elements in their own words using different contexts. We evaluate the summaries against a human oracle to find similarities and analyze the developers' eye gaze patterns to see what they look for and how they switch between different contexts. Our results indicate that Stack Overflow and bug reports are as helpful as source code in supporting code summarization tasks. Participants were more confident when using Stack Overflow and bug reports when compared to source code. The results of our study can be useful to researchers and practitioners interested in building context-aware code summarization tools that can help augment official documentation with the insightful information extracted from multiple sources.

Acknowledgements

Firstly, I would like to express my true thankfulness to my advisor, Dr. Bonita Sharif. I feel very honored to work with her whose motivation, understanding, and patience helped me to complete my research. I appreciate her vast knowledge and skills in many areas and her assistance in conducting study and writing my thesis report. I could not have imagined having a better advisor and mentor for my Masters. I will forever be thankful to her.

I would like to thank the esteemed members of my committee, Dr. Abdu Arslanyilmaz and Dr. Feng Yu for the assistance they provided in this research project.

A special thanks to my family for all the motivation, support given by them and the reason being for my graduate career. I would like to thank people who directly or indirectly helped me to conduct the study. I would also thank the Department of Computer Science and the STEM College for the financial assistance during my graduate studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software developers spend a lot of time reading source code and its documentation. (Paige Rodeghero et al. 2014). They need to understand the existing source code to perform various tasks such as fixing bugs and implementing new features. When developers perform tasks like fixing bugs, they typically need to read large amounts of code. The process is very complex and time consuming. It becomes even more difficult when the code is not properly documented. When working in teams, project documentation plays a key role to understand code written by other developers. To understand the existing code, software developers refer to multiple information sources including source code, official documentation or informal documentation such as bug reports, Stack Overflow, discussion forums etc. These sources give the purpose, context, implementation and usage of source code. Manual documentation of code is very expensive to develop, is very time consuming and often becomes outdated as soon a new line of source code is written. To help developers quickly understand the code elements, past research has suggested automated code summarization tools. But most of the tools they generate summaries based on the source code, which generates summaries that give purpose and usage of source code but not the context of code. However, a lot of tacit knowledge exchanged between developers in discussions that occur in informal settings. Not much work has been done using those informal settings in code summarization field. The goal of our study is to investigate the effect of multiple sources of information on code summarization tasks. The results of our study indicate that Stack Overflow (SO) and bug reports (BR) are as useful as source code in supporting code summarization tasks. The tools that are built by using the information extracted from these sources may generate context-aware summaries which may be more helpful for developers.

1.1 Motivation

Programmers spend a considerable amount of time in reading and navigating through the source code to understand it. It is complex and very time-consuming task. Manual documentation by developers is usually incomplete because of rapid changes occuring in software, which is not sufficient to understand the existing source code. Several works have been done in field of automatic code summarization. Actual research in this area has initiated a half-century ago. But the urge and need for automatic document generation is increasing which resulted in many published papers in this area. In initial stages summaries are generated by applying natural language processing techniques to source code (the processors which are used to automatically generate summaries for English text).(Haiduc, Aponte, and Marcus 2010). The generated document is of simple English statements that explain the functionality of the code. Later summarization techniques worked based on selected subset of the statements and keywords from code. The quality of summary is dependent on the selecting the subset. There are no specific standards followed to generate proper documentation. Various research works resulted in the relevant description of code by explaining the context in which different keywords and methods used in source code. Various techniques have been developed to automatically generate the document. Many tools like Doxygen and

Javadoc have been developed to generate automatic summaries. But most of them depend on specially formatted data which is provided by programmers. Several works on code summarization they depend on the source code to generate summaries. These summaries explain the purpose of the code elements and their functionality but not the context in which they are used. Software developers discuss code elements via various other forums. By using the information from other sources along with source code to build automatic code summarization tools may help us generate more context-aware summaries.

Our goal is to investigate the effect of multiple sources of information during code summarization tasks. An empirical study was conducted to understand how developers perform code summarization tasks when using multiple sources of information, how developers navigate between multiple sources of information while performing code summarization tasks and what they look at when searching for information in various resources. By analyzing these various resources, we might find the sources that are useful and helpful for supporting code summarization tasks. The insightful information from that resources can be helpful to create better automated tools that generate more useful documentation.

1.2 Contributions

The main contribution of this thesis is an empirical study conducted to understand how developers perform when using multiple sources of information during code summarization tasks. Data collection was done using a Tobii TX-300 eye tracker. Subjects were students at Youngstown State University. The goal is to measure the effects of four different types of information sources namely source code, Stack Overflow, bug reports, and their combination on code summarization. The contributions of this study are listed as follows:

- First study to look at how different informal sources play a role in summarization
- First study to use eye-tracking on Stack Overflow and bug reports
- Collection of 10.27 hours of eye tracking data from three different information sources
- Bug reports and Stack Overflow are important sources for code summarization

1.3 Research Questions

The thesis presents the following research questions.

RQ1: How do developers perform when using multiple sources of information for code summarization tasks?

RQ2: How do developers navigate between multiple sources of information?

RQ3: What do developers look at when searching for information in source code, Stack Overflow, and Bug reports?

RQ4: Which source of information is the most preferable for developers while they are summarizing code?

1.4 Organization

This thesis is organized as follows. The next chapter gives a brief introduction to source code summarization related works and eye-tracking related work. Chapter 3 presents the details of the experimental design, the process followed, and tasks used for the study. Chapter 4 discusses observations and results. Chapter 5 concludes the thesis and presents future work. Parts of this thesis will be submitted to peer-reviewed conferences and journals.

1.5 Acknowledgements

We would like to thank all the participants who took the time to be part of our study. This work has been funded in part by the National Science Foundation under Grant Numbers CCF 1553573 and CNS 17-30307/30181.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter presents an overview of various work done on automated code summarization tools. Next, the chapter presents related work in eye tracking and its use in the software engineering and other domains.

2.1 Code Summarization Techniques

In initial stages of research, code summarization is a challenging task. In 2010, Haiduc et.al proposed a technique for automatic code summarization by using lexical and structural information. (Haiduc, Aponte, and Marcus 2010). Text retrieval techniques which are commonly used for natural processing languages have been used to extract most important terms from the source code. The structural information, which describes the context of different methods and their use, parameters, and type of variables was used. They considered structural information as important source to generate good summary. Structural information along with retrieved information (keywords and description about them) are included to generate final document. Their work even reflects domain semantics. Methods which are used to generate document are evaluated in two ways. Internal quality of generated summaries was evaluated using the pyramid method. Its goal is to determine if generated summary expresses same content as the manual summary. The second test is done determine the utility and usability of summaries. They conducted a study with six graduate students on 12 java methods from different classes. They have methods with different properties. Comparision was done between the summaries generated by the proposed tool, manual summaries and pyramid score was given for certain methods. The results stated that by adding more structural information into the document will increase the quality of summaries. From their work, they learned TR methods also work for source code and more structural information in summaries results in a good summary.

There are no specific standards followed to generate good source code documentation. Different tools use different approaches and contexts for generating the summaries. There is no clear understanding of which characteristics to be considered to generate a good summary. McBurney in one of his papers has focused on this issue (McBurney 2015). He proposed three specific research objectives and approaches to those objectives to generate a good summary. The first objective is to answer how much similar should be the text in summaries to the text and keywords in the code. He answered by testing the textual and semantic similarity of the summary and the source code. He conducted a study with author (the person who wrote the code) and readers (the persons other than author who reads the code) he found reader summary is similar to code than the author summary. After analysis he told that similarity between the summary and source code is not required for good documentation. The second objective is to answer whether contextual information about source code inclusion improves the summary quality. The author has developed a tool which summarizes the java methods by considering its methods signature, context, and usage. The tool utilizes the call graph to get all methods, use page rank to prioritize the methods and generate the summary. He compared sumslice with Sridhara et al's tool. (G. Sridhara, Pollock, and Vijay-Shanker

2011). (Giriprasad Sridhara et al. 2010). (Giriprasad Sridhara, Pollock, and Vijay-Shanker 2011). He conducted a study in which programmers were asked to rate the summaries based on accuracy, content adequacy, and conciseness. Result Metrics stated summary with context and usage information of methods are better to understand when compared to plain textual summaries of source code.

The third objective the author proposed is, the problem of similarity in source code structure and source code summarization. Author has explained the source code structure and document structure. He used an approach which uses classification algorithm to generate a hierarchy of methods. More general methods are focused higher in the tree and specific methods at end of the tree. Conducted a preliminary study to test the above objective, the results stated that tool developed by the author has helped programmers in the better understanding of methods.

Another technique related to automatic code summarization which was proposed by McBurney and McMillan is automatic document generation via source code summarization of method context. (McBurney and McMillan 2014). Summaries generated by using automatic source code summarization tools usually help us to understand the purpose of method, functionality of it. But how interaction occur between various methods and their usage cannot be understood. The automatic summaries generated by using proposed technique will help us to understand the purpose of the method and its functionality, how to use them and how they interact with other methods. The approach followed to achieve this is first they used PageRank to find the most important methods in the source code. Later they used Software Word Usage Model (divides statements into verbs, nouns Etc) and extracted keywords that describe the action performed by important methods. Finally uses NLG to generate English statements describing the functionality of method. A case study was conducted to evaluate their approach and compared it to state-of-the-art. Study was conducted using 12 participants, each participant was asked to study summaries and answer to what extent both summaries differ and to what extent they included important information, excluded unnecessary information. The answers by participants were in favour of authors' approach.

McBurney et al. worked on to improve Topic model for better code summarization of Java source code. This was mainly to improve the automatic source code summarization for Java. They have created a Topic model from java source code as the graph where methods from source code are nodes and method calls are represented as edges. (McBurney et al. 2014). Hierarchical Document Topic Model (HDTM) is used with the graph. Their approach generates a hierarchy which gives us different levels of details. More generalized method is on the top of the hierarchy. Highest functionality represented on the top and lowest details at the bottom. A hierarchical structure of topics generated and displayed in web interface. A study was conducted by three programmers and asked them to examine the methods randomly picked in source code from various projects like Siena, nanoXML, jTopas, and jEdit and asked to answer questions how exactly the given words describe the functionality of methods. Do they agree that they understood the purpose of the method, and if they were asked to select any five keywords which one they would choose? The results stated that their approach does not include all necessary keywords to generate better summaries and suggested to include some necessary keywords to improve the summary.

Fowkes et al. have proposed TASSAL (Tree-based Auto-folding Software Summarization Algorithm). The main purpose of this tool is to help developers who read the unfamiliar code and try to understand and makes it easier for them. It achieves it by folding away less informative regions of source code and allowing them to focus their efforts on more informative one. (Fowkes et al. 2016). The input to the tool is set of source files along with the desired compression ratio. The output will be summary of each file with uninformative regions being folded. To achieve this first find the suitable regions to be folded. The tool determines the most informative region to unfold while achieving the given desired compression ratio. The region which is not important is folded and represented by the symbol. By doing this reader can spend time on the important region than the overall project.

TASSAL is evaluated against simple heuristic folding methods such as Shallowest folding, Javadoc, Largest folding techniques. All four systems were allowed folding code blocks and comments. 6 experienced developers from the computer science field were asked to rate brevity, completeness and usefulness of each summary. The evaluation demonstrates that this method is favored by experienced developers over other methods currently used in different IDEs.

To automatically generate the code summarization of complex artifacts Moreno proposed a method by combining natural language processing and static analysis, and software repository mining. (Moreno 2014). In automatic code summarization of Java classes, the aim of generated summaries to support quick understanding of class by describing its design and ignoring context. Stereotypes generated for classes and methods are used along with predefined rules are included in the summary. Final generated summary consisted of the overview of the class which is obtained from interfaces and stereotypes, details about the structure of the class, its behaviour, and list of existed inner classes. A case study was conducted with 22 programmers and was asked to access the content adequacy, conciseness, and expressiveness. The results are 69% contain important information and 96% are concise, readable and understandable.

He also proposed ARENA (Automatic Release Notes Generator) it considers information from different sources like bug tracker, commit logs and generate notes. The generation of notes done in different stages. (Moreno et al. 2014). In the first stage, the changes made between two subsequent releases are recorded. In second, the structural changes are hierarchically organized and prioritized. It next gets information about fixed bugs, improvements and new features. A case study was conducted with 10 independent evaluators, they have compared 8 original release notes with the generated one. The results are 86% of original notes are appeared from the generated one. 88% suggested to include the additional information generated by ARENA. Another study conducted with 38 professionals and open source developers who accessed the original and generated release notes and were asked to compare the presence of important information. Results were most of the information is present in generated one and one case suggested the increase in the level of granularity of modifications would result in even better summary.

Change Scribe tool proposed by Mario Linares-Vasquez et.al aimed at assisting developers when committing changes by automatically generating commit messages. (Linares-Vásquez et al. 2015). This tool implements the Summarization based approach. Delta doc (Buse and Weimer 2010) and Arena (Moreno et al. 2014) are tools similar to Change Scribe that generates the program changes. It acts as eclipse plugin and works with Git-based repositories. It describes the source code changes as for each row inserted, modified and deleted code snippet generates descriptive phrases for class/method/statement. It also gives the overview of commit changes. The tool uses Change distiller to get the Fine-grained node modifications. It has changes subset and fine-grained changes it uses them to generate general descriptions and detail descriptions. It even uses JStereocode tool. It used to document changes of Java methods. It uses an elegant heuristic depend on impact analysis. In the description, it includes only the classes which have a higher impact due to large changes. The threshold value which is provided to decide which classes to be included whether higher or lower. Change Scribe able to describe initial commits and descriptions. This tool can be downloaded as eclipse plugin.

Most of the results in papers, journals, conferences, scientific researches are depicted in form of document elements, searching those elements always not results in relevant information. Bhatia et.al in their work proposed method to generate synopsis for document elements (tables, algorithms, figures) searched. (Bhatia, Lahiri, and Mitra 2009). By generating such synopsis saves end users time to analyze and compare the results. Scientists they spend most of the time in searching for document elements and comparing the results with others. When they try to search, they may get correct results, but analysis of elements will be difficult. By generating synopsis analysis and search becomes easier. To create synopsis initially they pre-processed the document and proposed a grammar to distinguish the captions of elements from other sentences in the document. With good writing style, they can find reference document related to the element. They performed some mathematical calculations and selected sentences based on content features and context. To maintain the size sentences selection is done. By selecting generated synopsis for elements in digital documents.

2.2 Eye Tracking Overview

Eye tracking has been studied in computer fields and non-computer fields. It is a process of measuring the position of eyes where a person is looking at or motion of an eyes relative to the head. An eye tracker is a device that helps to detect eye positions and eye movements. We can also know the amount of time spent looking at elements (duration) and order of gazes too. Eye trackers monitor persons visual attention by tracking eye movement data. Eye tracker equipment captures various types of eye movements. *Fixation,* looking at an element for a certain amount of time, and the *saccade*, which is a quick movement of the eyes from one part to another. *Saccades* and *fixation* are the types of eye movements which let us know where the reader is looking at. (Busjahn et al. 2015). During a *saccade,* it would be recorded as a blur because the eyes would be moving very fast across the stable visual stimulus. Visual information is received from *fixations*. Eye tracking is a source of valuable information, which cannot be obtained by other methods. For instance, a programming instructor will ask students to write answers after tracing code or debugging a program code. We can know the final result after performing a specific task and the amount of time spent doing the task, that will not help us to get information that is necessary to understand how and why a student chooses an answer. (Busjahn et al. 2014). This was discussed in their paper. Discussion of various studies using eye tracking equipment in different fields is presented below. First sub-section presents eye tracking in software engineering field then next sub-sections presents eye tracking studies in program comprehension and code summarization areas.

2.2.1 Eye-tracking Studies in Software Engineering

Software Engineering is an engineering field that is related with all aspects of software production. The modern information system is formed by software systems. Most of these are very complex systems. These software systems concern the advancement of new, or modification of existing technologies to help software engineering activities. Let us now discuss some research studies related to this field.

An eye tracking was conducted to study the effect of two layouts schemes for class diagrams in the context of identifying classes and their roles in design pattern with respect to effectiveness, efficiency and visual effort. (Sharif and Maletic 2010b). This study is a replication of an online questionnaire-based study. This was conducted using students and faculty of Kent state university. Visual effort is determined using eight measures and provides an objective metric to measure the quality of UML class diagrams. The results of the study reports that accuracy rate is high for role detection in multi-cluster layout in case of Strategy pattern. With Observer pattern higher accuracy is for Observation role and less amount of spent on task in the multi-cluster layout.

A plugin called iTrace, an Eclipse plugin that records eye movements of developers while they work on change tasks. It is the first eye-tracking environment that makes plugin possible to conduct eye-tracking studies on huge systems. (Sharif, Shaffer, et al. 2015). An overview and design of iTrace plugin were discussed in the paper. iTrace design is so flexible to record eye gazes on various types of software artifacts such as Java code, text/html/xml documents, and diagrams. They have conducted a study with 22 developers using iTrace as Eclipse plugin. It was compared with mylyn interaction history data and results showed that it captures more contextual information about source code elements from different aspects of developer's activity when compared to interaction data. Their main contributions are an eye-aware eclipse plugin, easy to comprehend gaze export format for source code and demonstrated usage of iTrace for software traceability and program comprehension tasks.

A systematic literature review (SLR) was performed covering eye-tracking studies published from 1990 to end of 2014 in software engineering field. This SLR investigates the uses of eye-tracking technology in software engineering. (Sharafi, Soh, and Guéhéneuc 2015). To analyze the studies, they have performed automated search and found 649 publications, they identified 35 relevant papers related to uses of eye-tracking in SE. The results of eye-tracking studies determine how participants perform different software engineering tasks and how they use different models and representations along with the source code to understand software systems. The analysis was done on eye-tracking studies and they were categorized into model comprehension, debugging, traceability, code comprehension, and collaborative interactions. They provided general recommendations for the SE community and suggestions for researchers who are interested and new to this area.

Investigation about the behavior of developers while doing a change task was done by Sharif, et al. (Sharif, Kevic, et al. 2015). This is the first study that collects simultaneously both eye-tracking and interaction data while developers performing tasks. The study was conducted with 22 developers, they were asked to work on three change tasks of JabRef open source system. The approach they developed for study automatically links eye gazes to source code entities in the IDE and supports scrolling and switching behavior of normal IDE. The analysis shows that the gaze data collected by the eye tracker contains more data than the interaction data. Developers working on a real time change tasks look at very few lines within a method when working on single methods. When comes to switch between methods developers chase variables flows within the methods, rarely follow call graph links and mostly only switch to the elements close to the method within the class.

2.2.2 Program Comprehension

When developers working in integrated environments, receive compiler error messages through a variety of textual and visual mechanisms. Error messages can be used to communicate the defects with developers. Researchers have very limited knowledge of how developers use these error messages to resolve the defects in source code. Barik and et.al conducted an eye tracking study with 56 participants to understand how developers read the error messages and how they use them to resolve the defects . (Barik et al. 2017). The participants were asked to resolve common yet problematic defects in Java code within Eclipse environment. A comparison was made between source code reading, error messages, and prior work on silent reading. They found that participants read error messages, but they find difficult to read error messages when compared to the source code. They also found that developers they spend considerable amount time on error messages even though there is only single error message present in the task. They found several problematic areas in the way development environments presents the compiler messages which are making them difficult. By addressing those problems results in improving the compiler error messages for developers.

Does programming language affect a person ability to read and understand source code? To investigate about this an eye tracking study was conducted comparing the languages C++ and Python by Turner et.al. Eye gazes of 38 participants were tracked while performing overview and find-bug tasks. Results on task accuracy, speed, and visual effort are reported. (Lazar et al. 2014). They found no statistical difference between C++ and Python languages with respect to accuracy and speed. But there was a significant difference between two languages in fixation rate on buggy lines of codes while performing the find-bugs task.

Program comprehension plays an important role for developers. Understanding the source code is important for the learners. An empirical study was conducted by Sharif et.al. to determine if identifier naming conventions such as camelCase and underscore affect source code comprehension and readability. (Sharif and Maletic 2010a). They have used two main styles of identifiers to determine if one is significantly better than others. They have used eye tracker to analyze the effect of identifier styles on accuracy, time, and visual effort with respect to the task of recognizing a correct identifier, when given a sentence. They have conducted study with 15 participants. They found there is no significant difference between identifier styles with respect to accuracy, results indicated that there was a significant improvement and lower visual effort when using underscore style. Novice developers performed better when using underscore whereas with experience the no difference in performance between two styles.

