

UNIVERSITY OF CALIFORNIA,
IRVINE

How Software Developers Solve Problems by Searching for Source Code on the Web:
Studies on Judgments in Evaluation of Results and Information Use

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Information and Computer Sciences

by

Rosalva Eulogia Gallardo Valencia

Dissertation Committee:
Professor Susan Elliott Sim, Chair
Professor Cristina Videira Lopes
Professor James Arthur Jones

2012

UMI Number: 3499527

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent on the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI 3499527

Copyright 2012 by ProQuest LLC.

All rights reserved. This edition of the work is protected against unauthorized copying under Title 17, United States Code.



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

DEDICATION

This dissertation is dedicated to my parents
Vitaliano Gallardo Cuadra and Yolanda Valencia Lara
whom I admire and love so much.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	vi
LIST OF TABLES	vii
ACKNOWLEDGMENTS	viii
CURRICULUM VITAE	ix
ABSTRACT OF THE DISSERTATION	xii
1 Introduction	1
1.1 Introduction	1
1.2 Thesis Statement	5
1.3 Contributions	8
1.4 Organization	11
2 Background and Motivation	13
2.1 Searching for Source Code on the Web: a Common Practice	13
2.2 Previous Studies	14
2.2.1 Source Code Search Process	14
2.2.2 Empirical Studies and Methods	15
2.2.3 Our Focus	17
3 Empirical Studies	19
3.1 Overview	19
3.2 Online Questionnaire	21
3.2.1 Motivation	21
3.2.2 Design	21
3.2.3 Participants	22
3.2.4 Key Results	22
3.3 Focus Group	23
3.3.1 Motivation	23
3.3.2 Design	23
3.3.3 Participants	25
3.3.4 Key Results	25

3.4	Laboratory Study	27
3.4.1	Motivation	27
3.4.2	Design	27
3.4.3	Participants	30
3.4.4	Key Results	31
3.5	Field Studies	32
3.5.1	Motivation	32
3.5.2	Design	32
3.5.3	Participants	34
3.5.4	Key Results	36
3.6	Summary	37
4	Software Problems that Motivate Web Searches	39
4.1	Opportunistic Searches	40
4.1.1	Web Search is Used to Explore Further and Find Missing Information to Complete Software Development Tasks	41
4.1.2	Knowledge is Partially and Incrementally Gathered while Developers Search on the Web	45
4.1.3	Searches on the Web do not Follow a Well-Planned Process	47
4.2	Non-Opportunistic Searches	49
4.2.1	Web Search is Used to Find Open Source Projects to Reuse As-Is	49
4.2.2	Knowledge is Gathered for Criteria to Evaluate Open Source Candidates	50
4.2.3	Searches Commonly Follow a Planned Process	51
4.3	Discussion	51
4.3.1	Understanding of What Motivates Web Searches	52
4.3.2	Software Development Problems and Search Targets	56
4.3.3	Looking for Code Snippets and Looking for Open Source Projects are Different Problems	61
4.3.4	Classification of Tools for Opportunistic and Non-Opportunistic Searches	62
4.4	Summary	68
5	Judgments in Evaluation of Results	69
5.1	Types of Judgments in Source Code Search on the Web	69
5.1.1	Overview of Results	72
5.2	Relevance Cues Used to Evaluate Search Results	73
5.2.1	Relevance Cues Used for Opportunistic Searches	76
5.2.2	Relevance Cues Used for Non-Opportunistic Searches	85
5.3	The Process to Evaluate Search Results	88
5.3.1	Characteristics of Evaluation of Search Results in Source Code Search on the Web	89
5.3.2	Naturalistic Decision Making in Source Code Search on the Web	90
5.3.3	Developers Make Recognition-Primed Decisions for Opportunistic Searches	99

5.3.4	Developers do not Make Recognition-Primed Decisions for Non-Opportunistic Searches	111
5.4	Discussion	115
5.4.1	Comparison of Our Results and Results in the Literature	115
5.4.2	Evaluation of Results in Opportunistic Searches	115
5.4.3	Evaluation of Results in Non-Opportunistic Searches	117
5.4.4	Laboratory Experiment versus Field Studies	119
5.5	Summary	123
6	Use of Information in Web Searches	125
6.1	Use of Information in Web Searches	126
6.2	Use of Information in Opportunistic Searches	127
6.3	Use of Information in Non-Opportunistic Searches	132
6.4	Discussion	133
6.4.1	Comparison of Our Results and Results in the Literature	133
6.4.2	Use of Information in Opportunistic Searches	134
6.4.3	Use of Information in Non-Opportunistic Searches	136
6.4.4	Are Developers Efficiently Solving Software Problems by Searching on the Web?	137
6.4.5	Copying and Pasting Source Code versus Reading and Using it as Reference	141
6.5	Summary	143
7	Implications and Discussion of Research Methods Used	145
7.1	Implications for Tools	145
7.1.1	Implications for Tools for Opportunistic Searches	146
7.1.2	Implications for Tools for Non-Opportunistic Searches	148
7.2	Implications for Researchers	150
7.2.1	Extending Understanding of Source Code Search on the Web	150
7.2.2	Searches for Code Snippets and Software Components are Different	151
7.2.3	Applying Naturalistic Decision Making Theories	152
7.3	Implications for Developers	153
7.4	Discussion of Research Methods Used	154
8	Conclusion and Future Work	158
8.1	Conclusion	158
8.2	Future Work	164
	Bibliography	166
	Appendices	172
A	Glossary	172
B	Online Questionnaire	175

LIST OF FIGURES

	Page
2.1 Source Code Search Process	14
2.2 Empirical Studies for Phases of the Source Code Search Process . . .	15
2.3 Phases of the Source Code Search Process Emphasized in Our Approach	18
3.1 Overview of Empirical Studies	20
4.1 Problems that Motivate Web Searches	41
4.2 Length of Web Searches by Type of Software Problems	42
4.3 Frequency of Self-Reported Web Searches in Our Empirical Studies .	48
4.4 Comparison of Results from Online Questionnaire and Field Studies for Motivations behind Web Searches	53
4.5 Online Questionnaire: Motivations and Search Targets	55
4.6 Concept Lattice for Examples and Categories in Focus Group	60
5.1 Correlation between Emphasis of Judgments and Certainty Level of Expected Results by Type of Software Problems	71
5.2 Sections in Search Results Presented by Google	74
5.3 Sections in Search Results Presented in our Laboratory Experiment .	74
5.4 Time of First Visit to Promising Candidate	90
7.1 Compare Projects Feature by Ohloh	149

LIST OF TABLES

	Page
1.1 Overview of Empirical Studies	4
1.2 Summary of Results	6
2.1 Summary of Empirical Studies in the Literature	16
3.1 Source Code Search Targets Used in Card Sorting Task	24
3.2 List of Tasks in Laboratory Experiment	29
3.3 Summary of Participants and Companies in Field Studies	35
3.4 Summary of Empirical Studies	38
4.1 Query Reformulation by Type of Software Problems	46
4.2 Search Targets by Type of Software Problems	57
4.3 Tool Classification by Type of Searches	63
5.1 Summary of Evaluation of Results by Type of Software Problems	72
5.2 Relevance Cues by Type of Software Problems from Field Studies	75
5.3 Relevance Cues by Type of Software Problems from Laboratory Study	76
5.4 Number of Relevance Cues Used by Type of Software Problems	85
5.5 Summary of NDM's Characteristics Applied to Source Code Search on the Web	92
5.6 Actions and Number of Results Visited by Type of Software Problems	103
5.7 Option Evaluation by Type of Software Problems from Field Studies	106
5.8 Option Evaluation by Type of Software Problems from Laboratory Study	107
5.9 Simulation Used by Type of Software Problems	110
5.10 Relevance Cues Reported in Literature for Opportunistic Searches	116
5.11 Relevance Cues Reported in Literature for Non-Opportunistic Searches	118
6.1 Use of Information by Type of Software Problems from Field Studies	127
6.2 Use of Information by Type of Software Problems from Laboratory Experiments	128
6.3 Use of Information Reported in Literature for Opportunistic Searches	134
6.4 Success and Length of Efficient Searches by Type of Software Problems	138
6.5 Failure and Length of Non-Efficient Searches by Type of Software Problems	139
6.6 Reasons for Giving Up Search Sessions by Type of Software Problems	140
6.7 Actions after Giving Up Search Sessions by Reasons for Giving Up	141

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Professor Susan Elliott Sim for giving me the opportunity to pursue my graduate studies at UCI and for her unconditional support during the entire Ph.D. journey. Her patience, dedication, encouragement, willingness to help and listen, and support on academic and personal issues were key for my success in graduate school. Her guidance helped me to be a better researcher and a better person. Thank you so much.

I would also like to thank the members of my dissertation committee: Professor Cristina Lopes and Professor Jim Jones, for their encouragement, insightful comments, and challenging questions.

Thanks to my friends and lab-mates: Sukanya Ratanotayanon, Marisa Cohn, Hye Jung Choi, Albert Thompson, Phitchayaphong Tantikul, Leyna Cotran, Medha Umarji, Megha Agarwal, and Vivian Olivera, for their friendship, great discussions, and the nice time we spent together. I also thank my friends at UCI: Alegria Baquero, Joel Ossher, Nicolas Mangano, Ruy Cervantes, Francisco Servant, Nicolas Lopez, Sushil Bajracharya, Kristina Winbladh, Yongjie Zheng, Nithya Sambasivan, Tijana Milenkovic, and Vesna Memisevic, for their friendship, valuable feedback, and fun times in graduate school.

Special thanks to Phitchayaphong Tantikul who helped me implement the browser plug-in to collect information on Web searches. Also, thanks to Fernando Figueira Filho who granted me permission to extend his browser plugin. My gratitude to Arthur Valadares and Nilmax Moura for their help collecting data from field studies and to Marcos Mercado, Jim Milewski, Susan Lin, and Buda Chiou for their help collecting data from laboratory experiments. I would also like to thank Vanessa Abanto and Lucia Lopez for their help transcribing interviews.

I am very grateful to Carlos Uribe, Guillermo Pacheco, Jose Pacheco, Gerald Bortis, and Andrew Mutz for allowing me to conduct field studies in their companies. Thanks to all the participants in the field studies. This research was possible due to their generosity in sharing their time and work with us. Also thanks to our participants in online questionnaires, focus groups, and laboratory experiments.

I would like to thank the Donald Bren School of Information and Computer Sciences at UCI and NSF (under Grant No. IIS-0846034) for their financial support.

Last but not least, I would like to thank my parents, whom I dedicate this dissertation, for their love, their encouragement, and support every day of my life. Also, thanks to my siblings: Vitaliano, Beatriz, and Julian who were always there to help me. Thanks to my family for the sacrifices they made to give me the best education and life. Thanks also to Juana Gomez who took care of me since I was born. Special thanks to Sylvia Lauwerys for making me feel at home in the US all these years.

CURRICULUM VITAE

Rosalva Eulogia Gallardo Valencia

EDUCATION

Ph.D. in Information and Computer Sciences University of California, Irvine	2012 <i>Irvine, California</i>
M.S. in Information and Computer Sciences University of California, Irvine	2009 <i>Irvine, California</i>
B.S. in Computer Science Engineering Pontifical Catholic University of Peru	2000 <i>Lima, Peru</i>

RESEARCH EXPERIENCE

Graduate Research Assistant University of California, Irvine	2006–2012 <i>Irvine, California</i>
Graduate Level Co-op/Summer Intern IBM Research. T.J.Watson Research Center	2008 <i>Hawthorne, New York</i>

PROFESSIONAL EXPERIENCE

Localization Software Engineer Intern Laserfiche Corporate	2007 <i>Long Beach, California</i>
Project Manager and Analyst Programmer Novatronic	2003–2006 <i>Lima, Peru</i>
Analyst, Programmer, and Technical Support Pontifical Catholic University of Peru	1999–2003 <i>Lima, Peru</i>

TEACHING EXPERIENCE

Teaching Assistant and Reader University of California, Irvine	2007–2009 <i>Irvine, California</i>
Teacher Assistant and Laboratory Assistant Pontifical Catholic University of Peru	1999–2002, 2004 <i>Lima, Peru</i>

REFEREED CONFERENCE AND WORKSHOP PUBLICATIONS

- Software Reuse Through Methodical Component Reuse and Amethodical Snippet Remixing** Feb. 2012
Conference on Computer Supported Collaborative Work (CSCW 2012)
- What Kinds of Development Problems Can Be Solved by Searching the Web?: A Field Study** May 2011
Third International Workshop on Search-Driven Development (SUITE 2011)
- Information Used and Perceived Usefulness in Evaluating Web Source Code Search Results** May 2011
Work-In-Progress Track at the Conference on Human Factors in Computing Systems (CHI 2011)
- Searching for Reputable Source Code on the Web** Nov. 2010
Group Conference 2010
- Internet-Scale Code Search** May 2009
First International Workshop on Search-Driven Development (SUITE 2009)
- Continuous and Collaborative Validation: A Field Study of Requirements Knowledge in Agile** Sep. 2009
Second International Workshop on Managing Requirements Knowledge (MARK 2009)
- Supporting Program Comprehension in Agile with Links to User Stories** Aug. 2009
Agile 2009 Conference
- Practical Experiments are Informative, but Never Perfect** Nov. 2007
First Workshop on Empirical Assessment of Software Engineering Languages and Technologies (WEASELTech 2007)
- Are Use Cases Beneficial for Developers Using Agile Requirements?** Oct. 2007
Fifth International Workshop on Comparative Evaluation in Requirements Engineering (CERE 2007)

REFEREED JOURNAL PUBLICATIONS

- Planning and Improvisation in Software Processes** 2007
ACM Crossroads

BOOK CHAPTER

Agile Requirements Engineering. In Maalej, Walid and Thurimella, Anil (Eds.), *Managing Requirements Knowledge*. To appear. 2012
Springer

BOOK

Finding Source Code on the Web for Remix and Reuse. 2012
To appear.
Springer

SELECTED HONORS AND AWARDS

Miguel Velez Fellowship Award 2007–2008
University of California, Irvine

Information and Computer Sciences (ICS) Fellowship 2006–2009
University of California, Irvine

Novatronic Overall Performance Award 2004
Novatronic

2nd in the 2000-II Computer Science Graduating Class 2000
Pontifical Catholic University of Peru

PROFESSIONAL SERVICE

Student Volunteers Chair 2011
International Conference of Software Engineering *Waikiki, Hawaii*

Program Committee 2011–2012
International Workshop on Search-Driven Development

Organizing Committee 2009–2011
Agile Open Southern California *Irvine, California*

Student Volunteer 2008
International Requirements Engineering Conference *Barcelona, Spain*

Organizing Committee 2008
ISR Graduate Student Research Symposium *Irvine, California*

PROFESSIONAL MEMBERSHIPS

Association for Computing Machinery (ACM)
Institute of Electrical and Electronics Engineers (IEEE)

ABSTRACT OF THE DISSERTATION

How Software Developers Solve Problems by Searching for Source Code on the Web:
Studies on Judgments in Evaluation of Results and Information Use

By

Rosalva Eulogia Gallardo Valencia

Doctor of Philosophy in Information and Computer Sciences

University of California, Irvine, 2012

Professor Susan Elliott Sim, Chair

The large amount of information available on the Web has changed the way people develop software. Even though looking for source code on the Web is a common practice among developers, little is known about what motivates developers to look for source code on the Web, how developers evaluate search results, how they use the information they find, and how efficient are these Web searches in helping them complete software tasks. We found that looking for source code on the Web is a common activity for software developers because it helps them solve software development problems efficiently. Developers evaluate search results by making quick judgments and examining options in a serial fashion until a good-enough candidate is found. Information found on the Web is used to build developers knowledge or to guide their coding. Using a series of empirical studies including online questionnaires, focus groups, laboratory experiments, and field studies in the US and abroad, we gained a better understanding of how software developers solve problems by searching for source code on the Web. We found that 83% of developers performed at least one Web search during a work day and on average they did 3.6 searches per day. We also found that 82% of Web searches are done to solve opportunistic problems, such as when developers need to remember syntax details, to clarify implementation details or fix

bugs, and to learn new concepts. These searches are not planned ahead of time; they are done as they are needed. Using a naturalistic decision making approach, we found that developers make rapid judgments to evaluate search results in a serial fashion to find a good-enough candidate to solve their opportunistic software problems. We also found that developers are able to successfully solve 63% of their opportunistic software problems in 4.9 minutes on average by using the information they found on the Web to build their knowledge or to guide their coding. Results from these empirical studies have implications for tool designers, researchers, and developers.

Chapter 1

Introduction

In this chapter, we present the introduction of this dissertation, the thesis statement that summarizes our results, and our contributions.

1.1 Introduction

Code search is a critical part of software development. A study of software engineering work practices found that searching was the most common activity for software engineers [57]. They were typically locating a bug or a problem, finding ways to fix it and then evaluating the impact on other segments. Program comprehension, code reuse, and bug fixing were cited as the chief motivations for source code searching in that study. A related study on source code searching by Sim, Clarke, and Holt [54] found that the search goals cited frequently by developers were code reuse, defect repair, program understanding, feature addition, and impact analysis. They found that programmers were most frequently looking for function definitions, variable definitions, all uses of a function and all uses of a variable.

The recognition that search is powerful and useful has led to advances in code search tools. Software developers have needed tools to search through source code since the appearance of interactive programming environments. It started with simple keyword search and when regular expressions were added, it became possible to specify patterns and context [62]. An important improvement was made when search techniques started using program structure, such as identifiers of variables and functions, directly in expressing search patterns [1, 43].

Another approach to syntactic search involves processing the program and storing facts in a database file of entity-relations [9, 32]. Alternatively, the code can be parsed and transformed into other representations, such as data flow graphs or control flow graphs, and searches can be performed on those structures [39]. While some of these ideas have not been widely adopted, searches using regular expressions and program structure are standard in modern IDEs.

The prevalence and need for search is just as important, if not more, in the context of Internet-scale code search [13]. Similarly, the mechanisms for specifying searches and making source code searchable are just as applicable. However, there are two important differences: scale and the information sought. Conventional code search is concerned with searching for specific information within the context of a single project. A project could be very large and even have multiple programming languages, it would still be a single project with a constrained set of compilation units. With Internet-scale code search, the challenge is to examine the contents of many different projects, i.e. thousands of projects and billions of lines of code. Also, in this context, the software developer is less concerned with finding whether a variable or function is defined, and more interested in finding functionality, a typical concern in code reuse. It is also common to use the Web as a giant desk reference. Gone are the days of programmers keeping thick reference manuals on their desks.

Nowadays, they search the Web to find information in order to solve the problems they encounter while working on software development tasks. A recent study [5] have found that developers spend 19% of their programming time on the Web.

Looking for source code on the Web is a common practice among developers, but little is known about what motivates developers to look on the Web, how they evaluate source code results, and how they use the information they find. To fill this gap, we studied the phenomenon of looking for source code on the Web with the goal of answering the following research questions:

- RQ1: What motivates developers to search the Web to find source code to complete their software development tasks?
- RQ2: What information and strategies do developers use to evaluate search results when they perform Web searches to find source code?
- RQ3: What strategies do developers use to reuse/use source code found on the Web?
- RQ4: Are source code searches on the Web efficient to complete software development tasks?
- RQ5: What are the implications of our results for tool designers, researchers, and developers?

To answer these research questions, we conducted a set of complementary empirical studies. We gathered developers' opinions on how they use Web searches to complete tasks by collecting their reflections via an online questionnaire and team reflections via a focus group. We wanted to observe developers not only talking about code search but also actually performing code search on the Web. For that purpose,

Study	Subjects	Methods
Online Questionnaire	26 professional developers	Questionnaire with multiple choice questions
Focus Group	24 graduate and undergraduate students in Informatics and Computer Science	Card sorting game Follow up questions
Laboratory Experiment	16 graduate and undergraduate students in Informatics and Computer Science	Complete four programming tasks using pre-made search results
Field Studies	12 professional developers were observed in the US 12 professional developers were observed in Peru	Observations Automatically logged Web searches

Table 1.1: Overview of Empirical Studies

we conducted field studies in companies in the US and abroad, which provided insights into how developers use code search in the context of other workplace activities. In addition, we also conducted laboratory experiments to test specific hypothesis where we needed to control variables.

We conducted our empirical studies with around 90 software developers with a good balance of professional developers and students: 56% of our participants were professional developers and 44% of participants were students. We have a representative number of developers with different levels of expertise, domains of expertise, cultural backgrounds, and ages. Table 1.1 gives a summary of our empirical studies including the number of participants and the methods used.

Data collected in our empirical studies has been analyzed separately for each study and subsequently integrated and compared to other results from the literature. From this work, we found that looking for source code on the Web is a common activity for software developers because it helps them solve software development problems efficiently. Developers evaluate search results by making quick judgments and examining options in a serial fashion until a good-enough candidate is found.

Information found on the Web is used to build developers knowledge or to guide their coding. Table 1.2 shows a summary of our results for our first four research questions. This table also shows the mapping between the phases of our model of source code search on the Web, our research questions, and our results.

1.2 Thesis Statement

The thesis statement of this dissertation is that looking for source code on the Web is a common activity for software developers because it helps them solve software development problems efficiently. Developers evaluate search results by making quick judgments and examining options in a serial fashion until a good-enough candidate is found. Information found on the Web is used to build developers knowledge or to guide their coding.

Our thesis statement has four parts. The first part claims that “looking for source code on the Web is a common activity for software developers because it helps them solve software development problems.” We make this claim based on results from our empirical studies, which showed that eighty-three percent of developers performed at least one Web search during a work day and on average they did 3.6 searches per day. We found that 82% of Web searches are performed with the goal of finding a solution to an opportunistic problem discovered while working on a software task. These opportunistic problems arise when developers do not have complete information to finish their tasks. We identified three main reasons that motivated opportunistic Web searches: when developers needed to remember syntax details, to clarify implementation details or fix bugs, and to learn new concepts [5]. Developers formulate queries to Web search engines to find solutions to these problems in terms of examples or source code snippets. Developers partially and incrementally learn

		Identify Software Development Problem			Evaluate Results				Use Suitable Results			
		RQ1			RQ2				RQ3			RQ4
Motivation	Frequency	Search Target	Actions	Evaluation Strategy	Relevant Cues	# Cues Used	Simulation	Use of Information	Success	Avg. Time Successful		
Opportunistic Searches	25%	- Example/Code Snippet - API Documentation/Tutorial	1 Query 1 Result	Only One Result	- Result Order - Example/Code Snippet - Web Host Domain - Social Cues	2	Coding/ Testing	- Read and Code - Read and Understand	82%	2.7 min		
	43%	- API Documentation/Tutorial - Example/Code Snippet - Error Related Information	1-8 Queries 2-13 Results	Serial	- Result Order - Example/Code Snippet - Page Type - Web Host Domain	3	Coding/ Testing	- Read and Understand - Read and Code	49%	6.2 min		
	14%	- API Documentation/Tutorial - Example/Code Snippet	2-8 Queries 2-13 Results	Serial	- Result Order - Web Host Domain - Page Type - Title	2 or 3	Mental	- Read and Understand - Read and Code	75%	5.9 min		
Non-Opportunistic Searches	18%	- Open Source Project/ Software Tool	1 Query 1 Result	Comparison Between	Within Search Sessions: - Result Order - Web Host Domain - Page Type - Title Between Search Sessions: - System Architecture - Installation Requirements - Cost - What other people think	2	Mental	- Read and Understand - Download and Try it	88%	Search Session: 11.5 min Solve Problem: More than a day		

Table 1.2: Summary of Results

more about the problem and the keywords they should use while evaluating search results. Opportunistic searches were not planned ahead of time; they were done as they were needed. For the other 18% of Web searches, which were non-opportunistic searches, developers wanted to find an open source project to reuse.

The second part of our thesis statement claims that “developers evaluate search results by making quick judgments and examining options in a serial fashion until a good-enough candidate is found.” This claim is based on our analysis of empirical evidence using a naturalistic decision making approach that shows that evaluating search results and finding a solution to developers’ software problems on the Web did not require a careful comparison of options. Instead, developers made rapid judgments to evaluate search results serially to find a good-enough solution to solve their software problems. We found that developers visit the first promising candidate in less than 10 second for 79% of queries.

The third part of our thesis statement claims that “information found on the Web is used to build developers knowledge or to guide their coding.” This claim is based on empirical evidence that shows that developers mainly read the information from the Web and use it to understand it or to guide their coding. Copy and paste from the Web was not very common. Choosing between copying and pasting source code and reading and typing it depends on personal preferences of developers or also on the problem they want to solve and the information they find on the Web. However, some developers did not see any distinction between these two practices.

The fourth part of our thesis statement claims that “looking for source code on the Web helps software developers solve software development problems efficiently.” We define efficiency in terms of the success of a Web search to help solve a software development problem and in terms of the time it takes to solve a problem using a Web search. Our fourth claim is based on our results that show that developers were

able to successfully solve 63% of their opportunistic software problems in 4.9 minutes on average using information found on the Web. The time taken to solve problems ranged between 1 second and 38 minutes. Most of the opportunistic problems that developers could not solve using Web searches were to clarify implementation details or find how to fix errors. In these cases, developers did not find anything useful on the Web or they found a starting point for a solution. After developers gave up on a Web search, they tended to solve the problem by coding a solution by themselves or asking for advice to a co-worker. In other cases, developers decided to postpone dealing with the problem for the moment.

1.3 Contributions

In this section, we list the contributions made with this dissertation in four areas: A) new model, B) research methods used, C) application of theories, and D) extending the understanding of source code search on the Web regarding motivations, evaluation of results, use of information, and efficiency of Web searches to solve software problems.