Identifiers play an important role in program comprehension and code readability. A study was conducted to investigate the impact of gender on the performance of developers during code reading and program understanding activities. (Sharafi et al. 2012). This is done based on subject's visual effort, time and ability to recall identifiers in source code reading. They did not find any significant difference in terms of accuracy, time and effort. They found male and female subjects follow different plan while reading. Female are more focused on the options that are given to them than male. Female subjects spent more effort on the wrong answers than male subjects. Female subjects carefully examine all options and ruled out wrong options, which lead to higher accuracy rate. whereas male they quickly made up their minds and answered possibly wrong ones.

2.3 Eye Tracking in Code Summarization

Several researches have been done in code summarization field but there are no studies on how programmers read and understand source code especially in case of summarizing that source code. Rodeghero et.al has conducted an eye-tracking study using 10 professional java programmers in which programmers were asked to read Java methods and to write English summaries of those methods (Paige Rodeghero et al. 2014). The goal of their study is to find which keywords are important from source code to be included in summary and to develop tool that generates summary using the selected keywords. They tried to answer four research questions related to areas of code that are helpful to derive important keywords. And tried to answer research questions by using various approaches. Their work contributed to program comprehension literature with the evidence from the study about the programmer's behavior during code summarization. They found programmers they tend to read certain areas of code over others and control flows which are suggested as critical to program comprehension are not read as frequently as another part of the code. Method signatures and invocations are concentrated more often. Proposed a tool to generate based on the keywords selected using eye tracking study.

An eye-tracking study was conducted to understand the patterns of eye movement during summarization tasks. The order of eye movements that people use to read are the patterns of eye movement. This study with the help of eye tracker examines where people focus when they are looking at text or images. (P. Rodeghero and McMillan 2015). They tried to answer some the research questions through their study. Do programmers read the source code from top to bottom and from left to right, similar to English text? Do programmers tend to skim or thoroughly read the code? And some other questions. They presented a qualitative and quantitative study of eye movement patterns of programmers during source code summarization. They showed that programmers read source code a little differently when compared to natural language text, programmers prefer to read source code by skimming and jumping around but not depth and sectional reading. Programmers they tend to read and summarize source code following specific patterns. Programmers also use similar patterns when compared to each other.

Rodeghero et.al conducted an eye tracking study with 10 professional Java developers to understand what developers look at when performing code summarization tasks. (P. Rodeghero et al. 2015). Source code summarization is an emerging area of research. Current research techniques they depend on the subset of statements and keywords from the source code and then include the information about that keywords and statements in the summary. The quality of the summary it depends on the subset of words selected. A subset is a high quality when it matches with the keywords and statements that are chosen by programmers while performing code summarization tasks. Not much has been done to know about the keywords and statements that programmers view as important while they are summarizing source code. The study conducted by authors gives us information about the developers' view, the findings from the study are applied to build a novel summarization tool. They evaluated this novel tool against other existing tools. They are working on the summaries by programmers to explore specific keyword usage and provide evidence to support the development of source code summarization systems.

Program comprehension is task of interpreting a program's behavior usually by reading its source code. blind programmers usually use a screen reader when reading the source code, programmers with sight can skim the code with their eyes. This difference is important that can impact the generalizability of software engineering studies and approaches. (Armaly and McMillan 2016). Authors they have investigated how code comprehension of blind programmers differs from that of programmers with sight. They have conducted a comprehension study with both sighted and non-sighted programmers. Both groups were asked to summarize the given five methods. Sighted programmers were tracked using cursor script. It records all cursor movements and commands. Blind programmers they used screen reader to read the code. they found that no statistical significant differences between the areas of code that both groups found to be important. Sighted and blind programmers both spent considerable amount of time on method signatures. The similarity between two groups suggest that productivity-enhancing tools and software engineering studies are applicable to both sighted and blind programmers, provided that any newly-developed tools are accessible.

2.4 Discussion

Software developers spend most of the time in reading and browsing the existing source code to perform various tasks, which is complex and time-consuming with improper documentation. Proper documentation helps developers to understand code elements easily and they can invest their time in devising new solutions. Manual documentation is costly and often missing or outdated. There was need to generate documentation without human intervention. Past research has suggested automatic code summarization. As stated above, most of the work they depend on the subset selected from source code to generate summaries and there are no specific standards to be followed to generate better summaries. Several tools either they depend on developers' comments about source code or selected subset of keywords from code to generate summaries. These summaries they explain about purpose and functionality of code elements used in code. But they are not completely context-aware. It means summaries do not explain in detail the context in which that code elements are used. Besides source code developers they use Stack Overflow and Bug reports to discuss the code elements. A lot of tacit knowledge is exchanged between the developers during discussions that happen in informal settings. Not much work has been done using these various resources in code summarization. My study is to investigate the effect of these multiple sources of information on code summarization. Our emerging results and findings can be useful to researchers and practitioners interested in building context-aware code summarization tools that can help augment official documentation with the insightful information extracted from various kinds of informal documentation. Students and professional developers benefit from good source code documentation for program comprehension.

The next chapter covers the details of the eye tracking study including the experimental design and task selection.

CHAPTER 3

The Eye Tracking Study

This chapter presents the details of the empirical study conducted as part of this thesis. It gives details on the experimental design, contexts used, participants data, data collection methods, tasks, and how the study was instrumented.

3.1 Experimental Design

To understand how developers, perform when using multiple sources of information during code summarization tasks. The goal is to measure the effects of four different types of information sources namely source code, Stack Overflow, bug reports, and their combination on code summarization. For this, a user study was designed to summarize API elements in given context. Each participant is asked to summarize in their own words four code elements, which include two methods and two classes. Four different contexts based on the four information sources presented above are used. We have investigated whether code summarization can leverage on other information sources rather than source code. In the study, participants were given a pre-questionnaire where demographic data is collected. Responses were collected based on five-point Likert scale. Each participant was given four code elements two methods and two classes and was asked to summarize in their own words using given context.

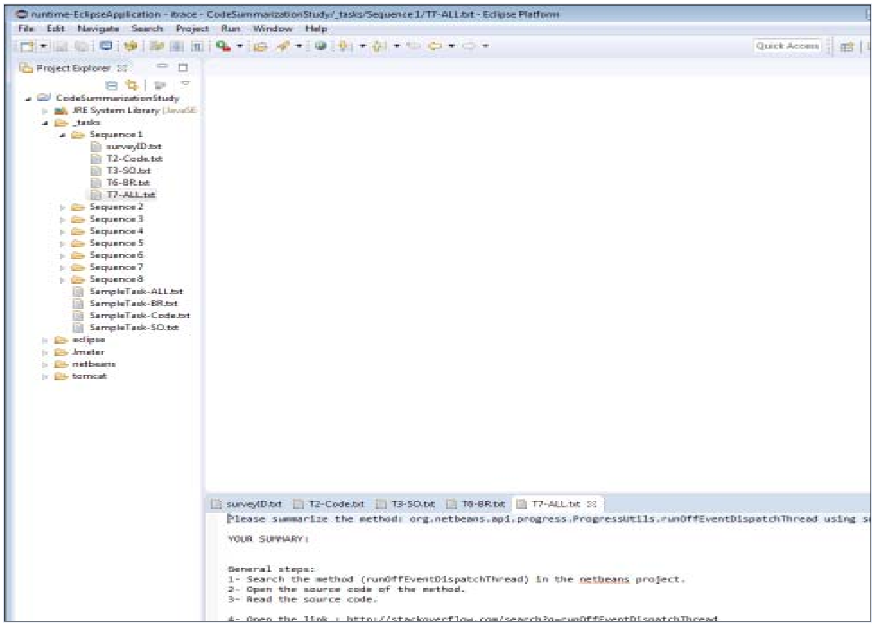
Participants were asked to summarize the code elements using following contexts:

- Only source code (which method or class should be referred will be mentioned)
- Only Stack Overflow (link to Stack Overflow will be provided in the question)

- Only bug reports (link to Bug report will be provided in the question)
- Using ALL (source code project, link to Stack Overflow and Bug reports will be provided)

Participants were instructed to understand the code in given context and summarize their view. They were also asked to answer difficulty level, the confidence level in their own degree of understanding about the given code elements after each task. After finishing the given four tasks the they will be given a post-questionnaire to answer.

A plugin called iTrace was developed in the Software Engineering Research and Empirical Studies lab. This plugin gives us eye tracking information at word level and supports scrolling. We have integrated this plugin with eclipse workspace. We have set up different systems used and task files under one eclipse project. Figure 1 shows the snapshot of code summarization study workspace. The left section contains four systems used for the study. All task files are opened in the bottom section. When using SO and BR context participants can open the link provided to them using the Eclipse internal browser.



```
Please summarize the method: org.netbeans.api.progress.ProgressUtils.runOffEventDispatchThread using s
YOUR SUMMARY:


General steps:
1- Search the method (runOffEventDispatchThread) in the netbeans project.
2- Open the source code of the method.
3- Read the source code.

4- Open the link : http://stackoverflow.com/search?q=runOffEventDispatchThread
```

Figure 1 Code Summarization study workspace screenshot

An overview of the experiment is given in Table 1. The experiment seeks to analyze summaries and eye tracking data using different contexts for purpose of evaluating their impact on code summarization tasks. Participants doing the study are instructed to read, understand the task, and the context using which they need to summarize the method/class. They need to summarize the method/class in their own words. Based on the accuracy of summary and other analysis, the usefulness of different contexts for code summarization can be understood. The independent variables for the study are different information sources such as source code, Stack Overflow, Bug reports, and ALL. Dependent variables are accuracy score, time, confidence level, level of Difficulty and Visual Effort (characterized by fixations and saccades).

Table 1. Experiment overview

Goal	Impact of multiple sources (source code, Stack Overflow, Bug reports, the combination of all) on code summarization
Independent variables	Information source (source code, Stack Overflow, Bug reports, all)
Dependent variables	Score, Time, Confidence level, Level of Difficulty, Visual Effort

Code elements that are given to participants to summarize are chosen from four different systems such as Eclipse, JMeter, Tomcat, and Netbeans. Two API elements, one method and one class are chosen from each of above mentioned systems. In total 8 code elements are selected. These code elements are selected based on their difficulty level, which is calculated by considering Lines of code (LOC) and other factors. Table 2 gives the overview of systems used, version of system, and link redirects to download system's source code.

Table 2 Systems used in the study

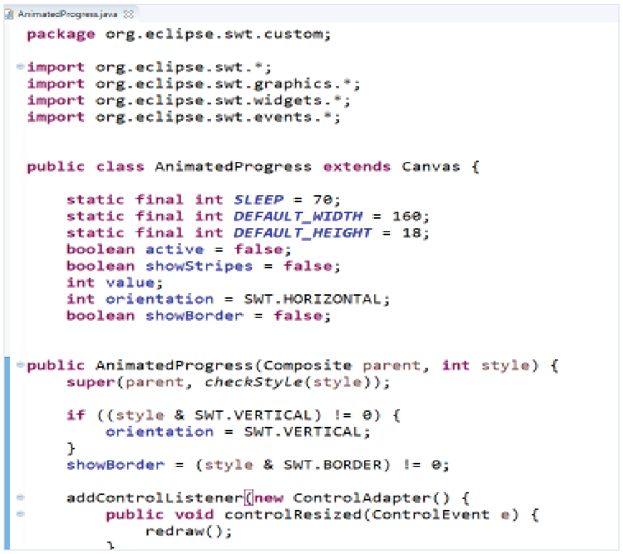
System	Version	Source code (link)
Eclipse	4.2	http://archive.eclipse.org/eclipse/downloads/drops4/R-4.2-201206081400/download.php?dropFile=eclipse-SDK-4.2-win32-x86_64.zip
Jmeter	3.2	http://jmeter.apache.org/download_jmeter.cgi
Tomcat	7	http://tomcat.apache.org/download-70.cgi
NetBeans	7.4	https://netbeans.org/downloads/7.4/start.html?filename=zip/netbeans-7.4-201310111528-src.zip&pagelang=

3.2 Contexts (Treatments) Used

Four contexts were used in our study. They are Source code, Stack Overflow, Bug reports and combination of all three sources. To complete the task given to them, participants they need to use the context stated in the task file and gather information related to code element and write the summary in their own words. By investigating the effect of these multiple sources of information on code summarization, we can find the usefulness of sources besides source code in supporting code summarization tasks. If they are as helpful as source code, the information from other sources can also be used to generate more context-aware summaries.

3.2.1 Source Code

Source Code is the code written by developers. Four open source Java projects - Eclipse, JMeter, Tomcat, and NetBeans - are used in the study. Projects are imported into Eclipse workspace. Participants when performing summarization task using CODE context. They can find the project from which code element is taken in the task file and can operate on the entire source code of that project that is placed in eclipse workspace. They can navigate through entire project to gather the required information to summarize the code element. Their access will be limited to only source code, which means they cannot use any other sources except source code to complete the task. Figure 2 It shows the sample source code file. The projects which are given to participants does not contain any type of comments in them. By looking at only source code they need to summarize the code element given to them.

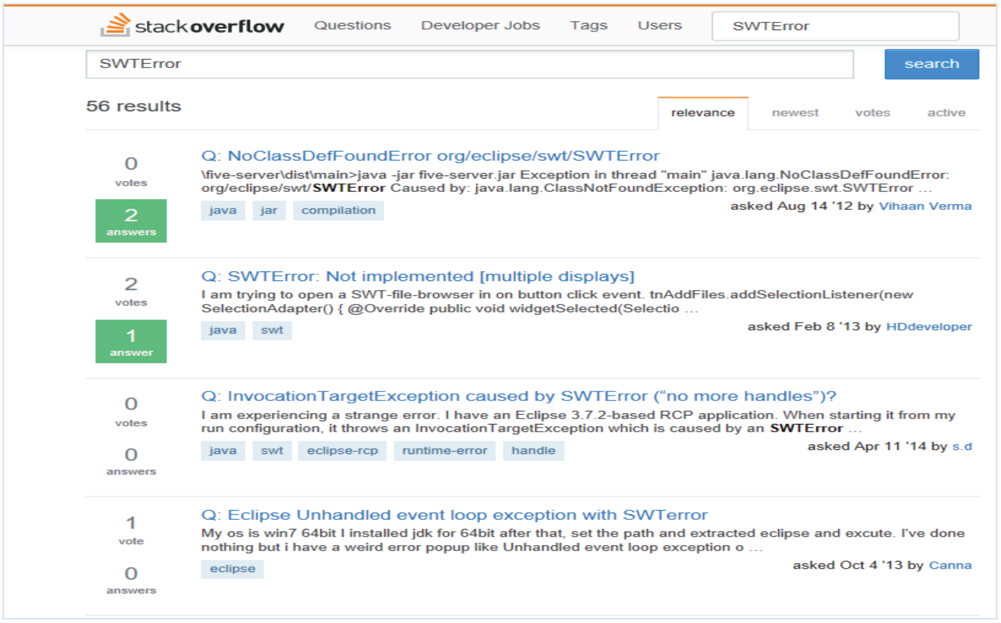


```java
package org.eclipse.swt.custom;

import org.eclipse.swt.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.*;


public class AnimatedProgress extends Canvas {

    static final int SLEEP = 70;
    static final int DEFAULT_WIDTH = 160;
    static final int DEFAULT_HEIGHT = 18;
    boolean active = false;
    boolean showStripes = false;
    int value;
    int orientation = SWT.HORIZONTAL;
    boolean showBorder = false;


public AnimatedProgress(Composite parent, int style) {
    super(parent, checkStyle(style));

    if ((style & SWT.VERTICAL) != 0) {
        orientation = SWT.VERTICAL;
    }
    showBorder = (style & SWT.BORDER) != 0;

    addControlListener(new ControlAdapter() {
        public void controlResized(ControlEvent e) {
            redraw();
```

Figure 2 Sample of source code used in the study

3.2.2 Stack Overflow

Developers they discuss code elements via various information sources, Stack Overflow is one of them. It is one of the largest and most trusted online community for developers to learn and share knowledge. It is a question and answer site used a lot by developers to discuss issues. Besides source code, this is also an important platform which developers use to discuss the code elements. Participants when asked to perform summarization task using SO context, they need to follow the instructions given to them in task file. They are provided with a link to Stack Overflow posts mentioning the given code element they can open the link using eclipse internal browser and can navigate through different pages in Stack Overflow to gather necessary information required to complete the task. When performing the task using SO context, participants will have access only to Stack Overflow and not to any other sources. Figure 3 shows the sample file of Stack Overflow site.



56 results

0 votes, 2 answers
Q: NoClassDefFoundError org/eclipse/swt/SWTError
\five-server\dist\main>java -jar five-server.jar Exception in thread "main" java.lang.NoClassDefFoundError: org/eclipse/swt/**SWTError** Caused by: java.lang.ClassNotFoundException: org.eclipse.swt.SWTError ...
java jar compilation
asked Aug 14 '12 by Vihaan Verma

2 votes, 1 answer
Q: SWTError: Not implemented [multiple displays]
I am trying to open a SWT-file-browser in on button click event. tnAddFiles.addSelectionListener(new SelectionAdapter() { @Override public void widgetSelected(Selectio ...
java swt
asked Feb 8 '13 by HDdeveloper

0 votes, 0 answers
Q: InvocationTargetException caused by SWTError (“no more handles”)?
I am experiencing a strange error. I have an Eclipse 3.7.2-based RCP application. When starting it from my run configuration, it throws an InvocationTargetException which is caused by an **SWTError** ...
java swt eclipse-rcp runtime-error handle
asked Apr 11 '14 by s.d

1 vote, 0 answers
Q: Eclipse Unhandled event loop exception with SWTerror
My os is win7 64bit I installed jdk for 64bit after that, set the path and extracted eclipse and excute. I've done nothing but i have a weird error popup like Unhandled event loop exception o ...
eclipse
asked Oct 4 '13 by Canna

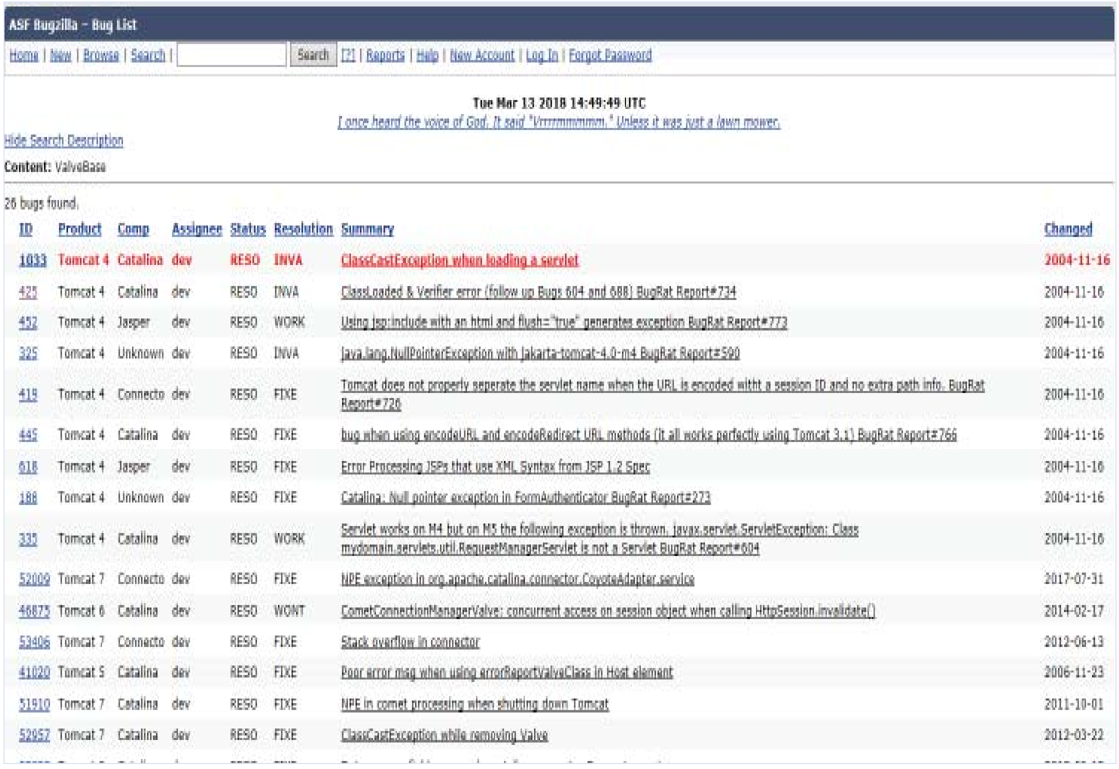
Figure 3 Sample of Stack Overflow page

3.2.3 Bug Report

It is another platform where developers discuss the code elements. Bug reports is a site where developers usually post the bugs, then fixes them. Thereby unofficially documenting the usage of code elements. Example Bugzilla is one of the famous bug reporting sites used by developers. These sites are used to keep track of bugs and generate bug report forms. The goal of my study is to find the effect of bug reports in code summarization. How developers perform when using only bug reports and combination all resources while doing code summarization tasks. These questions are answered in our research questions.

Participants when asked to summarize the given code element using BR context. They can follow the general steps in the task file to complete the task. They are provided with a link to bug reports discussing the code elements.

They can open the link using eclipse internal browser and gather the necessary information to summarize the given code element. They can navigate through various pages in bug reports, but their access is limited to only bug reports. Figure 4 shows the sample bug report


ASF Bugzilla – Bug List
Home | New | Browse | Search | Search | [?] | Reports | Help | New Account | Log In | Forgot Password

Tue Mar 13 2018 14:49:49 UTC
I once heard the voice of God. It said "Vrrrrmmmmm." Unless it was just a lawn mower.

Hide Search Description

Content: ValveBase

26 bugs found.

ID	Product	Comp	Assignee	Status	Resolution	Summary	Changed
1033	Tomcat 4	Catalina	dev	RESO	INVA	ClassCastException when loading a servlet	2004-11-16
425	Tomcat 4	Catalina	dev	RESO	INVA	ClassLoaded & Verifier error (follow up Bugs 604 and 688) BugRat Report#734	2004-11-16
452	Tomcat 4	Jasper	dev	RESO	WORK	Using jsp:include with an html and flush="true" generates exception BugRat Report#773	2004-11-16
325	Tomcat 4	Unknown	dev	RESO	INVA	java.lang.NullPointerException with jakarta-tomcat-4.0-m4 BugRat Report#590	2004-11-16
419	Tomcat 4	Connecto	dev	RESO	FIXE	Tomcat does not properly seperate the servlet name when the URL is encoded witht a session ID and no extra path info. BugRat Report#726	2004-11-16
445	Tomcat 4	Catalina	dev	RESO	FIXE	bug when using encodeURL and encodeRedirect URL methods (it all works perfectly using Tomcat 3.1) BugRat Report#766	2004-11-16
618	Tomcat 4	Jasper	dev	RESO	FIXE	Error Processing JSPs that use XML Syntax from JSP 1.2 Spec	2004-11-16
188	Tomcat 4	Unknown	dev	RESO	FIXE	Catalina: Null pointer exception in FormAuthenticator BugRat Report#273	2004-11-16
335	Tomcat 4	Catalina	dev	RESO	WORK	Servlet works on M4 but on M5 the following exception is thrown. javax.servlet.ServletException: Class mydomain.servlets.util.RequestManagerServlet is not a Servlet BugRat Report#604	2004-11-16
52009	Tomcat 7	Connecto	dev	RESO	FIXE	NPE exception in org.apache.catalina.connector.CoyoteAdapter.service	2017-07-31
46875	Tomcat 6	Catalina	dev	RESO	WONT	CometConnectionManagerValve: concurrent access on session object when calling HttpSession.invalidate()	2014-02-17
53406	Tomcat 7	Connecto	dev	RESO	FIXE	Stack overflow in connector	2012-06-13
41020	Tomcat 5	Catalina	dev	RESO	FIXE	Poor error msg when using errorReportValveClass in Host element	2006-11-23
51910	Tomcat 7	Catalina	dev	RESO	FIXE	NPE in comet processing when shutting down Tomcat	2011-10-01
52957	Tomcat 7	Catalina	dev	RESO	FIXE	ClassCastException while removing Valve	2012-03-22

Figure 4 Sample of Bug reports page

3.3 Participants

We recruited graduate and undergraduate students at YSU. The minimum requirement of a participant is to have knowledge of Java programming. There were 12 undergraduate students and 3 graduate students.

All the code elements given to participants to summarize are selected from four different open source Java projects. In pre-questionnaire, participants were asked to selfassess their skills in Java programming, industrial experience in years and the IDE which they are familiar with. In the post questionnaire, they were asked questions related to their experience in the study. See the appendix for details on the questionnaire materials. All participants were compensated with a $25 gift card.

Table 3 Characteristics of Participants

Characteristics	**Level**	**# of Participants**
Study program	Bachelor	12
	Master	3
Active programming experience	Between 6 & 10 years	2
	Between 3 & 5 years	7
	Between 1 & 2 years	5
Knowledge in Java	Poor	1
	Fair	8
	Good	4
	Very Good	2
Familiarity with Stack Overflow	Yes	14
	No	1
Familiarity with Bug repositories	Yes	10
	No	5

From the above Table 3 one can understand that there are 12 bachelor students 3 graduate students in participants. More than half of the students have greater than two years of active programming experience. Only one participant have poor knowlegde in Java programming, rest all the participants have fair to very good knowledge in Java. 14 out of 15 participants used Stack Overflow to find solutions to their coding problems when working on projects. 10 out of 15 participants use/read bug reports to find solutions to their issues encountered while coding.

Participants in our study are divided into two groups Novice and Non-novice based on certain criteria. Each of our research questions were answered by using data of overall participants and also by comparing the novice and non-novice group data.

The criteria used to divide into groups is based on the pre-questionnaire data by participants and accuracy of summaries given by participants. The participants with Fair to Very Good Java programming knowledge and having more than one year of active programming experience and whose summaries does not consists of more than one incorrect summary are considered to be Non-novice. Participants who do not follow at least one of the rules from above criteria fall into Novice group. Out of 15 participants we have 6 in Novice group and 9 in Non-novice group. The table below shows the participants ID and the group they fall into.

Table 4 Participants in Novice and Non-novice group

ID number	Accuracy of summaries	Expertise
IDY1	2 F, 2 P	Non-novice
IDY2	3 P, 1 N	Non-novice
IDY3	2 P, 2 N	Novice
IDY4	2 F, 1 P, 1 N	Non-novice
IDY5	1 F, 1 P, 2 N	Novice
IDY6	1 F, 1 P, 2 N	Novice
IDY7	3 P, 1 N	Novice
IDY8	1 P, 3 N	Novice
IDY9	3 F, 1 P	Non-novice
IDY10	2 P, 2 N	Novice
IDY11	2 F, 2 P	Non-novice
IDY12	1 F, 2 P, 1 N	Non-novice
IDY13	3 P, 1 N	Non-novice
IDY14	1 F, 3 P	Non-novice
IDY15	2 F, 2 P	Non-novice

In Accuracy of summaries column, the F, P, and N denote F- Fully Correct, P- Partially Correct, N-Not Correct. In Table 4, the greyed cells represent participants fall in novice group and while cells represent the participants in the non-novice group.

Participants they are divided into groups in order to understand the difference in their approach to analyze their performance when using different contexts.

3.4 Tasks

Code elements used in the study are chosen from four different open source projects named Eclipse, Tomcat, Netbeans, Jmeter. Selection of API elements, methods and classes are done by considering the level of difficulty, lines of code and other factors. The experiments are divided into four tasks. In each task, the participant will be asked to summarize the given method or class from any of the four projects using the context mentioned in task. One method and one class are selected from four Java projects. We have generated 8 sequences and in each sequence 3 versions (by changing the order of tasks given to participants). Each sequence contains four tasks T-SO, T-BR, T-CODE, and T-ALL. Task T-SO implies the participant needs to summarize the given method/class using only Stack Overflow context. They will not have access to either source code or any other sources except Stack Overflow. Link to Stack Overflow related to that method/class will be provided in the task file. Task T-BR implies that the participant needs to summarize the given method/class using only Bug reports context. They will not have access to either source code or any other sources except Bug reports. Link to Bug reports related to that method/class will be provided in the task file. Task T-CODE implies that the participant needs to summarize the given method/class using only source code. They can search for method/class in the related project in Eclipse project. They will not have access to either Stack Overflow or any other sources except source code. Task T-CODE implies that the participant can summarize the given method/class using the source code, Stack Overflow, and Bug reports. They can access all three sources to summarize the code elements. They will provide with source code project, links to Stack Overflow and Bug reports related to the method/class. Table 5 below shows different code elements used in the study. 8 code elements were chosen from four systems.

Table 5 Different tasks used in the study

Task ID	Projects	Version	Package	API Level
T1	Eclipse	4.2	`org.eclipse.core.databinding`	Method
T2	Eclipse	4.2	`org.eclipse.swt`	Class
T3	JMeter	3.2	`org.apache.jmeter`	Method
T4	JMeter	3.2	`org.apache.jmeter.samplers`	Class
T5	Tomcat	7	`org.apache.catalina.realm`	Method
T6	Tomcat	7	`org.apache.catalina.valves`	Class
T7	Netbeans	7.4	`org.netbeans.api.progress`	Method
T8	Netbeans	7.4	`org.openide.nodes`	Class

Different API elements from various systems are selected based on difficulty level and Lines of code. Table 6 below shows different tasks used in the study, their API Level, difficulty level and LOC.

Table 6 Tasks and their difficulty level and LOC

Task ID	Projects	API Level	Difficulty	LOC
T1	Eclipse	Method	Difficult	18
T2	Eclipse	Class	Medium	123
T3	JMeter	Method	Medium	58
T4	JMeter	Class	Medium	1523
T5	Tomcat	Method	Difficult	59
T6	Tomcat	Class	Difficult	245
T7	Netbeans	Method	Difficult	42
T8	Netbeans	Class	Difficult	27

Different sequences used in the study are shown in Table 7. We have a total of 8 sequences. Each sequence contains 4 tasks one in each context. Note that the same method/class are not given to the same participant for all three contexts (source code, Stack Overflow, Bug reports, and a combination) due to learning effects. In each sequence, we have different tasks for every context.

Table 7 Sequence table with tasks

Sequences	API1	API2	API3	API4
Seq1	T3-SO	T6-BR	T7-All	T2-Code
Seq2	T4-BR	T1-All	T6-SO	T7-Code
Seq3	T5-Code	T4-SO	T1-BR	T8-All
Seq4	T6-All	T3-Code	T2-BR	T1-SO
Seq5	T5-SO	T2-All	T1-Code	T8-BR
Seq6	T2-Code	T5-BR	T8-SO	T3-All
Seq7	T5-All	T8-Code	T7-BR	T2-SO
Seq8	T6-Code	T7-SO	T2-All	T3-BR

Table 8 Different sequences and versions table

		API1	API2	API3	API4
Seq 1	**Version 1**	T6-BR	T7-ALL	T3-SO	T2-Code
	Version 2	T6-BR	T3-SO	T7-ALL	T2-Code
	Version 3	T3-SO	T2-Code	T6-BR	T7-ALL
Seq 2	**Version 1**	T6-SO	T4-BR	T7-Code	T1-ALL
	Version 2	T1-ALL	T4-BR	T6-SO	T7-Code
	Version 3	T4-BR	T6-SO	T1-ALL	T7-Code
Seq 3	**Version 1**	T4-SO	T1-BR	T5-Code	T8-ALL
	Version 2	T1-BR	T4-SO	T5-Code	T8-ALL
	Version 3	T4-SO	T1-BR	T8-ALL	T5-Code
Seq 4	**Version 1**	T1-SO	T6-ALL	T2-BR	T3-Code
	Version 2	T6-ALL	T2-BR	T1-SO	T3-Code
	Version 3	T6-ALL	T3-Code	T1-SO	T2-BR
Seq 5	**Version 1**	T1-Code	T4-ALL	T8-BR	T5-SO
	Version 2	T5-SO	T8-BR	T1-Code	T4-ALL
	Version 3	T5-SO	T4-ALL	T1-Code	T8-BR
Seq 6	**Version 1**	T3-ALL	T4-Code	T8-SO	T5-BR
	Version 2	T5-BR	T3-ALL	T4-Code	T8-SO
	Version 3	T3-ALL	T5-BR	T4-Code	T8-SO
Seq 7	**Version 1**	T8-Code	T5-ALL	T2-SO	T7-BR
	Version 2	T7-BR	T2-SO	T8-Code	T5-ALL
	Version 3	T8-Code	T2-SO	T5-ALL	T7-BR
Seq 8	**Version 1**	T6-Code	T3-BR	T7-SO	T2-ALL
	Version 2	T2-ALL	T7-SO	T6-Code	T3-BR
	Version 3	T2-ALL	T3-BR	T7-SO	T6-Code

See Table 8 for various sequences and their versions which are used in the study. The tasks in each sequence are rearranged in their order to generate different versions. This was done randomly. Versions in any sequence they differ only in order of tasks. We have total 8 sequences, for each sequence we have three versions. In total, we have 24 versions for our study. We randomly choose anyone version and give it to the participant. We have make sure to cover all sequences in order to cover all tasks using different contexts.

List of different sequences and versions given to each participant during the study are shown in Table 9. Study was conducted using 15 participants and almost all the sequences repeated twice in the study.

Table 9 Sequence number and its version given to each participant

ID of participant	sequence used	version used
IDY1	sequence 1	version 1
IDY2	sequence 2	version 1
IDY3	sequence 7	version 1
IDY4	sequence 3	version 1
IDY5	sequence 6	version 1
IDY6	sequence 5	version 1
IDY7	sequence 4	version 1
IDY8	sequence 8	version 1
IDY9	sequence 3	version 3
IDY10	sequence 5	version 2
IDY11	sequence 1	version 2
IDY12	sequence 4	version 2
IDY13	sequence 8	version 2
IDY14	sequence 2	version 2
IDY15	sequence 1	version 3

We make sure that we cover all sequences to cover tasks in all contexts. Table 10 lists various tasks and contexts that were used during the study. The numbers indicate how many people were assigne to each of these treatments.

Table 10 Number of tasks in each context

	Task Number							
Context	**T1**	**T2**	**T3**	**T4**	**T5**	**T6**	**T7**	**T8**
Stack Overflow	2	1	3	2	2	2	2	1
Code	2	3	2	1	2	2	2	1
Bug Report	2	2	2	2	1	3	1	2
Code+SO+BR	2	2	1	2	1	2	3	2

Table 11 Tasks and their Stack Overflow link

Task ID	Projects	SO
T1	Eclipse	https://stackoverflow.com/search?q=databinding+dispose+
T2	Eclipse	http://stackoverflow.com/search?q=SWTError
T3	JMeter	https://stackoverflow.com/search?q=Jmeter+convertSubTree
T4	JMeter	https://stackoverflow.com/search?q=SampleResult
T5	Tomcat	http://stackoverflow.com/search?q=DeltaSession.setAttribute
T6	Tomcat	http://stackoverflow.com/search?q=ValveBase
T7	Netbeans	http://stackoverflow.com/search?q=runOffEventDispatchThread
T8	Netbeans	http://stackoverflow.com/search?q=ChildFactory.Detachable

Table 11 lists the tasks with links to Stack Overflow posts related to code element asked in the task. This links related to the code element is provided to the participants when performing the task using SO context. If the participant, they are performing T2-SO task, they are provided with the SO link related to that element, they can open the link using eclipse internal browser to complete the task.

Table 12 Tasks and BR Link

Task	Project	BR Link
T1	Eclipse	https://bugs.eclipse.org/bugs/buglist.cgi?quicksearch=binding%20dispose
T2	Eclipse	https://bugs.eclipse.org/bugs/buglist.cgi?quicksearch=SWTError
T3	JMeter	https://bz.apache.org/bugzilla/buglist.cgi?quicksearch=Jmeter%20convertSubTree
T4	JMeter	https://bz.apache.org/bugzilla/buglist.cgi?quicksearch=SampleResult
T5	Tomcat	https://bz.apache.org/bugzilla/buglist.cgi?quicksearch=DeltaSession.setAttribute
T6	Tomcat	https://bz.apache.org/bugzilla/buglist.cgi?bug_status=__all__&content=ValveBase&no_redirect=1&order=Importance&query_format=specific
T7	Netbeans	https://netbeans.org/bugzilla/buglist.cgi?bug_status=__all__&content=runOffEventDispatchThread&no_redirect=1&order=Importance&product=&query_format=specific
T8	Netbeans	https://netbeans.org/bugzilla/buglist.cgi?quicksearch=ChildFactory.Detachable

The above Table 12 shows the link to bug reports discussing the code element. These links are used in BR and ALL contexts.

3.5 Data Collection

The responses from pre-questionnaire and post-questionnaire are collected using google forms. Pre-questionnaire gives us the demographic data related to participants. post-questionnaire that helped us gain insights and feedback about the usefulness of different context when performing summarization tasks. The summaries of API elements written by participants were collected from the task files given to them. The responses of confidence level and difficulty level questions after each task were also collected from task files. There was no time constraint while performing the tasks. Each task was timed, the participants' eyes were tracked using eye tracker. TX-300 eye tracker was used to collect data for our study. It can generate up to 300 samples per second, we used 60 Hz frequency thereby generated 60 samples per second. Responses from eye tracker are collected in the form of JSON and XML files. In addition to the above, Camtasia was used to record the screen while participants performing the tasks. External video recording of participants' screen was done for each task. We obtained IRB approval (155-2017) and training before we began the study.

3.6 Eye-Tracking Apparatus

The TX-300 eye tracker was used in this study which is present in the Software Engineering and Empirical Studies Lab at the Computer Science and Information Systems Department at Youngstown State University. This is an integrated eye tracker which comes with 23'' inch TFT removable monitor. Removing the TFT monitor transforms the integrated eye tracker into stand alone eye tracker. This can generate 300 samples of eye data per second and the user does not require to wear any headgear. We used sample as 60Hz per second for our study. Its large head movement box allows the subject to move during tracking while maintaining the accuracy and precision of recorded data. A dual monitor extended desktop setting was used for the study. The monitor with integrated eye tracker was used to by participant to do the study. The second monitor is used by moderator to monitor the eye movements of the subject using Tobii pro manager. The eye tracker specifications Sampling rate (binocular) is 300 Hz, Latency Processing latency is 1.0 - 3.3 ms, Total system latency <10 ms, Timestamp precision Via sync-out port is <0.1 ms, Time to tracking recovery For blinks is Immediate, Head movement Freedom of head movement at 65 cm (width x height) 37 x 17 cm (15 x 7''), Operating distance (eye tracker to subject) 50-80 cm (20-31''), Max head movement speed is 50 cm/s (20''/s), Max gaze angle is 35°, Tracking technique is Dark pupil tracking. We have developed a plugin called iTrace in Seresl lab which can record gazes at word level and supports scrolling. We used this plugin with eclipse workspace to conduct the study.

3.7 Conducting the Study

The test was conducted in the Software Engineering Research and Empirical Studies Lab (SERESL). Participants are asked to fill the sign-in sheet. Then they are provided with the consent form and study instructions sheet. When they finish reading the instructions, they will decide whether to participate in the study or not. They can withdraw themselves if they have any concerns. After that participant are requested to sign the consent form and were asked to fill out the pre-questionnaire that include their background details, their active experience in programming, level of expertise in Java, their familiar IDE and other questions. The test can be attempted by one student at a time as the lab can have one student in at a time. Before starting the actual study, participants are shown sample tutorial which contains sample tasks one in each context. Moderator will explain how participant needs to complete the study in using different contexts. In sample tasks they were not expected to write summaries it is just to make them familiar with the environment. After showing sample tutorial, calibrations will be done to make sure that the eyes of subject are in sync with the eye tracker.

The participants are asked to maintain a position in the chair during the study so that we do not lose the tracking of their eyes. Subjects will see a red circle on the screen when the calibration is started, they need to follow the circle. Once we get good calibration results, Moderator can start the session. A good calibration appears with the green vector in the circle and not too far away from the circle. If the researcher does not find a good calibration, the re-calibration must be done. Subjects can ask questions to make sure they understand the study instructions and what was expected out of them.

Moderator is always present with the subject during the study to make sure that the eyes are always tracked which is determined by the track status in the Tobii Pro Manager keeping it on the left screen. The subjects are encouraged to think out loud while reading the methods and look at the screen all the time except when they need to type an answer keeping the head movement limited. Moderator fills the session info and then starts Camtasia screen recording, external video recording (which records the screen) and eye-tracking recording after that subject can continue and complete the task in given sequence which is opened at the bottom tab of Eclipse. We do not pause or end the session once the recording starts unless they want to withdraw themselves due to emergency and start a new session if they wish to continue. After completion of each task, moderator stops all recordings. The subject can fill the confidence level and difficulty level related to that task. A similar procedure is repeated for all the four tasks. In the next chapter, analyses and results of our study are presented.