A) New Model

1. A model to characterize the process of source code search on the Web. *No model for the source code search on the Web process existed before.*

B) Research Methods Used

2. First study to understand source code search on the Web based on direct observations of developers in the workplace. *Previous work reported on laboratory experiments, interviews, surveys, and analysis of system's logs.*

C) Application of Theories

3. Application of the Opportunistic Problem Solving approach to analyze motivations of developers to look for source code on the Web. *First to find an explanatory theory to understand motivations of developers and identify opportunistic and non-opportunistic searches.*
4. Application of the Naturalistic Decision Making theory and Recognition-Primed Decision Model of Rapid Decision Making to analyze judgments made by developers to evaluate search results. *First to apply these theories to code search on the Web and to try to find an explanatory theory to the evaluation of result in code search on the Web.*

D) Extended Understanding of Source Code Search on the Web

5. Empirical evidence that source code search on the Web is more common than what developers report. *First to report this.*
6. Empirical evidence that looking for code snippets and open source projects are two different problems. *First to report this.*
7. Identified 9 categories of existing tools that support source code search on the Web and classified these tools on their support for searches to look for code snippets and searches for open source projects. *First to do this classification.*
8. Provided implications for tool designers, researchers, and developers about source code search on the Web. *First time implications are given based on direct observation of developers in the workplace.*

D.1) Extended Understanding of Source Code Search on the Web: Motivations

9. Confirmed the three motivations reported in the literature[5] to look for source code on the Web and added a new motivation (looking for open source projects). *First to confirm the three previously reported motivations and added a new category.*

For each type of motivation to look for source code on the Web:

10. Provided frequency of motivations. *It has not been previously reported.*
11. Identified 5 categories of search targets and their frequency. *First to do this categorization for each motivation.*

D.2) Extended Understanding of Source Code Search on the Web: Evaluation of Results

12. Empirical evidence that the evaluation of search results to select code snippets was quick and options were evaluated in a serial fashion until a good-enough candidate was found. *First to report this.*

For each type of motivation to look for source code on the Web:

13. Identified 9 relevance cues and their frequency used to evaluate search results. *The literature reported 3 cues (source code, cosmetic features, and official documentation), we added 6 cues.*
14. Identified 4 strategies to evaluate source code search results and provide their frequency. *First to report this.*

D.3) Extended Understanding of Source Code Search on the Web: Use of Information

15. Empirical evidence that developers more often use the information they found on the Web to read it and understand it or to read it and guide their coding than

to copy and paste it. *First to report this. The literature reports that developers often copy and paste.*

For each type of motivation to look for source code on the Web:

16. Identified 5 categories of how developers use found information from the Web and their frequency. *First to do this categorization for each motivation.*

D.4) Extended Understanding of Source Code Search on the Web: Efficiency

17. Empirical evidence that developers efficiently solve software problems by searching on the Web. *First to study the efficiency of Web searches to solve software development problems.*

For each type of motivation to look for source code on the Web:

18. Provided the ratio of success of Web searches to solve software problems.
19. Provided the minimum, average, and maximum time that took developers to solve a software problem by performing Web searches or to give up Web searches.
20. Identified 4 reasons that motivate developers to give up Web searches.
21. Identified 5 actions that developers follow to solve problems when they did not find what they were looking for on the Web.

1.4 Organization

This dissertation is organized as follows. Chapter 2 presents our motivation to do this work and what has been done so far in this field. Chapter 3 discusses the set of empirical studies we designed and conducted to better understand the phenomenon

of looking for source code on the Web. The next three chapters focus on each of the areas of the search process we are focusing and present results based on our empirical studies. Chapter 4 presents a classification of the software development problems that motivate developers to search on the Web. Our results related to the evaluation of source code search results and the use of suitable results are presented in Chapter 5 and 6 respectively. Chapter 7 discusses the implications of our results for tool designers, researchers, and developers. Finally, Chapter 8 presents the conclusions of this dissertation and future work.

Chapter 2

Background and Motivation

In this chapter, we report on what motivates our work as well as on the current knowledge on how developers look for source code on the Web from the literature. Our main motivation to study developers looking for source code on the Web is that it is a common practice among developers. However, it has not been extensively studied in the literature. Specifically, we do not know much about the kinds of problems that motivate developers to look for source code on the Web, and the strategies used to evaluate search results and to use source code found on the Web. We conducted a series of empirical studies to gain knowledge on these areas.

2.1 Searching for Source Code on the Web: a Common Practice

Searching the Web to look for ways to solve software development problems is a very common practice among software developers. In our field studies, we found that 83% of developers looked for source code on the Web the day they were observed.

Although this is a common practice among developers, little is known about the problems that developers want to solve by looking for source code on the Web, as well as how developers evaluate results and how they use the source code they find.

2.2 Previous Studies

We surveyed the literature to better understand what is the current knowledge about the phenomenon of looking for source code on the Web. We first present our source code search process model used to review the literature and then we provide an overview of the 8 empirical studies found in the literature.

2.2.1 Source Code Search Process

We characterized the source code search process in five stages as showed in Figure 2.1. We used this model to review the empirical studies that report how developers look for source code on the Web. Other models in the information seeking, information retrieval, and information behavior literature influenced this model. Marchionini's [37] and Sutcliffe and Ennis's [60] models influenced ours by their problem solving approach that motivates the search process. Wilson's model [66] motivated the inclusion of the information use phase.

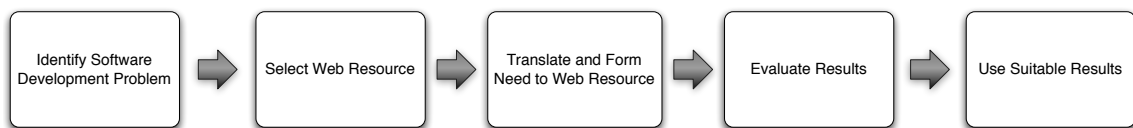


Figure 2.1: Source Code Search Process

2.2.2 Empirical Studies and Methods

There are few empirical studies on how developers look for source code on the Web. We analyzed the eight studies we found in the literature and we noticed that none of these empirical studies observed professional developers in the workplace. We also found that even though these studies provide some initial understanding on how developers look for source code on the Web, still little is known about how they evaluate search results and how they use the information they find to solve development problems. Table 2.1 presents a summary of the characteristics of these empirical studies including the authors' names, studies' goal, research methods used, participants in the studies, and the tool proposed by researchers to support source code search on the Web.

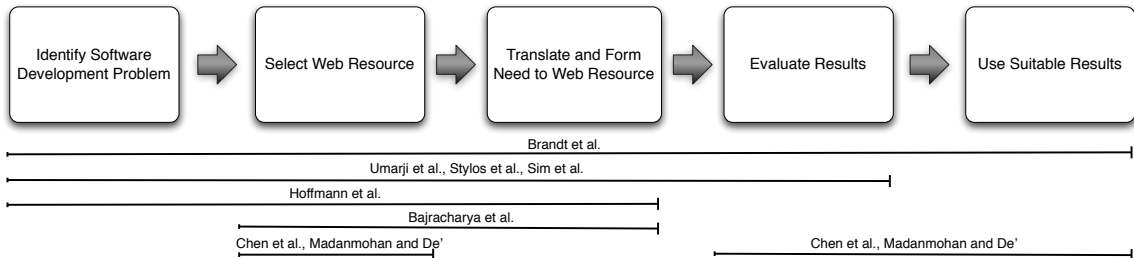


Figure 2.2: Empirical Studies for Phases of the Source Code Search Process

We analyzed the research methods used in the empirical studies in the literature. We found that two studies report on search engines logs [2, 20], two on laboratory experiments [55, 59], two on questionnaires [8, 65], one on interviews [34], and one on a search engine log and a laboratory experiment [5]. However, none of them reports on observations from developers in the workplace.

We also analyzed the phases of the source code search process each study reports on. Results of this analysis are shown in Figure 2.2. Only three studies report on how developers use results from the search. One study [5] provides information on

Author and Year	Goal	Research Methods	Participants	Tool
Brandt et al., 2009 [5]	To understand how programmers use online resources	- Laboratory experiment - Analyzed queries to an online programming portal and examined their lexical structure, their refinements, and pages visited	- 20 participants' Web use was observed while building an online chat room - A month of queries from 24,293 programmers making 101,289 queries about the Adobe Flex Web app in July 2008	-
Bajracharya and Lopes, 2009 [2]	To understand what users of code search engines are looking for	Topic modeling analysis of a year long usage log of Koders, a code search engine on the Web	User activity log for Koders. 10 million activities from more than 3 million users during the whole 2007 year	-
Sim et al., 2009 [55]	To evaluate the effectiveness of the search sites that software developers use	Laboratory experiment	36 subjects performed an assigned search scenario on five Web sites to search for source code and judged the relevance of the 10 first hits returned	-
Chen et al., 2008 [8]	To investigate the challenges that development with OSS components poses for Chinese software companies	Survey using a structured questionnaire	47 developers from 43 small, medium, and large software companies reported on 47 completed development projects	-
Umarji et al., 2008 [65]	To understand how and why programmers search for source code on the Web	Online survey with 13 closed-ended questions and two open-ended questions	69 participants contributed 58 anecdotes about how they search for source code on the Web	-
Hoffmann et al., 2007 [20]	To better understand what developers are searching for on the Web	Analysis of query logs and click-through data	Query logs submitted to the MSN search engine from May 2006. 15 millions queries and 339 sessions on Java programming	Assieme
Stylos and Myers, 2006 [59]	To understand how programmers used Web resources to support their programming activities	Observed three small programming projects in Java and a collection of screen-captures of Java programmers	The projects involved creating a new GUI Java app, creating an Eclipse plug-in, and modifying an unfamiliar open source app	Mica
Madanmohan and De', 2004 [34]	To understand the practices that companies use when incorporating open source components	Structured interviews with project developers in large and medium enterprises in the US and India	16 developers from 12 companies in the US and India who worked on 13 small projects and of short duration	-

Table 2.1: Summary of Empirical Studies in the Literature

how developers use results when they are looking for code snippets on the Web and the others [8, 34] comment on the use of information when developers look for open source projects. However, the emphasis in none of these studies is on how developers use search results returned by search engines.

Similarly, none of the studies emphasizes how developers evaluate results from Web searches. There are six empirical studies that provide information related to how developers evaluate results. Two studies [5, 59] briefly report on the evaluation criteria used to select source code snippets, and four [8, 34, 55, 65] on the criteria to select open source projects. We did not find information on the strategies developers use to evaluate source code on the Web.

Although empirical studies mention some criteria used by developers to evaluate search results and briefly comment on the use of results, none of them provide information on how the evaluation of results and information use is performed by developers in the workplace depending on the software problems they need to solve.

We discuss how the results from these empirical studies agree or contradict results from our empirical studies in the next chapters. Chapter 4 discusses findings for what motives Web searches, Chapter 5 for how developers evaluate search results, and Chapter 6 for how developers use the information they found on the Web.

2.2.3 Our Focus

From our review of the literature we learned that little is known about what motivates Web searches, how developers evaluate search results, and how they use the information they find on the Web. Also, we learned that current knowledge we have on this topic is not based on direct observation of how developers on the workplace

use the Web as a resource to solve software development problems. We propose to fill this knowledge gap by conducting a series of empirical studies that will give us a better understanding of the phenomenon of looking for source code on the Web from different but interrelated perspectives. We will emphasize our studies on the phases of the source code search process where we identified there is a lack of knowledge. These phases are shown with a white background in Figure 2.3.

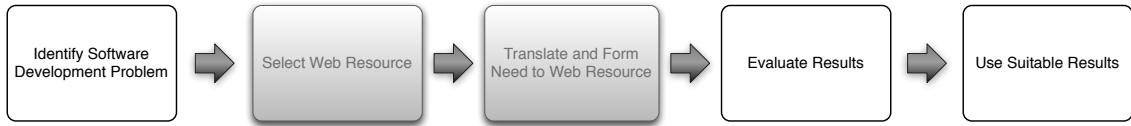


Figure 2.3: Phases of the Source Code Search Process Emphasized in Our Approach

In this dissertation, we do not discuss the phase of “select Web resource” because we found in our online questionnaire that most developers (96%) use general-purpose search engines such as Google and Yahoo!, followed by blogs which are used by 50% of developers. Similarly, we do not include a discussion for the “translate and form need to Web resource” phase because developers mainly use query formulation and there are empirical studies [2, 5, 20] based on code specific search engine logs that report on the characteristics of these queries written by professional developers.

In the following chapters, we will explain the series of empirical methods we used and present our results on the problems that motivate developers to look for source code on the Web, as well as how developers evaluate search results and how they use the information they find on the Web.

Chapter 3

Empirical Studies

In this chapter, we discuss the series of empirical studies we conducted to answer our research questions to better understand the phenomenon of source code search on the Web. We discuss the motivation, design, participants, and key results for each study. At the end of this chapter, we present a summary of these studies. The results of these empirical studies will be reported in the following chapters.

3.1 Overview

We used a set of complementary empirical studies to understand the phenomenon of looking for source code on the Web from different but interrelated perspectives. Figure 3.1 shows how these studies fit together.

We first performed a survey of the literature in order to have an overview of what we know so far about source code search on the Web. We reported our results from this study in the previous chapter. In sum, we found that little is known about the evaluation of results and the use of information found on the Web to solve different

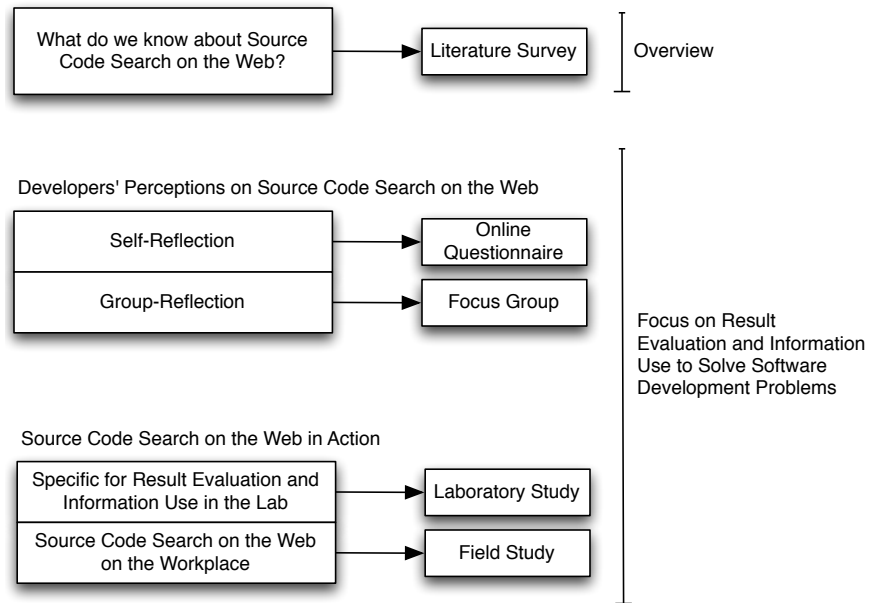


Figure 3.1: Overview of Empirical Studies

software development problems. Results from this survey of the literature motivated our work. For that reason, our next studies are focused on these areas where a better understanding is needed.

Following the literature survey, we emphasized our studies on the evaluation of results and the use of information found on the Web. We used two complementary approaches to study developers' perceptions and actions in these areas. The first approach was to gather developers' opinions on the activity of looking for source code on the Web by collecting developers' reflections via an online questionnaire and team reflections via a focus group. The second approach was to observe developers not only talking about code search but also actually performing code search on the Web. For that purpose, we conducted a laboratory experiment to test specific hypothesis where we needed to control some variables and also field studies in companies in the US and abroad.

Data collected from all these studies has been thoroughly analyzed together. The goal of this analysis was to identify the motivations of developers to look for source code on the Web, to identify the strategies developers use or not to evaluate results and use information from the Web, as well as to determine the effectiveness of Web searches to help developers complete software tasks.

In the following subsections, we explain the motivation, design, participants, and key results from each of the four empirical studies we conducted. In subsequent chapters, we will discuss the combined results in detail.

3.2 Online Questionnaire

3.2.1 Motivation

To gather developers' perceptions on how, when, and what developers look for source code on the Web, we conducted an online questionnaire on professional developers. We presented developers with multiple choice questions which options were taken from our literature survey results. The online questionnaire was the more suitable method to add quantitative data to our literature survey results and also to get individual developers' opinions on this topic.

3.2.2 Design

The questionnaire had 16 multiple choice questions related to the motivations and search targets of Internet-scale source code search, as well as the tools and selection criteria used to look for source code snippets, software components, and open source projects. Appendix B shows all the questions included in the questionnaire. We

wanted to answer the following questions related to source code search on the Web:

- How often developers perform this activity?
- What are developers trying to accomplish?
- What are they looking for?
- What resources developers use?
- What are the sites that developers use to search for source code snippets, software components, and open source projects?
- What criteria are more important for developers when evaluating candidates for code snippets, components, and open source projects?

3.2.3 Participants

We conducted the online questionnaire between September and November 2010. We solicited participation at two conferences of professional developers: Agile Open SoCal (Southern California) and SoCal Code Camp. A total of 26 professional developers participated in our questionnaire. They have in average 13.8 years of development experience.

3.2.4 Key Results

We learned that the most common motivation to search on the Web was to find examples (96%) followed by to remember syntax (65%). We also found that developers are mainly looking for a few lines of source code (92%) and for tutorials

and libraries/APIs (69%). We also learned that most developers (96%) use general-purpose search engines such as Google and Yahoo!, followed by blogs that are used by 50% of developers. Source code specific search engines are the least used resource by only 12% of developers. Developers indicated that they do not use them mainly (79%) because they are not aware that they existed.

3.3 Focus Group

3.3.1 Motivation

Software developers usually work in teams and they tend to express themselves slightly different when interacting with other developers. To gather developer teams' perceptions on resources, evaluation criteria, and how they use the information they found on the Web, we conducted a focus group where developers had to classify different search targets found in the literature survey and answer some questions about the groups they formed.

3.3.2 Design

We designed a focus group where participants had to first complete a card sorting task and then reflect on the resources they use to look for source code in each group, as well as on the criteria used to evaluate search results and how they used the information found. We recorded the focus group sessions with 2 video cameras and one researcher was taking notes in all sessions.

Participants completed a card sorting task on different kinds of search targets. We had the participants work in pairs, because we tend to articulate rules and procedures

only when there is a breakdown between our expectations and the world and/or when we want to make our expectations accountable. This study design allowed us to find regularities in communication about code search targets.

We provided subjects with a stack of 27 index cards that contained code search targets that could be found on the Web. Table 3.1 lists the code search targets in the index cards. The types of examples that we included were motivated by the examples we found in our literature survey. We asked participants to classify these examples into at least 2 and no more than 8 categories based on similarities or differences. The focus group sessions lasted between 1–1.5 hours.

1. Tomcat bug report on improper shutdown of AJP connector	15. Class to connect to Oracle using JDBC
2. A JavaScript tutorial	16. Class to represent a bank transaction
3. Apache Struts 2.2.1	17. Patch for ANT to allow multiple elements in a single property
4. Openbravo ERP	18. Form post on different behaviors in a Thread
5. Implementation of diff	19. OpenMRS
6. Implementation of binary search	20. MySQL bug report on strings short than defined length
7. Implementation of circular linked list	21. Java.util package
8. Implementation of stack	22. Patch for WebdavServlet
9. Javadocs for Log4J	23. Spring Framework
10. Javadocs for Java 2 Platform 5.0	24. Apache Commons Collections API
11. Code to convert Array to Map in Java	25. JUnit package
12. Method to validate email address	26. Forum post on how Hibernate binds values to prepared statements
13. Article on Java Management Extensions JMX	27. MySQL server
14. The Standard Widget Toolkit	

Table 3.1: Source Code Search Targets Used in Card Sorting Task

After participants were done with the card sorting task, we asked them about the criteria that they used for classification and to provide a name for each category. Then, for each category, we asked questions to gather information on:

- What resources/tools do developers use to look for each group?
- How do developers formulate queries for each group?

- What do developers consider to be a good candidate for each group? What characteristics do developers use to assess a candidate is good?
- How using or reusing source code from each group is different?
- Would developers classify each group as Information or Resources?

3.3.3 Participants

We ran a total of 12 sessions with 24 participants: 22 graduate and 2 undergraduate computer science students. We placed the undergraduate students in the same session. In our group of participants, 19 were men and 5 were women. There were between 20 and 49 years old. They had between 1 and 15 years of experience developing software. Twelve participants indicated that they look for source code on the Web a couple of times per week, seven almost every day, three several times per day, and two less than once per month. Twenty one participants indicated that they use Google to look for source code on the Web. Participants received \$20 as compensation for their time.

3.3.4 Key Results

We learned that developers typically classified searches in different groups such as: searches for source code snippets, open source projects, and documentation. Developers mainly use Google, Yahoo!, or Bing as resources to search for source code, but asking to other people for recommendations of source code to use or sites to visit was also mentioned by our participants.

Regarding the evaluation criteria, we found that the used criteria vary depending on the size of the search target. The most important criterion in all cases was that the

source code should comply with the functionality developers need. When developers were looking for code snippets they cared about the appearance of the Web page, for example they preferred a page that did not have many pop-up windows. The source of information also matters, for example a page from Java Sun or IBM will be trusted. Developers preferred source code over pseudo code. Also, when consulting forums, developers also checked other people's comments about a piece of source code, and they often clicked first candidates from very well known forums such as Stack Overflow.

When developers were looking for documentation, they cared about how current is the information, clarity of organization and explanation, and also if it included source code. The source of the documentation also matters. We found that looking for open source projects was not very common among developers, which made it difficult for our participants to articulate what criteria was important to them in this case.

Regarding how developers use the information, we also found differences between search target sizes. When developers were looking for source code snippets, we found mixed answers here. Some developers preferred to copy and paste and try the source code, while others preferred to understand a piece of code and then write it by themselves. For documentation, they mainly read the information and they tried the examples included in it. For open source projects, developers downloaded the components and tried it, but these type of searches are not common.

We also collected many anecdotes from developers about instances when they search for source code. In some cases, the index cards served as motivation for them to tell us about stories and anecdotes about their searches, mentioning specific problems, resources, tools, and issues they had while looking for source code on the Web.

3.4 Laboratory Study

3.4.1 Motivation

From the online questionnaire and focus group, we gathered developers' perception on how they evaluate search results and how they use the information they found on the Web. However, commonly there is a mismatch between what people report doing and what they actually do. For that reason, we decided to conduct a laboratory study where developers could evaluate source code search results in a controlled environment, while trying to solve a software development problem.

3.4.2 Design

We designed a laboratory study where developers were asked to complete software development tasks using hand-built search results so that we will answer the following research questions:

- What information do developers use when they evaluate source code results from the Web?
- What is the relationship between the frequency of information use and its perceived usefulness?
- What is the relationship among the frequency of information use, the likelihood of selecting the best match, and the time to complete a task?
- What strategies do developers use to integrate source code found on the Web?

The experiment session was organized in three stages: training, experiment, and debriefing. We recorded audio and screen activity while participants completed all the stages of the study. In the next subsections, we will explain each of these stages.

Training

We started the session with a warm-up task. The goal of this task was to familiarize participants with the experiment settings, the procedure for each task, and the procedure of thinking aloud. Participants were given the task to find a source code snippet that implements a binary search in Java. We provided a set of three search results with one obvious best match that had the same signature as the method provided. A simple copy and paste, and rename of a variable were enough to make this match work. We also provided a print out of how to run a JUnit test case and a tutorial to add an external jar in Eclipse.

Experiment

We gave participants a written description of the common context for all the tasks. This description includes the fictitious company, characteristics of the software team, and a brief explanation of the software system being developed. All tasks involved modifications to a CRM (Customer Relationship Management) system that tracks the professors and universities that currently use the various textbooks that Tome Textbooks publishes.

Participants performed four tasks (see Table 3.2) and were given a maximum of 15 minutes to complete each of them. We created these four tasks based on the categories of a previous code search experiment [55]. The motivation for all of the tasks was to find a piece of source code to reuse in order to solve an implementation problem. We had two tasks where the source code target size were blocks and the other two were subsystems [65]. In each group, we included one easy task and one

Task	Source Code Target	Target Size
1	Find a library to parse a CSV (comma-separated values) file and place the data into a list of Strings.	Subsystem
2	Find a library to compare two text files and show the lines that are different.	Subsystem
3	Find a small snippet of source code that will validate email addresses. Valid and invalid addresses were included in test cases.	Block
4	Find a small snippet of source code that will convert a String date from one time zone to another one.	Block

Table 3.2: List of Tasks in Laboratory Experiment

difficult task.

For each task, we provided a problem description, a software system to modify, and a JUnit test case. Participants were given a set of search results and asked to select the best match to complete the task. Then, they used the selected match to modify the software system to pass the given test case.

Every subject received the same set of 10 results, but in random order, on a pre-made results page. We used a random order, because we did not want subjects to automatically go to the first result. By giving each subject the same results, we can focus on their evaluation process, without the influence of their ability to form queries. These 10 results were taken from the Web and were chosen with the aim of having one best matching result based on the requirements given in the task description, the test cases, and the method signature. We also included four results that partially met the requirements, four results that were related to the requirements but were only partial solutions, and one that was not related.

Participants performed two tasks to find and reuse components or subsystem, one with search results in the baseline interface and the other in the treatment interface. Similarly, search results were shown using the baseline interface for one task to find

code snippets and using the treatment interface for the other task. The search results in the baseline interface included for each match: title, description, and links to source code. The treatment interface included the same information as the baseline interface plus technical cues (number of lines of code, number of classes, and number of methods) and social cues (number of favorites, number of copies, and percentage of positive reviews). The order of the tasks and the interface were randomized for each participant.

Debriefing

We debriefed subjects at the end of each task and at the end of the experiment. At the end of a task, subjects were asked to rank the usefulness of the information shown using a scale of 1 (low) to 5 (high). They were also asked about the process used to make their final selection. After participants finished all the four tasks, we interviewed them about how the information shown in the baseline and treatment interfaces affected their evaluation process.