CHAPTER 4

Analyses and Results

This chapter presents analyses and results of our controlled experiment. We have conducted the study with 15 participants and collected a total of 60 data points across all the tasks. Each of these data points was analyzed to answer our research questions. Each of the research questions and their results are presented.

4.1 RQ1: How do developers perform when using multiple sources of information for code summarization tasks?

This question is answered based on the accuracy of the summaries given by participants and the time taken to finish the task. The summaries which are written by participants are taken from task files and the total time spent in seconds to finish the task is calculated from eye tracking data.

4.1.1 Accuracy

This sub-section presents the results of participants based on accuracy. The accuracy of summaries by participants was obtained by manually evaluating the summaries given by participants against the human-generated oracle and marking them as fully correct, partially- correct and not correct.

We initially had three sets oracles of summaries for each task given by YSU, Carlton University, and ETS. After analyzing all three possible summaries, three organizations they have agreed upon one final oracle. We have used that final oracle and evaluated summaries given by participants against them and marked into three categories such as fully correct, partially correct and not correct summaries. The final oracle for each task is mentioned in the appendix study material section.

The evaluation of summaries given by participants is done is illustrated with an example. Let us consider Task-T1 used in the study. It is to summarize the method: `org.eclipse.core.databinding.Binding.dispose.`

Task-T1 is chosen from Eclipse open source project. Participants need to summarize dispose method form Binding class in their own words using given context. Below are the three summaries given by three different participants performing Task-T1 using different contexts. Final oracle is the summary given by human annotators for Task-T1 against which the summaries of participants are evaluated. Oracle used for all the tasks is given in Appendix section A.6. The three different cases mentioned below are three summaries by different participants.

Case 1: In the method dispose, the if condition is used. The first if condition is used to check the context and then assign the null value it. Secondly, disposeListener is checked and within it, target and model is checked and then assign the null value to disposeListener, target and model. Super function is imposed at the end.

Case 2: The Binding.dispose method is a method use by DataBinding classes to remove the event listener on the data.

Case 3: Dispose is a method that removes the binding with the context. For instance, in a UI thread, a dispose would destroy a widget at the termination of a program, if it's not null yet.

Final Oracle: Disposes of the Binding object this method is called from. A call of this method usually results in the binding stopping to observe its dependencies by unregistering its listener(s).

Summaries by participants are evaluated manually to improve the accuracy of grading. Human evaluation generates better results than just comparing the summary to a bag of words which in itself is a threat since sentence meaning and context is lost.

<u>Case 1 example</u>: Given a summmary by some participant A, we evaluate it against the Final oracle. It was observed that participant A has explained about the variables and conditional statements in the source code of that method instead of explaining its functionality and purpose. We felt that the summary does not explain the intent of what code element does and marked as **Not Correct**.

<u>Case 2 example</u>: If the summary implies the almost same meaning as in final oracle, even if it was not answered using same words as in final oracle, it was marked as **Fully Correct**.

<u>Case 3 example</u>: If the summary only partially explained the intent but did not cover all important aspects of the code, it is marked as **Partially Correct**.

There are total of 60 summaries given by 15 participants. There are 15 tasks performed (15 summaries collected) using each context. The figure below (Figure 5) shows accuracy of summaries given by participants which are grouped by four different contexts.

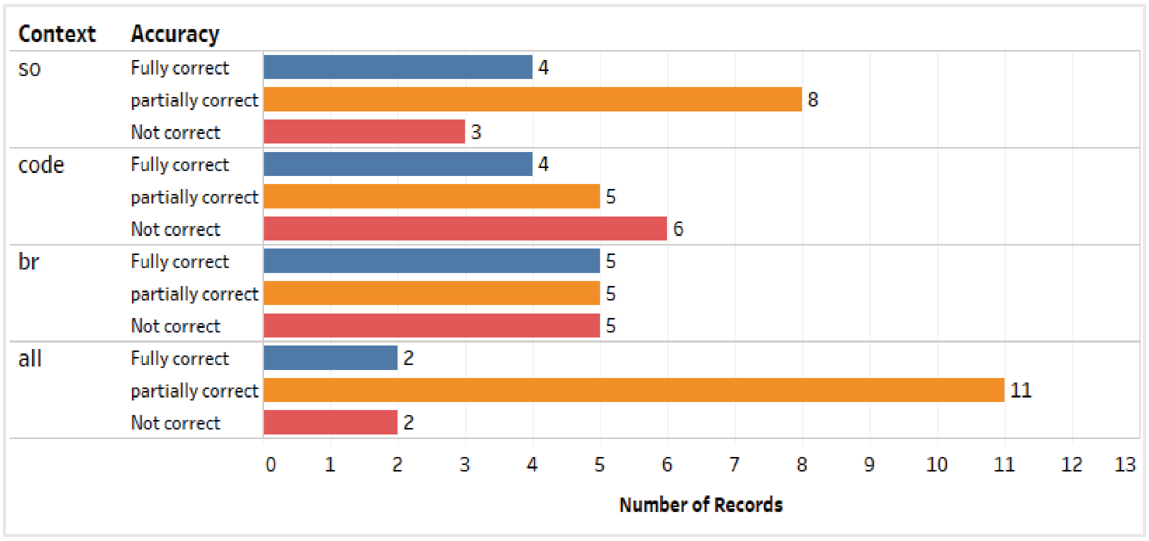


Figure 5 RQ1: Accuracy of summaries grouped by context.

From the above Figure 5, one can infer that there is no one such context using which participants were able to give better summaries. All the contexts they are equally important. Using Bug reports(BR) participants they were able to get the highest number of correct summaries when compared to other contexts. Almost all the participants were able to give at least partially correct summaries when using ALL context. From this we can understand that using all the sources they are able to gather information to understand the code elements in a better way when compared to performing tasks using individual sources. Using CODE participants, they got highest incorrect summaries when compared to others. Figure 6 shows accuracy of summaries grouped by tasks. We have 8 tasks in total that we used in our study. We almost divided evenly tasks to all participants.

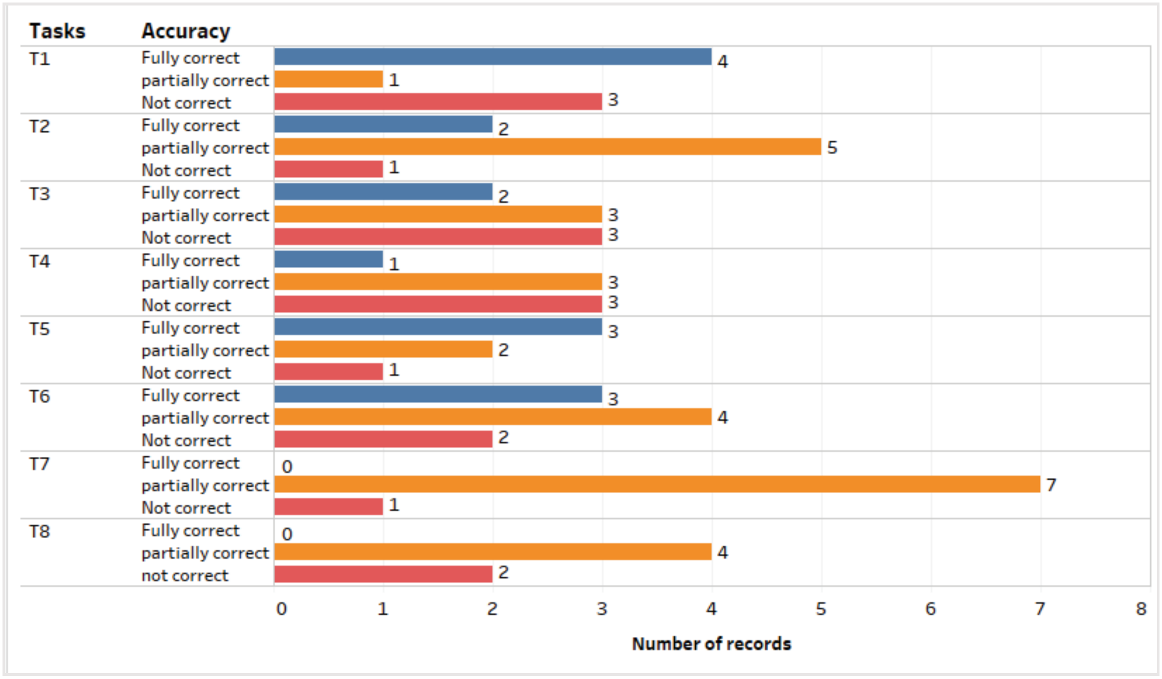


Figure 6 RQ1: Accuracy of summaries grouped by Task

The accuracy of summaries grouped by tasks they depend on the context using which the task is performed, difficulty level of task, participants expertise in using the context given to them. These various tasks are chosen from four different open source projects based on their difficulty level and it varied for all tasks. We have created sequences given to participants in a way that all the participants they receive equal standard of questions. Each sequence contains 4 tasks taken form above 8 tasks. Though different participants they summarized same code elements they used different contexts to summarize them. By considering above things we cannot really say anything just based on above graph. But we can observe participants answered 5 correct summaries when performing Task-T1. When performing Task 7, 7 out of 8 participants answered summaries as Partially Correct.

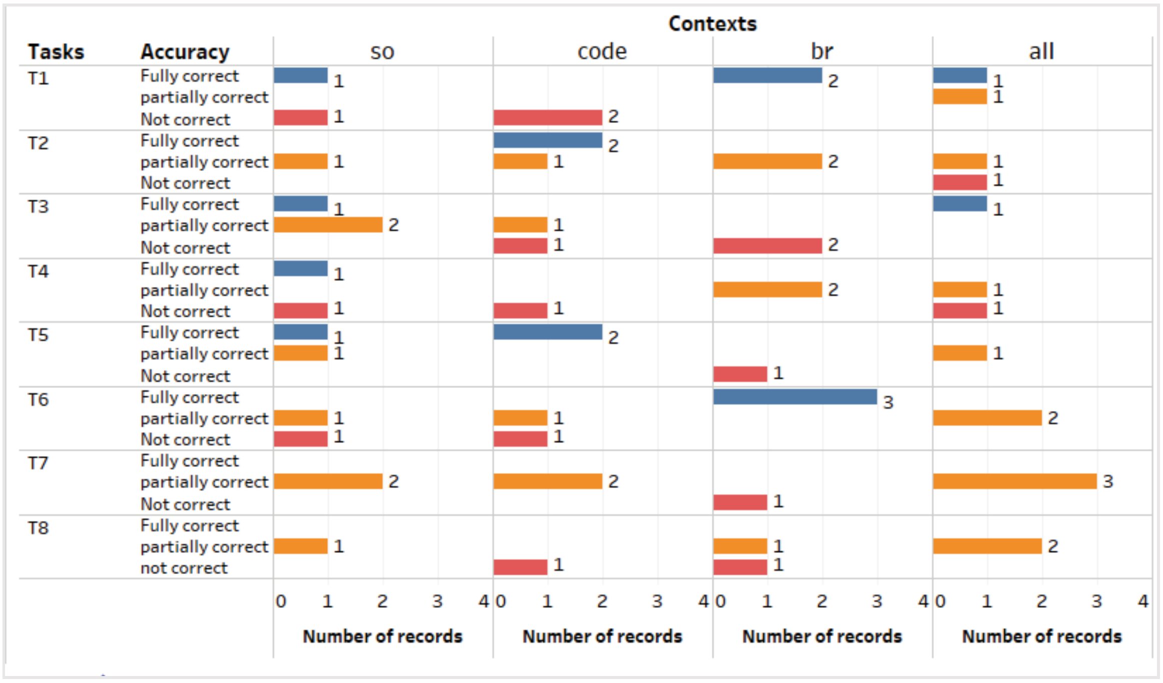


Figure 7 RQ1: Accuracy of summaries grouped by Task and Context

Above figure shows that when participants were given the same tasks using different contexts, they have performed in different ways. Task-T1 was given to a total eight participants using different contexts. Each context was given to two participants. Two participants performed task using Stack Overflow, one of the participants summary was fully correct and another participant summary was incorrect. Another two participants they have given incorrect summaries when using source code context for the same task T1. When using BR both participants they answered correctly. Using ALL context one summary was correct and other was partially correct. If we observe all the tasks and contexts by using ALL context, participants they were able to write summaries in a better way. 2 out of 15 participants have given incorrect summaries which are low when compared to other contexts.

Performance of developers in terms of accuracy, there is no one such context by using which participants they were able to give better summaries. All the four contexts have their own importance.

4.1.2 Time

This section presents the results of participants based on the Time spent to write the summaries. Time spent on each task by the participant is calculated from eye tracking data. We got responses recorded from eye tracker in the form of XML and JSON files. By looking at the Timestamp attribute from XML data we got the start and stop time of task from the first and last record. By calculating the difference of two Timestamps we have calculated the duration in secs. Eye tracker it records the time in milliseconds. The figure below (Figure 8) shows the average time spent in seconds by participants on each task grouped by contexts.

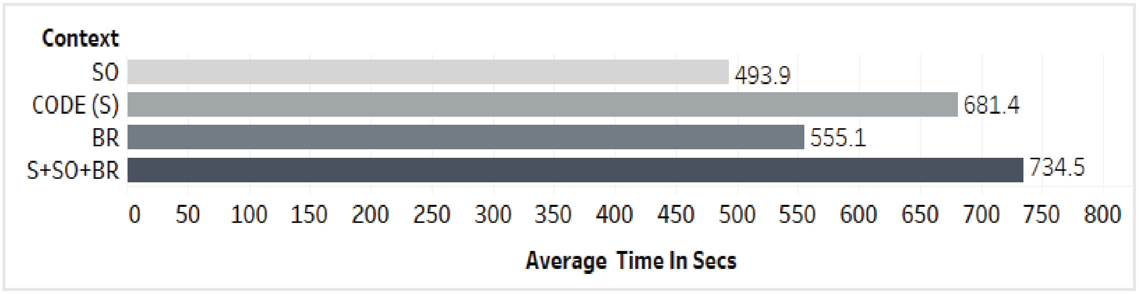


Figure 8: RQ1: Average time spent on each task grouped by context.

Using Stack Overflow(SO) context participants they spent less amount of time to finish the tasks. Whereas using the combination of all three sources they have spent more amount of time when compared to other contexts to finish the tasks. This is expected because using ALL context they have more things to look at.

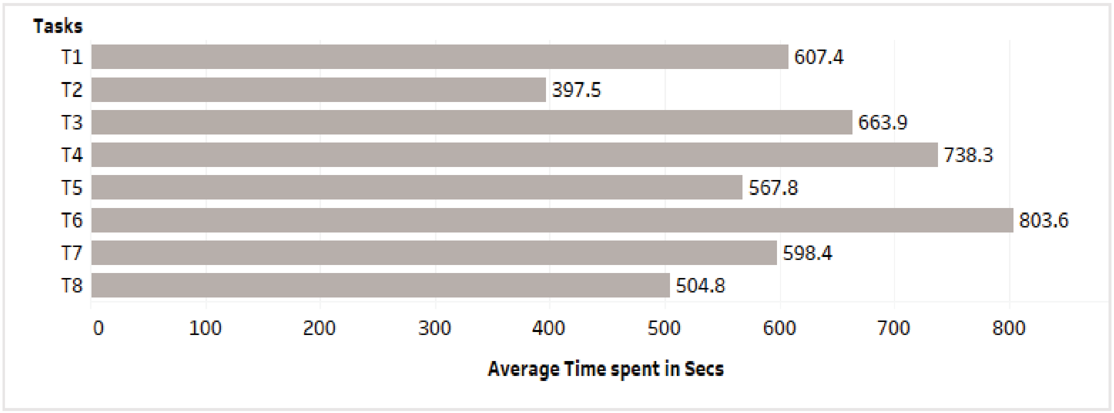


Figure 9 RQ1: Average time spent on each task grouped by task

The above figure shows the average time spent in seconds by participants on each task grouped by tasks. Participants they finished Task-T2 in less amount of time when compared to other tasks. T2 is medium level difficulty task. They spent more amount of time while performing Task-T6. T6 is a difficult task. Figure 10 below shows the average time spent in seconds by participants on each task grouped by context and tasks.

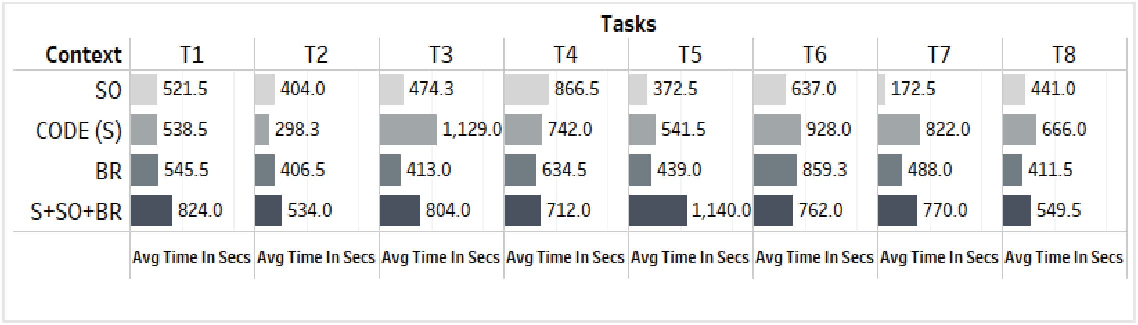


Figure 10 RQ1: Average time spent in secs grouped by task and context

While performing Task-T2 participants they have spent less amount of time. To complete task T6 participants they have spent more amount of time. Performance of participants in terms of time, average amount of time spent to complete the task when grouped by context, participants they have spent less amount of time when using Stack Overflow and highest amount of time when using combination of all three sources.

Average amount of time spent when grouped by task, highest amount of time spent on Task-T6. Almost all the participants have no knowledge of the source code for the Tomcat Project. It might have affected the performance of participants when doing Task-T6.

4.1.3 Accuracy and Time

There is no proportionality between Accuracy of summaries and amount of time spent to write the summaries. If you consider Figure 11, it gives us the overall view of the average amount of time spent on task using given context and accuracy of summaries grouped by context. The amount of time spent on task using SO is less when compared to others, its accuracy is good. Using SO participants, they were able to give 4 fully correct summaries and 8 partially correct summaries. Whereas if you consider ALL context the average amount of time spent to complete the task using ALL is more (this is expected because using all they have more things to look at), the accuracy of summaries given by participants is good. 2 summaries were fully correct and 11 are partially correct.

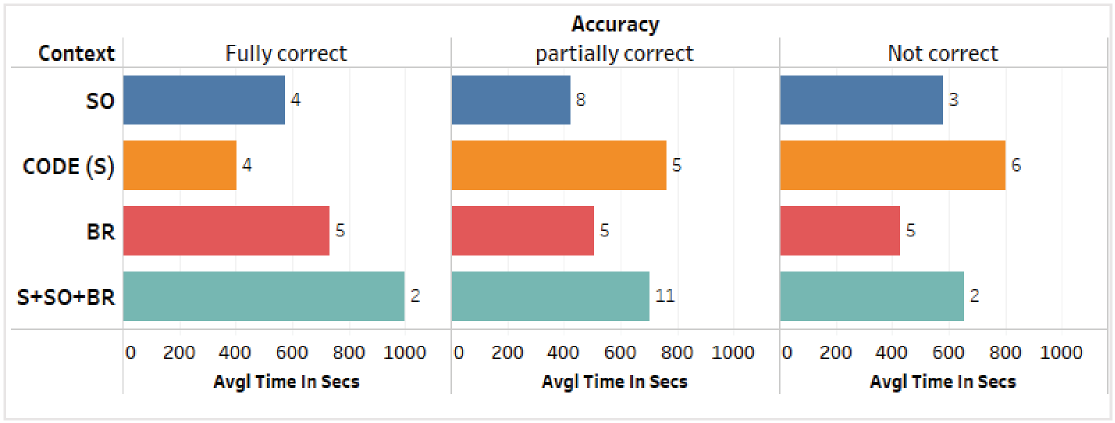


Figure 11 RQ1: Accuracy and Time grouped by context

The below Figure 12 shows the accuracy of summaries, the average amount of time spent in seconds to complete the task are grouped by contexts and tasks.

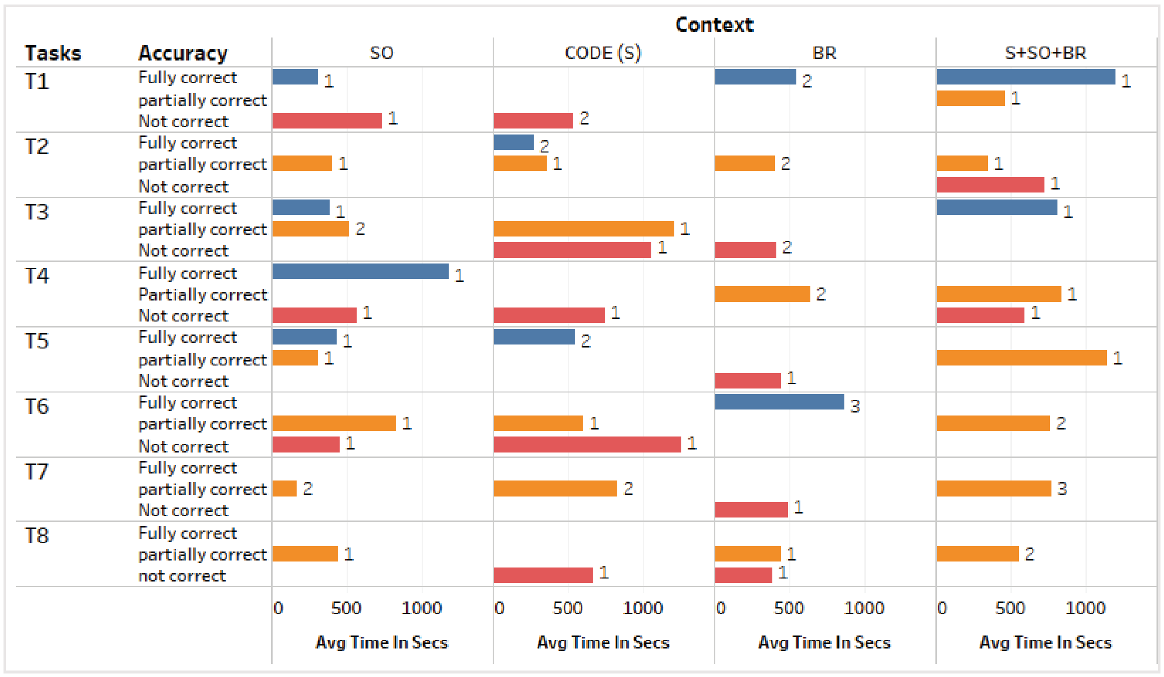


Figure 12 RQ1: Accuracy and Time grouped by context, task

There is no specific pattern followed to perform the tasks in terms of accuracy and time. Performance of participants varied from task to task as well as context to context. Using ALL context participants, they were able to perform all tasks well. Next, by using Stack Overflow context, they performed well. Participants they did not give correct summaries to T7 and T8 tasks using any of the contexts.

4.1.4 Comparing accuracy of summaries by novices and no-novices

To find whether there is a difference in performance of novice and non-novice participants we have compared their accuracy of summaries given by them, amount of time spent to complete the tasks. There are 6 participants in novice group and 9 participants in non-novice group. Below Figure 13 shows the accuracy of summaries given by participants grouped by context. Since each participant, they perform tasks using all four contexts, in each context we have 6 summaries given by different participants. We can observe that by using SO and SO+BR+Code participants they were able to give good summaries when compared to Code and BR contexts. Novices they did not perform well when using Source Code. 5 out 6 summaries given by them using Code context are marked as Incorrect. The same case with BR 4 out of 6 summaries given by them is marked as incorrect.

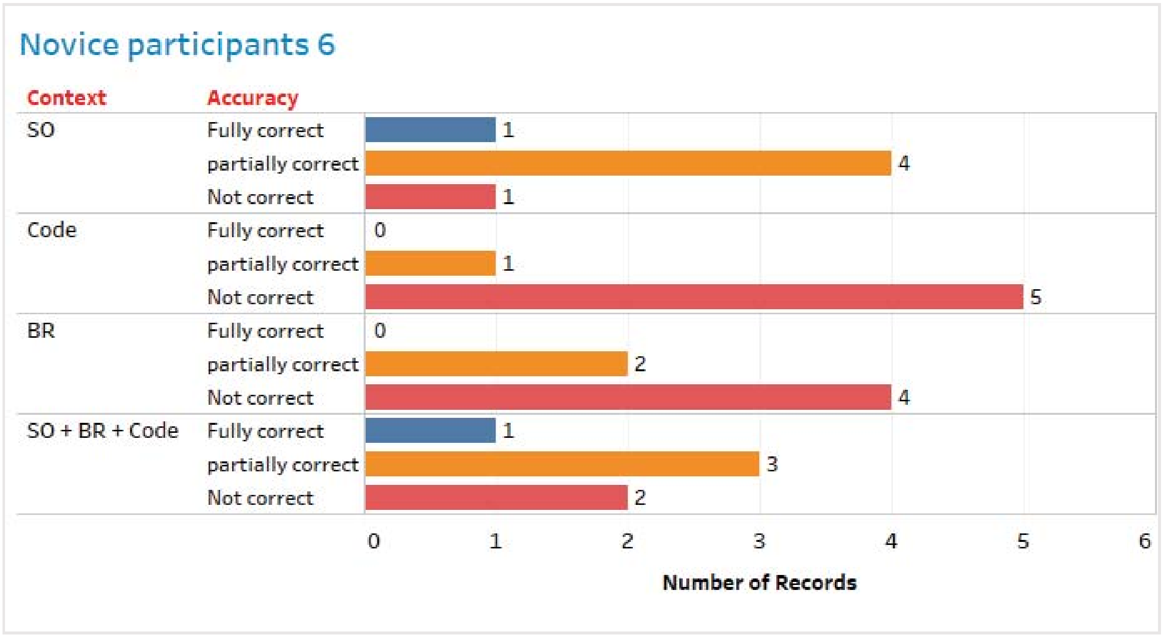


Figure 13 RQ1: Accuracy of summaries by novice participants grouped by context

Non-novices they performed well using all the four contexts un like novice participants. Using SO+BR+Code, all the summaries given by 9 participants are either partially correct or fully correct (See Figure 14). They were able to perform in a better way when using all three sources when compared to their performance using individual sources.

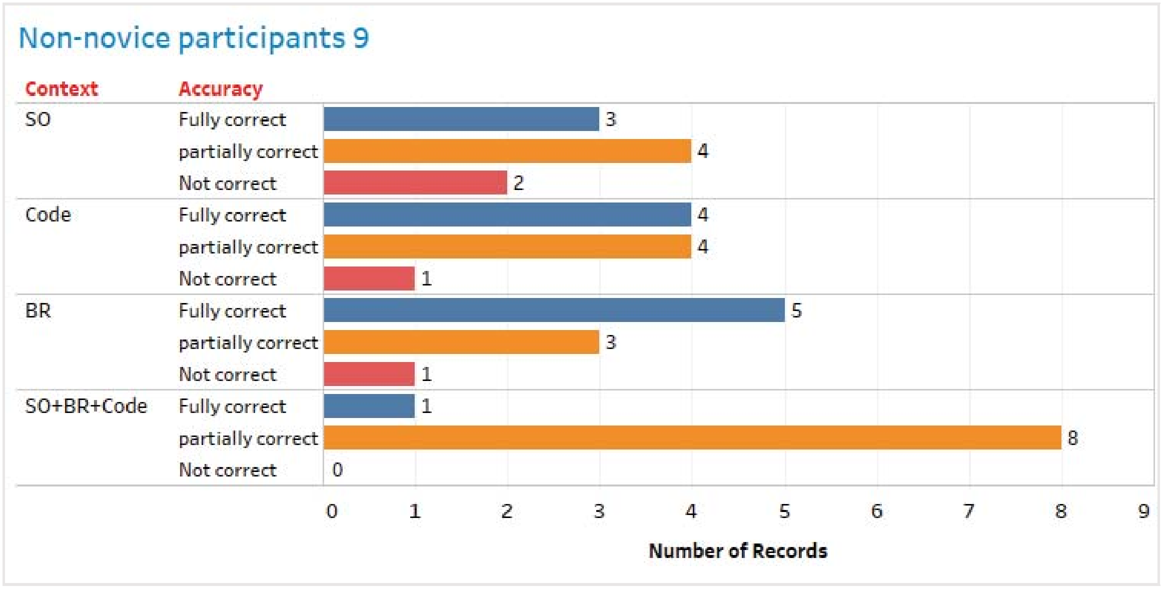


Figure 14 RQ1: Accuracy of summaries by non-novice participants grouped by context

If considered novice and non-novice groups, two groups they have given better summaries when using SO and SO+BR+Code when compared to other two contexts. Overall non-novices their summaries are more accurate when compared to novice participants. There is a significant difference between the accuracy of summaries by both groups.

4.1.5 Comparing time spent by novices and non-novices

Time spent to complete each task was calculated from eye tracking data. The average amount of time spent by novice participants to write summaries grouped by context. Figure 13 shows novices have spent more amount of time to complete the tasks using source code context. Both novices and non-novices have spent almost same amount of time when using SO and BR context.

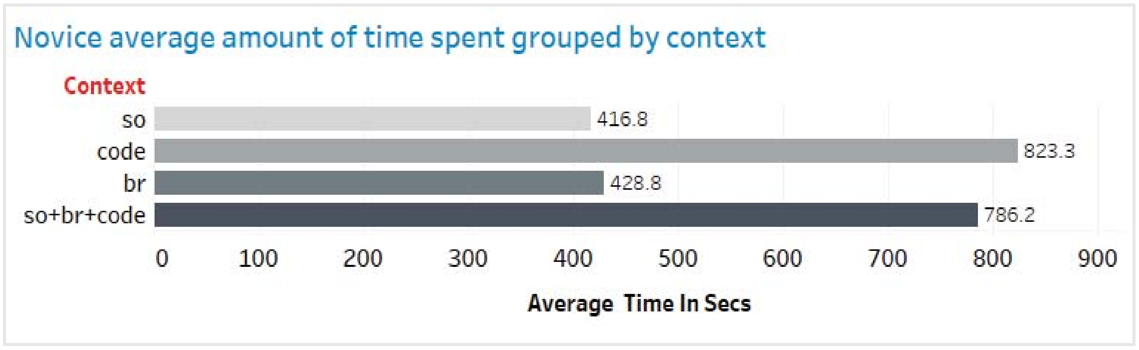


Figure 15 RQ1: Average Time spent on task by novices grouped by context

Figure 16 shows the average amount of time spent on each task in seconds by non-novice participants grouped by context. Using combination of all three sources they have spent more amount of time to finish the task which is expected.

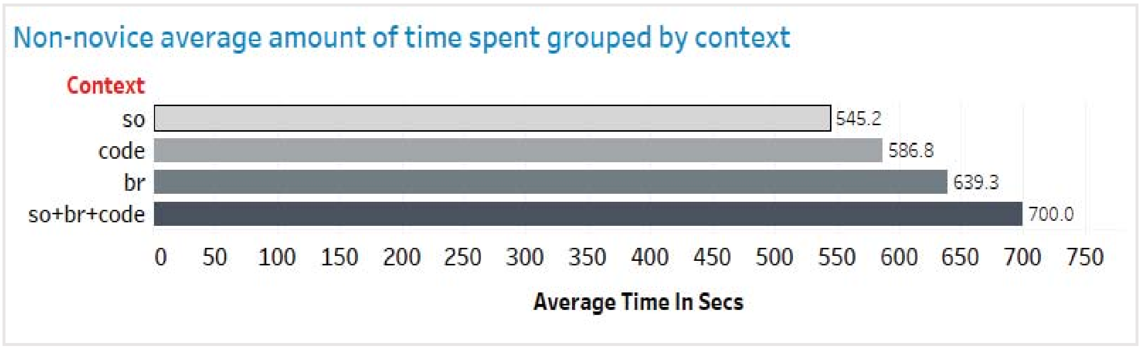


Figure 16 RQ1: Average Time spent on task by non-novices grouped by context

By comparing novices and non-novices in terms of amount of time spent, we see that novices they have spent more amount of time to complete the task using source code context, when compared to the non-novice group. Average time spent on task when using

SO and BR context are less for the novice group when compared to the non-novice group.

4.2 RQ2: How do developers navigate between multiple sources of information?

This research question is answered based on visual effort data collected from eye tracker. We have tried to find when given various sources, which information source does participant prefer to use. How subject navigates between multiple sources and amount of time spent on each source.

How many times does subject switches between multiple contexts to finish the given task? The eye tracker records responses in milliseconds.

Context switch term here implies moving from one source to another source while perfoming task. For instance, if a participant was asked to summarize the given code element using all the three sources. Initially, participant may look at source code after some time he might switch to Stack Overflow or Bug reports and so on. While performing the task, navigating from one source to another is treated as a context switch. We have considered three different contexts (text file, source code, and browser) and calculated the number of context switches made by each participant shown in Figure 17. The number of context switches are calculated by analysing eye tracking data of each participant performing the task using SO+BR+Code context. To weed out small gazes that involved mainly stray glances, we discard context switches with less than two seconds duration because when all three sources were opened, in the process of navigating from one source to another they are chances of participant looking at another source. After discarding these short gazes, we calculated the number of times context switches done by each participant.

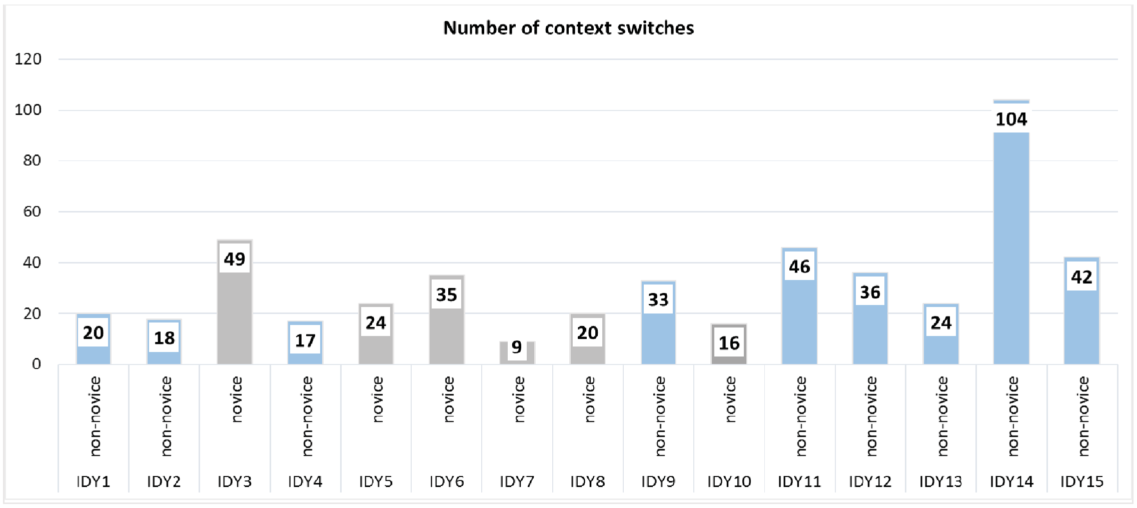


Figure 17 RQ2: Number of context switches made by each participant while using all three sources

The number of times a participant switched between sources it varied from person to person. In above figure bars in blue represent non-novice participants and bars in grey represent novice participants. There is no pattern followed for novice group and non-novice group. There were many individual differences between participants. IDY14 has made the highest number of context switches between sources. There is a relation between the number of switches made by participant and accuracy of summary given by them.

See Figure 18 for the data of one of the participants performing a task using ALL context. It shows how the subject navigated between multiple sources when performing task. The amount of time spent on each source in seconds. This data differs from participant to participant.

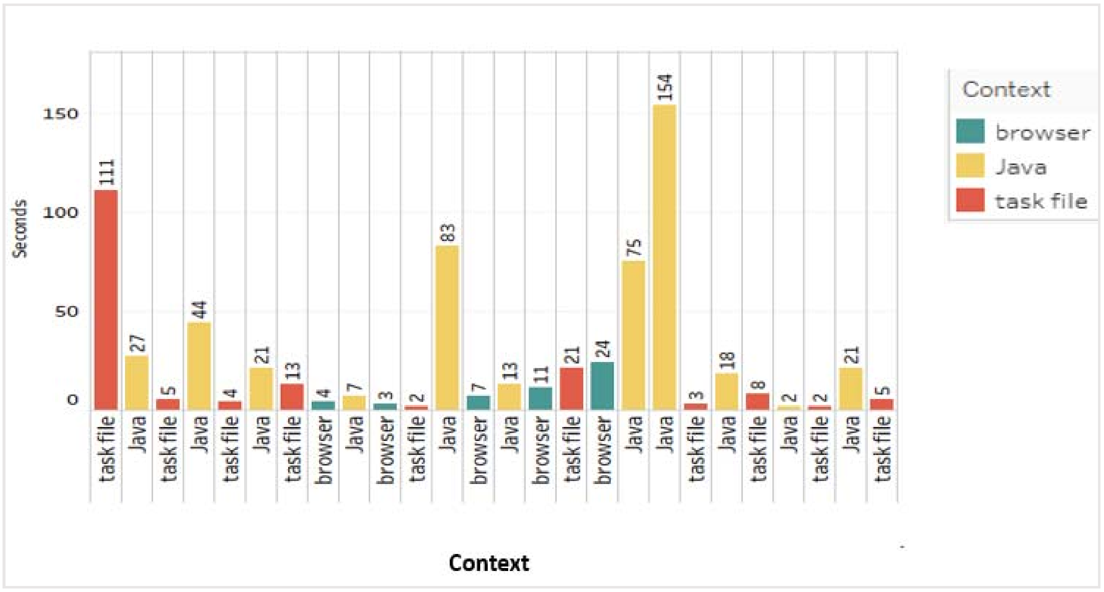


Figure 18 RQ2: Context switch when using multiple sources.

From the above figure we can observe that participant initially looked at text file then switched to Java file, then spent time looking at source code in between switched to the browser (so or br). Overall participant has spent more amount of time looking at source code. One might understand participants preferable source of information when given all three sources is source code from the data.

By analyzing the navigation patterns of all participants, we may know their preferable context when given all three sources to them.

4.3 RQ3: What do developers look at when searching for information in source code, Stack Overflow, and Bug reports?

The elements developers look at when searching for information varies from context to context. When searching for source code they look at elements like if statements, method declarations, variable declarations any other conditional statements in code etc. Whereas when they search in Stack Overflow they look different questions, answers, and their comments, tags, and images. Similarly with bug reports, they look at different questions, answers and comments related to the code elements.

By knowing different elements focused, participants if they spent more amount of time looking at certain elements it implies those elements are important and we may need to consider them to include in the summary.

A plugin called iTrace was developed in our SERESL lab (Sharif, Shaffer, et al. 2015) at YSU, it gives us eye-tracking information at word level and support scrolling. iTrace plugin with Eclipse IDE was used to conduct the study. When using this plugin with Eclipse it shows iTrace perspective. It provides us iTrace controller window from which we can control our eye tracker. The eye tracker records gazes at word level it will give us the words/tokens we looked at, its coordinates, its type, corresponding file and other book keeping information. It supports collection of data at the word level in source code, Stack Overflow, and Bug Report pages.

In next subsections presents each context and what are the different elements usually developers look at when searching for information in those contexts. We also compared the different elements focused by a novice participant and non-novice participant when performing the same task using the same context.

4.3.1 Source Code

Developers when searching for information in the source code there are many elements they focus on. In the past, many researches have been done to find the elements focused by developers when reading source code and pattern followed to read the source code. In our study, we focused to find various elements that developer has focused on when searching for information using different contexts.

Different elements focused by developers when performing source code was shown in Figure 19. It shows the data of one of the participants performing the task using source code context. Participant has focused on if statement method body, declaration, variables and amount of time spent on each element was given in seconds. From our data, we can find the different elements looked at the word level. Here we have presented the outer most view. We are also able to tell different elements participant have focused on that type.