3.4.3 Participants

The 16 participants were computer science students (13 graduate and 3 undergraduate) between the ages of 23 and 39. Of these, 13 were men and 3 were women. They had 1–11 years of experience programming in Java. All of them have worked with Eclipse at least once. Six participants indicated that they look for source code on the Web almost everyday, four a couple of times per week, four less than once per month, one several times per day, and one never. Participants received \$20 as compensation for their time and additional \$10 if they completed all the tasks successfully.

3.4.4 Key Results

We learned that participants primarily looked at the title, links to source code, and description. Even when participants have access to additional social and technical information, they still relied on these three. The social and technical cues were used only 12.5–50% of the time.

The information used most often was not perceived to be the most useful. For instance, the title was used 100% of the time, but had a Usefulness of only 3.8 out of 5. The opposite was true as well; information that was perceived to be very useful was used infrequently. For instance, the number of uses had a Usefulness of 4.4, but was used only 18.8% of the time.

We found three patterns of relationships among use of a metric, frequency that a match was selected correctly, and the time taken to complete the task. The two most frequently used metrics, percentage of positive reviews and number of lines of code, were related to lower rates of correct selection and longer completion times. The next two metrics that were used moderately frequently, number of favorites and number of copies, were associated with better rates of correct selection, but longer completion times. The final two metrics, number of classes and number of methods, were used infrequently, but were related to both better rates of correct selection and shorter completion times. It is not clear whether these differences can be attributed to properties of the information or characteristics of the participants who chose to use the information. Further research could explore a possible relationship between personality traits or background knowledge and evaluation strategies. A detailed explanation of the results from this study have already been reported [14].

3.5 Field Studies

3.5.1 Motivation

To collect data on Web searches in situ, that is, within the context of the work being done, we conducted field studies in software companies in the US and abroad. Conducting field studies in companies allows researchers to see the phenomenon of source code search on the Web when it arises naturally and not motivated by researchers. In a field site, researchers can observe the context of the search including: What motivated developers to do a Web search? Were other people involved in the search process? Why developers visited some results and not others? What were the cues that they were following to evaluate candidates? How do they use what they found on the Web? Those are some of the answers that can be better answered by observing developers in the workplace, in their own environment, while they work in real world problems under time constraints and stress.

3.5.2 Design

We conducted a field study of 24 developers in three software development companies. We observed them as they worked with particular attention to how and when they searched for source code on the Web. We augmented these observations with fine-grained data collected using a Web browser extension.

Companies

We had three field sites, one in Peru (Novatronic) and two in Southern California (Health Connection and AppFolio).

Novatronic is a consulting company with 64 employees that develops transactional software. Its clients are banks, telecommunication companies, the government, and other firms located throughout the Americas. The company has achieved CMMI level 3 and its processes are certified as ISO 9001-compliant. We contacted the owners of Novatronic who agreed to participate in the study. One of the owners and the company's product managers identified 25 developers who were coding at the time of the study. Among these 25 developers, we randomly selected 12 developers to observe. Developers did not receive any compensation for their participation.

Health Connection is an open source health information technology company; its system is used to securely exchange health information between health-care organizations such as hospitals, clinics, and laboratories. The company has approximately 45 employees working on various products. We contacted a Senior Software Engineer from Health Connection who agreed to participate in the study. He selected 7 developers, and 4 other developers volunteered to be observed. Developers did not receive any compensation for their participation. We use the name Health Connection as a pseudonym for this company to protect its confidentiality.

Finally, we also conducted one observation at AppFolio, which develops property management software. This company develops a web-based application to manage rental properties, such as apartment complexes. The company has around 120 employees. We contacted the Director of Software Engineering who agreed to participate in the study. He identified one developer to be observed. The developer we observed did not receive any compensation for his participation.

Methods

We shadowed each developer for one day of work. We took notes about the activities they were performing and time stamped switches between activities. We

paid particular attention to Web searches. After they performed a search, we asked some questions to understand the goal of the search, expectations for the search, how the candidates were evaluated, and the use of the information.

At the end of the day of observations, the researcher conducted a short debriefing interview. The participant was asked to reflect on the activities performed and the patterns of searches on the Web.

Because the searches were conducted so quickly, we developed an extension for the Chrome Web browser to collect data. The extension automatically recorded searches as well as the results developers visited after each search. Data collected included the search engine used, terms in the query, and time of query. For visited results, the extension recorded: the number of result page, the position of the visited result, the time of visit, the title, and the URL. A subset of participants in the US (10) installed and used the Chrome extension the day they were observed.

We analyzed the data inductively and iteratively [33]. We used open coding to identify categories, sometimes revisiting data to apply new categories. We used axial coding to relate different categories to each other to create descriptions. The names of participants that we use in this document are pseudonyms to protect their confidentiality.

3.5.3 Participants

Table 3.3 shows a summary of our participants and companies. We observed 12 developers at Novatronic, 11 men and 1 female. Eleven participants completed the background questionnaire. They were between 23 and 38 years old and had 1–15 years of programming experience. They use the following programming languages at work:

Java, C/C++, SQL, Visual Basic, and JavaScript. They used these IDEs: Netbeans, Eclipse, Visual Studio, and Notepad++ (for C). Five participants indicated that they look for source code on the Web a couple of times per week, four almost every day, one several times per day, and one less than once per month.

	Novatronic	Health Connection*	AppFolio
Type of Software	Transactional software	Open source software to exchange health information	Property management software
Country	Peru	USA	USA
# of Employees	64	45	120
# of Developers Observed	12	11	1
Years of Programming Experience of Participants	1-15	2.5-13	2.5
Programming Languages Used by Participants	Java, C/C++, SQL, Visual Basic, Java Script	Java, SQL, JSP, JavaScript	Ruby

*Health Connection is a pseudonym

Table 3.3: Summary of Participants and Companies in Field Studies

We observed 11 software developers, all men, from Health Connection. Developers were between 23 and 36 years old with 2.5–13 years of programming experience. They use the following programming languages at work: Java, SQL, JSP, and JavaScript. The IDEs that they used were Netbeans, IntelliJ IDEA, and Eclipse. Five participants indicated that they look for source code on the Web almost every day, three a couple of times per week, and three several times per day.

One participant was from AppFolio. He has 2.5 years of work experience. He uses TextMate to code in Ruby at work. He indicated he looks for source code on the Web several times per day.

3.5.4 Key Results

We observed a total of 87 distinct Web search sessions, 34 from Peru and 53 from the US. We analyzed these searches and found that performing Web searches is a common practice among developers to solve software development problems. We also found that the length of Web searches depended on the kind of problem they want to solve. Finally, we observed that judgments to select promising candidates occur quickly, in less than 10 seconds for more than three-quarters of the searches. A detailed presentation of our results from observations in Peru have already been reported [15].

Eighty-three percent of developers we observed performed searches on the Web to help them solve software development problems. Only four developers out of twelve we observed did not perform any searches related to software development, three in Peru and one in the US. In the US, a developer who was an expert in JavaScript did not perform any searches while coding the whole day in JavaScript. He did not need to consult for any implementation detail and when he had exceptions he knew how to solve them. The other three developers in Peru had different situations and coding was not the main activity during the day. In one observation, a developer was running around trying to solve a problem with a system in production, the other was writing documentation for a system, and the last one was working his last day at the company.

Developers perform Web searches to solve software development problems. We identified four types of problems. The first three (remember, clarify, and learn) are taken from Brandt et al. [5]. To these, we added a fourth: finding tools or open source projects. Developers use the Web to remember syntax details or facts. Other times, developers have an idea of what they want to implement but they do not know

how to actually do it in a programming language, or sometimes they have an error in their program and they do not know how to fix it. In this case, developers use the Web to clarify implementation details. Developers also use the Web to learn new concepts and to find open source projects to reuse.

The length of the Web searches varied depending on the type of problem developers wanted to solve. Searches for remembering/fact finding had the lowest median time and the smallest distribution. The median of searches for clarification is higher than for remembering but close to the median for learning. Searches for open source projects and software tools had the second highest median, but the most skewed distribution, and the longest tail. In other words, these kinds of searches typically took the same amount of time as searches in support of learning, but many of them could take a very long time. One participant spent an entire day looking for an open source project, and still was not finished with the task.

Judgments to choose a promising candidate are quick. Developers visit the first promising candidate in less than 10 seconds for 79% of queries. In addition, developers have a hard time verbalizing why they chose promising candidates. They make decisions so quickly that they do not even think about how they do it.

3.6 Summary

Table 3.4 presents a summary of the series of empirical studies we did to better understand how developers look for source code on the Web. Specifically, what kind of development problems they can solve using the Web as a resource, and also how they evaluate search results and how they use information they find on the Web.

Study	Goal	Participants	Design	Research Questions
Online Questionnaire	Gather individual developers' perceptions	26 professional developers	16 multiple choice questions	<ul style="list-style-type: none"> • How often developers perform source code search on the Web? • What are developers trying to accomplish? • What are they looking for? • What resources developers use? • What are the sites that developers use to search for source code snippets, software components, and open source projects? • What criteria are more important for developers when evaluating candidates for code snippets, components, and open source projects?
Focus Group	Gather developers' perceptions and articulation while working in a group	24 participants. 22 graduate students and 2 undergraduate students in Informatics and Computer Sciences	Card sorting game and follow up questions in groups of two developers for 1–1.5 hours	<ul style="list-style-type: none"> • What resources/tools do developers use to look for each group of search targets? • How do developers formulate queries for each group? • What do developers consider to be a good candidate for each group? What characteristics do developers use to assess a candidate is good? • How using or reusing source code from each group is different? • Would developers classify each group as Information or Resources?
Laboratory Experiment	Observe developers actions to evaluate search results and use information	16 participants. 13 graduate students and 3 undergraduate students in Informatics and Computer Science	Complete four programming tasks using pre-aid search results so that JUnit test cases will pass. They had 15 min to complete each task	<ul style="list-style-type: none"> • What information do developers use when they evaluate source code results from the Web? • What is the relationship between the frequency of information use and its perceived usefulness? • What is the relationship among the frequency of information use, the likelihood of selecting the best match, and the time to complete a task? • What strategies do developers use to integrate source code found on the Web?
Field Studies	Observe developers performing code searches and gather automatically collected searches in the workplace	24 professional developers were observed	Observations and automatically collected searches	<ul style="list-style-type: none"> • What kinds of problems motivate developers to look for source code on the Web? • What information and strategies do developers use to evaluate source code results? • What strategies do developers use to reuse/use source code found on the Web? • Are source code searches on the Web efficient to complete software development tasks?

Table 3.4: Summary of Empirical Studies

Chapter 4

Software Problems that Motivate Web Searches

Developers are mainly using Web searches to opportunistically solve software development problems (82% of Web searches). Opportunistic searches are ad hoc and are done to remember syntax details, clarify implementation details or fix bugs, and learn new concepts. On the other hand, non-opportunistic searches (only 18% of Web searches) are done following a systematic process and are performed to find open source projects.

Analyzing Web searches from the perspective of opportunistic problem solving helps us understand that developers' searches on the Web are motivated by the software problems that they want to solve. These problems define the search targets that developers are looking for. Using this perspective also helps us see that searches for snippets of code and searches for open source systems are two different types of searches.

In this chapter, we explain the characteristics of opportunistic searches as well

as the characteristics of non-opportunistic searches. We also discuss the insights that come from understanding Web searches from an opportunistic problem solving perspective. Finally, we provide a summary of the chapter.

4.1 Opportunistic Searches

Robillard [49] argues that developers use a mixture of systematic and opportunistic problem solving to complete their tasks. Developers use a systematic approach when they have the knowledge for completing a task and can follow a well-structured plan. In contrast, developers use an opportunistic approach when they need to find missing information for completing software development tasks. For that activity, they incrementally collect knowledge when the opportunity arises. Opportunistic approaches do not follow structured plans but instead happen ad hoc.

Opportunistic problem solving originally helped explain software engineering in general [49]. Here, we are applying it to Web search as an aspect of software development, where developers want to solve problems.

We argue that developers mainly use Web searches for opportunistic problem solving. In the next subsection, we show our results for each of the characteristics of opportunistic problem solving: Web search is used to explore further, knowledge is partially and incrementally gathered, and Web search does not follow a well-planned process.

4.1.1 Web Search is Used to Explore Further and Find Missing Information to Complete Software Development Tasks

Developers often do not have all the information they need to complete their software development tasks, and this is why they search the Web. They are looking for information that will help them solve their software development problems.

In our field studies, we identified four types of problems using the classification proposed by Brandt [5]. First, developers need to remember syntax details or find a fact. Second, they need to clarify how to implement functionality given that they have a high level understanding of how to implement it. Third, they need to learn some concepts. Finally, developers need to look for tools or open source projects. This last classification was not reported in Brandt’s study. Figure 4.1 shows the frequency of each type of problem. We identified that the first three types of problems fit into opportunistic problem solving, but not the last one. Searches done to find an open source project to reuse does not meet all the three characteristics of opportunistic problem solving. For that reason, searches for open source projects will be discussed in the next section.



Figure 4.1: Problems that Motivate Web Searches

The length of the Web search session varied depending on the type of problem developers wanted to solve. We obtained this result by looking at the distribution of the duration of each kind of search. Our definition of a search session can be found in Appendix A. Figure 4.2 shows a box and whisker plot for the length of the searches by the type of search performed. Each box in the graph shows the range of 50% of the data and the black dot shows the median. The whiskers above the boxes show the 25% of searches that took the longest time and the whiskers below the boxes show the 25% of searches that took the shortest time. Triangles show the outliers (three additional outliers have been omitted).

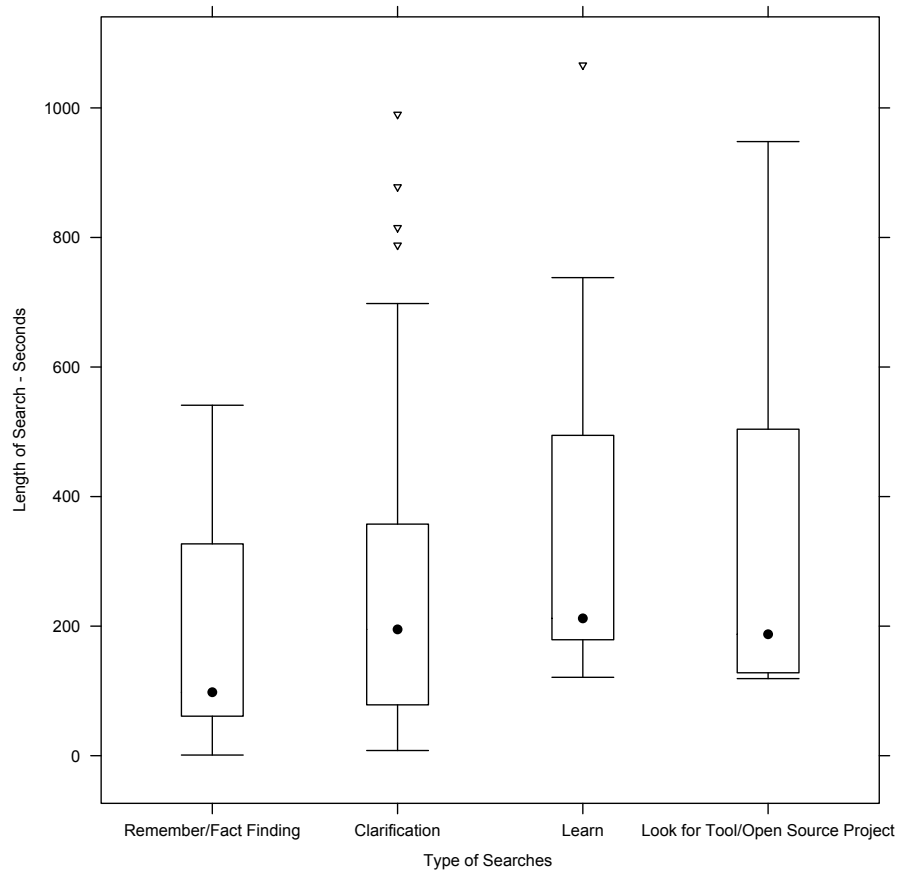


Figure 4.2: Length of Web Searches by Type of Software Problems

Here, we include a description for each type of software problem identified for

opportunistic searches. Descriptions are accompanied by quotations from developers that exemplify the searches our participants performed in field studies or talked about in focus groups. We use pseudonyms to identify our participants in field studies.

Remembering/Fact Finding Developers know exactly what they are looking for when they perform Web searches to remember syntax details or find facts. In these cases, developers can recognize the answer as soon as they see it.

Developers perform searches to remember syntax details of commands or parameters of a method. As Bob, a developer in our field studies, mentioned when he did one of these searches to remember the syntax of a SQL command: *“I always forget how to do this.”* Developers use the Web as memory aid [5]. Developers do not bother remembering information which they know is on the Web, as indicated by a participant in focus group 11: *“When I look for tutorials, I search for the name of a method in the documentation and read examples. It is more understanding. I will not care about remembering because I know it will be available there.”*

Developers also know exactly what they are looking for when they are trying to find facts. Sometimes, they need specific pieces of information. For example, the goal of one developer was *“to find what was the last version of HtmlUnit.”* Twenty five percent of the searches we observed are to remember or to find facts as shown in Figure 4.1. Developers know exactly what they are looking for and they recognize it easily. These searches had the lowest median time and the smallest distribution as seen in Figure 4.2. Query refinements are uncommon in these cases. Developers tend to visit few results and in some cases they find the answer just by looking at the search results.

Clarifying Often developers have a high level understanding of what they want to implement, but they do not know precisely how to do it. They are looking for

examples of how to use APIs or they are looking for solutions on how to fix bugs or exceptions. One instance of this type of search is when Michael needed to send messages to a JMS queue from his php application so that the messages will appear in the log. When we asked him about his goal for the search, he indicated he wanted to *“find an example of how to have a php application send JMS messages to a queue so they will be logged.”* Another instance is when Gregory was *“trying to find out any issue or forum post about the exception org.eclipse.jetty.util.log EOF.”*

Developers have a rough idea of what they want, but are not sure what would be a good answer. In such cases, it is hard for developers to create effective queries. They need to evaluate more results and they reformulate queries as they learn from the search results that they evaluate. These searches had a median time longer than the one for remembering but smaller than the one to learn new concepts as seen in Figure 4.2. Forty three percent of the searches we observed fall in this category.

Learning Developers need to learn new concepts. Once, Joaquin needed to learn about *“OS4690 v6.0”* because he was asked to implement an application for that operating system, which was unfamiliar to him. For searches in support of learning, developers mainly looked at explanations and examples or tutorials. Fourteen percent of the searches observed fall into this category. Developers spent more time reading the documentation than evaluating the relevance of results. They did many query reformulations and visited many results. These searches had the longest median time as seen in Figure 4.2.

4.1.2 Knowledge is Partially and Incrementally Gathered while Developers Search on the Web

Developers learn more about the problem they want to solve and how to formulate effective queries when they are engaged in a Web search session. Developers use query reformulation to refine their searches based on the partial knowledge they collect from previous queries in a Web search session. When developers examine search results from queries, they sometimes find the solution to their problems. In other cases, the search results help them identify keywords that are more appropriate to describe what they are looking for.

In our field studies, developers used query reformulation for 50% of the searches to learn new concepts and for 35% of the searches when they were trying to clarify implementation details or fix bugs. Table 4.1 shows the number of query reformulations done for searches to solve each type of problem. The first column shows the number of searches with 0 refinements. That means, developers only entered 1 query for those searches. The second column shows the number of query reformulation between one and seven. In this case, developers performed between 2–8 queries. In the third column, we show the searches that did not include query reformulation because developers visited bookmarks or links directly. In the last column, we show the searches for which we did not have information about query reformulation.

One instance of a search that involved query refinement was when Manfred was looking for a solution to solve an exception he was having when he was using HtmlUnit. He entered these 3 queries:

Query 1: *htmlunit "The data necessary to complete this operation is not yet available"*

Query 2: *htmlunit doScroll*

Query 3: *htmlunit "The data necessary to complete this operation is not yet available"*

		Query Reformulation			
		0	1-7	Bookmark	No info
Opportunistic Searches	Remembering/ Fact Finding	13	5	4	0
	Clarifying	15	14	3	5
	Learning	6	6	0	0
	SUBTOTAL	34 (48%)	25 (35%)	7 (10%)	5 (7%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	14	2	0	0
	SUBTOTAL	14 (88%)	2 (13%)	0 (0%)	0 (0%)
TOTAL		48 (55%)	27 (31%)	7 (8%)	5 (6%)

Table 4.1: Query Reformulation by Type of Software Problems

doscroll

In the first query, Manfred entered the name of the library and the error message he received. After examining some results, he learned that this problem could be related to the “doScroll” method, for that reason he replaced the error message for the name of the method in the second query. Then, he examined the results and did not find the answer he was looking for, so he tried to include both the error message and the name of the method. This example shows that developers collect partial knowledge from the search result evaluation they perform during a Web search session.

4.1.3 Searches on the Web do not Follow a Well-Planned Process

Web searches happen in an ad hoc manner and they happen very often. Typically, developers do not start their day by planning the Web searches they are going to perform during the day. In fact, they do not know if they will even perform searches in a given day or how many they will need. The number of searches they will perform depends on the problems they would need to solve during the day.

Also, if Web searches were a planned process, people would be more aware of the searches they perform. However, we found a mismatch between what people reported doing and what people actually did.

Based on our observations of developers in field sites, 83% of them perform many searches on the Web during a day of work to help them solve software development problems. However, when asked in all our surveys, only 45% of developers reported that they perform searches almost everyday or several times per day.

Twenty developers we observed performed searches on the Web to help them solve software development problems. Only four developers out of the twenty four that we observed did not perform any searches related to software development (three in Peru and one in the US). One developer who was an expert in JavaScript did not perform any searches while coding the whole day in JavaScript. He did not need to consult for any implementation detail and when he had exceptions he knew how to solve them. The other three developers had different situations and coding was not the main activity during the day. In one observation, a developer was running around trying to solve a problem with a system in production, the other was writing documentation for a system, and the last one was working his last day at the company.

Among those who did search the Web, they performed on average 3.6 searches per day ($\sigma = 3.4$), with a low of one search and a high of 15. Web searches were an important and integral part of their day. One developer at Health Connect, Brian, said: *“I could not code without Google.”* He performed 7 searches the day he was observed.

Figure 4.3 shows a pie chart with the frequency of self-reported Web searches that developers perform to help them in their software development tasks. This graph includes data reported from 88 developers who participated in our online questionnaire, focus group, laboratory experiment, and field studies. Two developers did not provide this information.

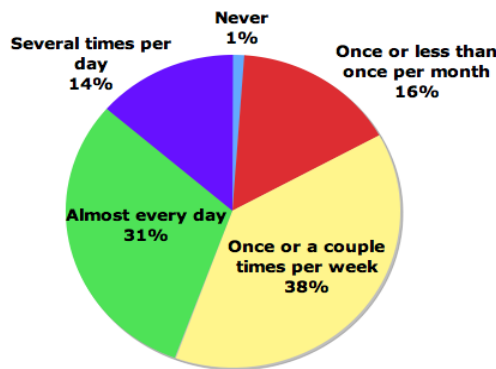


Figure 4.3: Frequency of Self-Reported Web Searches in Our Empirical Studies

Most developers (38%) indicated that they perform Web searches to help them solve development problems only once or a couple times per week. Thirty one percent of developers look for source code on the Web almost everyday and 14% several times per day. Only one developer indicated that he never performs searches on the Web, he prefers to write his own code because he cannot trust what is on the Web.

This discrepancy between what we observed and what developers reported is likely influenced by two factors. First, doing Web searches is such an ubiquitous action for developers that they do not notice when they perform Web searches. This was clear

when we asked one of the developers about the 4 searches he just did and he replied:
“Really? Four searches? I remember doing maybe only one.”

Second, the number of searches that developers do is not the same everyday and it depends on the problems they are solving on a given day. Developers mentioned that they do more searches when they need to learn something new or when they have to deal with a problem that they have not faced before. In contrast, days in which developers *“just code”*, as they said, have a low probability of performing searches unless an exception is raised in their code.

4.2 Non-Opportunistic Searches

Not all Web searches done by developers meet the three characteristics of opportunistic searches. When developers look for open source projects, Web searches are mainly to find software to reuse as is, not to find code snippets or explanations. Also, Web searches are not ad hoc but instead they follow a methodical process.

4.2.1 Web Search is Used to Find Open Source Projects to Reuse As-Is

Web searches are non-opportunistic when developers are trying to find open source projects to reuse. When they do this type of search, they are looking for software to reuse as-is and developers do not intend to make any changes to this open source projects or software tools. In this case, developers are not looking for code snippets or explanations, but instead for complete systems to reuse.

Looking for Software Tools or Open Source Projects Developers look for open

source projects that they can reuse and integrate into their current projects. For example, Oscar was looking for an open source project to do data mining of logs and to manage alerts. In other cases, developers need to find some tools to support their programming tasks. For instance, Malcom was trying to find a tool to do performance testing of Java programs. In this type of search, developers look at many alternatives and evaluate each of them very carefully according to criteria such as functionality, cost, popularity, and support. Unlike opportunistic searches, this type of search often requires multiple search sessions, each requiring evaluating different options.

Searches for open source projects and software tools have the second highest median in search time, but the most skewed distribution, and the longest tail as seen in Figure 4.2. In other words, these kinds of search sessions typically took the same amount of time as search sessions in support of learning, but many of them could take a very long time. Also, developers often required more than one search session to solve the problem of finding a suitable open source project to reuse. One participant spent an entire day looking for an open source project, and still did not finish the task.

4.2.2 Knowledge is Gathered for Criteria to Evaluate Open Source Candidates

When developers evaluate open source projects, they do Web searches to find information related to each of the evaluation criteria that they use to compare them.

When we observed Oscar, he did Web searches using the name of the systems that he was evaluating. He did a few query reformulations, as seen in Table 4.1. For each candidate system, he read information related to the architecture of the system,

requirements for installation, cost, and support, which were the evaluation criteria he was using. He looked for the same information for all the systems he evaluated.

4.2.3 Searches Commonly Follow a Planned Process

Oscar, a developer we observed, knew at the beginning of the day that he would be doing many Web searches to find an open source project to do data mining of logs and to manage alerts. At the beginning of the day, Oscar did a Web search to find information about the type of software he was looking for. He found an article that provided a list of this type of open source systems. During the day, he methodically followed that list to search for information related to each open source system. For each system in the list, he did a Web search using the name of the system for the query.

4.3 Discussion

We found that 82% of Web searches are used for opportunistic problem solving. These searches are done to remember syntax details, to clarify implementation details or fix bugs, and to learn new concepts. In opportunistic searches, developers do not use a systematic or well-planned engineering process where they carefully consider candidates to solve a problem, but instead they partially and incrementally collect knowledge to find missing information as opportunities arise.

Using an opportunistic problem solving approach helped us to have a clear understanding of what motivates Web searches, to identify common search targets for different motivations, to differentiate searches for code snippets and open source projects as two different problems, and to classify tools for opportunistic and non-opportunistic

searches.

4.3.1 Understanding of What Motivates Web Searches

We look at Web searches not just as entering a query, but instead we look at them in a more ecological fashion by studying their context as well as what happens before, during, and after the search. By opening up our perspective from querying to problem solving, we can more clearly identify that what motivates searches are the software problems developers want to solve, and that these problems define the search targets they are looking for.

In the literature, we did not find a clear statement on what motivates Web searches. In some cases, the motivation was given in terms of search targets, such as find examples, but it was not clear why developers were looking for examples. In other cases the motivation was given in terms of the purpose of the search, for example, to reuse source code, but it was not clear why developers wanted to reuse.

We are interested in understanding what motivates developers to do searches on the Web. For that reason, we first conducted a review of the literature to find out what has been reported. Then, informed by results from the literature review we conducted an online questionnaire to find out what motivates developers to do Web searches and what they are looking for. Finally, we conducted field studies to observe searches in situ, putting emphasis on the context of the search, including what happens before and after the search. Here we synthesize our results.

When we compared results from our online questionnaire (whose options were taken from the literature) and field studies, we did not find a perfect match between the motivations identified, as seen in Figure 4.4. This mismatch is due to the fact that

motivations identified in the online questionnaire were in terms of software problems and search targets, but the motivations identified in the field studies were in terms of software problems, which define the search targets. There are two categories from the online questionnaire that do not match with the categories from our field studies: 1) find examples and 2) reuse source code. Using the opportunistic problem solving perspective, we found that developers look for examples when they want to remember syntax details, they want to clarify implementation details or fix bugs, and they want to learn new concepts. Developers want to reuse source code to solve any of the problems identified in our field studies.

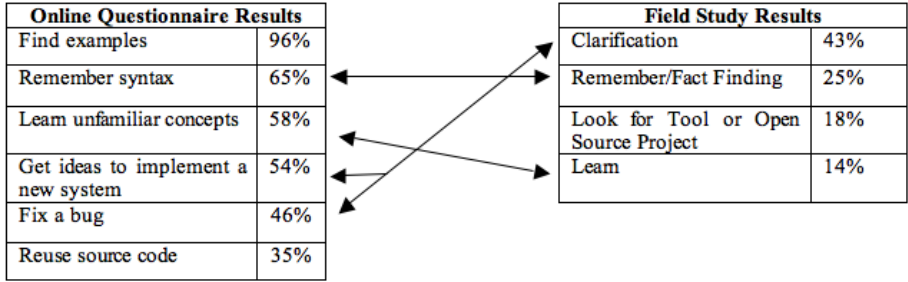


Figure 4.4: Comparison of Results from Online Questionnaire and Field Studies for Motivations behind Web Searches

In our literature review, we found that some empirical studies [5, 55, 59, 65] report on the motivation and also the search target developers expect to find, but some studies [2, 20] only report on the search target. None of these studies report on observations of developers in industry.

We analyzed the different motivations and search targets reported by empirical studies. Based on this analysis we found that developers are looking for information for seven categories of reasons:

- R1. To reuse source code as-is.
- R2. To find examples of usage for GUI widgets or API/libraries.

R3. To remember syntactic details or frequently used functionality.

R4. To find examples to clarify how to implement functionality in a specific language or how to implement an algorithm or data structure.

R5. To learn unfamiliar concepts.

R6. To fix a bug.

R7. To get ideas to implement a new system.

Results from our literature review informed the options we included in our online questionnaire for the question “*What are you trying to accomplish by looking for source code on the Web.*” We came up with 7 types of motivations based on our survey of the literature. Two of these types were very similar and we collapsed them in one to be included in our online questionnaire. We collapsed R2 (To find examples of usage for GUI widgets or API/libraries) and R4 (To find examples to clarify how to implement functionality in a specific language or how to implement an algorithm or data structure) in a category called “Find examples” because both motivations are related to looking for source code examples. We also asked developers “*What are you looking for when you are searching for source code on the Web*” to identify what are the search targets they were trying to find. See Appendix B to see all the questions and options included in the online questionnaire.

In our online questionnaire, we found that the most common motivation for developers to look for source code on the Web was to find examples (96%), followed by searches to remember syntax for programming language details or frequently used functionality (65%), as shown in Figure 4.5. We also found that developers commonly expect to find just few lines of source code (92%), followed by libraries or APIs and tutorials (69%) as a result of their searches, as shown in Figure 4.5.

Results from the online questionnaire showed that developers are mostly looking for examples and they expect to find few lines of source code. However, we still

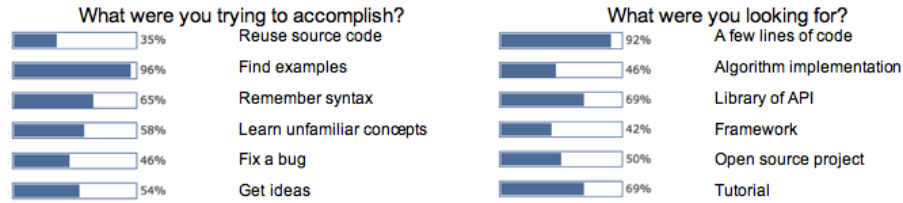


Figure 4.5: Online Questionnaire: Motivations and Search Targets

did not know why developers were looking for examples. Similarly, we knew that developers were looking for a few lines of source code or APIs but we did not know what motivated developers to look for them. To find answers to these questions, we conducted field studies of developers in the workplace.

From our observations in field studies, we learned that what motivates developers to look for source code on the Web are the software problems that they are trying to solve. We found that developers were looking for examples or a few lines of source code when they wanted to remember implementation details or find facts, or when they wanted to clarify implementation details or fix errors. We also found that developers looked for API documentation or tutorials when they wanted to clarify implementation details.

When we compared our results from the online questionnaire and the field study, there is no a perfect match between the motivations reported and the motivations observed. This is mainly because in our online questionnaire, the options given were in terms of both, software problems and search targets. However, in our field study the motivations identified were software problems. Figure 4.4 shows this not so perfect match. Notice that results from the online questionnaire do not sum 100% because participants were allowed to select more than one option. Two categories match exactly between the two studies: Remember syntax and learn unfamiliar concepts. In both studies, remembering was the second most common motivation among de-

velopers. Learn new concepts was the third most common motivation in the online questionnaire, but in our observations it was the least common. In our field studies, we observed that developers do searches to clarify implementation details and fix errors. This category matches with these categories in the online questionnaire: get ideas to implement a new system and fix a bug. The categories find examples and reuse source code in the online questionnaire do not match exactly to only one of the categories in our field studies. By using an opportunistic problem solving perspective, we now know that developers look for examples when they want to remember, clarify, and learn. Similarly, developers want to reuse source code in any of the four categories identified in the field study.

Understanding software problems as the motivation behind Web searches provide an appropriate context to identify the search targets developers are looking for.

4.3.2 Software Development Problems and Search Targets

Software problems that developers want to solve define the search targets. We found that developers look for examples, code snippets, syntax, or API documentation when they want to remember or find a fact. When developers want to clarify implementation details or find solution to a bug, they look for API documentation, examples, code snippets, and error related information. If developers need to learn new concepts, they usually look for tutorials or API documentation. When developers look for open source projects, they try to find information related to the projects, specifically with respect to the installation requirements, architecture, and reputation.

Table 4.2 uses circles to show how often developers look for each search target when they are trying to solve each type of software problem. The bigger the circle, the more frequent a search target is used to solve a development problem. For example,

Search Target		API Documentation/ Tutorial	Example/Code Snippet/Syntax	Open Source Project/ Software Tool	Error Related Information	Others
Opportunistic Searches	Remembering/ Fact Finding	● 7	●● 11		· 1	● 3
	Clarifying	●●● 18	●● 10		● 7	· 2
	Learning	● 8	· 3			· 1
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools			●●● 16		

Table 4.2: Search Targets by Type of Software Problems

we can see in this table that for 18 searches (out of 87) developers were looking for API documentation or tutorials to clarify implementation details. We identified the following search targets:

API Documentation/Tutorial Developers are looking for documentation of APIs or tutorials, mainly when they want to clarify some implementation details. Less frequently, developers look for these search targets when they want to learn new concepts or remember syntax details.

Example/Code Snippet/Syntax We grouped examples (14), syntax (7), and code snippets (3) together because when developers are looking for these search targets, most of the time they are trying to find few lines of code to be used as a reference or to be reused. Developers look for this set of search targets when they are trying to remember, find a fact, or clarify implementation details.

Open Source Project/Software Tool When developers are evaluating open source project candidates or they are looking for a tool to help them with programming, they look for information related to each candidate. This information includes requirements to install the project, description of the architecture, and what other people think about a project.

Error Related Information When developers get exceptions after they compile, run, or test their code, they look to the Web to find error related information that could be helpful to understand the error and fix it. Developers are interested in finding the cause of the error, how to solve it, and experiences other developers have with the same issue.

Others In this category we grouped search targets that did not fall in any of the previous categories. We found that developers are also trying to find again a Web page recently visited that was useful but they do not have a link to it (2), and trying to find the meaning of a word (1). In one case, a developer did not know exactly what he was looking for and in other two cases developers describe their search targets in terms of the problems that they were trying to solve.

Search targets identified in our field studies are consistent with the search targets that developers identified in our focus group. In our focus group, 12 pairs of software developers were given 27 index cards with examples of search targets they can find on the Web. Participants were asked to classify these examples into at least 2 and no more than 8 categories based on similarities or differences.

We took the examples and the categories generated by every group and performed a Formal Concept Analysis (FCA) on them. FCA is a data analysis technique that takes a matrix of objects and properties of objects and derives an ontology, called a concept lattice. This technique has been used in a number of areas, including artificial intelligence, software clustering, and genetics.

The concept lattice in Figure 4.6 was created using the data from our study with the Concept Explorer application. The white boxes are the search targets from the study. The grey boxes are the categories created by the participants. When targets or categories always appeared together, they were collapsed into a single box.

The organization and layout of the lattice are generated automatically and it reflects the relationships between the concepts. More specific concepts appear closer to the bottom of the lattice, and more general ones appear to the top. Circles are placed where different objects are joined to create more general concepts. Larger circles indicate greater confidence in the concept. Some circles are barely visible and not labeled, as these have been created by the analysis process and were not part of the input data. Circles where the bottom half is colored black indicate an exact match with categories created by the participants. Circles where the top half is colored blue indicate an exact match with search targets in the study. Edges in the lattice depict a relationship between concepts, with line thickness showing the strength.

In this concept lattice, examples and snippets are in the left half, tutorials, forums, patches, and bugs are in the middle, Javadoc appear next, and system, product, and frameworks were placed on the far right. There are 8 clear, consistent concepts: example, snippet, tutorials, forums, patches, bugs, Javadocs, and systems. Open source projects and snippets are distinct from each other, but the definitions are fuzzy. There are five “snippet” concepts, and there are many close concepts, such as “partial tool,” “tutorial,” “forum,” and even “example.” Open source projects have a similar variety of categories, such as “system,” “product,” “project,” “framework,” and “dependency.” These are not merely different labels, but also include different search targets. One pair of participants considered a binary search implementation, code to validate email addresses, and code to convert a Java Array to a Map as the only snippets. Another pair had a broader definition of snippet, one that included all the implementations, classes, and specific code. Interestingly, this same set of search targets were labeled as partial tools by another group.

These eight categories of search targets found in the focus group fit almost exactly with the five categories observed in our field studies. Tutorials and Javadocs

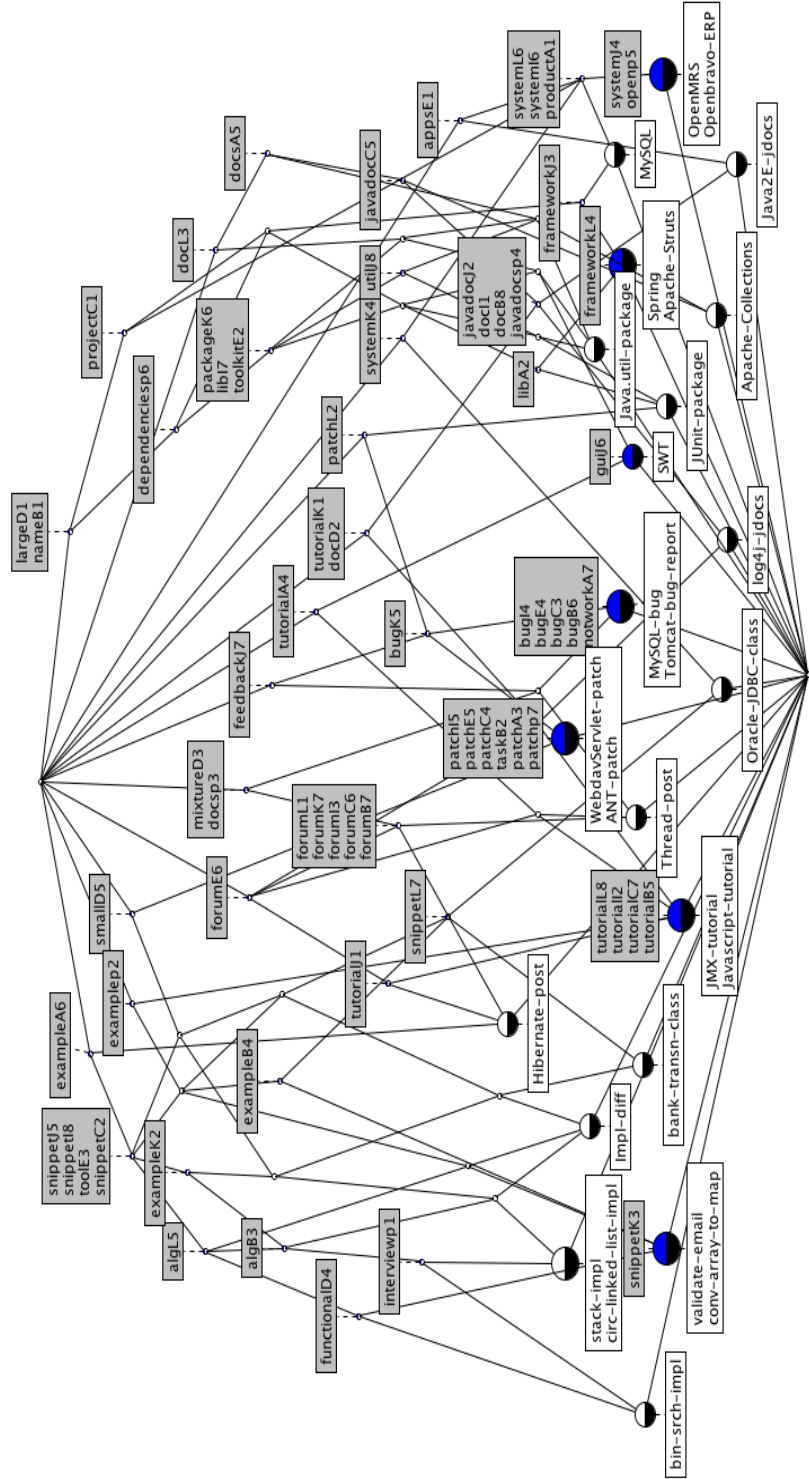


Figure 4.6: Concept Lattice for Examples and Categories in Focus Group

found in the focus group which are related in the concept lattice fit into the API Documentation/Tutorial found in the field study. Similarly examples and snippets from the focus group fit into the example/snippet/syntax found in the field studies. Also, patches and bugs found in the focus group belong to the category of error related information identified in the observations of developers. The category systems in the focus group is related to the open source projects identified in the field studies. Finally, the category forum identified in the focus group was not identified as a search target but instead as a relevance cue used to select search candidates.

4.3.3 Looking for Code Snippets and Looking for Open Source Projects are Different Problems

Analyzing the motivation of Web searches from an opportunistic problem solving perspective, makes evident the differences between searches for code snippets and searches for open source projects.

When developers search for code snippets or explanations to remember syntax details, clarify implementation concepts or fix bugs, and learn new concepts, they are performing opportunistic searches. These searches do not follow a planned process, instead, they are ad hoc. Developers perform opportunistic searches to find missing information and incrementally gather information.

On the other hand, Web searches to look for open source projects are done following a methodical or planned process. Developers methodically evaluate each open source candidate by looking into the Web for further information about a set of criteria. These criteria includes cost, installation requirements, functionality, architecture, and reputation.

Finding these differences between searches for code snippets and searches for open source projects makes it clear that developers need different tool support for these two types of searches.

4.3.4 Classification of Tools for Opportunistic and Non-Opportunistic Searches

We present the classification of tools on the support they provide to opportunistic and non-opportunistic searches as shown in Table 4.3. In this subsection, we first provide an overview of software engineering tools in the areas of code search, mining software repositories, software reuse, and program comprehension by classifying them according to their main goal. Then, we present the classification of these tools on their support to opportunistic and non-opportunistic searches.

Overview of Source Code Search Tools

We classified source code search tools in nine categories based on their main goal. The tools included for each category are shown in Table 4.3. Categories with more members are shown first. The nine categories that we identified are:

API and Example Code Search In this category, we included the tools that help developers to identify an appropriate API and/or to find examples that show how to use APIs, frameworks, or libraries. More precisely, these tools help

¹<http://demo.spars.info/j/>

²<http://www.koders.com>

³<http://www.google.com/codesearch/>

⁴<http://www.krugle.com/>

⁵<http://snipplr.com/>

⁶<http://www.smipple.net/>

⁷<http://sourceforge.net/>

⁸<https://github.com/>

⁹<http://www.google.com/>

	Opportunistic Searches	Non-Opportunistic Searches	Both
API and Example Code Search	Prospector [36] Strathcona [22] MAPO [67] Mica [59] XSnippet [50] PARSEWeb [63] STeP_IN_Java [69] SNIFF [7] Blueprint [4] SAS [3]	JSearch [56] XFinder [12]	Assieme [20]
Web-based Code Search Engine		Agora [52] SPARS-J ¹ [25] JBender [19]	Koders ² Google Code Search ³ Krugle ⁴ Merobase [24] Sourcerer [31] S6 [48] Exemplar [18]
Test-driven Code Search		Code Conjurer [24] CodeGenie [30]	Extreme Harvesting [23]
Code Snippet Web Search Engine	Sniplr ⁵ Smipple ⁶		
Project hosting Site		SourceForge ⁷	Github ⁸
Reuse Opportunity Recommender	CodeBroker [68] Rascal [38]		
Source Code Integration	Jigsaw [10]	Gilligan [21]	
General Purpose Web Search Engine			Google ⁹
Others	Codetrail [17]		JIRISS [47]

Table 4.3: Tool Classification by Type of Searches

developers identify an appropriate API to use (Mica, Assieme, STeP_IN_Java, SNIFF), look for more information about an API (Mica, Assieme), look for examples of how to use an API (Strathcona, JSearch, MAPO, Mica, Assieme, STeP_IN_Java, XFinder, SNIFF, SAS), look for how to instantiate an object of a class type derived from another class type (Prospector, PARSEWeb), look for how to instantiate an object (XSnippet), look for experts on an API (STeP_IN_Java) or look for code examples in general, not necessarily related with APIs (Blueprint). Some of these tools are web-based (Mica, Assieme, STeP_IN_Java, SAS) and others are integrated in an IDE (Strathcona, Prospector, JSearch, MAPO, XSnippet, PARSEWeb, XFinder, SNIFF, Blueprint) as an Eclipse plug-in.

Web-based Code Search Engine In this category we have Web applications that help developers look for source code on open source projects. All of these tools provide a Web interface that allows developers to enter a keyword query to describe the source code need. The tools match the query with the source code they have in their repositories, which has been crawled from the Web. Finally, these tools show the results in a Web page to be evaluated by developers. Some of these are commercial tools (Koders, Google Code Search, Krugle), and others are research prototypes (Agora, SPARS-J, Merobase, Sourcerer, S6, Exemplar, JBender).

Test-driven Code Search These tools receive a test case as an input and look for source code that matches the test case structure and passes successfully the execution of the test case. Tools in this category include Extreme Harvesting, Code Conjurer, and CodeGenie. S6 was placed in the Web-based Code Search Engine category and not in this category because including a test case as input is optional for this tool. S6 also allows users to specify their needs using keywords,

contracts with pre and post conditions, and security constraints.

Code Snippet Web Search Engine Tools in this category help developers to find code snippets on the Web. Generally, these tools provide an online repository where users can submit code snippets and also look for them. These tools do not look for source code in open source projects. Instead, they look for source code in their own repositories of code snippets. There are many Code Snippet Web Search Engines available on the Web. In this review of the tools, we will include two of them: Sniplr and Smipple.

Project Hosting Site A project hosting site provides a source code repository for open source projects and also other features that help maintenance of projects and collaboration among open source developers using the Web. SourceForge provides support for several revision control systems such as Subversion and CVS, while Github supports only the Git revision control system.

Reuse Opportunity Recommender Tools in this category help developers identify source code reuse opportunities. The user does not invoke these tools directly. Instead, the tools proactively suggest source code to be reused. These tools automatically form a query to a repository and recommend source code that can be potentially useful based on the current context and task of developers. These tools are usually integrated into IDEs as in the case of CodeBroker and Rascal.

Source Code Integration In this group, we included tools which main goal is to help developers integrate source code that they want to reuse into an existing software project. These tools do not offer help to search source code. Instead, they assume that developers already have both source code to reuse and a software project where they want to reuse source code. These tools support integration at different levels of granularity, ranging from integration of a method

into another method or class (Jigsaw) to integration of a software project into another project (Gilligan).

General Purpose Web Search Engine These tools search for information on the Web. The search results that they present consist of Web pages, images, and other types of files. They are used for general purposes including source code search. However, they are not specially designed for that kind of search, as is the case of Web-based Code Search Engines. In this survey, we include one tool of this category, namely Google, which has been reported to be commonly used for code search [65]. There are other tools available on the Web that belong to this category such as Yahoo! and Bing.

Others In the last category we included tools that do not belong to any of the previous categories. Here we include JIRISS that performs code search inside an IDE with a repository consisting of only the code in the IDE. In this category, we also included Codetrail which goal is to improve developer's use of Web resources by connecting Eclipse IDE and the Firefox Web browser. Codetrail helps users to automatically identify JavaDoc browsed in Firefox and to create bookmarks to Web pages that have source code pasted in the IDE.

Classification of Tools by Type of Search Supported

We analyzed the tools previously presented to identify which tools support opportunistic searches, non-opportunistic searches, or both. Table 4.3 shows the classification of these tools.

Developers perform opportunistic searches to clarify implementation details, learn how to use an API, to acquire a new concept, or to remind them of syntax. Developers expect to find some lines of code that they could reuse with or without the need to adapt the code to integrate it to their current development task. The types of tools

that mainly support developers perform opportunistic searches are API and example code search tools [36, 22, 67, 59, 50, 63, 69, 7, 4, 3, 20], code snippet Web search engines, reuse opportunity recommenders [68, 38], and source code integration tools [10]. Many of these tools, especially the ones in the first and third group in the list, make use of the current context of the user to suggest potentially related code snippets. Few tools in the project-hosting sites, test-driven code search, and general-purpose search engine groups also help developers look for code snippets. Many tools in the listed groups support both opportunistic and non-opportunistic searches, but only the reuse opportunity recommender group and code snippet Web search engines support exclusively opportunistic searches. When developers evaluate the result set given by the tools, they mainly pay attention to the functionality of the code snippet. In this case, developers do not need to worry much about licenses, support, reputation, and other criteria. Not many tools offer support for integrating code snippets, mainly, because they assume developers will copy and paste them.

Developers perform non-opportunistic searches when they want to reuse complete frameworks or systems. Developers expect to find complete components or systems that they could reuse, but the expectation changes according to what they find available on the Web. The types of tools that mainly support developers seeking open source projects are web-based code search engines [52, 25, 19, 24, 31, 48, 18], source code integration tools [21], test-driven code search [24, 30, 23], and project-hosting sites. Only few of these tools take into consideration the current development context of the developer to suggest components or to help in the evaluation of results. Many tools in the listed groups support both opportunistic and non-opportunistic searches, but there is no a group that supports exclusively non-opportunistic searches. When developers evaluate open source components and projects, they look not only at the functionality, but also at other aspects such as compatibility of the license, the support and level of activity of the open source community in case of problems and questions,

the quality of the software, and the reputation of the developers. After selecting a suitable component or project, developers will adapt their current code, and possibly the found code, to integrate them. There are few tools that help developers with this integration.