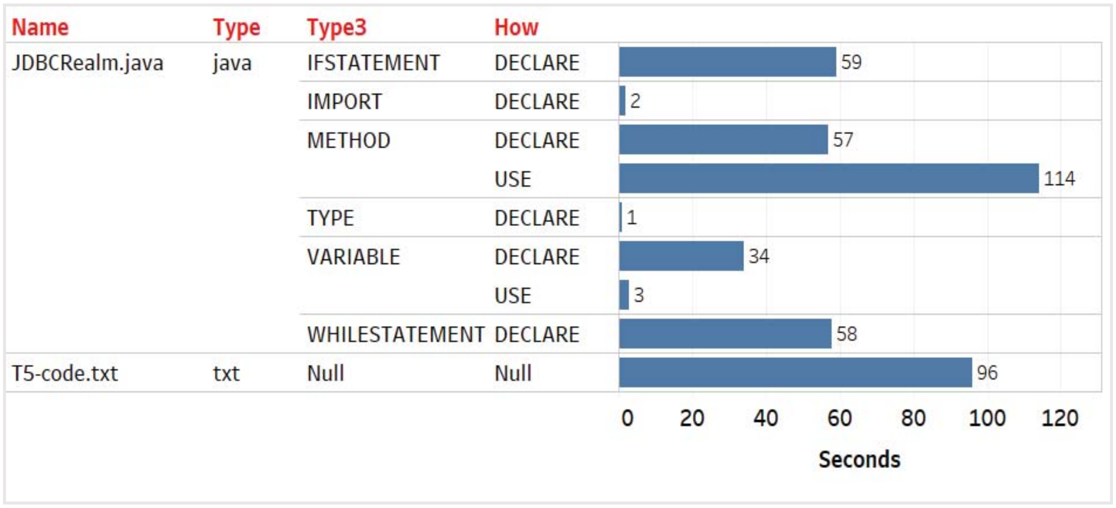


Figure 19 RQ3: Different elements focused in Source Code

Below Figure 20 and Figure 21 illustrates what novice participant and the non-novice participant have looked at when searching for information in the source code to perform the same task.

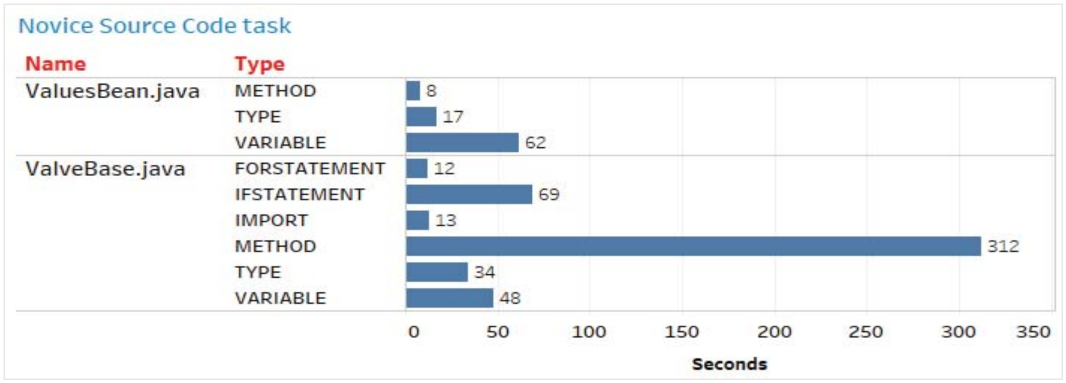


Figure 20 RQ3: Novice participant performing Source Code task

Novice participant has opened two Java files and focused on method, type, variables and conditional statements to gather information whereas non-novice participant see Figure 21 has looked at multiple Java files to gather information.

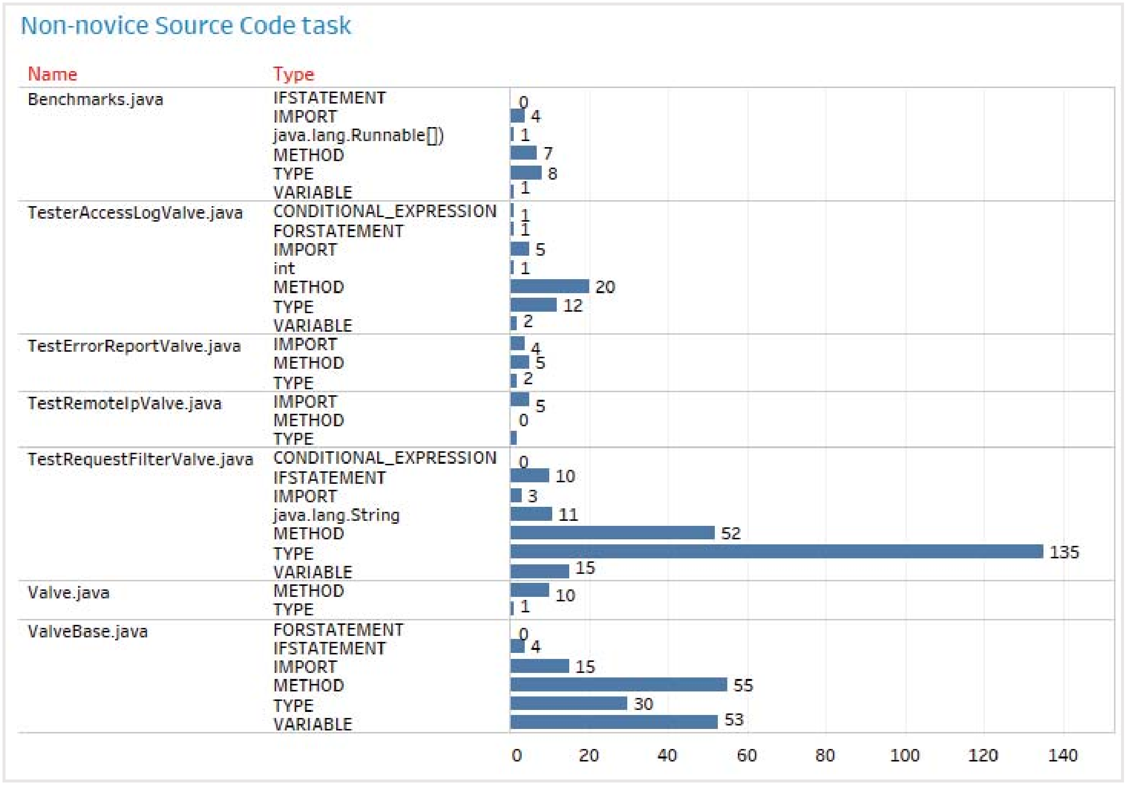
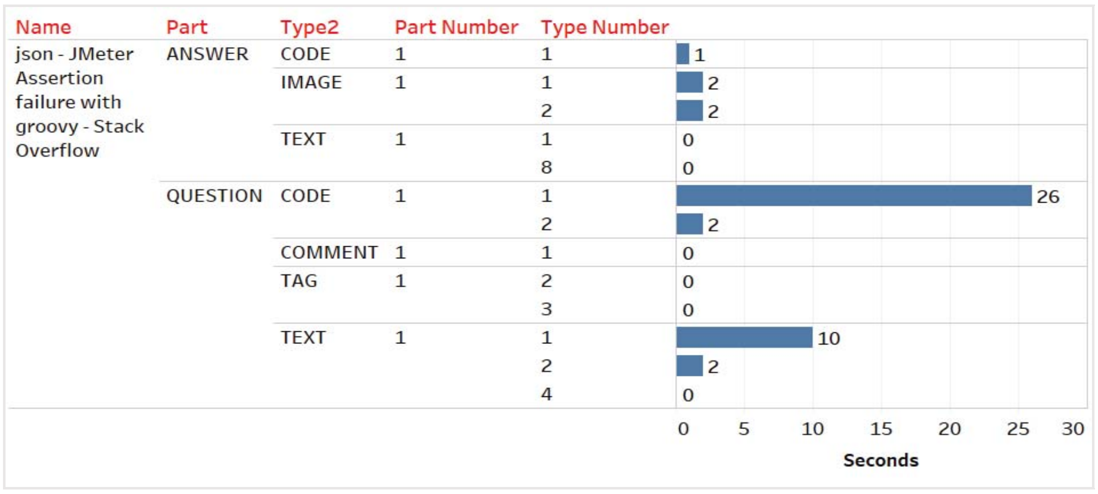


Figure 21 RQ3: Non-novice participant performing Source Code task.

Both participants they have spent more amount of time looking at method body, declarations and if statements. By looking into individual participant data, we can find the commonly focused elements by developers when using source code context, which may help to be included in the summaries.

4.3.2 Stack Overflow

Stack Overflow is a platform which is widely used by software developers to disuss the issues. Developers they post their issues related to code, other developers they help by writing answers and comments to the questions. By viewing at vote tag, one can know how many people have supported the answer. We can also green tick mark for correct solutions. One can look timestamp to know when the question was posted.



Name	Part	Type2	Part Number	Type Number	Seconds
json - JMeter Assertion failure with groovy - Stack Overflow	ANSWER	CODE	1	1	1
		IMAGE	1	1	2
				2	2
		TEXT	1	1	0
				8	0
	QUESTION	CODE	1	1	26
				2	2
		COMMENT	1	1	0
		TAG	1	2	0
				3	0
		TEXT	1	1	10
				2	2
				4	0

Figure 22 RQ3: Different elements focused when performing Stack Overflow task

The above Figure 22 shows various elements that are focused by developers when searching for information in Stack Overflow. It is from one of the participants' data performing the task using Stack Overflow context. Name in figure represents the name of the page which is opened, part indicates whether they looked at question or answer, type indicates the type of element they focused in that part, part number indicates the question or answer number that participant has focused on and type number indicates different comments looked at.

Consider the first row from Figure 22 it shows participant has looked at that Stack Overflow page, answer number 1 and first comment which is code in it for 1 second. One can interpret the figure in this way. Participant has spent highest amount of time looking at code element in question1.

For some elements time spent is shown as 0 seconds it does not mean that participant has not looked at that element. Eye tracker it records data in milliseconds, participant might have spent less than second on it might have glanced at it. Since time in figure shown in seconds, so it is represented as 0. Less than a second is enough to give a glance and to know the information is relevant or not to what we are searching for.

Below Figure 23 and Figure 24 illustrates what novice participant and the non-novice participant have looked at when searching for information in Stack Overflow while performing the same task.

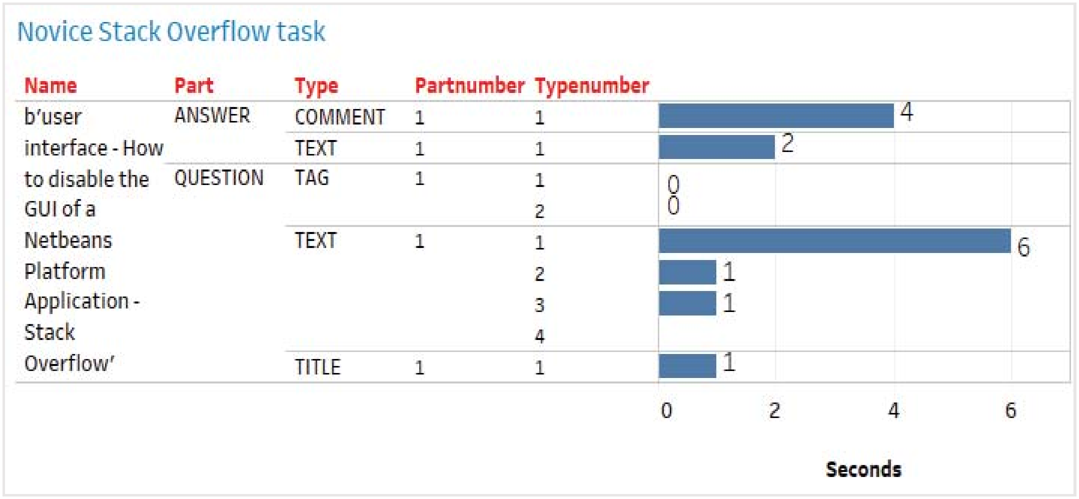
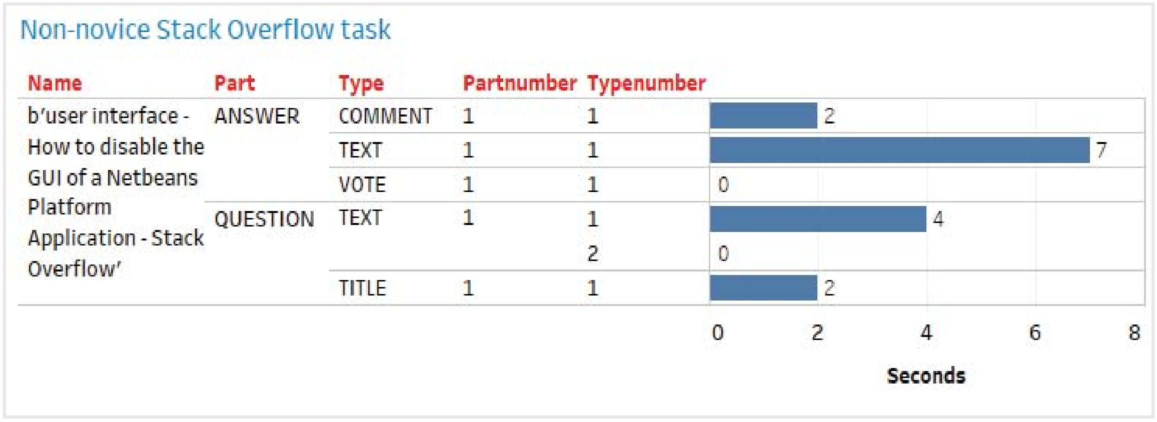


Figure 23 RQ3: Novice participant performing Stack Overflow task



Name	Part	Type	Partnumber	Typenumber	Seconds
b'user interface - How to disable the GUI of a Netbeans Platform Application - Stack Overflow'	ANSWER	COMMENT	1	1	2
		TEXT	1	1	7
		VOTE	1	1	0
	QUESTION	TEXT	1	1	4
				2	0
		TITLE	1	1	2

Figure 24 RQ3: Non-novice participant performing Stack Overflow task

Above Figure 23 and Figure 24 show that novice participant and the non-novice participant both focused on similar elements when performing the same task using Stack Overflow. The amount of time spent by them looking at different elements is also similar. There is no huge difference. The summary given by two participants are partially correct. As mentioned above time duration '0 seconds' does not imply that they have not looked at that element, they might have focused on them for less than a second. They have spent more amount of time looking at the textual description of question and answers.

4.3.3 Bug Reports

A bug reporting site is where developers usually post bugs and then fixes for them. There are many bug reporting sites exists in market. Which help developers not only to solve bugs but also keep track of all the bugs and helps in generating reports. This is also one of the famous platform used by developers to discuss code elements. Developers when searching for information in Bug Reports they mainly focus on three elements questions, attachments, and answers. Different comment numbers related to each of them. Figure 25 shows data of one of the participants performing summarization task using Bug Reports. One can observe that participant has looked at two bug report pages and different attachments and questions in each page. We can interpret the image in this way, name attribute gives the name of the page which is opened, part tells us which elements participant has looked at the question, answer or attachment and type number give detail about which question, or attachment participant has looked at.

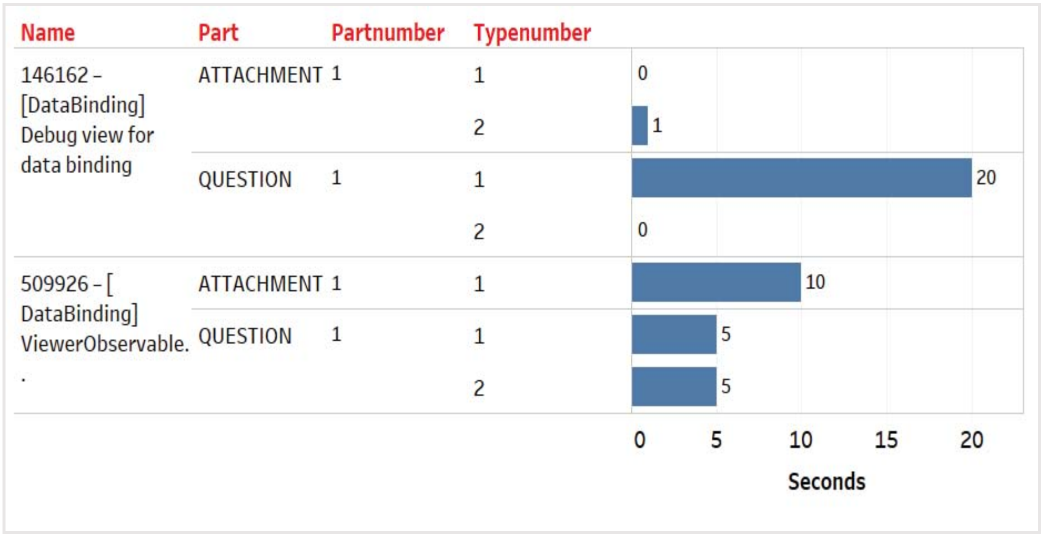
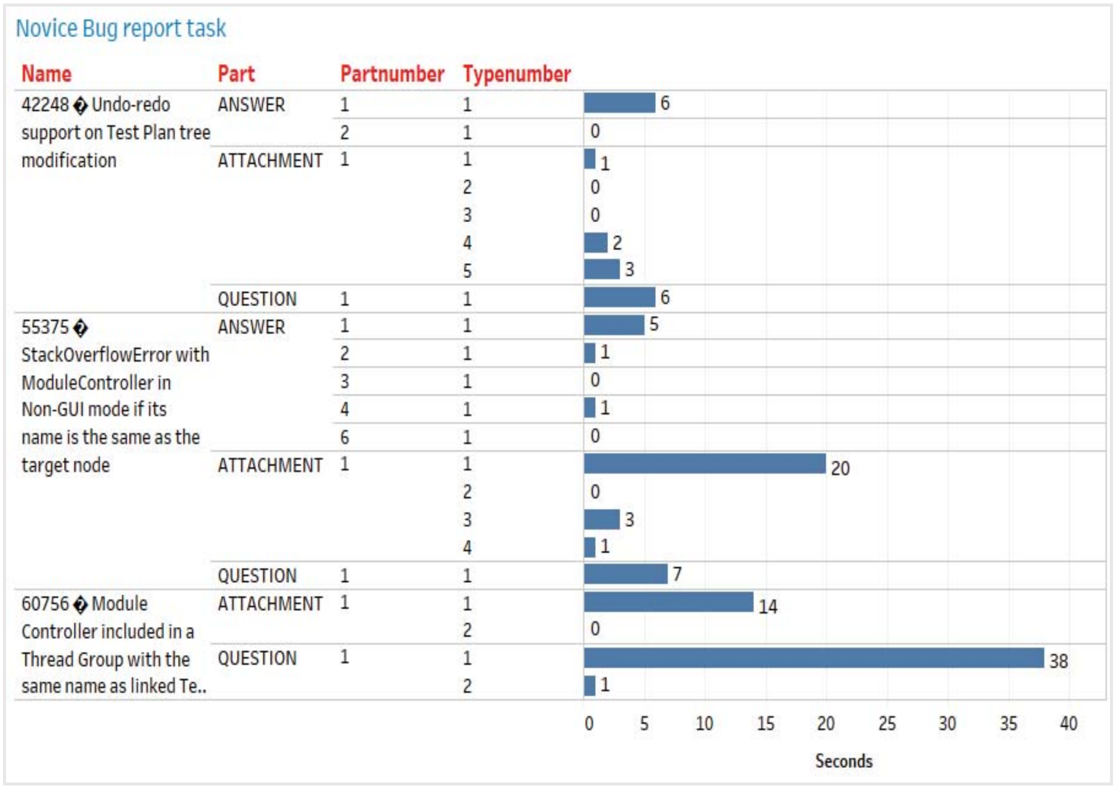


Figure 25 RQ3: Different elements focused when performing Bug Reports tasks

Above figure shows participant has spent more amount of time looking at question 1 for 20 seconds. Below Figure 26, Figure 27 shows on which elements novice participant and the non-novice participant have focused when searching for information in Bug Reports while performing the same task.



Name	Part	Partnumber	Typenumber	Seconds
42248 � Undo-redo support on Test Plan tree modification	ANSWER	1	1	6
		2	1	0
	ATTACHMENT	1	1	1
			2	0
			3	0
			4	2
			5	3
	QUESTION	1	1	6
55375 � StackOverflowError with ModuleController in Non-GUI mode if its name is the same as the target node	ANSWER	1	1	5
		2	1	1
		3	1	0
		4	1	1
		6	1	0
	ATTACHMENT	1	1	20
			2	0
			3	3
			4	1
	QUESTION	1	1	7
60756 � Module Controller included in a Thread Group with the same name as linked Te..	ATTACHMENT	1	1	14
			2	0
	QUESTION	1	1	38
			2	1

Figure 26 RQ3: Novice participant performing Bug Report task

Novice participant has gathered information by looking at three different pages in the bug report. He has spent more amount of time looking questions and attachments. Below Figure 27 shows non-novice participant performing the task using Bug Reports. Non-novice participant has looked at two Bug Report pages to summarize the code element. He has spent more amount of time looking at answer and attachment in Bug Reports.