4.4 Summary

In summary, we analyzed Web searches using an opportunistic problem solving approach to find out what motivates developers to look for information on the Web. We found that developers mainly perform searches to opportunistically solve software development problems (82% of Web searches). Opportunistic searches are ad hoc and are done to remember syntax details, clarify implementation details or fix bugs, and learn new concepts. On the other hand, non-opportunistic searches (only 18% of Web searches) are done following a systematic process and are performed to find open source projects. Using opportunistic problem solving lenses we changed the level of granularity to understand the motivation behind Web searches from search targets to software development problems. This change on focus allow us to clearly understand that what motivates Web searches are software development problems and they define the search targets developers are looking for. Using the opportunistic approach also help us understand that searches for code snippets and searches for open source projects are two different problems that should be investigated separately.

Chapter 5

Judgments in Evaluation of Results

In this chapter, we present the judgments that developers make when evaluating the results of source code search on the Web. We identified that developers use different evaluation processes and relevance cues when they perform opportunistic searches versus when they perform non-opportunistic searches. We first present the differences between judgments in these two types of searches. Then, we present the evaluation cues used for each type of search as well as the evaluation processes we identified using a naturalistic decision making approach. Finally, we discuss how our results differ from what is currently reported in the literature and present a summary of our results.

5.1 Types of Judgments in Source Code Search on the Web

We reviewed the literature on the judgments that developers make when they evaluate candidates while looking for source code on the Web. We found that the literature

reports two types of judgments: relevance judgments and suitability judgments [13]. Relevance judgments are made when developers identify promising candidates. Relevance decisions are quick, lasting on the order of seconds, and use little information. A lab experiment [55] reported that developers spent 32 seconds on average to make relevance judgments. Suitability judgments are made when developers decide whether a promising candidate is appropriate for the development problem at hand. Suitability decisions take more time and involve a cost-benefit analysis. A lab experiment [55] reported that developers spent days or weeks to make suitability judgments.

We started with this classification of judgments: relevance and suitability. After performing our empirical studies, we have found that these two judgments happen in all types of searches to different extents depending on the type of problem that is being solved. We observed that there is a correlation between the type of judgments developers made and the certainty level of expected results as shown in Figure 5.1.

When the expected results are close to certainty, developers mainly make relevance judgments. When the expected results are highly uncertain, developers mainly make suitability judgments, and when the expected results are somewhere in-between certain, developers make both relevance and suitability judgments. Thus, when developers look for source code to remember or find facts, they usually find what they are looking for in an average of 3 minutes and make mainly relevance judgments often using only one result or just looking at the search results page. Developers use a combination of relevance and suitability judgments when they look for source code to clarify implementation details and to learn. For example, in the case of developers trying to clarify implementation details, they make relevance judgments based on the match of a result with the API they want to use and make suitability judgments based on how close the found source code fits with the specific problem they want to solve and with the source code developers already have. The time spent for doing the

combination of relevance and suitability judgments is on the order of 5–6 minutes. When developers look for open source projects or tools they have a high uncertainty regarding the expected results and developers will mostly make suitability judgments including: matching of functionality, type of software license, social characteristics of the project, and availability of a community of practice, among others.

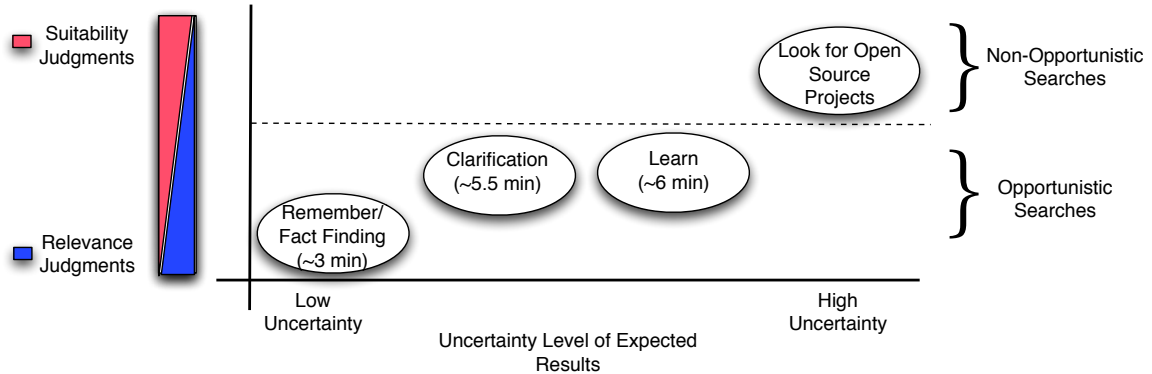


Figure 5.1: Correlation between Emphasis of Judgments and Certainty Level of Expected Results by Type of Software Problems

The fact that these two types of judgments happen in all types of searches makes it difficult to study them separately. On the other hand, we have observed a clear distinction between the types of judgments, cues, and candidate selection processes used when developers are doing opportunistic versus non-opportunistic searches. For that reason, we have restructured our classification of judgments. We find it clearer to talk about judgments made in opportunistic versus non-opportunistic searches than relevance versus suitability. Figure 5.1 also presents the relationship between these two classifications of judgments.

In the previous chapter, we discussed our identification of two types of searches: opportunistic searches and non-opportunistic searches. Opportunistic searches happen when developers want to remember syntax details or find facts, when they want to clarify implementation details or find fix for bugs, and also when they want to

learn new concepts. On the other hand, non-opportunistic searches are common when developers are looking for open source projects and tools.

We observed that developers use different evaluation processes and relevance cues for these two types of searches. For that reason, we discuss these two types of evaluations separately in this chapter.

5.1.1 Overview of Results

Here we present an overview of our results for judgments in opportunistic and non-opportunistic searches. Table 5.1 shows a summary of our results.

		Actions	Evaluation Strategy	Relevant Cues	# Cues Used	Simulation
Opportunistic Searches	Remembering/ Fact Finding	1 Query 1 Result	Only One Result	- Result Order - Example/Code Snippet - Web Host Domain - Social Cues	2	Coding/ Testing
	Clarifying	1-8 Queries 2-13 Results	Serial	- Result Order - Example/Code Snippet - Page Type - Web Host Domain	3	Coding/ Testing
	Learning	2-8 Queries 2-13 Results	Serial	- Result Order - Web Host Domain - Page Type - Title	2 or 3	Mental
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	1 Query 1 Result	Comparison Between	<i>Within Search Sessions:</i> - Result Order - Web Host Domain - Page Type - Title <i>Between Search Sessions:</i> - System Architecture - Installation Requirements - Cost - What other people think	2	Mental

Table 5.1: Summary of Evaluation of Results by Type of Software Problems

When developers perform opportunistic searches, they often enter only one query and evaluate one result to remember syntax details, but they enter multiple queries and evaluate multiple results one after the other to clarify implementation details or

learn new concepts. When evaluating results, developers use two or three relevance cues including the result order, the presence of examples or code snippets, and the Web host domain. During the evaluation of results, developers test results by actually coding or testing the source code in an IDE when they want to remember or clarify syntax details. When developers find information to learn new concepts, they often read the information and understand it. They do not usually test it in an IDE.

On the other hand, when developers perform non-opportunistic searches to look for open source projects, they often perform only one query and evaluate one result to find information about a candidate system. After they collect information for several candidate systems, they compare the systems based on the system architecture, installation requirements, cost, and the opinions other people have about the system. Developers often mentally process the information they find for each candidate and select one.

5.2 Relevance Cues Used to Evaluate Search Results

In this section, we discuss results from our field studies, laboratory experiment, and focus group regarding the criteria developers use to evaluate results for opportunistic and non-opportunistic searches.

In our field studies, we observed the cues that developers use to evaluate results returned by a Web search engine such as Google. Figure 5.2 shows an example of search results returned by Google. We took notes about the cues we observed. Also, after each Web search we observed, we asked our participants: *“What criteria did you use to select a candidate from the result list?”* Based on our field notes and

the answers that developers gave us, we identified the cues that developers use to solve different types of problems. Table 5.2 shows the number of searches in which developers used the cues organized by the type of searches. For example, this table shows that developers used the result order as a cue to visit promising candidates for 18 Web searches performed to remember syntax details.

Title	Bit array - Wikipedia, the free encyclopedia en.wikipedia.org/wiki/Bit_array [+]
Description	To obtain the bit mask needed for these operations, we can use a bit shift operator to A word array , however, is probably not justified due to the huge space ... Daily One Algorithm: Rotate Array dsalgo.blogspot.com/2006/07/rotate-array.html [+] Jul 24, 2006 – Rotate Array . Rotate the array about an index k. So if the input was 0 1 2 3 4 5 6 rotating the array about index = 2the o/p would be 3 4 5 6 0 ...
URL	# - How to Rotate a 2D Array of Integers - Stack Overflow stackoverflow.com/questions/.../how-to-rotate-a-2d-array-of-integers [+] 5 answers - Mar 14, 2009 I am programming a Tetris clone and in my game I store my tetromino ... If they're a 2D array , you can implement rotation by copying with different ...

Figure 5.2: Sections in Search Results Presented by Google

		Technical Cues	Social Cues
Title	How do I convert time between timezone? I have been looking for an example of how to convert time between timezones. However, I could not find one that was exactly what I was looking for. I decided to write it myself. In this example you can see how to convert certain time in my local time to Germany time. ...more	Metrics Lines of Code: 23 Number of Classes: 1 Number of Methods: 1	Popularity Number of Favorites: 48 Number of Copies: 42 Reviews: 60% recommended
Description	TimeZone.getTimeZone('Europe/Paris') This example shows the use of the Java classes TimeZone and Calendar. It shows how to get the current time in Paris, but you can adapt it so that you can use it for any time zone or even to convert times between time zones. ...more	Metrics Lines of Code: 10 Number of Classes: 1 Number of Methods: 1	Popularity Number of Favorites: 64 Number of Copies: 56 Reviews: 80% recommended
	Convert from One Timezone to Another Converts a date/time from one timezone to another. ...more Links to Source Code	Metrics Lines of Code: 25 Number of Classes: 0 Number of Methods: 1	Popularity Number of Favorites: 64 Number of Copies: 56 Reviews: 80% recommended

Figure 5.3: Sections in Search Results Presented in our Laboratory Experiment

For our laboratory experiment, we analyzed the cues that developers used to complete the tasks assigned using the treatment interface. Each of our 16 participants completed 2 tasks using the treatment interface. One task was to find a snippet of source code, which falls in the category of opportunistic searches. The other task was to find an open source project, which falls in the category of non-opportunistic searches. Developers were given a list of 10 search results randomly ordered and they

were asked to evaluate them and choose one result to complete the task. We decided not to include the tasks where participants used the baseline interface because this interface only shows a subset of cues that the treatment interface shows. The search results in the treatment interface include for each match: title, description, links to source code, technical cues (number of lines of code, number of classes, and number of methods) and social cues (number of favorites, number of copies, and percentage of positive reviews) as shown in Figure 5.3. We considered that a participant used a piece of information, if the participant talked positively about that during the think aloud process or during the debriefing, and also if the participant highlighted or pass the mouse over that piece of information while working on a task. Table 5.3 shows how often participants used the identified cues organized by opportunistic and non-opportunistic searches.

Cue		Title	Description	Result Order	Page Type	Web Host Domain	Example/Code Snippet	Cosmetic Appearance	Social Cues	Others	No Cue Observed
Opportunistic Searches	Remembering/Fact Finding	2	3	18	2	8	11	1	4	0	2
	Clarifying	7	6	24	12	11	15	0	6	1	7
	Learning	2	1	9	4	5	1	1	0	0	1
	SUBTOTAL	11 (15%)	10 (14%)	51 (72%)	18 (25%)	24 (34%)	27 (38%)	2 (3%)	10 (14%)	1 (1%)	10 (14%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	1	1	12	2	15	2	0	1	0	0
	SUBTOTAL	1 (6%)	1 (6%)	12 (75%)	2 (13%)	15 (94%)	2 (13%)	0 (0%)	1 (6%)	0 (0%)	0 (0%)
TOTAL		12 (14%)	11 (13%)	63 (72%)	20 (23%)	39 (45%)	29 (33%)	2 (2%)	11 (13%)	1 (1%)	10 (11%)

Table 5.2: Relevance Cues by Type of Software Problems from Field Studies

We also report on the answers that developers gave during our focus groups when we asked them: *“What do you consider to be a good candidate for each group? and what characteristics do you use to assess if a candidate is good?”*

We will now discuss the cues we observed in our empirical studies organized by opportunistic and non-opportunistic searches.

Cue	Title	Description	Links to Code	Lines of Code	Number of Classes	Number of Methods	Number of Users	Number of Downloads	Reviews	Example/Code Snippet	Page Type	Others
Opportunistic Searches	16 (100%)	15 (94%)	16 (100%)	5 (31%)	2 (13%)	2 (13%)	5 (31%)	4 (25%)	10 (63%)	0 (0%)	0 (0%)	2 (13%)
Non-Opportunistic Searches	16 (100%)	16 (100%)	16 (100%)	4 (25%)	2 (13%)	2 (13%)	2 (13%)	2 (13%)	6 (38%)	6 (38%)	6 (38%)	1 (6%)
TOTAL	32 (100%)	31 (97%)	32 (100%)	9 (28%)	4 (13%)	4 (13%)	7 (22%)	6 (19%)	16 (50%)	6 (19%)	6 (19%)	3 (9%)

Table 5.3: Relevance Cues by Type of Software Problems from Laboratory Study

5.2.1 Relevance Cues Used for Opportunistic Searches

Based on our empirical studies, we identified 10 cues that developers use when they evaluate search results to find solutions to their software development problems. Developers commonly use multiple cues at the same time. We found that the number of cues they use varies by the type of problem they want to solve.

The cues that we identified and we explain below are: title, description, result order, page type, Web host domain, example/code snippet, cosmetic appearance, technical cues, social cues, and other.

Title This refers to the first sentence that identifies a search result. This sentence is shown as a link to the Web resource as seen in Figure 5.2 and Figure 5.3. We identified developers as using the title when they explicitly mentioned they were using it during the observation, when they highlighted the title with the mouse, or when they indicated they used it when answering to our question about the evaluation criteria used. When developers were skimming results it was hard to tell if they were using the title or the description. In those cases, we did not count Title or Description as a cue.

We had mixed results about the use of the Title in our field studies, laboratory experiment, and focus groups. In our field studies, we found that developers

did not explicitly mention or show that they used the title to evaluate results. According to our observations and notes from field sites, developers only used the title as a cue for 15% of opportunistic Web searches. Similarly, in the focus group, few groups explicitly mentioned they used the title. Only a participant in focus group 7 indicated that *“In general, I read the title to verify if it is related to the search or not. There should be a similarity between the title and the task I have to do.”* However, results from our laboratory study showed that developers used the Title to evaluate search results to complete all the treatment tasks.

It seems that developers indeed use the title for evaluation quite often, but they just do not articulate this in their answers or do not make an evident use of it. In the case of the laboratory experiment, we explicitly asked if participants used the title in a paper questionnaire.

Description This refers to the couple of sentences that are shown after the title and URL of a search result as illustrated in Figure 5.2 and Figure 5.3. Similarly to the analysis of use of the Title, we identified developers as using the description when they explicitly mentioned they were using it during the observation, when they highlighted the description with the mouse, or when they indicated they used it when answering to our question about the evaluation criteria used.

Similar to the Title, we also had mixed results from our field studies and laboratory experiments. In our field sites, we observed that the description was used only in 14% of opportunistic Web searches. However, it was used for 94% of opportunistic Web searches in our laboratory experiment.

In the case of the Description, it also seems that it is commonly used in evaluation, but developers do not articulate it in their answers when they were observed or interviewed. Only one group in the focus group mentioned that

they use the “*summary that Google provides*” to evaluate candidates for all the types of source code searches they perform.

Result Order This refers to the order in which the results are presented in the search result list. In our field site observations, we identified developers as using the order of results if they examined the results in the order given and they clicked on one of the first three top results, or if they explicitly mentioned in the interview that they used the order of results.

In our field study, we found that result order was the most used cue. Developers used the result order for 72% of their opportunistic Web searches. It was common for developers to see the search results and click the first or second result without even considering the other results below. Developers pay attention to the order given by the search engine, as indicated by a participant in our focus group. He indicated: “*I always use the relevance given by the search engine to decide which results to evaluate.*” In our laboratory experiment, even though we provided the search results in random order and we informed of that fact to our participants, almost all of them read the results in the order given.

Page Type Developers chose to visit some search results because they recognized the type of page where they could find an answer to their problems. By type of page, we mean the format in which Web pages show information. The type of pages that developers visited include: forums, mailing lists, documentation, tutorials, and JavaDocs. Developers inferred the type of page from the information given by the search result page that shows the URL, title, and description for each result. Different types of pages were helpful for different problems developers wanted to solve. For instance, forums and mailing lists were useful to find bug fixes while tutorials and documentation were helpful to learn new concepts or clarify implementation details.

In our field studies, we found that developers used Page Type as a cue for 25% of opportunistic Web searches. However, in our laboratory experiment the Page Type was not used as a cue. That is mainly because the interface used in our laboratory experiment provided the same type of page for all the 10 results given. In our focus group, developers mentioned same characteristics of tutorials and forums that they are looking for. A participant in focus group 2 mentioned that: *“Tutorials with examples are the best. If I have many tutorials, I will go to the one that has examples. If I have many with examples I will, in many cases, mix the examples.”* The same participant also mentioned that *“In the case of forums, you judge on the quality based on who is posting or on the quality of the comments. You start reading and you can see if they are joking or if they know what they are talking about. You have to check who post it.”*

Web Host Domain The domain or URL of a page was also used as a cue to visit search results. This was the third cue most used by developers solving opportunistic problems. Developers used this cue in 34% of opportunistic Web searches observed in our field sites. Developers trust certain pages because they are “official” or because these pages were helpful in the past to find answers. The sources of pages that developers preferred in our field studies are: Stack Overflow, IBM, Microsoft, Wikipedia, and official API Web pages. For instance, when we asked Scott why he chose a search result, he mentioned: *“I went there probably because it is Stack Overflow. It often solves my problems.”* Similarly in our focus group, developers also indicated they prefer official pages. For example a participant in focus group 1 indicated that he prefers official sites when he is looking for information about bugs: *“Definitely it has to look official. It has to be a real bug tracking tool or something where you can say here is the bug, this is the problem, here is the status, whether it is solved or not. Ideally it would have an estimation of when it is going to be solved. Or ideally, it will*

have the patch for the release.” We did not include results from our laboratory experiment because all the search results showed in the experiment belong to the same Web host domain.

Example/Code Snippet Developers also used the presence of source code or examples in a Web page as a cue to evaluate search results. This was the second most common cue used to evaluate results to solve opportunistic problems. Developers used this cue for 38% of opportunistic Web searches observed. They wanted to find: syntax details, source code in specific programming languages, examples, and links to download examples. It was not straightforward for developers to know just by looking at the text of search results if the Web page includes examples or source code. Sometimes the description of a search result will show source code. However, developers inferred that a result contains examples based on the type of Web page or the URL. For example, developers knew that they have a high probability of finding examples or source code in Stack Overflow, tutorials, or pages that include the word examples in the title. In some cases, developers included the word “example” in their queries to make sure they will have search results that contain examples.

The characteristics that developers look for while evaluating snippets of source code include simplicity, readability, and running examples. Participants in focus groups 1 and 4 agreed in that they will be more willing to use a snippet of source code if the source code looks simple. A participant in focus group 1 mentioned: *“I should be able to copy and paste samples and they should work, ideally. For the best ones you should not need to change much. That also means that it is easy to understand. If I do not know what the code is doing, I will not use it. Yeah, I will be confused and I will look for another example because it could have behaviour that I do not know where it comes from or that you do not want. The simpler, the better for me.”* A participant in focus group 4 also indicated

that he looks for simplicity when evaluating snippets: *“Simplicity. The simpler it looks the more willing I would be to copy and use it.”* The other participant in focus group 4 emphasized the importance of finding source code that is easy to read. He indicated: *“For me it is readability, if it is easy to read and it is understandable.”* Finally, developers also wanted snippets that run and work. A developer in focus group 4 also indicated that: *“When looking for a set of functionality, I just try to run them. If they run without any modification, then I might use them to modify them. If they look like they are not going to function by themselves or if they do not look like they are being maintained or they will not work, I would just drop them.”*

Cosmetic Appearance Developers visited Web pages that did not have ads and looked clear and serious. This cue was only used in 3% of the opportunistic searches we observed. When we asked Bob why he chose one page over the other. He said: *“The first one was shady, ugly, it gave the same information and it had ads. Maybe the information is the same but this is more serious. This is the official one. Avoid the ones that have ads.”* Developers in our focus group also mentioned that they prefer websites with few pop-ups, as in the case of a participant in focus group 8 who indicated: *“When looking for snippets, I look for a Web site with less pop-ups.”* Developers also emphasized the importance of how results are presented so that source code can be easy to read. For instance, a participant in focus group 12 mentioned: *“You really want code, so you do not want a very descriptive text, you want the actual code, you want to see what is less wordy. It is also about the presentation of the results. If it is text only, text in the same layout, it is difficult to distinguish, some pages have the code different how you would see it in an IDE. The presentation matters.”*

Technical Cues Developers also use technical cues to evaluate potential candidates that could help solve software development problems. We define technical cues as software metrics, such as number of lines of source code or number of classes, that could be useful in comparing several pieces of source code. Although general-purpose search engines do not show technical cues in the search results, we observed that developers in our field sites look for these cues, in particular for lines of source code. We observed that developers preferred pieces of source code with fewer lines of code when they were looking for source code snippets. In our laboratory experiment, we included three technical cues in our search results: number of lines of source code, number of classes, and number of methods. Among these cues, the number of lines was the most used. Developers used the lines of code for 31% of opportunistic Web searches as seen in Table 5.3. Participants in our focus group also mentioned that they pay attention to the number of lines of source code while evaluating source code. For instance, a participant in focus group 5 indicated: *“If I am looking for how to code an array in Java, if I find a 21 line solution, I am not going to like it, I would rather find one line solution. It should not take many lines of code to do that. I am looking for simplicity but not over simplicity. My decision will be based on my intuitive sense of how complex a solution should be.”*

Social Cues It is also important for developers to know what are the experiences of other developers with a piece of source code. Developers used social cues to determine which results to evaluate for 14% of opportunistic Web searches observed. Developers were looking for what other people think about a piece of source code (Web users as well as co-workers), what are the comments that people left in Web pages related to the effectiveness of a solution or source code, and how other people solve a bug.

In our laboratory experiment, we included three social cues in the search results: number of copies, number of favorites, and percentage of positive reviews. Among these three social cues, the most used was the percentage of positive reviews, which was used in 63% of the tasks to find snippets of source code in the laboratory experiment. Developers in the focus group use social cues to decide which code snippets they can trust based on the reputation of the author of a piece of source code, if available, as indicated by one participant in focus group 10: *“Some forums have some kind of reputation. Some have a profile, then I use that. I see if they [developers] work in a big application. If I did not find anything, random example random person, I will just try it.”* Developers can also identify if they can trust a piece of source code based on how useful was that piece for other developers, as mentioned by a participant in focus group 3: *“For snippets, it is easier if you would find the solution for example in Stack Overflow because someone else knows that the solution actually works and you have a bunch of answers that say it worked for me in this or this situation. It is easier to trust the solution.”*

We found mixed results for the use of social cues in our field studies and laboratory experiment. It was used for 14% of searches in our field studies but 63% of searches in our laboratory experiment. One explanation for this mismatch is how the cues were presented in these two studies. In the case of the laboratory experiment, we showed the social cues in the result page as shown in Figure 5.3. Developers used these cues to evaluate results by just looking at the search results and without visit them. On the other hand, in our field studies, developers were using Google search results most of the time, which did not include social cues in the results list. In this case, developers had to click a search result and look for the social cues, such as number of download, reviews, and comments. The presentation of social cues has an impact on the use of them.

Others In our field studies, we observed that one developer used a cue that did not fall in the categories described before. He used the date in which the content of a Web page was published to determine if the search result was relevant or not. In this case, the developer was trying to clarify what is the difference between the event of clicking with the mouse or doing it programmatically. In our laboratory experiment and focus group, the cues observed fall in the categories described before.

No Cues Observed We did not have enough information to determine the cues developers used for 14% of the opportunistic Web searches observed. In some cases, the search was so quick that the researcher missed most of the search while taking notes. In other cases, no clear cues were observed and developers had a hard time articulating why they chose a search result.

In our field studies, we also found that the number of cues used to evaluate search results has a relationship with the type of software problem developers want to solve.

We classified the searches in which developers used zero, one, two, three, or more than three cues to evaluate results. This classification can be seen in Table 5.4.

Developers most often used two or three cues to evaluate results for opportunistic searches. They used three cues for 34% of searches and two cues for 27% of opportunistic searches. They used more than three cues for only 11% of searches. When developers wanted to remember syntax details or find facts, they often used two or three cues (result order, example/code snippet, and Web host domain). When they were trying to clarify implementation details they usually used three cues (result order, example/code snippet, and page type). When developers wanted to learn new concepts they used up most three cues (result order, Web host domain, and page type). Developers sometimes used zero cues when they went directly to bookmarks.

		Cues Used				
		Zero	One	Two	Three	More than Three
Opportunistic Searches	Remembering/ Fact Finding	2	3	8	6	3
	Clarifying	7	4	7	14	5
	Learning	1	3	4	4	0
	SUBTOTAL	10 (14%)	10 (14%)	19 (27%)	24 (34%)	8 (11%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	0	3	9	3	1
	SUBTOTAL	0 (0%)	3 (19%)	9 (56%)	3 (19%)	1 (6%)
TOTAL		10 (11%)	13 (15%)	28 (32%)	27 (31%)	9 (10%)

Table 5.4: Number of Relevance Cues Used by Type of Software Problems

We also considered that developers used zero cues when we did not have enough information about the evaluation.

5.2.2 Relevance Cues Used for Non-Opportunistic Searches

When developers look for source code on the Web to solve software development problems, they perform two types of searches: opportunistic and non-opportunistic searches. In the previous section, we discussed the relevance cues that developers use to evaluate search results for opportunistic searches. In this section, we discuss the relevance cues used by developers when they perform non-opportunistic searches to find open source projects.