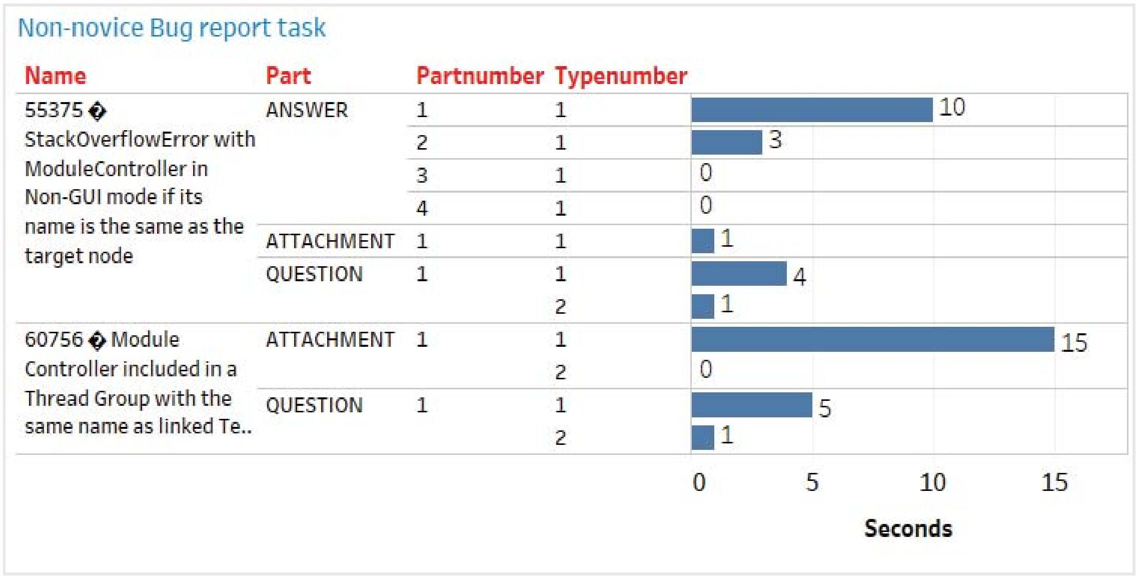


Figure 27 RQ3: Non-novice participants performing Bug Reports task

When we compare the different elements focused by novice and non-novice participant in Bug Reports to perform the same task, one can observe novice participant has spent more amount of time, looked at more Bug Reports when compared to the non-novice participant. The elements they focused were similar, the amount of time spent looking at them varied.

4.4 RQ4: Which source of information is the most preferable for developers while they are summarizing code?

This was answered based on the post-questionnaire data given by participants. After completion of the study participants were given post questionnaire to fill, which contained questions related to study. In this subsection presents responses given by participants to various questions which were asked in post-questionnaire. Most of the answers are based on Likert scale. The numbers in the table indicate the count of participants. Participants have answered their familiarity with the code of the projects or parts of these projects which are used in tasks before this study. By familiar, we mean that have they seen or worked with this code, bug report, Stack Overflow document before and does that knowledge helped them in answers to the study. Table 13 shows the response of participants to this question.

Table 13 Rating familiarity with the source code of different projects before study

Familiarity	Extremely Familiar	Moderately Familiar	SomeWhat Familiar	Slightly Familiar	Not at all Familiar
Eclipse	0	2	1	6	6
Netbeans	1	1	2	3	8
JMeter	0	0	1	1	13
Tomcat	0	0	0	1	14

Before doing the study not many participants were familiar with the source code of projects which are used in our study. Only a few participants are familiar with Netbeans and Eclipse projects source code. Almost all the participants they were not familiar with Tomcat project. From above responses, we may assume that their previous knowledge about code elements might not have affected their performance.

Participants were asked to rate the usefulness of each type of information with respect to how helpful they were to summarize the API elements in the study. Table 14 shows the responses given by participant to that question. Almost all the participants felt Stack Overflow + Bug Reports + Source Code context as extremely helpful when performing summarization task. They have also rated Source Code and Stack Overflow as Very helpful. Most of the participants felt Bug Reports as Somewhat helpful while performing summarization tasks.

Table 14 Rating usefulness of different contexts when doing summarization tasks

Usefulness	Extremely helpful	Very helpful	SomeWhat helpful	Slightly helpful	Not at all helpful
Stack Overflow	2	7	4	2	0
Bug Reports	0	2	9	3	1
Source Code	0	8	3	4	0
Stack Overflow + Bug Reports + Source Code	11	3	1	0	0

Next participants were asked to rate the usefulness of the different types of contexts present in Stack Overflow (SO) documents that helped them to summarize the API elements. Below Table 15 shows different types of contexts in Stack Overflow in the second column and their usefulness in the first row.

Table 15 Rating usefulness of different types of contexts in Stack Overflow that helped to summarize the API elements

	Stack Overflow Elements	Extremely helpful	Very helpful	Some What helpful	Slightly helpful	Not at all helpful	I do not know what this is	None given
1	Code examples	5	6	2	0	0	0	2
2	Comments by users	2	9	2	2	0	0	0
3	Stack traces	0	3	2	5	0	3	2
4	Textual descriptions (other than code examples and stack traces)	2	9	2	2	0	0	0
5	Votes	1	1	2	5	5	0	1
6	Tags of questions	0	1	4	5	3	1	1
7	User reputation/profile	0	0	3	1	10	0	1
8	Date of a comment/answer	1	0	2	1	10	0	1

The cell value gives us the count of participants who chose that response. Out of all the elements, participants they rated code examples, comments by users, and textual descriptions as very helpful elements in Stack Overflow when summarizing API elements. User reputation/profile and date of a comment/answer as Not at all helpful elements when using Stack Overflow. 5 out of 15 participants rated stack traces and votes as slightly helpful.

Participants were asked to rate the usefulness of the different types of contexts present in the source code that helped them to summarize the API elements. Table 16 shows the different source code elements and the rating giving to them based on their usefulness by participants. Source code contexts such as Comments, identifier names, good indentation were rated as Very helpful to summarize API elements by participants. They felt Javadoc comments and lines of code as helpful. Almost all the elements were useful when summarizng API elements.

Table 16 Rating usefulness of different types of contexts in Source Code that helped to summarize the API elements

	Source Code elements	Extremely helpful	Very helpful	SomeWhat helpful	Slightly helpful	Not at all helpful	I do not know what this is
1	Comments (line and block comments)	3	7	2	3	0	0
2	Javadoc comments (e.g., @return, @param)	1	4	4	2	4	0
3	Identifier names	3	5	5	2	0	0
4	Lines of code	2	1	6	2	4	0
5	General readability of the code (e.g., good indentation, structure, etc...)	3	7	2	3	0	0

Participants were asked to rate the usefulness of the different types of contexts present in bug reports that helped them to summarize the API elements. The responses to this question were presented in below table.

Table 17 shows that more than half of the participants they rated code samples, comments by users, and the textual description of the bug report as very helpful, Test cases and proposed patch as somewhat helpful to summarize the API elements using Bug Reports.

Based on above questions and responses, participants they felt the combination of all three sources as extremely helpful when compared to individual sources when performing summarization tasks. From their responses, we can also say that Stack

Overflow and Bug reports are helpful as source code in supporting code summarization tasks. From the responses of rating usefulness of different types of elements in each source, we can understand the elements that are helpful and can extract information from those elements to generate better summaries.

Table 17 Rating usefulness of different types of contexts in Source Code that helped to summarize the API elements

	Bug reports Elements	Extremely helpful	Very helpful	SomeWhat helpful	Slightly helpful	Not at all helpful	I do not know what this is	None given
1	Code examples	1	7	5	1	0	0	1
2	Comments by users	1	6	7	1	0	0	0
3	Stack traces	0	2	3	4	1	3	2
4	Proposed patch	1	1	7	3	2	1	0
5	Test cases	0	5	5	1	1	0	3
6	Textual description of the bug report (other than stack traces or code examples)	2	9	4	0	0	0	0

4.5 Observations and Discussion

In previous sections, we have discussed our experiment design and results in detail. In this section some observations that are found during the study are discussed. Amount of time spent on task using source code may be affected by partitcipants' familiarity with Eclipse IDE. Participants who are familiar working with eclipse IDE were able to search and find required code elements in project given to them in less amount of time when performing tasks using Code and ALL contexts. Whereas the participants who are not familiar with Eclipse spent almost half of the time in searching code elements while performing the task.

Participants who are familiar with Stack Overflow and bug reports were more comfortable when completing the tasks given to them than others. When subjects are performing the tasks using ALL context, the order of the tasks given to them may affect performance. For instance, consider a participant who is not familiar to Stack Overflow, Bug reports. Case 1 if the participant was given task using ALL context before performing tasks using BR and SO context. Case 2 if the participant was given task using ALL context after performing the tasks using SO and BR context. In both cases the approach of participants may differ.

Number of incorrect summaries using ALL context is low, participants they were able to give at least partially correct summaries when using all three sources. Above data shows by using information from all the three sources, we may generate better summaries. Novice and Non-novice groups they were able to give good summaries when using Stack Overflow and combination of sources. Novice group they were not able to perform well when using source code context. Most of the novices they are not familiar with the source code projects used in our study.

From post questionnaire data we can say that both groups rated Stack Overflow + Bug Reports + source code as extremely helpful context and Stack Overflow as Very helpful. Novices and the non-novices are somewhat familiar with Eclipse project source code before the study. They were not at all fmilar with Tomcat and Jmeter projects code.

While performing tasks using source code participants have rated comments, identifier names, good indentation as Very helpful contexts to summarize API elements. When using Stack Overflow to summarize API elements participants they felt code examples, comments by users, and textual descriptions as very helpful elements. Participants stated that when using bug reports code samples, comments by users, and the textual description are very helpful to them to summarize API elements.

4.6 Threats to Validity

Every experiment is subjected to various threats to validities. This section presents some of them.

4.6.1 Internal Validity

Randomly generated sequences of treatments and assigned to participants making sure that each task is assigned to same number of participants and with the same proportion of various contexts used in the study. Participants they do not know the actual hypothesis of the research this may affect their approach during the study. They just know that they are doing the summarization study by reading code elements using different contexts. Each participant was given with only four summarization tasks to limit participants fatigue effect. There is no time constraint for participant to complete the task because it may affect their performance. Participants when they feel they gathered sufficient information related to code elements they can write the summary. Same method/class was not given to same participant for all tasks in different contexts because of learning effect. Different code elements for different contexts were given.

4.6.2 External Validity

Results from our study may be applied to various group of people because we make sure to select the participants with different backgrounds. We have participants range who have above 6 years of programming experience and having very good knowledge in Java to the participants who are novice and have fair amount of knowledge in Java. This allowed us to understand the performance of novice and non-novice in different aspects. Representative tasks taken from four open source projects are used.

4.6.3 Construct Validity

During study tasks given to participants were designed in a way that participants get to summarize the given code elements using individual sources in three tasks as well as combination of all sources in one task. In this way the effect of each individual source and their combination in supporting code summarization tasks can be known. Since visual attention while performing tasks is related to mental processing of the information, (Just and Carpenter 1980) the measures derived from the fixations and durations should be valid.

4.6.4 Conclusion Validity

To ensure conclusion validity, we use appropriate methods for analysis of our results. We make sure all assumptions are met for the statistical tests used. Since we used qualitative methods based on pure observation of navigation strategies, we do not have any threats with respect to this.

CHAPTER 5

Conclusions and Future Work

Source code summarization is important for developers to keep up to speed with what the code does. To investigate the effect of multiple sources of information on code summarization we have conducted an empirical study using an eye tracker, with 15 participants performing summarization tasks using different contexts. This is the first eye tracking study to look at role of various informal sources on code summarization. After analyzing the data from our empirical study, we conclude that Stack Overflow and bug reports are as helpful as source code in supporting code summarization tasks. Participants felt more confident when using Stack Overflow and bug reports when compared to the source code. When we look at the performance of participants using different contexts in terms of accuracy there is no one context that stood out to produce better summaries; all the contexts have their own importance. In terms of time, average amount of time spent to complete the task using Stack Overflow context is less and when using all three sources of information they have spent more amount of time to complete the task. Participants rated Stack Overflow + Bug Reports + Source Code as extremely helpful context to summarize API elements. Novice and non-novice groups performed the given task well when using Stack Overflow and Stack Overflow + Bug Reports + source code sources. The average amount of time spent to complete the task using source code by novices is high and they were not able to accurate summaries when using the source code. The results of our study can be useful to researchers and practitioners interested in building context-aware code summarization tools that can help augment official documentation with the insightful information extracted from multiple sources.

As part of our future work, we are planning to conduct the study with professionals from industry. The goal is to compare students method of summarization with industry professionals. Another future goal is to develop high-level strategies used in Stack Overflow vs bug reports vs source code in trying to summarize code elements. We noted that there are many individual differences between our subjects however, we would like to find some common thread to link the main strategies used.

APPENDIX Study Material

This section it contains the IRB approval certificate to conduct the study, the study instructions given for participants, pre-questionnaire, post questionnaire used in the study, tasks used in the study and the oracle against which the summaries given by participants are evaluated.

A.1 Study Instructions

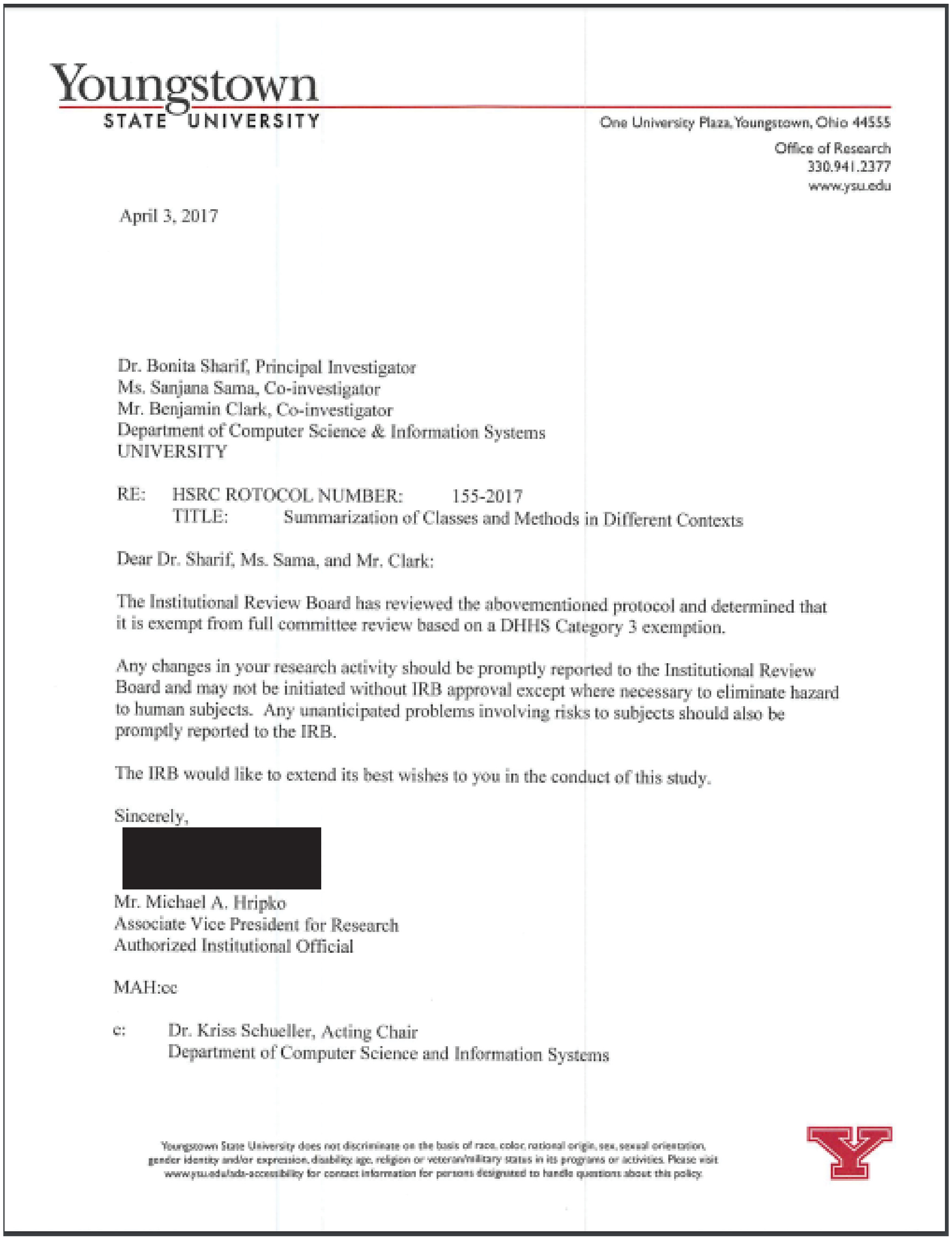
This study is concerned with summarizing methods and classes (i.e., API elements)

- In order to understand what your task is and what you need to do, please carefully read the printed document titled "A tutorial on the summarization tasks for API classes or methods" given to you. A sample task has been provided in the Eclipse workspace for you.
- You will need to work on four summarization tasks i.e., your task is:

  Summarize in a very concise and brief way the API Method or Class in your own words.
- After reading the task, you will follow the instructions given to you in the tutorial depending on what context you are asked to use. Please pay attention to what context the task is asking you to use. For e.g., if it asks to check Stack Overflow then please do not use the source code to summarize the task.

- You will be using the Eclipse built-in browser to browse Stack Overflow and Bug reports. Please do not use Chrome or anything outside of Eclipse to finish this task.
- You will type in your answers in the task text file that is shown at the bottom of the Eclipse window where it says, "YOUR SUMMARY:" Please do not close these text files. Save the files by hitting <Ctrl>+S at regular intervals.
- Please do not guess the answers.
- Let the moderator know when you are done with a task. She will stop eye recording, so you can complete confidence and difficulty ratings.
- For each question, you will be asked to rate your confidence level of the summary and to rate the difficulty level you faced while doing the summary. Check the appropriate box.
- Please try to maintain your position in the chair while you do the study so that we do not lose the tracking of your eyes. Moving the chair back or moving yourself back in the chair will cause the eye tracker to stop tracking. Small head movements such as looking at the keyboard to type should be fine.
- Find a comfortable position so we can begin. We will first begin with calibrating your eyes. Look at the black dot in the center of the red circle and follow it around on the screen.

A.2 IRB approval certificate to do the study



Youngstown
STATE UNIVERSITY

One University Plaza, Youngstown, Ohio 44555

Office of Research
330.941.2377
www.ysu.edu

April 3, 2017

Dr. Bonita Sharif, Principal Investigator
Ms. Sanjana Sama, Co-investigator
Mr. Benjamin Clark, Co-investigator
Department of Computer Science & Information Systems
UNIVERSITY

RE: HSRC ROTOCOL NUMBER: 155-2017
TITLE: Summarization of Classes and Methods in Different Contexts

Dear Dr. Sharif, Ms. Sama, and Mr. Clark:

The Institutional Review Board has reviewed the abovementioned protocol and determined that it is exempt from full committee review based on a DHHS Category 3 exemption.

Any changes in your research activity should be promptly reported to the Institutional Review Board and may not be initiated without IRB approval except where necessary to eliminate hazard to human subjects. Any unanticipated problems involving risks to subjects should also be promptly reported to the IRB.

The IRB would like to extend its best wishes to you in the conduct of this study.