Based on our empirical studies, we observed that developers perform two kinds

of evaluations when selecting an open source project. In the first kind, developers evaluate search results in individual search sessions to look for information related to a potential open source project. In the second kind of evaluation, developers already have information on multiple open source projects and they evaluate the information collected in different Web searches to select one open source project. We found that developers used different relevance cues when they did the evaluation of candidates within a search session from when they did the evaluation of candidates between several search sessions.

Relevance Cues Used within a Search Session

Based on our observations from field studies, when developers evaluate search results to find information about open source project candidates, there are two cues that are used the most: Web Host Domain and Result Order as can be seen in Table 5.2. Developers are looking for the official Web page for each open source candidate that they are evaluating, for that reason they prefer to use information from the Web host domain that seems the official one. In most of the cases, developers enter the name of an open source project candidate as the query. As a consequence, the top results are usually the official pages of these systems. In only few cases, developers also used the title, description, page type, example/code snippets, and social cues. Similarly, in our focus group developers mentioned the importance of using an official page and looking for documentation when they are trying to find an open source project. One participant in focus group 1 indicated: *“I will use a page that looks official and allows me to download it. It also has to say that it does what I need to do. It has to be well documented.”*

Results from our laboratory study do not match with results from our field studies. In our laboratory experiment, developers were given a list of 10 open source projects that they had to evaluate and then they had to choose one to complete an

implementation. To evaluate these open source projects, developers used the title, description, and links to source code for all searches. Among the technical cues showed, the most used one was the lines of code for 25% of non-opportunistic searches. The most used social cue was the percentage of positive reviews, used for 38% of searches. In addition, we observed that developers were also using other cues not included in the result list such as examples and page type (documentation). Developers indicated they were looking for documentation and examples that show how they should integrate the open source project with their own code to complete the implementation requested in the laboratory task.

We believe the difference in results between the field studies and the laboratory experiments was due to the fact that search results are presented differently and also the nature of the task they had to complete in 15 minutes. We will expand on these differences in Section 5.4.4.

Relevance Cues Used between Search Sessions

In each search session, developers collected information about each open source project candidate. This information was later used to compare and decide which candidate to use. While observing developers doing this type of searches, it was not very obvious what information they were collecting or paying attention for each open source candidate. After the developer was done with some searches, we asked him about the criteria he was using. He told us that the criteria he used to evaluate different open source candidates included: architecture of the system, requirements to install the system, cost, and what other people think about this project. Similarly, a developer in focus group 10 mentioned the importance of knowing who else is using an open source project, what is the environment needed, support available among others. He indicated: *“I would like to know who else is using this project or system. Complete technical information, environment in which you can run it, support available, if it is*

open source, community of developers, do they create extentions, do they modify it?"

5.3 The Process to Evaluate Search Results

Our results indicate that the evaluation of source code search results is a rapid, almost unconscious, and instinctive activity. Yet developers are able to successfully use Web search to solve their software development problems. So the question arises: How are developers able to make effective judgments so quickly?

Motivated by this question we looked into the decision-making literature and we found that developers were making “Recognition-Primed Decisions.” Such decisions are made by assessing the situation, rather than individual results in detail. A familiar situation is used to set expectations and to select cues. This assessment helps developers to identify promising candidates and select a good-enough solution in a serial evaluation of options.

Recognition-Primed Decisions can be seen in “naturalistic” environments in which people need to make decisions under time pressure and stress to solve ill-defined problems. These characteristics exactly match the environment in which developers need to make decisions to find a solution to their problems.

In this section, we first present the characteristics of evaluation of search results we observed. Next, we present Naturalistic Decision Making Theory and explain why Source Code Search on the Web can be considered a Naturalistic environment. Then, we discuss how Recognition-Primed Decisions happen in Source Code Search on the Web for both opportunistic and non-opportunistic searches.

5.3.1 Characteristics of Evaluation of Search Results in Source Code Search on the Web

Judgments to Choose a Promising Candidate are Quick

Developers visit the first promising candidate in less than 10 seconds for 79% of queries. During our observations, we were surprised by how quickly developers chose the candidates they evaluated. Sometimes developers were already integrating a piece of source code they found on the Web, while we were still taking notes about the initial search. This observation motivated us to implement a Web browser extension to record the time they entered a query in a search engine and the time they visited promising candidates. Ten developers, all in the US, used the browser extension on the day that they were observed. The browser extension collected 70 queries for 38 search sessions and developers visited at least one result for 56 queries. We calculated the time it took for developers to click the first result after doing the search. To do that, we calculated the time difference between the timestamp of the first click and the timestamp of the query formulation. Figure 5.4 shows in the x-axis the time of first visit to promising candidates grouped by 10 seconds. The y-axis shows the number of queries. From this graph, we can see that developers visit the first promising candidate in less than 10 seconds for 79% of queries.

It is Difficult for Developers to Explain or Remember their Judgments and Searches

Developers have a hard time verbalizing why they chose promising candidates. They make decisions so quickly that they do not even think about how they do it. We asked our participants about how they select promising candidates, and sometimes they were not able to articulate the criteria or metrics that they used. When pressed, we received answers such as *“I try always the first one and then the others. I am not*

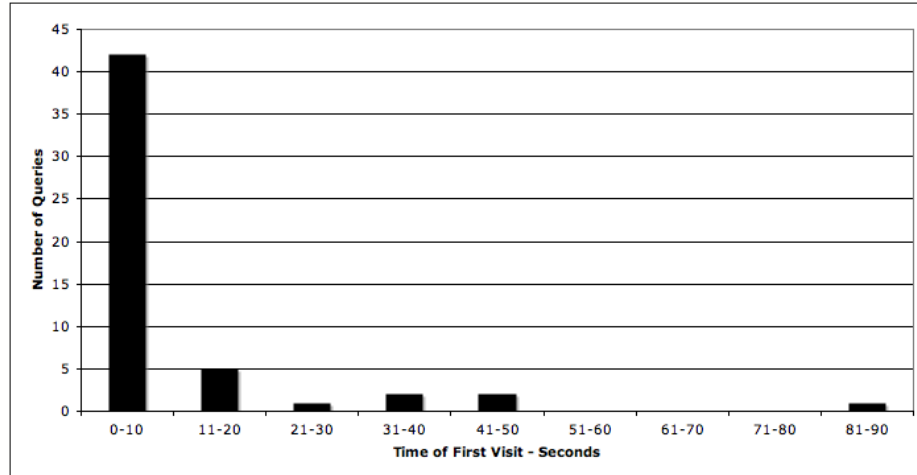


Figure 5.4: Time of First Visit to Promising Candidate

consciously thinking.” This developer is evaluating the results almost instinctively. Sometimes developers do not even remember performing a search. For instance, after Brian completed 4 searches, we interrupted him to ask some questions. He was surprised by our request and exclaimed: “Really? Four searches? I remember doing maybe only one.”

5.3.2 Naturalistic Decision Making in Source Code Search on the Web

The Naturalistic Decision Making (NDM) Theory [26] emerged in response to the lack of theories that explained how people actually make decisions in real-world settings. Previous theories mainly identified optimal ways of making decisions in settings that were well-structured and could be controlled, such laboratory experiments. Instead of creating a model of decision-making first and then evaluating it in a laboratory, NDM researchers conducted field research to discover how people make decision under time pressure with vague goals, uncertainty, and unstable conditions. The field settings that were studied included Navy commanders, jurors, nuclear power plant operators,

Army small unit leaders, anesthesiologists, airline pilots, and nurses.

A natural question to ask is: Why Naturalistic Decision Making? NDM seemed suitable to help us achieve our goal of better understanding how developers made judgments when they look for source code on the Web. This is mainly for two reasons. First, software development organizations present all the eight characteristics of a naturalistic environment [42, 51]. Second, judgments and decisions are a central piece for this theory as it is for our research goal.

We also considered other theories, such as the Everyday Problem Solving theory [58] and Opportunistic Problem Solving in Programming [49]. Although these two theories match very well with the everyday problems developers face to complete their software development activities, they both emphasize the problem solving activities over the decisions taken to solve the problems. However, these two problem-solving theories are applicable in naturalistic settings due to the fact that they both deal with ill-structured problems.

NDM provided us with a good foundation to understand the settings in which developers look for source code on the Web. There are eight factors that characterize NDM [26]. We will explain each of the factors and how they apply to source code search on the Web. A summary of how NDM's characteristics apply to code search on the Web is shown in Table 5.5.

Ill-structured Problems Ill-structured problems are found when both the goal and how to solve the problem are initially unclear. This type of problem is very common in everyday situations. When people face ill-structured problems, they first need to discover and clarify the end goal, and then discover and test the means for achieving the goal [58].

Seventy-five percent of the problems developers want to solve by looking for

	Naturalistic Decision Making Environment	Source Code Search on the Web
Ill-Structured Problems	Goal and means to solve problems are initially unclear.	75% of problems developers want to solve are ill-structured.
Uncertain Dynamic Environments	Decision makers have incomplete information. Environment changes quickly during the time a decision is being made.	Uncertainty due to interaction between people working in parallel and obscure nature of software bugs.
Shifting, Ill-defined, and Competing Goals	Decision makers have to deal with multiple and often conflicting constraints.	Competing goals: find good enough solution in short time.
Action/feedback Loops	Decision makers follow actions to deal with problems or to find out more about them.	Developers learn the "right" vocabulary to use while reading search results which help them refine their queries.
Time Stress	Decision are made under significant time pressure.	Developers have limited time to perform searches on the Web.
High Stakes	Stakes matter to the participant who will take an active role to arrive at a good outcome.	High cost in time and money in case a component is not the right one.
Multiple Players	Common to have more than one decision maker in the process of solving a problem.	What developers find on the Web and decide to use in their projects can affect many people in the team and the organization.
Organizational Goals and Norms	Decisions are not only driven by personal preference but also by goals, values, rules, and guidelines from the organization.	Organizational culture affected Web searches: US participants engaged in collaborative searches more often.

Table 5.5: Summary of NDM's Characteristics Applied to Source Code Search on the Web

information on the Web are ill-structured. When developers deal with this type of problems, they do not exactly know what they are looking for and it is hard for them to formulate queries and recognize a good-enough solution. They tend to have a better understanding of the problem by looking at intermediate results and refining their queries.

Developers face ill-defined problems when they are trying to learn, clarify, and look for tools or open source projects on the Web. Although, developers also deal with well-defined problems when they look on the Web to remember or find facts, these are only 25% of the problems they need to solve.

Uncertain Dynamic Environments Decision-makers often have incomplete and imperfect information. Information may be ambiguous or of poor quality. In addition, the decisions need to be taken in an environment that changes quickly

during the time a decision is being made.

Source code search on the Web also occurs in an uncertain dynamic environment. The goal of a search might change while in progress and be affected by some characteristics of this type of environment. Some sources of uncertainty and dynamism include people working in parallel to develop software and the obscure nature of software bugs.

The impact of uncertain and dynamic environments on source code search was evident in our observation of Scott at AppFolio. We saw him change his query in the middle of working on a problem, based on the actions of others on his team. Based on bug reports and stack traces, Scott thought the problem was caused by the “Solr core” component. About half an hour into the search session, Scott discovered that the stack trace had changed due to a colleague making a fix that caused some exceptions to go away. The new stack traces helped Scott to realize that the problem lay in how the Solr core was initialized. Consequently, he modified his query in response to the new information, more than one and a half hours after the initial query.

In this scenario, the goal of the search was affected by the uncertainty on the cause of the error. In addition, the dynamic environment generated by multiple people working in parallel and creating new versions on the repository also affected the search.

Shifting, Ill-defined, or Competing Goals It is unusual for a decision to be driven by a single, well-understood goal, or priority. The decision-maker has to deal with multiple, usually underspecified, and often conflicting constraints.

When developers perform Web searches, they want to find a good-enough solution in the shortest time possible. They also have competing goals when they evaluate results. For example, Joseph was looking for a tool to add performance

metrics to his Java program. He indicated that he was *“looking for a tool that other people say is good but at the same time do not add too much overhead to my program.”* He found a popular tool, but this added too much performance overhead, so he decided to go with *“old school performance metric capture”* and coded the solution by hand.

Developers also shift goals while they are doing Web searches. When they examine results, they sometimes are exposed to new information and they need to learn about those new concepts to continue their evaluation. Oscar was looking for an open source project to do data mining of logs. While doing the evaluation, he found some terms that he was not familiar with like “real time ETL,” so he performed some searches to learn about that.

Action/feedback Loops Decision-makers follow a chain of actions to deal with the problem or to find out more about it, or both. Mistakes in earlier actions provide helpful feedback for later corrective actions. These action/feedback loops can also generate new problems to be solved.

Developers experience many action/feedback loops while doing searches on the Web. These feedback loops are more common when they are trying to solve ill-structured problems.

Developers sometimes have a hard time translating into a query what they want to look for. They enter a query and then read the results. While reading the results, they learn the “right” vocabulary to use and refine their queries. A single query refinement is common, and usually consists of adding a new word that will filter out some unrelated results. For example, Michael refined his query from “openmq http” to “openmq http php” to find results only for php. We observed few query refinements that included 2 refinements, and only one that included 7 refinements, which occurred when a developer was learning a

new concept.

Time Stress Developers have limited time to perform searches on the Web. The time they are willing to spend searching can vary depending on the type of search. On one side of the spectrum, developers spend only seconds doing searches to remember syntax. On the other side of the spectrum, searches to judge the suitability of open source components can take hours or days.

In our field studies, we observed this notion of limited time for searches mainly when developers were having trouble finding what they were looking for. In these cases, our participants verbalized at which point they decided to stop searching and mentioned some reasons that motivated to give up the search.

Gregory: *“If I do not find anything useful in Stack Overflow, I will give up.”*

Manfred: *“I would look more later. This is not stopping me more, so I will look at this later, not now.”*

Derek: *“I will email the mailing list so they will work on that. They are experts, I can work on other things while they do the research on that.”*

These three examples show the fact that developers are aware of the limited time they have to conduct searches on the Web to solve their software problems.

High Stakes Naturalistic Decision Making typically occurs in situations where the stakes are high. A poorly executed search can lead to errors in the code or the selection of the wrong open source project.

In an example already mentioned, one developer spent days looking for an open source component. Oscar’s searches showed that he was doing a careful evaluation of the potential candidates and he was not giving up even though he already spent one full day in this task. He knew the decision is an important one for the project. We observed him performing 15 searches. Thirteen of them

were to find information related to promising candidates. The queries for these searches were the name of the systems he found in a MS power point presentation he found in his first search. The shortest search took 2 minutes and the longest took 44 minutes. He read information about 10 different components to judge their suitability. At the end of the day, he had some potential component candidates but he was planning on looking for more components the next day. When performing these types of searches, developers spend a great deal of time making suitability judgments to evaluate the requirements of each promising component and to make sure that it will integrate well with the current components. In case the chosen component is not the right one, there will be a high cost in time and money depending on how late the error is discovered. This cost might include: time and effort developing the integration layer with the component, time and effort testing the integration layer with the component, need to redo the selection of a new component, and delay in the delivery date.

Multiple Players It is common to have more than one decision-maker involved in the process of solving a problem. What developers find on the Web and decide to use in their projects can affect many people in the team and the organization. Eduardo had to make a presentation to a Technical Committee about the architecture of the system the team was implementing. The purpose of the presentation was for the Technical Committee to evaluate the design proposed by the team, provide feedback, and suggest improvements. The Committee has at least 5 members including the owner of the company, the Research and Development manager, the Product Manager, the Support Manager, and the Project Manager.

Eduardo needed to include in the presentation why the team chose to use Jahia as the Content Management System for the project. Even though Eduardo was

the only one presenting the design, 2 other team members helped him to find information on the Web that would justify why they chose Jahia. They found a Web page that included a comparison of some CMS including Jahia and they used that information in the presentation. During the presentation, a member of the committee mentioned that he have heard about Alfresco and he suggested to also take a look at it. After the presentation, Eduardo went directly to do a Web search. He entered the query “JAHIA VS ALFRESCO” in Google and read about the differences between them.

This scenario shows that the decision of choosing one promising candidate over others have an impact on the project in which many people is involved. Also the decision is made by more than one person and affects not only the team members but also the whole organization.

Organizational Goals and Norms Naturalistic Decision Making usually takes place in organizational settings. Decisions are not only driven by personal preference but also by goals, values, rules, and guidelines from the organization.

The organizational goals and norms affected how developers perform searches on the Web. We carried out our study in two countries: Peru and the US. We observed some similarities and differences in how the organizational culture affected the searches.

Developers in both the US and Peru preferred to use free or open source software for tools that support programming. The teams generally avoided software that had monetary costs.

A major difference that we observed was that developers in the US engaged in collaborative searches more often. Collaborative searches are those where more than one person was involved. We observed 14 collaborative searches in the US and 2 in Peru.

At Health Connection, when a developer asked a question, other developers in earshot often rushed to the keyboard to conduct a Google search for the best answer. Although they were trying to help each other, there was also a competitive edge, as if coming up with a better result also raised the status of the developer. For example, upon hearing a co-worker call a method *“deprecated,”* Brian corrected him saying, *“it is not deprecated, it is deprecated”* while simultaneously doing a Google search for evidence. At the same time a third developer also performed a Google search and said *“Yes, it is deprecated, not deprecated!”*

We also observed that having good Google search skills is important for developers. One of our participants was teased by co-workers because sometimes he does *“bad Google searches.”* Curious, we asked the heckler for an explanation, he said: *“Uhm bad Google searches are those that are very specific and you do not find any good results. You need to make it little bit general so you will find something. For example, if you enter a query like ‘I am having an error out of memory when I run my Java program’ you might not find many relevant results, but if you enter ‘out of memory error java’ you might have more relevant results.”*

In addition, we were surprised to find that developers in Peru mostly wrote their queries in English even though their native language is Spanish. Ramiro, said *“I always write my queries in English even though I do not know much English.”* There were only four queries in Spanish. In two of these queries, developers immediately switched to queries in English because good results were not returned. Jose explained, *“Sometimes there is more information in English. For example, if you are looking for information in Wikipedia, you will find more information in the pages written in English.”*

5.3.3 Developers Make Recognition-Primed Decisions for Opportunistic Searches

Having established that software developers engage in Naturalistic Decision Making when they are searching for code on the Web, we turn to Recognition-Primed Decision (RPD) Model of Rapid Decision Making, which explains how developers make decisions so quickly. The RPD Model [26] helps explain how decisions are actually made by experienced practitioners working under high stress and time pressure. In these circumstances, people make an assessment of the situation to identify the actions that could work. Actions are evaluated one after the other through mental simulation of possible outcomes. A good-enough solution that solves the problem is chosen instead of the best one.

We argue that developers make Recognition-Primed Decisions while evaluating candidates when they perform opportunistic Web searches to solve development problems. They first assess the situation to set search targets, expectations, and relevance cues. This situation assessment helps them to quickly identify potentially promising candidates, which are then evaluated, one after the other, until a good-enough solution is found to solve opportunistic problems. Promising candidates are mainly evaluated by software testing in an IDE and by mental simulation.

A reasonable question to ask is: Why Recognition-Primed Decision Model? RPD was a suitable decision model to use in our research study for two main reasons. First, RPD models rapid decisions that are made in situations of time pressure and high levels of stress. In our field studies, we observed that judgments in evaluation of search results took seconds. This model helps us understand how these rapid decisions can be made. Second, RPD models how experts make decisions, which is very useful in our study due to the fact that developers are experts in Web navigation.

We also considered other theories to understand how developers make decisions when they look for source code on the Web. We considered using the Information Foraging Theory [45]. This theory has been used before in software development settings to understand information-seeking behavior of programmers [41]; to determine relevant information when developers seek, relate, and collect information in maintenance tasks [27]; to explain and predict code navigation during debugging [29]; and to predict code navigation [40].

We decided not to use the Information Foraging theory because it has been used more to predict behavior [44, 46] than to explain it [28]. Also, this theory mainly considers only text or word matching as “scents,” but we observed other “scents” like: type of Web page, Web host domain, technical cues, and social cues that developers use to evaluate results. In addition, the theory does not take into account time constraints, which are very important in our study. However, we think that the concept of Information Scent [45] proposed by this theory could complement the set of naturalistic theories exposed here.

There are three phases in the RPD model: Situation Recognition, Option Evaluation, and Simulation. We will explain each of these phases and how they apply to opportunistic searches.

Situation Recognition in Source Code Search on the Web

The first step in situation recognition that decision-makers take is to make an assessment of the situation they are facing. Situation assessment has four important aspects: goals, expectations, actions, and relevance cues.

Goals Decision-makers determine the types of goals that can be reasonably accomplished in a situation.

The goal of Web searches done by developers is to solve the software problems they are facing. In our field studies, we asked developers “What was the goal of your search?” after each Web search they performed. Developers’ answers to this question were given in terms of the problems they wanted to solve. We identified 3 types of opportunistic software problems developers want to solve:

- Remembering/Fact Finding
- Clarifying Implementation Details
- Learning New Concepts

We discussed each of these types of software problems that motivate opportunistic Web searches in Section 4.1.1.

Expectations Decision-makers form expectations, which can serve as a check on the accuracy of the situation assessment.

Developers set expectations based on the problems they want to solve and previous experiences with similar searches. In our field studies, we asked developers “What were the expectations of your search?” after each search they performed. Based on developers’ answers, we found that most of the time developers had a good understanding of what they expected to find and they verbalized it mainly in terms of *search targets*. In a few cases, developers had a vague idea of what they expected to find and in only one case a developer did not know what he expected to find.

Developers assessed the situation to determine a reasonable search target. For example, Joseph was trying to find out what is the typical functionality of a check/uncheck all checkboxes. When we asked him what was the goal of his search he said: *“Ideally, I would like to find a user study that says what is the functionality that works best. I do not think I would find that, but examples*

of what other people do.” The ideal search target for Joseph was to find a user study. However, he thought that was not realistic based on his previous experience and he set his search target to find examples of what other people do.

We observed that developers set the following expectations, or search targets, for opportunistic searches:

- API Documentation/Tutorial
- Example/Code Snippet/Syntax
- Error Related Information
- Other

We discussed each of these search targets and how frequently they are used for each type of software development problem in Section 4.3.2.

Actions Decision-makers identify the typical actions to take when they are making an assessment of the situation.

Developers commonly formulated queries in a Web search engine to find solutions to their software problems. The number of queries and number of results evaluated depended on the type of problem they wanted to solve. When developers wanted to remember syntax details or find facts, they usually formulated only one query and evaluated only one result. When developers wanted to clarify implementation details they evaluated more than two results from one or multiple queries. In cases where developers wanted to learn new concepts, they performed one query or multiple queries and they evaluated one or more results.

Based on the 87 Web searches we observed in our field studies, we found that developers performed queries in a Web search engine, such as Google most of the time (86%). For a few searches (10%), they went to bookmarks or

URLs from sites they visited in the past. For almost half of the opportunistic search sessions, developers formulated only one query. They formulated multiple queries for 36% of opportunistic search sessions. We did not have information related to the actions developers took for 7% of the searches. Table 5.6 shows the actions taken by developers as well as the number of results visited for searches done to solve the different types of software problems.

	Action # Results Visited	Bookmark	1 Query			2-8 Queries		No info
			Zero	One	2-7	One	2-13	
Opportunistic Searches	Remembering/ Fact Finding	4	1	10	2	1	4	0
	Clarifying	3	1	2	12	2	12	5
	Learning	0	0	3	3	1	5	0
	SUBTOTAL	7 (10%)	2 (3%)	15 (21%)	17 (24%)	4 (6%)	21 (30%)	5 (7%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	0	0	12	2	0	2	0
	SUBTOTAL	0 (0%)	0 (0%)	12 (75%)	2 (13%)	0 (0%)	2 (13%)	0 (0%)
TOTAL		7 (8%)	2 (2%)	27 (31%)	19 (22%)	4 (5%)	23 (26%)	5 (6%)

Table 5.6: Actions and Number of Results Visited by Type of Software Problems

We found that developers take one of the following actions to find a solution to their software development problems:

- Go to bookmark or URL. Developers sometimes do not perform Google searches, but instead they go directly to Web sites for official documentation or Web sites that worked in the past. When Joseph had to find details of a JQuery method, he went directly to the Web page for jQuery. He said “*I knew I will find the answer there.*” He entered the word “Checked” in the jQuery Web site and searched for the property “checked” and went to

the first link and found the answer.

Developers decided to visit bookmarks or URLs when they needed to remember syntax details or when they wanted to clarify implementation details.

- **Query Formulation.** The most common action developers took was to go to a search engine such as Google and enter a query or multiple queries. When developers did not find a solution to their problems after evaluating the results from the first query, they tended to refine their queries.
 - **One query.** For 48% of the opportunistic search sessions, developers only entered one query. When developers wanted to remember syntax details or they were looking for open source projects, they visited only one search result after they performed a query. Developers visited between two to seven search results when they wanted to clarify implementation details. In few cases, developers found an answer only by looking at the search results and not visiting a single one. Most of the results visited were in the first page of results which commonly contains ten search results. Only in two searches did developers look up to page 2 and 3 of results.
 - **Two to Eight Queries.** For 36% of opportunistic searches, developers decided to refine their queries. This mainly occurred when developers wanted to clarify implementation details or find solutions to errors. When developers refined their queries they visited at least one result and a maximum of 13 search results. Only for 2 searches, developers visited results that were in the result pages 2 and 4. In those cases, developers were trying to find solutions to errors.

Relevance Cues Decision-makers increase the salience of some cues that they con-

sider important for the situation they are assessing.

When developers make judgments to decide which search results they should visit and which one they should choose to solve their problems, they use some cues that align with the problem they want to solve, their expectations, and their previous search experiences.

In our field studies, we explicitly asked developers which criteria they used to choose search result candidates. They commonly had a hard time answering the question and in some cases developers did not even remember the searches they performed.