Sincerely,

[redacted]

Mr. Michael A. Hripko
Associate Vice President for Research
Authorized Institutional Official

MAH:cc

c: Dr. Kriss Schueller, Acting Chair
Department of Computer Science and Information Systems

A.3 Pre-Questionnaire

1. ID: * ______________________________

2. Gender *
   a. Female
   b. Male

3. Your age range is: *
   a. < 18 years
   b. 18 - 25 years
   c. 26 - 30 years
   d. 31 - 35 years
   e. 36 - 40 years
   f. 41 - 45 years
   g. 46 - 50 years
   h. > 50 years

4. How many years of active programming experience do you have? *
   a. < 1 year
   b. Between 1 and 2 years
   c. Between 3 and 5 years
   d. Between 6 and 10 years
   e. > 10 years

5. What is your level of expertise in the Java programming language? *
   a. Poor
   b. Fair
   c. Good
   d. Very Good
   e. Excellent

6. Please select all the degrees you have and are currently enrolled in. *
   Check all that apply
   [ ] Bachelors
   [ ] Masters
   [ ] Ph.D.
   [ ] Other

7. Current Positions - Select all that apply *
   Check all that apply
   [ ] I currently work in industry
   [ ] I currently work in academia
   [ ] I am currently a student

[ ] I am currently a faculty member
[ ] I am currently a post doc

8. How many years of work experience do you have in industry? *
a. None
b. < 1 year
c. Between 1 and 2 years
d. Between 3 and 5 years
e. Between 6 and 10 years
f. > 10 years

9. Do you use Stack Overflow to find solutions to your coding problems? *
a. Yes
b. No

10. Do you use/read bug reports to find solutions to issues while coding? *
a. Yes
b. No

11. Have you contributed (code and/or documentation) to an *
a. Yes
b. No

12. Which of the following IDEs are you familiar with? (By familiar we mean you are able to work in fairly well). *

Check all that apply
[ ] Eclipse
[ ] Visual Studio
[ ] Netbeans
[ ] IntelliJ
[ ] Other

A.4 Post-Questionnaire

1. ID: * ______________________________

2. Were you familiar with the code of the following projects or parts of these projects before this study? *

(By familiar, we mean you have seen or worked with this code, bug report, Stack Overflow document before and that knowledge helped you in your answers to the study.)

1. Eclipse
   a. Extremely familiar
   b. Moderately familiar
   c. Somewhat familiar
   d. Slightly familiar
   e. Not at all familiar

2. Netbeans
   a. Extremely familiar
   b. Moderately familiar
   c. Somewhat familiar
   d. Slightly familiar
   e. Not at all familiar

3. JMeter
   a. Extremely familiar
   b. Moderately familiar
   c. Somewhat familiar
   d. Slightly familiar
   e. Not at all familiar

4. Tomcat
   a. Extremely familiar
   b. Moderately familiar
   c. Somewhat familiar
   d. Slightly familiar
   e. Not at all familiar

2. Rate the usefulness of each type of information with respect to how helpful they were to summarize the API elements in the study**.** *

1. Stack Overflow
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful

2. Bug Reports
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful

3. Source Code
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful

4. Stack Overflow + Bug Reports + Source Code
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful

3. Rate the usefulness of the different types of contexts present in **source code** that helped you to summarize the API elements. *

1. Comments (line and block comments)
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is

2. Javadoc comments (e.g., @return, @param)
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is

3. Identifier names
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is

4. Lines of code
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is

5. General readability of the code (e.g., good indentation, structure, etc...)
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is

4. Rate the usefulness of the different types of contexts present in **Stack Overflow (SO) documents** that helped you to summarize the API elements *

1. Code examples
   a. Extremely helpful
   b. Very helpful
   c. Somewhat helpful
   d. Slightly helpful
   e. Not at all helpful
   f. I do not know what this is
   g. None given

2. Comments by users
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

3. Stack traces
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

4. Textual descriptions (other than code examples and stack traces)
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

5. Votes
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

6. Tags of questions
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful

f. I do not know what this is
g. None given

7. User reputation/profile
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

8. Date of a comment/answer
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

5. Rate the usefulness of the different types of contexts present in **bug reports** that helped you to summarize the API elements *

1. Code examples
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

2. Comments by users
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

3. Stack traces
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

4. Proposed patch
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

5. Test cases
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

6. Textual description of the bug report (other than stack traces or code examples)
    a. Extremely helpful
    b. Very helpful
    c. Somewhat helpful
    d. Slightly helpful
    e. Not at all helpful
    f. I do not know what this is
    g. None given

6. Overall, how difficult did you find the study? *

7. Overall, do you feel you spent sufficient time to summarize the *

8. Comments

(Please list any comments you had on the study. We value your feedback in this research.)

A.5 Tasks and Comprehension

There are total eight API elements selected from four different open source projects such as Eclipse, JMeter, Tomcat, and Netbeans. Participants were asked to summarize the above code elements using given context such as Source Code, Stack Overflow, Bug report and Source Code+ Stack overflow + Bug report. We have created eight sequences by randomly choosing 4 code elements in each.

Task T1

Please summarize the method:

`org.eclipse.core.databinding.Binding.dispose`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the method (dispose) in the Eclipse project.

2- Open the source code of the method.

3- Read the source code.

4- Summarize in a very concise and brief way the given method.

Using Stack Overflow:

1- Open the link: https://stackoverflow.com/search?q=databinding+dispose+

2- Search the method (dispose) in Stack Overflow, while considering the context (`org.eclipse.core.databinding.Binding`).

3- Summarize in a very concise and brief way the given method.

Using Bug Report:

1- Open the link:

https://bugs.eclipse.org/bugs/buglist.cgi?quicksearch=binding%20dispose

2- Search the method (dispose) in Bug reports, while considering the context (`org.eclipse.core.databinding.Binding`).

3- Summarize in a very concise and brief way the given method.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T2

Please summarize the class:

`org.eclipse.swt.SWTError`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the class (SWTError) in the Eclipse project.

2- Open the source code of the class.

3- Read the source code.

4- Summarize in a very concise and brief way the given class.

Using Stack Overflow:

1- Open the link:

http://stackoverflow.com/search?q=SWTError

2- Search the class (SWTError) in Stack Overflow, while considering the context (`org.eclipse.swt`).

3- Summarize in a very concise and brief way the given class.

Using Bug Report:

1- Open the link:

https://bugs.eclipse.org/bugs/buglist.cgi?quicksearch=SWTError

2- Search the class (SWTError) in Bug reports, while considering the context (`org.eclipse.swt`).

3- Summarize in a very concise and brief way the given class.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T3

Please summarize the method:

`org.apache.jmeter.Jmeter.convertSubTree`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the method (convertSubTree) in the JMeter Project.

2- Open the source code of the method.

3- Read the source code.

4- Summarize in a very concise and brief way the given method.

Using Stack Overflow:

1- Open the link:

https://stackoverflow.com/search?q=Jmeter+convertSubTree

2- Search the method (convertSubTree) in Stack Overflow, while considering the context (`org.apache.jmeter.Jmeter`).

3- Summarize in a very concise and brief way the given method.

Using Bug Report:

1- Open the link:

https://bz.apache.org/bugzilla/buglist.cgi?quicksearch=Jmeter%20convertSubTree

2- Search the method (convertSubTree) in Bug reports, while considering the context (`org.apache.jmeter.Jmeter`).

3- Summarize in a very concise and brief way the given method.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T4

Please summarize the class:

`org.apache.jmeter.samplers.SampleResult`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the class (SampleResult) in the JMeter Project.

2- Open the source code of the class.

3- Read the source code.

4- Summarize in a very concise and brief way the given class.

Using Stack Overflow:

1- Open the link: https://stackoverflow.com/search?q=SampleResult

2- Search the class (SampleResult) in Stack Overflow, while considering the context (`org.apache.jmeter.samplers`).

3- Summarize in a very concise and brief way the given class.

Using Bug Report:

1- Open the link:

https://bz.apache.org/bugzilla/buglist.cgi?quicksearch=SampleResult

2- Search the class (SampleResult) in Bug reports, while considering the context (`org.apache.jmeter.samplers`).

3- Summarize in a very concise and brief way the given class.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T5

Please summarize the method:

`org.apache.catalina.realm.JDBCRealm.getRoles`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the method (getRoles) in Tomcat project.

2- Open the source code of the method.

3- Read the source code

4- Summarize in a very concise and brief way the given method.

Using Stack Overflow:

1- Open the link: https://stackoverflow.com/search?q=JDBCRealm+getRoles

2- Search the method (getRoles) in Stack Overflow, while considering the context (`org.apache.catalina.realm.JDBCRealm`).

3- Summarize in a very concise and brief way the given method.

Using Bug Report:

1- Open the link:

https://bz.apache.org/bugzilla/buglist.cgi?bug_status=UNCONFIRMED&bug_status=NEW&bug_status=ASSIGNED&bug_status=REOPENED&bug_status=NEEDINFO&bug_status=RESOLVED&bug_status=VERIFIED&bug_status=CLOSED&query_format=advanced&short_desc=JDBCRealm%20getRoles&short_desc_type=allwordssubstr

2- Search the method (getRoles) in Bug reports, while considering the context (`org.apache.catalina.realm.JDBCRealm`).

3- Summarize in a very concise and brief way the given method.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T6

Please summarize the class:

`org.apache.catalina.valves.ValveBase`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the class (ValveBase) in the Tomcat project.

2- Open the source code of the class.

3- Read the source code.

4- Summarize in a very concise and brief way the given class.

Using Stack Overflow:

1- Open the link: http://stackoverflow.com/search?q=ValveBase

2- Search the class (ValveBase) in Stack Overflow, while considering the context (`org.apache.catalina.valves`).

3- Summarize in a very concise and brief way the given class.

Using Bug Report:

1- Open the link:

https://bz.apache.org/bugzilla/buglist.cgi?bug_status=__all__&content=ValveBase&no_redirect=1&order=Importance&query_format=specific

2- Search the class (ValveBase) in Bug reports, while considering the context (`org.apache.catalina.valves`).

3- Summarize in a very concise and brief way the given class.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T7

Please summarize the method:

`org.netbeans.api.progress.ProgressUtils.runOffEventDispatchThread`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the method (runOffEventDispatchThread) in the netbeans project.

2- Open the source code of the method.

3- Read the source code.

4- Summarize in a very concise and brief way the given method.

Using Stack Overflow:

1- Open the link: http://stackoverflow.com/search?q=runOffEventDispatchThread

2- Search the method (runOffEventDispatchThread) in Stack Overflow, while considering the context (`org.netbeans.api.progress.ProgressUtils`).

3- Summarize in a very concise and brief way the given method.

Using Bug Report:

1- Open the link:

https://netbeans.org/bugzilla/buglist.cgi?bug_status=__all__&content=runOffEventDispatchThread&no_redirect=1&order=Importance&product=&query_format=specific

2- Search the method (runOffEventDispatchThread) in Bug reports, while considering the context (`org.netbeans.api.progress.ProgressUtils`).

3- Summarize in a very concise and brief way the given method.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

Task T8

Please summarize the class:

`org.openide.nodes.CHildFactory.Detachable`

YOUR SUMMARY: <participant writes summary here>

Using Source Code:

1- Search the class (Detachable) in the NetBeans project

2- Open the source code of the class.

3- Read the source code.

4- Summarize in a very concise and brief way the given class.

Using Stack Overflow:

1- Open the link: http://stackoverflow.com/search?q=ChildFactory.Detachable

2- - Search the class (Detachable) in Stack Overflow, while considering the context (`org.openide.nodes.CHildFactory`).

3- Summarize in a very concise and brief way the given class.

Using Bug Report:

1- Open the link:

https://netbeans.org/bugzilla/buglist.cgi?quicksearch=ChildFactory.Detachable

2- Search the class (Detachable) in Bug reports, while considering the context (`org.openide.nodes.CHildFactory`).

3- Summarize in a very concise and brief way the given class.

Please note that if the participant was instructed to use all contexts, they would need to check all the links given including source code.

A.6 Oracle against which summaries are evaluated

Task T1

Please summarize the method:

`org.eclipse.core.databinding.Binding.dispose`

Final Oracle

Disposes of the Binding object this method is called from. A call of this method usually results in the binding stopping to observe its dependencies by unregistering its listener(s).

Task T2

Please summarize the class:

`org.eclipse.swt.SWTError`

Final Oracle

A class representing an error internal to the SWT module. It contains a integer code and a throwable object.

SWTErrors are thrown when something fails internally which either leaves SWT in an unknown state or when SWT is left in a known-to-be-unrecoverable state.

Indicates that an internal error occurred in SWT, displaying the error code and a description of the problem.

Task T3

Please summarize the method:

`org.apache.jmeter.Jmeter.convertSubTree`

Final Oracle

Remove disabled elements. Replace the ReplaceableController with the target subtree.

Task T4

Please summarize the class:

`org.apache.jmeter.samplers.SampleResult`

Final Oracle

This is a nice packaging for the various information returned from taking a sample of an entry. Several helper methods are available for timing.

Task T5

Please summarize the method:

`org.apache.catalina.realm.JDBCRealm.getRoles`

Final Oracle

Returns a set of strings describing the roles associated with the user name. If no role store is defined and an authentication only configuration is used, it returns null.

Task T6

Please summarize the class:

`org.apache.catalina.valves.ValveBase`

Final Oracle

Convenience base class for implementations of the Valve interface. Handles the flow of requests between components in a container.

Task T7

Please summarize the method:

`org.netbeans.api.progress.ProgressUtils.runOffEventDispatch Thread`

Final Oracle

Runs operation out of event dispatch thread, blocks UI while operation is in progress. This method is supposed to be used by user invoked foreground actions, that are expected to run very fast in vast majority of cases.

Task T8

Please summarize the class:

`org.openide.nodes.CHildFactory.Detachable`

Final Oracle

Extension of child factory class providing methods for when the object is first used and last used

References

Armaly, Ameer, and Collin McMillan. 2016. “An Empirical Study of Blindness and Program Comprehension.” In *Proceedings of the 38th International Conference on Software Engineering Companion*, 683–685. ICSE ’16. New York, NY, USA: ACM. https://doi.org/10.1145/2889160.2891041.

Barik, Titus, Justin Smith, Kevin Lubick, Elisabeth Holmes, Jing Feng, Emerson Murphy-Hill, and Chris Parnin. 2017. “Do Developers Read Compiler Error Messages?” In *Proceedings of the 39th International Conference on Software Engineering*, 575–585. ICSE ’17. Piscataway, NJ, USA: IEEE Press. https://doi.org/10.1109/ICSE.2017.59.

Bhatia, Sumit, Shibamouli Lahiri, and Prasenjit Mitra. 2009. “Generating Synopses for Document-Element Search.” In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2003–2006. CIKM ’09. New York, NY, USA: ACM. https://doi.org/10.1145/1645953.1646287.

Buse, Raymond P.L., and Westley R. Weimer. 2010. “Automatically Documenting Program Changes.” In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 33–42. ASE ’10. New York, NY, USA: ACM. https://doi.org/10.1145/1858996.1859005.

Busjahn, Teresa, Roman Bednarik, Andrew Begel, Martha Crosby, James Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. “Eye Movements in Code Reading: Relaxing the Linear Order.” In *International Conference on*

Program Comprehension, 255–65. IEEE. http://dl.acm.org/citation.cfm?id=2820320.

Busjahn, Teresa, Carsten Schulte, Bonita Sharif, Simon, Andrew Begel, Michael Hansen, Roman Bednarik, et al. 2014. "Eye Tracking in Computing Education." In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 3–10. New York, NY, USA: ACM. https://doi.org/10.1145/2632320.2632344.

Fowkes, Jaroslav, Pankajan Chanthirasegaran, Razvan Ranca, Miltiadis Allamanis, Mirella Lapata, and Charles Sutton. 2016. "TASSAL: Autofolding for Source Code Summarization." In *Proceedings of the 38th International Conference on Software Engineering Companion*, 649–652. ICSE '16. New York, NY, USA: ACM. https://doi.org/10.1145/2889160.2889171.

Haiduc, S., J. Aponte, and A. Marcus. 2010. "Supporting Program Comprehension with Source Code Summarization." In *Software Engineering, 2010 ACM/IEEE 32nd International Conference On*, 223–26.

Just, M., and P. Carpenter. 1980. "A Theory of Reading: From Eye Fixations to Comprehension." *Psychological Review* 87: 329–354.

Lazar, Alina, Rachel Turner, Michael Falcone, and Bonita Sharif. 2014. "An Eye-Tracking Study Assessing the Comprehension of C++ and Python Source Code." In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 231–234. ETRA '14. New York, NY, USA: ACM. https://doi.org/10.1145/2578153.2578218.

Linares-Vásquez, Mario, Luis Fernando Cortés-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. “ChangeScribe: A Tool for Automatically Generating Commit Messages.” In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, 709–712. ICSE ’15. Piscataway, NJ, USA: IEEE Press. http://dl.acm.org/citation.cfm?id=2819009.2819144.

McBurney, Paul W. 2015. “Automatic Documentation Generation via Source Code Summarization.” In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, 903–906. ICSE ’15. Piscataway, NJ, USA: IEEE Press. http://dl.acm.org/citation.cfm?id=2819009.2819210.

McBurney, Paul W., Cheng Liu, Collin McMillan, and Tim Weninger. 2014. “Improving Topic Model Source Code Summarization.” In *Proceedings of the 22Nd International Conference on Program Comprehension*, 291–294. ICPC 2014. New York, NY, USA: ACM. https://doi.org/10.1145/2597008.2597793.

McBurney, Paul W., and Collin McMillan. 2014. “Automatic Documentation Generation via Source Code Summarization of Method Context.” In *Proceedings of the 22Nd International Conference on Program Comprehension*, 279–290. ICPC 2014. New York, NY, USA: ACM. https://doi.org/10.1145/2597008.2597149.

Moreno, Laura. 2014. “Summarization of Complex Software Artifacts.” In *Companion Proceedings of the 36th International Conference on Software Engineering*, 654–657. ICSE Companion 2014. New York, NY, USA: ACM. https://doi.org/10.1145/2591062.2591096.

Moreno, Laura, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2014. "Automatic Generation of Release Notes." In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 484–495. FSE 2014. New York, NY, USA: ACM. https://doi.org/10.1145/2635868.2635870.

Rodeghero, P., C. Liu, P. W. McBurney, and C. McMillan. 2015. "An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization." *IEEE Transactions on Software Engineering* 41 (11): 1038–54. https://doi.org/10.1109/TSE.2015.2442238.

Rodeghero, P., and C. McMillan. 2015. "An Empirical Study on the Patterns of Eye Movement during Summarization Tasks." In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–10. https://doi.org/10.1109/ESEM.2015.7321188.

Rodeghero, Paige, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney D'Mello. 2014. "Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers." In *Proceedings of the 36th International Conference on Software Engineering*, 390–401. New York, NY, USA: ACM. https://doi.org/10.1145/2568225.2568247.

Sharafi, Zohreh, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. 2015. "A Systematic Literature Review on the Usage of Eye-Tracking in Software Engineering." *Inf. Softw. Technol.* 67 (C): 79–107. https://doi.org/10.1016/j.infsof.2015.06.008.

Sharafi, Zohreh, Zéphyrin Soh, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2012. "Women and Men - Different but Equal: On the Impact of Identifier Style on Source Code Reading." *Program Comprehension (ICPC), 2012 IEEE 20th International Conference On*, 27–36. https://doi.org/10.1109/ICPC.2012.6240505.

Sharif, Bonita, Katja Kevic, Braden M. Walters, Timothy R. Shaffer, David C. Shepherd, and Thomas Fritz. 2015. "Tracing Software Developers' Eyes and Interactions for Change Tasks." In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 202–213. ESEC/FSE 2015. New York, NY, USA: ACM. https://doi.org/10.1145/2786805.2786864.

Sharif, Bonita, and Jonathan I. Maletic. 2010a. "An Eye Tracking Study on CamelCase and Under_Score Identifier Styles." In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*, 196–205. ICPC '10. Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/ICPC.2010.41.

Sharif, Bonita, and Jonathan I Maletic. 2010b. "An Eye Tracking Study on the Effects of Layout in Understanding the Role of Design Patterns." *Software Maintenance (ICSM), 2010 IEEE International Conference On*, 1–10. https://doi.org/10.1109/ICSM.2010.5609582.

Sharif, Bonita, Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, and Michael Falcone. 2015. "ITrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks." In *Proceedings* *of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 954–957. ESEC/FSE 2015. New York, NY, USA: ACM. https://doi.org/10.1145/2786805.2803188.

Sridhara, G., L. Pollock, and K. Vijay-Shanker. 2011. “Generating Parameter Comments and Integrating with Method Summaries.” In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference On*, 71–80. https://doi.org/10.1109/ICPC.2011.28.

Sridhara, Giriprasad, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. 2010. “Towards Automatically Generating Summary Comments for Java Methods.” In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, 43–52. New York, NY, USA: ACM. https://doi.org/10.1145/1858996.1859006.

Sridhara, Giriprasad, Lori Pollock, and K. Vijay-Shanker. 2011. “Automatically Detecting and Describing High Level Actions Within Methods.” In *Proceedings of the 33rd International Conference on Software Engineering*, 101–110. ICSE ’11. New York, NY, USA: ACM. https://doi.org/10.1145/1985793.1985808.