Based on our observations and answers from developers in our field studies, laboratory experiment, and focus group, we identified 10 types of cues developers use to evaluate search results. We describe these evaluation cues in section 5.2.

Option Evaluation

The decision-maker evaluates action alternatives one at a time until a satisfactory one is found, in a serial manner. They do not usually do a comparison of alternatives. Experienced decision-makers usually try to find a satisfactory course of action, not the best one.

We analyzed the 87 search sessions we observed in our field sites and the 32 search sessions from our laboratory experiment to determine the type of option evaluation that developers used to find a good-enough solution that works to solve opportunistic problems. From our field study, we found that developers mainly use serial evaluation of candidates and comparison within a search session was rare. We found the opposite from our laboratory experiments, developers often compared search results within search sessions. We believe that the difference in our results has been impacted by the way relevance cues are shown in the laboratory experiment which made it easier to

do comparison of results just by looking at the result page. In many cases, developers did not use any of the previously mentioned option evaluation strategies because they only evaluated one result or went directly to bookmarks. In other cases, we did not have enough information about the evaluation of results.

We identified four types of option evaluation: serial, comparison within a search session, comparison between search sessions, and no serial or comparison. Table 5.7 shows the frequency of searches for each option evaluation type identified by type of software development problems from our field studies and Table 5.8 shows the same information for the laboratory experiment results. Here we explain, the three option evaluation types we observed for opportunistic searches: serial, comparison within a search session, and no serial or comparison.

Option Evaluation		Serial	Comparison within	Comparison between	No Serial or Comparison
Opportunistic Searches	Remembering/ Fact Finding	6	0	0	16
	Clarifying	22	2	0	13
	Learning	7	1	0	4
	SUBTOTAL	35 (49%)	3 (4%)	0 (0%)	33 (46%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	4	0	11	1
	SUBTOTAL	4 (25%)	0 (0%)	11 (69%)	1 (6%)
TOTAL		39 (45%)	3 (3%)	11 (13%)	34 (39%)

Table 5.7: Option Evaluation by Type of Software Problems from Field Studies

Serial We considered the option evaluation “serial” if developers visited more than one result and they chose to use a good-enough solution found without returning to previously evaluated results or comparing the different results visited. We

Option Evaluation	Serial	Comparison within	Comparison between	No Serial or Comparison
Opportunistic Searches	2 (13%)	12 (75%)	0 (0%)	2 (13%)
Non-Opportunistic Searches	8 (50%)	7 (44%)	0 (0%)	1 (6%)
TOTAL	10 (31%)	19 (59%)	0 (0%)	3 (9%)

Table 5.8: Option Evaluation by Type of Software Problems from Laboratory Study

consider developers chose a good-enough solution when they stopped evaluating candidates as soon as they found a candidate that helps solve the software problem. Developers did not look exhaustively for the best candidate. When developers did serial evaluation, they did it only for a single search session and not among different search sessions.

Based on our field studies, in 49% of opportunistic search sessions observed, developers did a serial evaluation of candidates. In this group, we found searches in which more than one search result candidate was evaluated but candidates were not compared. Serial evaluation was common when developers were trying to clarify implementation details or when they were trying to learn new concepts. In these cases, developers stopped their searches as soon as they found a good-enough solution that worked. Serial evaluation was less common in our laboratory experiment; participants only used it for 13% of opportunistic Web searches.

Comparison within a search session We considered that developers performed a “comparison within a search session” when they evaluated more than one candidate and they returned to previously visited results or compared multiple

results visited in the same search session.

Developers evaluated results by comparison within a search session for only 4% of opportunistic search sessions in our field study but it was used for 75% searches in the laboratory experiment. In our field sites, we only found three cases of this type of option evaluation done by three different participants. In these cases, developers were trying to clarify implementation details or learn new concepts. Developers evaluated more than one page to see which one was more “serious.” After comparing alternatives, they went back to the one that looked more serious or professional. In another case, a developer visited the first result but he was curious about other results. He visited two more results, but found some contradictions between them. So, he decided to use the first result.

We believe that the high number of developers performing comparison within searches in the laboratory experiments could be due to the fact that the search results in this study included a summary of technical cues and social cues, which made it possible for developers to compare different results by just looking at the result page.

No Serial or Comparison In this group, we included the searches in which developers evaluated less than two results in a search session or they did not compare the results between search sessions. We also included in this group the searches for which we did not have enough information to determine the type of option evaluation used.

In our field studies, we found that for 46% of opportunistic search sessions, developers did not use serial or comparative evaluation. These searches include 26% of searches where only one result was evaluated and also 3% of searches where developers found what they were looking for just by looking at the result

page without visiting a single result. It also includes 10% of searches in which developers went directly to bookmarks and no evaluation was involved. Finally, it also includes the 7% of searches for which we did not have enough information about how developers evaluated candidates. In our laboratory experiments, we found that for 13% of opportunistic search sessions, developers evaluated only one result.

Simulation

The decision-maker simulates an action in his/her imagination to evaluate if the action is satisfactory. The steps to be taken are simulated mentally to identify potential problems that could be encountered and how the problems might be handled. Depending on the simulation, decision-makers might implement the action as-is, modify it, or reject it and examine another action.

When developers evaluate search results, they do mental simulations of results but they also can do real simulations because they have access to source code that they can immediately copy and try in their development environment. We found that developers perform actual testing in an IDE for 45% of opportunistic searches, while they use mental simulation for 31% of opportunistic searches. For 24% of searches developer did not perform any simulation. Table 5.9 shows the frequency of searches for each type of simulation classified by the different software problems developers want to solve.

Coding/Testing For 45% of opportunistic search sessions, developers actually tested the source code they found on the Web in an IDE to evaluate a candidate. For example, when Bob was evaluating candidates he said *“I am not sure if this would work but lets try.”* Bob tried the source code and it did not work, then he made some changes and it worked and he stopped the search. This evaluation

	Simulation	Coding/ Testing	Mental	No Simulation
Opportunistic Searches	Remembering/ Fact Finding	13	8	1
	Clarifying	17	6	14
	Learning	2	8	2
	SUBTOTAL	32 (45%)	22 (31%)	17 (24%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	3	13	0
	SUBTOTAL	3 (19%)	13 (81%)	0 (0%)
TOTAL		35 (40%)	35 (40%)	17 (20%)

Table 5.9: Simulation Used by Type of Software Problems

by coding and testing mainly happened when developers found source code in the search results and they were trying to remember syntax, clarify implementation details, or fix bugs. Often, developers read the source code and typed it themselves in an IDE. It was not frequent for them to copy and paste lines of code. After writing the source code, they ran it, and they immediately knew if it worked or not. If it worked, they stopped the search and if it did not work they looked for another promising candidate or gave up the search.

Mental Simulation/Understand Information For 31% of search sessions, developers read and understood the information. In such cases, developers did not actually test source code but just read the information or watched videos of demos, which helped them make decisions. We observed this type of simulation when developers were trying to learn new concepts, or when they were looking for information about errors.

No Simulation For 24% of opportunistic search sessions, developers did not perform any simulation because they did not find what they were looking for and they decided to give up on the Web search to solve their software problem. Developers gave up mostly on searches to clarify implementation details and to find out how to fix bugs.

5.3.4 Developers do not Make Recognition-Primed Decisions for Non-Opportunistic Searches

In the previous subsection, we explained that developers make Recognition-Primed Decisions when they are performing opportunistic searches. For this type of search, developers evaluate candidates in a serial way and decisions are made quickly. In this subsection, we explain that when developers perform non-opportunistic searches they do not make recognition-primed decisions. Decisions in this type of search are usually done comparing several candidates and decisions are not rapid. Rather, deciding on an open source project can take more than one day. We describe what are the characteristics of these searches that make the recognition-primed decision model not completely suitable to identify these non-opportunistic searches.

Situation Recognition in Source Code Search on the Web

When developers perform non-opportunistic searches, they perform situation recognition but the actions that they take do not match the characteristics of the RPD model. Developers mainly perform comparison of alternatives given a set of criteria to evaluate different open source projects. We give details for each aspect of situation assessment: goals, expectations, actions, and relevant cues.

Goals The main goal of non-opportunistic searches is to find Software Tools or Open

Source Projects to be integrated into developers' source code or environment to solve a software development problem.

We discussed this type of software problem in Section 4.2.1.

Expectations When developers are performing non-opportunistic searches, the search targets they want to find are Open Source Projects, Software Tools, or information related to them.

We discussed these search targets and how frequently they were used for each type of software development problem in Section 4.3.2.

Actions When developers look for open source projects, they mainly (for 75% of non-opportunistic searches) read information they found after visiting one result from one query as seen in Table 5.6. In few cases, they evaluate more than one result when they perform one or multiple queries.

Relevance Cues Based on our observations and answers from developers from field studies, laboratory experiment, and focus group, we identified 10 types of cues developers use to evaluate search results. We described these evaluation cues for non-opportunistic searches in section 5.2.

Option Evaluation

We analyzed the search sessions in our field studies and laboratory experiment to determine how developers perform the evaluation of candidates for non-opportunistic searches. From our field study, we found that developers usually do comparison between search sessions to find a solution. We could not observe this behaviour in our laboratory experiment because developers only did one search session for a single task. Due to this restriction, developers did serial evaluation or compare candidates within the same search session.

Serial When developers were looking for information related to open source projects, they sometimes evaluated candidates one after the other without making a comparison between them. In our field studies, we found that developers did a serial evaluation of results for only 25% of non-opportunistic searches. However, we found from our laboratory experiments, that developers performed a serial evaluation of candidates for half of the non-opportunistic searches.

Comparison within a search session Developers also perform the evaluation of results by comparing promising candidates in a search session. We learned from our field studies that developers do not perform this type of evaluation when they look for open source projects. However, from our laboratory experiment, we found that developers did this type of evaluation for 44% of non-opportunistic searches. We believe this difference is influenced by the fact that developers could not do comparison between search sessions in the laboratory experiment. Instead, they did serial evaluation or comparison within the 10 results given in the result list.

Comparison between search sessions We considered that developers evaluated results by comparison between search sessions when they compared the results of a search session with results from another search session. The comparison was not done between candidates visited within a search session as in the previously discussed option evaluation strategy. In this case, developers needed multiple search sessions, or what we call a problem solving sequence as indicated in Appendix A, to select an open source project.

From our field studies, we found this type of option evaluation for 69% of non-opportunistic search sessions. We observed 11 of these searches and they were done to collect information about open source systems in order to select one to reuse. The results of each search session were compared to the results of other

search sessions. When developers looked for information for candidate systems, they mainly visited only one result to read documentation in a search session, but they then compared the result of that search session with results from other search sessions to find information about other open source candidates. When developers evaluated open source candidates, they compared the information of different systems in terms of price, installation requirements, architecture, type of support, popularity, and match with the project requirements. We did not observe this type of candidate evaluation in our laboratory experiments because participants only evaluated the 10 search results given by the researchers. Participants were not allowed to modify the query or perform another search session for a single task.

No Serial or Comparison In this group, we included the searches where developers evaluated only one result. In both our field studies and laboratory experiments, developers performed only one non-opportunistic search in which they evaluated only one search result and chose it.

Simulation

Based on our results from field studies, we found that developers perform mental simulation of results for 81% of non-opportunistic searches. This is mainly because developers cannot immediately try open source projects due to the installation requirements, so instead they read information about them. Only for 19% of non-opportunistic searches, developers actually tried the open source project as seen in Table 5.9.

5.4 Discussion

5.4.1 Comparison of Our Results and Results in the Literature

We found six empirical studies that report on how developers evaluate source code results to select one option that satisfies their source code needs. Two studies report on questionnaires [8, 65], two on lab experiments [55, 59], one on a laboratory experiment and log analysis [5], and one on interviews [34]. We compare our results with those found in the literature for the evaluation criteria used when developers are performing opportunistic searches, and then for the criteria used when developers are performing non-opportunistic searches.

5.4.2 Evaluation of Results in Opportunistic Searches

We summarized the empirical results in the literature that report on the evaluation of opportunistic searches in Table 5.10. This table includes information about the judgment strategy, judgment criteria, and the time spent during a session for some source code needs. Information in this table is based on findings from Brandt et al. [5] who reported on how developers evaluate code snippets to remember syntactic programming language details and routinely-used functionality, to clarify how to implement functionality in a specific programming language, and to learn unfamiliar concepts.

From our empirical studies we have learned that when developers try to remember syntax details, they often review only one search result. This partially matches with the literature stating that developers review zero or one search result when they try

	Judgment Strategy	Judgment Criteria	Time Spent
Remembering/ Fact Finding	- Zero or one result click - View only the search result snippets	- Developers knew exactly what information they were looking for	- Less than 1 minute
Clarifying	- Few result clicks	- Developers can easily recognize the code once they find it	- Around 1 minute
Learning	- Several result clicks - Rapidly skim several result pages opened in tab browsers	- Quality based on cosmetic features: prevalence of advertisement	- Tens of minutes

Table 5.10: Relevance Cues Reported in Literature for Opportunistic Searches

to remember syntax details. We found that it was not very common for developers to review only the search results to find a snippet. Also, we observed that this type of search took longer than reported in the literature. We found that on average, they take 3 minutes while the literature indicates they take less than a minute. This difference could be due to the fact that the time we consider is the time to solve a problem, which includes the use of the information, while the time reported in the literature only reports on the search on the Web. In addition, our studies have added to the current knowledge by finding that for this type of search, developers mainly use the result order, the presence of examples, the Web host domain, and social cues to evaluate search results. We also found that developers often use the source code found to test it in an IDE.

Our empirical studies showed that when developers are looking for information to clarify how to implement functionality in a specific language or to find information to fix bugs, developers visit multiple results from the result list. Our results agree with the literature in that developers visit many search results, but our studies give more precise information on this phenomenon. We found that developers perform between 1–8 queries for this type of search and they review between 2–13 results in a serial way. We also found that developers used 3 relevance cues among these: result order,

Web host domain, page type, and title. When developers found promising candidates, they mainly tested them in an IDE. In addition, we found that on average searches to clarify syntax information take around 5–6 minutes until the problem is solved. In contrast, Brandt et al. reported that this type of search takes around one minute.

We learned from our empirical studies that when developers want to learn new concepts, they commonly perform multiple queries and visit multiple results. This finding confirms the literature. However, we provide more details on the number of queries and results visited. We found that developers review between 2–13 results from 2–8 queries. Also, we found that developers mainly use 2 or 3 relevance cues to evaluate results including the result order, Web host domain, page type, and title. Developers prefer Web pages that come from trustworthy sources such as wikipedia, IBM, Microsoft, and official documentation. However, these results do not confirm the literature which states that cosmetic appearance is a relevance cue to evaluate results for learning. According to the literature, developers make judgments about the quality of the Web page based on the prevalence of advertisement, the fewer advertisements, the better the quality. Based on our empirical studies, this type of search takes shorter than what is reported in the literature. These searches take on average 6 minutes and not tens of minutes as indicated in the literature. We also found that developers mainly process mentally the information that they found.

5.4.3 Evaluation of Results in Non-Opportunistic Searches

Four empirical studies reported on how open source components and projects are evaluated [8, 34, 55, 65]. Table 5.11 shows a comparison of the criteria reported on these studies. The number that precedes some criteria indicates its ranking. For example (1) indicates that a criterion was the one most used by developers who

participated in the study. Chen et al. included 10 criteria in their study, but they only reported on the most used criterion and the two least used. Umarji et al. and Sim et al. reported the ranking of the 5 criteria they identified. Madanmohan and De' reported seven criteria but did not include ranking information for three of them.

Chen et al.	Umarji et al. and Sim et al.	Madanmohan and De'
(1) Requirements compliance	(1) Available functionality	(1) Functionality requirements
Functionality		
Open source licensing terms	(2) Licensing	Licensing
Quality of components (security, reliability, usability)		(2) Usability/Component characteristics
Architectural compliance		Architectural compatibility
Reputation of components and supplier		
Quality of documentation		
Environment or platform		
(9) Licensing price	(3) Price	(3) Costs
(10) Expected support from the open source community	(4) User support available	Maintenance and support
	(5) Level of project activity	
		(1) Familiarity

Table 5.11: Relevance Cues Reported in Literature for Non-Opportunistic Searches

All these four studies agree that requirements or functionality are the most used criteria to evaluate open source software. Licensing was mentioned in all the studies, but Umarji et al. and Sim et al. reported it as the second more important criterion. For Madanmohan, the second most important criterion was usability or component characteristics such as flexibility and design consistency. This criterion also matches with the quality of components mentioned by Chen et al. Cost was mentioned by all the studies, two of them reported it as the third most important criterion and Chen et al. reported it as the second least important criterion. Sim et al. and Umarji et al. reported user support availability as the fourth most important criterion. However, Chen et al. report it as the least important criterion. The last important criterion for Umarji et al. and Sim et al. was the level of project activity. These two last criteria match the maintenance and support criteria mentioned by Madanmohan and

De'. Finally, Madanmohan and De' indicated that familiarity, i.e. if developers have used the open source software before, is the most used criterion when resources and time are limited during the selection process.

From our empirical studies, we have found that looking for open source projects is not a common activity for developers in the workplace, but when they do it, this activity can take more than a day. We also found that there are two types of evaluation done when developers look for open source projects. One evaluation is done within a search session to find information about open source candidates and the other is to compare the information about different open source projects. When developers evaluate results within a search session they mainly use two relevance cues among the result order, Web host domain, page type, and title. The literature reports only on the criteria used by developers when they perform evaluations between search sessions and not on the criteria used to evaluate results within a search session for this type of search. The criteria we observed for the evaluation between search session have been reported in the literature. According to our observations, developers care about functionality, the system architecture, installation requirements, cost, and also what other people think about a system.

5.4.4 Laboratory Experiment versus Field Studies

Results from our laboratory experiment and field studies contradict each other on how developers evaluate search results. We believe that the difference in results is influenced by mainly four factors: the presentation of search results, the search results available for evaluation, the questions asked during the studies, and the nature of the open source project task. All these factors that influenced the difference in our results lead us to conclude that it is hard to observe naturalistic decision making in

a laboratory experiment.

The presentation of search results was different in the laboratory experiment and in the field studies as seen in Figures 5.2 and 5.3. For the field studies, developers used the search engine or Web page they preferred. Developers mostly used Google to start a Web search. Thus, they mainly evaluated search results provided by Google which shows the title, a small description, and a link with more information about the search result. In contrast, for the laboratory experiment, we presented search results augmented with technical information (lines of code, number of classes, and number of methods) and social information (number of favorites, number of copies, and percentage of positive reviews) about the source code in each result. The fact that we included these technical and social cues in the search results influenced our findings in two ways. First, if a relevance cue was available in the result list and was perceived as useful, developers used the information provided during evaluation of results with greater frequency. For example, in the field studies, developers on average used social cues for 13% of Web searches. However, in the laboratory experiment, developers used one of the social cues provided — the percentage of positive reviews — for half of Web searches. But showing the information is not enough; the information must be perceived as useful for developers. For example, although we showed some technical cues such as number of classes and number of methods, these were not used very often. Both were used for 13% of Web searches. Second, developers used the information shown to evaluate search results by looking only at the result list and without clicking search results. When developers evaluated Google search results, they needed to click a search result in order to read more details about the result and find out the number of lines of code or the number of positive reviews, if available. However, in the laboratory experiment we provided that information as a summary in the result list. Making the information available in the result list changed the way developers evaluated results because in some cases they evaluated results by just

looking at the result list and they were also able to make comparisons between results by just looking at the result list.

Another factor that explains the difference between the findings from our laboratory experiment and our field studies was the search results that were available for evaluation. In our field studies, developers did not have any restriction on the search results they could visit or the number of queries they could perform. However, in our laboratory experiment, we gave 10 results and we told participants that the right answer to complete the task was in one of them. Developers could only evaluate those 10 results and they were not allowed to perform query refinement. This restriction in the number of available search results influenced how developers evaluated the results. In the laboratory experiment, developers often methodically evaluated results one by one in order, which was rarely seen in our field study. Instead, in the field study, developers glanced at the results and started visiting some of them based on some cues. In addition, all the results showed in our laboratory experiment had the same layout and looked the same. The only difference was the text and code inside. That meant that we could not see if cosmetic appearance was used as an evaluation cue. In contrast, we observed in our field studies that developers used cosmetic appearance to evaluate results in 13% of Web searches.

The questions that researchers asked during the experiment also influenced the results. In the field study, we asked open-ended questions related to the evaluation of results; we asked *“What criteria did you use to select a candidate from the result list?”* In contrast, in our laboratory experiment, we asked participants to complete a paper questionnaire after each task. This questionnaire asked how useful each piece of information (i.e.: title, description, lines of source code) in the result list was on a scale of 1 to 5. Asking specific questions about each piece of information made participants think about whether they used a specific piece of information or not and

how useful it was. In our field studies, we did not ask explicitly for each piece of information; developers mentioned the ones that were relevant to them and that they remembered. For example, developers in the field studies often did not mention the title or the description as cues used. Developers mentioned that they used these cues for 14% and 13% of Web searches. However, in our laboratory experiment developers indicated using the title for all the searches and the description for 97% of searches. We believe that developers also used the title and description often in our field studies but developers just did not articulate it when asked and they did not make evident use of it. Developers often glanced at the title and description quickly and they did not mention it as a relevance cue when asked. A similarity between the laboratory experiment and the field studies, is that, in both cases, developers had a hard time remembering which cues they used even though we asked them about these just seconds or minutes after they finished a search.

Finally, the nature of the non-opportunistic searches also impacted the results. In the laboratory study, developers completed a task in which they had to evaluate 10 open source projects to find a piece of source code or a jar file that they had to reuse and integrate into their source code in 15 minutes so that a test case would pass. The task was one of these two: 1) to find source code to compare two text files and show the differences or 2) to look for some code that will read in a CSV file and place the data into a list of strings. In the laboratory, developers showed frustration completing this task and mentioned that this would not be the way they would look for source code. In our field study and focus group, we found that when developers look for a piece of source code, they do not look at open source project repositories to find it. They usually look on Google and find it in forums, tutorials, or official API Web pages. Developers usually look for open source projects when they are looking for a standalone system that they do not need to modify, and then do some research on how to integrate it into the other systems they have in place. This kind of search for

open source projects are uncommon and the evaluation of alternatives can take more than a day. We only observed one of our 24 developers doing this type of search in our field studies. He was looking for an open source project to do data mining of logs and to manage alerts. He looked at least at 10 different candidate systems and at the end of the day he was not done with the evaluation and he was planning on continuing the next day. In addition, in our laboratory experiment, participants were allowed to evaluate only the 10 open source projects we provided in the search result list. Due to this restriction, participants evaluated alternatives by comparing results within the search session. However, in our field study, we observed that developers devote one search session to look for information about each open source candidate, and then compare information between search sessions to select an open source candidate.

From our analysis of these differences between our findings, we conclude that laboratory settings make it hard to observe naturalistic decision-making being performed. This is due to the fact that in a laboratory experiment, there are many variables that we controlled (such as the time, the results showed, the layout of results, the nature of the task) that affected how developers actually searched for source code and imposed restrictions that sometimes do not exist in a workplace environment.

5.5 Summary

In this chapter, we have learned that developers evaluate source code search results differently when they are performing opportunistic searches and non-opportunistic searches. Decisions made during the evaluation of results are often quick and developers have a hard time articulating their evaluation process. When developers evaluate results for opportunistic searches they make quick judgments using Recognition-Primed Decisions. Developers first recognize the situation by identifying the goal,

relevance cues, expectations, and actions that match with a previous similar experience. Developers evaluate search candidates in a serial way by mental simulation or by actually testing a candidate in an IDE until they find a good-enough candidate that solves the problem. On the other hand, when developers evaluate candidates for non-opportunistic searches to find an open source project, they do a methodical and logical comparison of open source candidates. For each open source candidate, developers gather information for a set of characteristics such as the installation requirements, architecture, and cost. Developers compare candidates using these characteristics and choose an open source project. The decision to choose a specific open source project can take more than a day.

Chapter 6

Use of Information in Web Searches

In the previous chapters, we have discussed what kind of software problems motivate software developers to perform Web searches and also how they evaluate search results to find a solution to their software problems. In this chapter, we present results from our empirical studies on how developers use the information they found on the Web searches to efficiently solve their software problems.

We first present our results from field studies, laboratory experiments, and focus groups on how developers use the information they find on the Web. Then, we compare our results with those found in the literature. We also analyze the efficiency of Web searches in terms of success and length of time to help developers solve problems. In addition, we discuss when developers prefer to copy and paste source code from the Web or when they prefer to read it and use it to guide their coding. Finally, we present a summary of this chapter.

6.1 Use of Information in Web Searches

Based on our field studies, we learned that developers use the information they found on the Web differently for opportunistic and non-opportunistic searches. Developers often use the information they found for opportunistic searches by reading information from the Web and then understanding the information or using it as a reference for coding. Developers did not use any information for almost a third of their opportunistic searches because they did not find anything useful on the Web. When developers were performing non-opportunistic searches to find open source projects, they mainly read and understood the information they found for each candidate system. In few cases, they also download the software and try it. For 13% of these searches, developers did not find any useful information on the Web. Observations from our focus groups reinforce our field study results. However, our laboratory experiment results contradict the results from our field studies.

We analyzed our observations of developers and also the answers that developers gave us in our empirical studies. In our field studies, we observed how developers used the information they found on the Web. Also, after developers finished a Web search, we asked them: *“After you selected a piece of source code or information related to source code, how did you use it to complete the task you were working on?”* In our laboratory experiment, we observed how participants used the information they selected from the search results we provided. In our focus groups, we asked participants: *“How using or reusing source code from each group is different?”* after they finished classifying search targets in the card sorting game.

We present our results on how developers use the information they selected from the Web for opportunistic searches and for non-opportunistic searches. Table 6.1 and Table 6.2 summarize our results from field studies and laboratory experiments

respectively. These tables show the different uses that developers give to information they found on the Web by the types of problems developers want to solve.

Use of Information		Read and Understand	Read and Code	Copy and Paste	Copy, Paste, and Modify	Download and Try it	Nothing Useful Found
Opportunistic Searches	Remembering/ Fact Finding	8	10	1	1	0	2
	Clarifying	9	8	1	3	1	15
	Learning	8	2	0	0	0	2
	SUBTOTAL	25 (35%)	20 (28%)	2 (3%)	4 (6%)	1 (1%)	19 (27%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	12	0	0	0	2	2
	SUBTOTAL	12 (75%)	0 (0%)	0 (0%)	0 (0%)	2 (13%)	2 (13%)
TOTAL		37 (43%)	20 (23%)	2 (2%)	4 (5%)	3 (3%)	21 (24%)

Table 6.1: Use of Information by Type of Software Problems from Field Studies

6.2 Use of Information in Opportunistic Searches

Based on results from our field studies, when developers perform opportunistic searches to remember syntax details, to clarify implementation details, or to learn new concepts, they often read the information they find on the Web to understand it or to use it as a reference to code. Only in few cases, developers copy, paste, and modify or copy and paste, or download software and try it as seen in Table 6.1. Developers did not use the information they found because it was not useful for 27% of Web searches. However, results from our laboratory study did not match with our results from our field studies. In our laboratory experiments, we found that developers often copy, paste, and modify source code or just copy and paste source code.

Use of Information	Read and Code	Copy and Paste	Copy, Paste, and Modify	Download and Try it
Opportunistic Searches	0 (0%)	2 (13%)	14 (88%)	0 (0%)
Non-Opportunistic Searches	2 (13%)	4 (25%)	5 (31%)	14 (88%)
TOTAL	2 (6%)	6 (19%)	19 (59%)	14 (44%)

Table 6.2: Use of Information by Type of Software Problems from Laboratory Experiments

In our empirical studies, we identified 6 different types of use that developers gave to the information they found on the Web. We explain each of the categories of use we identified.

Read and Understand We observed that some developers use the information they found on the Web to read it and understand it. They did not use it directly to create or modify source code. Instead, they use it to understand or remember a concept, to understand why an error occurs, to form an opinion for a conversation, or to reply an e-mail. For example, a developer saw in a log file some warnings related to “fakeweb.” He did not know what was fakeweb, so he decided to look for that on the Web. He found the definition, read it, understood it, and continued with his activities. Another developer wanted to “*figure out if PostgreSQL supports time as data type.*” He was asked about that by a co-worker, so when he found the information on the Web, he read it, understood it, and prepared an e-mail to his co-worker.

In our field studies, we found that developers use the information they found on the Web to read it and understand it for 35% of opportunistic Web searches.

However, we did not observe this type of use in our laboratory studies. We believe these conflicting results are due to the fact that developers were asked to complete a coding task in the laboratory experiment and for that reason they often used the information to code. Similarly, in our focus groups, developers did not explicitly mention this type of use.

Read and Code When developers find a piece of source code on the Web that could help them solve a software problem, they often do not copy and paste it, but instead developers read it and use it as a reference to write their own source code. In our field studies, we observed that developers took source code from the Web as a reference for 28% of opportunistic Web searches, mainly when developers were trying to remember syntax details or when they wanted to clarify implementation details. For example, a developer wanted to remember the syntax for the SQL command JOIN. He found some examples on the Web, read them, and created his SQL statement without copying and pasting anything from the Web. In our focus group, many developers also indicated their preference for using information from the Web as reference examples or to get ideas for implementation. A participant in focus group 4 indicated that he rarely copies code, instead he uses it as a reference by putting the examples from the Web side to side to his coding environment, as he mentioned: *“It just guides my coding. I just keep it side by side while I am coding. When I have questions, I look at it. From documentation, I rarely copy code.”* Also, a participant in focus group 1 emphasized the importance of understanding the source code he finds. Our participant indicated: *“For documentation, you are not going to use their source. I mean, you will use it, but when you are looking at it using the documentation, you are not going to copy the documentation. You would never going to copy and paste, you are going to use what they say but you are supposed to understand what it means, because sometimes they will give you the*

parameters, but then you fill that in yourself, you are just looking for the name of the method. You apply the explanation, you apply what you learn to create your own source.” We did not observe any participant in our laboratory experiment who read the source code and used it as a reference. All participants completing tasks where they had to find a snippet of source code, copied and pasted source code from the Web.

Copy and Paste After developers found information that could solve their software problem, they copied it from the Web and pasted it to use it without making any modification to the information taken. In our field studies, we found that developers copied and pasted information without making any modification to it, only for 3% of opportunistic searches. In one case a developer copied a line with the URL to check the status of a server and pasted it into a Web browser to check the status of his server. Another developer needed to include a quick reference of log4j (a log management library) in a document, so he found it in a pdf on the Web and copied and pasted it from the pdf to his document. In our laboratory experiments, developers copied and pasted lines of source code for 13% (2 out of 16) of opportunistic Web searches. In both cases, developers copied a line of source code that included a regular expression. In our focus groups, participants indicated they sometimes copied and pasted source code from tutorials. One participant in focus group 1 emphasized that he does not copy and paste source code from tutorials often, except when he is learning, as he mentioned: *“For tutorial, you are not going to copy and paste the examples much, you might do it initially, for learning and that is it, so hopefully you will learn it and you will not need the examples anymore and you can do it yourself.”* Also, a participant in focus group 2 commented on the fact that sometimes developers find a piece of source code that they can reuse, he indicated *“Today, I actually copied and pasted a piece from a tutorial to do*

a unit test, that happened to me like an hour ago. Depending on the type of tutorial and forum, you can find pieces that you can copy and paste.”

Copy, Paste, and Modify Once developers found on the Web a good-enough solution to a software problem, we observed that some developers copied a piece of source code they found on the Web, they pasted it into their own source code, and they made some changes to tailor it to fit their needs. In our field studies, we observed that developers copied, pasted, and modified source code from the Web for only 6% of opportunistic Web Searches. They copied between 3–11 lines of source code. Developer used the information from the Web in that way when they wanted to remember syntax details or when they wanted to clarify implementation details. However, in our laboratory experiment, we observed that participants copied, pasted, and modified the source code from search results for 88% of opportunistic searches. We believe that the difference in results between these two studies is impacted by the fact that participants in the laboratory had to complete implementation tasks and they had source code available for all the search results. Similarly, most of participants in our focus groups mentioned that when they find examples or source code snippets, they often copy, paste, and modify it. As mentioned by a participant in focus group 10: *“For code snippets or examples, I copy and paste, modify variable names and add some comments to understand.”* Developers also mentioned copying, pasting, and adapting source code found in forums, as indicated by a participant in focus group 4: *“I think it is the same for forums and snippets because they look alike in the way they are small and they are used to solve a problem that others had but I might not have the same problem as them, so I would copy and paste but most of the time I would modify to suit my problem.”*

Download and Try it Rarely, developers had to download software to find solu-

tions to their opportunistic problems. In our field study, we observed only one search of this kind. In this case, a developer was having an error when running a web service. He performed a Web search and found a WSDL he could use to compile the Web service. He downloaded the WSDL, used it to compile his web service, and the error was fixed. We did not observe this type of use in our laboratory experiment.

Nothing Useful Found Sometimes, developers did not use information from the Web after performing a Web search because they did not find anything useful to solve their software problems. In our field studies, developers did not find useful information from the Web for 27% of opportunistic Web searches.

6.3 Use of Information in Non-Opportunistic Searches

In the previous section, we presented our results on how developers used the information they found on the Web when they were performing opportunistic searches. In this section, we will discuss our results from our empirical studies on how developers use information they find on the Web when they are looking for an open source project.

From our field studies, where we observed a developer performing Web searches for a day to find an open source project, we found that developers mainly read information from the Web about potential open source candidates. They read information about the architecture of the systems, watched some demos of the systems, and read what other people think about the project. Developers read information for 75% of non-opportunistic Web searches. In few cases, for 13% of non-opportunistic Web searches developers also downloaded software to actually install it and try it. Developers in our focus group also mentioned that they sometimes download the open source project,

as indicated by a participant in focus group 12: *“For systems, I will download and try it.”* Also, they commented on the importance of knowing others’ opinion on the system, as a participant in focus group 9 mentioned: *“For full systems, I read very carefully what other people think, who is using it. I download it and see the coding style. What other people think is very important.”*

Our results from our laboratory experiment did not match with our results from our field studies. In our laboratory experiment, developers downloaded a jar file from an open source project for 88% of non-opportunistic searches as seen in Table 6.2. Due to the fact that participants had to complete an implementation task using the jar file from the open source project, developers also had to look for information on how to use the jar file and what methods invoke to make the test case pass. Notice that the sum of percentages in the row for non-opportunistic searches in Table 6.2 is more than 100% because developers sometimes used the information in at least two ways: to find the appropriate jar file and then to find out how to use it. Developers copied, pasted, and modified examples on how to use the jar for 31% of non-opportunistic searches and they just copied and pasted for 25% of this type of Web searches. Only for 13% of non-opportunistic searches, developers use the examples they found as a reference to write their own source code.

6.4 Discussion

6.4.1 Comparison of Our Results and Results in the Literature

Once developers find a suitable piece of source code on the Web, they need to integrate it to their current knowledge or their current source code to solve the software devel-

opment problem that motivated the search. We are aware of three empirical studies that report on how developers use source code they found on the Web. One of them [5] reports on how developers use code snippets after they are found in opportunistic searches and two of them [8, 34] report on how developers use open source projects after they are selected in non-opportunistic searches. We first present our discussion on how our results agree or contradict the literature results for opportunistic searches and then for non-opportunistic searches.

6.4.2 Use of Information in Opportunistic Searches

Brandt et al. provide some evidence on how developers use code snippets on the Web to learn, clarify, and remember, as summarized in Table 6.3.

	Use of Information Reported in the Literature
Remembering/ Fact Finding	- View search results or API documentation to remember syntax - Copy and paste lines of code
Clarifying	- Copy several lines, paste, and adapt code to their needs
Learning	- Copy dozens of lines (approx. 10 lines each time), paste, and adapt code to their needs

Table 6.3: Use of Information Reported in Literature for Opportunistic Searches

We found from our empirical studies that when people perform Web searches to remember syntax details, they often read the information and use it to code or to understand it. In few cases, developers copy and paste source code. This agrees with the literature that indicates that developers use the information to read and also to copy and paste. However, the literature did not provide information on how frequent

each type of use is. We found that for 82% of this type of searches developers read the information and understand or copy, and only for 9% they actually copy information from the Web.

Based on our results from empirical studies, we learned that when developers perform searches to clarify implementation details or fix bugs, if they find useful information on the Web, they will often (46% of clarifying searches) use it to read it and then to guide their coding or understand the information. However, this type of use was not reported in the literature for clarifying searches. We also found that for few Web searches, only 11% of these type of searches, developers copied information from the Web. They copied between 3–11 lines of code. This result agrees with results in the literature that indicate that developers copy and paste several lines of source code when they want to clarify. We also found that for 41% of this type of searches developers did not find any useful information. However, the literature did not report on developers not finding the information they were looking for. The differences between our results and the ones found in the literature could be influenced by the fact that our results discussed here came from field studies and the ones reported in the literature came from a laboratory experiment.

From our empirical studies, we learned that when developers look for information on the Web to learn new concepts, they often use the information to read and understand the concepts and sometimes also to code. Our results contrast with the literature that reports that when developers look for information to learn, they copy, paste, and adapt dozens of lines of source code. We did not observe any case of that. Instead of copying and pasting, we observed developers reading the information they found on the Web to incorporate it to their knowledge or to guide their coding.

6.4.3 Use of Information in Non-Opportunistic Searches

In our empirical studies, we observed that when developers looked for open source projects, they used the information they found on the Web for each candidate system to evaluate it and decide which candidate is more appropriate for their needs. Sometimes developers also downloaded a candidate system and tried it. Our results are different from the literature. Our results emphasize the use of information for evaluation of candidates, and not the use after a system was selected as reported in the literature. In our observations, we saw a developer performing searches to evaluate open source candidates but none of our participants actually performed searches to use the chosen open source candidate.

Chen et al. and Madanmohan and De' studied how open source projects are used after they are selected. Both studies agree in that it is very common to adapt the open source software to integrate it to the developer's environment. Chen et al. reported that 45% of questionnaire participants needed to change the source code to fix bugs. This seems to be a very high percentage and could mean that the quality of open source software is not good. However, we expect the quality of open source to improve over the years as reported by the Coverity Report [11]. It was also reported that 39% of participants make changes for other reasons. These other reasons could include customization and integration changes.

In addition to source code adaptation, Madanmohand and De' also reported that sometimes developers contact the author of source code to seek permission for use and extension. In other cases, developers do not look at the source code even though they have access to it. This happens when developers do not need to modify the source code and the software teams do not have enough resources (knowledge, skills, manpower) to do it.

6.4.4 Are Developers Efficiently Solving Software Problems by Searching on the Web?

In our empirical studies, we found that developers efficiently solve software problems by searching the Web to remember syntax details, to learn concepts, or to find information to evaluate open source projects. However, when developers need to clarify implementation details or find solutions to fix bugs, searching for information on the Web is not very efficient to help them solve software problems. We define efficiency in terms of the success of a Web search to help solve a software development problem and in terms of the time it takes to solve a problem using a Web search, as indicated in Appendix A. We discuss our results for efficient and non-efficient problem solving by using Web searches. Table 6.4 shows the success rate and the time successful searches took for each type of software problem. Similarly, Table 6.5 shows the failure rate and the time failed searches took for each type of software problem.

Efficient Problem Solving by Searching on the Web

When developers perform opportunistic Web searches, they are highly efficient when they look for information on the Web to remember syntax details or to learn new concepts. Developers who searched the Web to remember or to find facts found useful information on the Web and were able to use it to solve their software problems on average in 2.7 minutes for 82% of this type of searches as shown in Table 6.4. Performing Web searches to remember facts helped developers to solve software problems in 1 second in the fastest case and in 9 minutes in the longest case. Similarly, developers who decided to search on the Web to learn new concepts were able to successfully solve 75% of their software problems in 5.9 minutes on average. However, developers efficiently solved only 49% of software problems that required clarification of implementation details in 6.2 minutes on average.

In the case of non-opportunistic searches, developers were very successful when they looked on the Web to find information related to open source projects they were evaluating. Developers successfully found the information they were looking for for 88% of this type of searches and they did it in 11.5 minutes on average. From our empirical studies, we have information on how successful developers were finding information to evaluate open source candidates. However, we do not have information on how successful developers were by actually using the open source project they selected.

		Success	Length of Searches		
			Min.	Avg.	Max.
Opportunistic Searches	Remembering/ Fact Finding	18 (82%)	1 sec	2.7 min	9 min
	Clarifying	18 (49%)	41 sec	6.2 min	38 min
	Learning	9 (75%)	2 min	5.9 min	18 min
	SUBTOTAL	45 (63%)	1 sec	4.9 min	38 min
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	14 (88%)	2 min	11.5 min	46 min
	SUBTOTAL	14 (88%)	2 min	11.5 min	46 min
TOTAL		59 (68%)	1 sec	8.2 min	46 min

Table 6.4: Success and Length of Efficient Searches by Type of Software Problems

Non-Efficient Problem Solving by Searching on the Web

Looking for information on the Web is not very efficient to solve software problems that required clarifying implementation details or finding solutions to bugs. Developers failed to solve these types of problem in 51% of cases as seen in Table 6.5. On the other hand, less than 26% of the other types of opportunistic and non-

opportunistic searches failed. Although developers were not very successful to solve problems to clarify implementation details or fix bugs, they decided to give up unsuccessful searches in 4.8 minutes on average which is 1.4 minutes faster to what an average successful search of this type takes. Similarly, developers were faster in giving up unsuccessful searches to look for information to evaluate open source candidates. Developers gave up this type of searches in 3.4 minutes on average which is 8.1 minutes faster than an average successful search of this type.

		Failure	Length of Searches		
			Min.	Avg.	Max.
Opportunistic Searches	Remembering/ Fact Finding	4 (18%)	1 min	4.3 min	7 min
	Clarifying	19 (51%)	8 sec	4.8 min	15 min
	Learning	3 (25%)	3 min	6.3 min	12 min
	SUBTOTAL	26 (37%)	8 sec	5.1 min	15 min
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	2 (13%)	3 min	3.4 min	4 min
	SUBTOTAL	2 (13%)	3 min	3.4 min	4 min
TOTAL		28 (32%)	8 sec	4.3 min	15 min

Table 6.5: Failure and Length of Non-Efficient Searches by Type of Software Problems

We found that developers gave up 32% of Web searches as shown in Table 6.5. We analyzed the reasons that motivated developers to give up search sessions as well as the actions they followed to solve the software problem. Table 6.6 shows the reasons that developers had to give up search sessions by type of software problems and Table 6.7 shows the actions that developers followed to solve a problem after they gave up finding a solution on the Web.

	Reason for Giving Up Search	No Solution Found	Found Starting Point to Solution	Source Code did not Work	Inconclusive Results
Opportunistic Searches	Remembering/ Fact Finding	2	0	1	1
	Clarifying	15	4	0	0
	Learning	2	0	1	0
	SUBTOTAL	19 (27%)	4 (6%)	2 (3%)	1 (1%)
Non-Opportunistic Searches	Looking for Open Source Projects or Software Tools	2	0	0	0
	SUBTOTAL	2 (13%)	0 (0%)	0 (0%)	0 (0%)
TOTAL		21 (24%)	4 (5%)	2 (2%)	1 (1%)

Table 6.6: Reasons for Giving Up Search Sessions by Type of Software Problems

Developers mainly gave up searches because they did not find a solution that could help them to clarify implementation details. After developers decided to stop a search session for this reason, they commonly code the solution by themselves, postponed dealing with the problem, or ask for help to a co-worker. Only in one case a developer decided to look for the information she was looking for in a book and another developer sent the question to a mailing list. We do not have information about the actions developers took after they gave up 7 Web searches.

For 5% of Web searches, developers did not find the solution to their software problems, but they found information that they used as a starting point to implement a solution by themselves. For the 2% of Web searches that were given up because the source code found did not work, developers solved the problem by coding a solution by themselves or asking for help to a co-worker. Requesting advice from a co-worker was also a way to solve a problem when developers found inconclusive results on the Web.

Action after Giving Up Search	Code Solution by Themselves	Communicate with Co-worker	Postpone Dealing with the Problem	Look for Info in a Book	Send email to Mailing list	No Info about Action
No Solution Found	6	2	4	1	1	7
Found Starting Point to Solution	4	0	0	0	0	0
Source Code did not Work	1	1	0	0	0	0
Inconclusive Results	0	1	0	0	0	0
TOTAL	11 (13%)	4 (5%)	4 (5%)	1 (1%)	1 (1%)	7 (8%)

Table 6.7: Actions after Giving Up Search Sessions by Reasons for Giving Up

6.4.5 Copying and Pasting Source Code versus Reading and Using it as Reference

In our field studies, we found that developers prefer to read the information from the Web and they use it to guide their coding instead of copying and pasting information. However, this preference has not been reported in the literature that reports how developers use information from the Web. We examined responses from our focus groups and found mixed answers regarding which practice developers prefer: copying and pasting source code versus reading and using it as reference. Some developers have a personal preference for one of these practices, for other developers the practice they use depends on the goal of the search, and other developers do not make any distinction between these two practices.

Participants in focus group 5 started a conversation about their preferences for copying and pasting source code versus reading and using it as reference after we asked them about how they use information from the Web. Participant B clearly prefers reading the information and using it to guide his coding by typing it by himself because this practice allows him to understand and learn the code. However, participant

A prefers copying and pasting. Here we show an extract of this conversation.

“B: I will synthesize the documentation into my own understanding.

A: But what about the code in there?

B: I would take the ideas, added to my own understanding of the world. Close the window and code my own thing.

A: It depends on how often I need to use it. I do not know.

B: Even if I do not close the window, I will not copy and paste.

A: You do not?

B: I will type it. I will personalize it to have fun, i will not copy and paste it.

A: I will copy and paste.

B: I will type it.

A: No, I will copy it and then delete what I do not need. What about if there are 50 lines of code?

B: Well, if I do not know something, I want to learn it.

A: OK.

B: Well, I just do not think I will learn if I just copy and paste.

A: That is true. I would rather copy and paste, but I think yours is a good principle.”

For other developers the decision of copying and pasting source code or reading and typing by themselves depends on the goal of the search and the information they find. For example, if developers are trying to fix a bug and they find information of a similar problem in a forum, they do not expect to copy and paste the solution but instead to adapt the solution to their needs as indicated by a participant in focus group 4: *“You are not looking for code that you are going to copy, you are looking for a solution to a problem that you have or you are looking for something that will allow you to change your code but you would very unlikely copy something from those forums into your code. If somebody has a similar problem, has a similar solution, you*

are hoping that you will be able to adapt that solution to yours but it is very very rare the case in which you can copy the code from a forum to your code.” Some developers prefer to copy and paste when they already know what they are looking for, but if they want to learn they prefer to follow the instructions step by step as indicated by a participant in focus group 9: *“If it is new information, I follow step by step. If I already knew the solution, I go directly and take the code snippet. If it is something general, I type it. I prefer to type it myself.”*

In other cases, developers do not make any distinction between these two practices as indicated by a participant in focus group 3: *“Often it is easier to copy and paste than to type it all over again. But I do not see a huge difference between copying and typing it. Usually they are short. I have not looked for multiple methods that i want to copy. It is usually very small, you know, an idiom that you need.”*

6.5 Summary

In this chapter, we discussed how developers use the information they find on the Web to solve software development problems. From our empirical studies we learned that developers mainly read the information from the Web and use it to understand it or to guide their coding. Copy and paste from the Web was not very common. Choosing between copying and pasting source code and reading and typing it depends on personal preferences of developers or also on the problem they want to solve and the information they find on the Web. However, some developers did not see any distinction between these two practices.

We also analyzed how efficient are Web searches in terms of their success to solve software problems and their length in time. We found that developers are highly

efficient to solve problems when they are trying to remember syntax details, learning new concepts, or looking for information to evaluate open source projects. Web searches are not very efficient to find information to clarify implementation details or to fix bugs. Less than half of this type of searches helped developers to find a solution to their problems. When developers did not find a useful solution on the Web, they decided to implement a solution by themselves, postpone dealing with the problem, or ask for advice to a co-worker. In few cases, developers look for an answer in a book or send questions to mailing lists.

Chapter 7

Implications and Discussion of Research Methods Used

In this chapter, we present the implications of the results from our empirical studies for tool designers, researchers, and developers. We end this chapter with a discussion of the research methods we used in this dissertation.

7.1 Implications for Tools

Our empirical studies show that developers perform Web searches differently to look for code snippets (opportunistic searches) and to look for open source projects (non-opportunistic searches). In this section, we provide implications for tools for these two types of searches separately.

7.1.1 Implications for Tools for Opportunistic Searches

From our empirical studies, we have learned that developers are highly successful when they do Web searches to find code snippets to remember syntax details or to find facts. In this case, developers know exactly what they are looking for and recognize it easily. However, developers are not highly successful when they search on the Web for code snippets to clarify implementation details or fix errors, which is the most common motivation for searching on the Web. We believe that improving tools to help developers be more successful in this type of searches can make a positive impact on the effectiveness of solving software problems. Based on our findings, we give the following recommendations to tool designers.

Make Examples and Source Code Snippets more Visible

Developers are mainly looking for examples or code snippets on the Web. Even when developers look for API Documentation or tutorials, they also want to see examples of how to use certain functionality. However, Web browsers do not facilitate the identification of examples or code snippets in the search results.

Due to the fact that we observed that developers often look for search results that contain examples or code snippets, we believe that it would be helpful if developers could know which search results contain examples or code snippets. One possibility is to augment the search results gathered from a search engine such as Google and analyze which ones have source code. Mica [59] and Assieme [20] are two tools that recently have shown that augmenting Web search results to make developers aware of which results contains source code examples of API can make Web searches more effective. Also, after identifying that a result has source code, we believe that it would be useful to show technical and social information related to a piece of source code. Recently, a prototype that augments Web search results with reputation information

has been developed [16].

Another possibility to make examples or code snippets more visible is to create a crawler to gather source code snippets from tutorials, forums, API documentation on the Web and create a repository of them. This repository could associate code snippets with text surrounding them in Web pages so that code snippets would have actual text associated with them to facilitate the matching between code snippets and keywords in queries [61].

Show Error Related Information when an Exception Occurs

Nineteen percent of searches to clarify information were to fix errors. When developers compile, run, or test their source code, they copy and paste the exception in a Web search engine and they try to find what causes an error, how it can be solved, and what are the experiences of other people with that same issue.

Due to the fact that we observed that developers look for error related information when their program shows an error, we believe that it would be helpful for developers if the IDE that throws an exception will also run a query on the Web with that exception. In that way, the IDE can show the developers not only the line where the exception was detected and the stacktrace, but also information found on the Web related to this error including potential causes, potential solutions, and what other people did when they encounter the same error.

Present Results from Web Searches in the Development Environment

We observed that sometimes after developers found a solution on the Web, they put side by side a window with the source code found and a window with their source code in an IDE. For this reason, we believe that it would be useful if developers could have both, their source code and results from the Web in the same environment.

Recent tools such as CodeGenie [30] and Blueprint [4] have explored this integration between the IDE and the Web.

We expect that in the future developers will have online IDEs and the integration of search results from the Web and their own code stored in the cloud would be easier and more natural.

7.1.2 Implications for Tools for Non-Opportunistic Searches

In our empirical studies, we observed that developers look for the same type of information to compare open source projects. Based on this observation, we believe that it would be helpful if developers would have a tool that gathers this information and shows it in a comparative way. There are not many applications that have this functionality. One of the few is Ohloh, which allows developers to enter the name of three open source projects and shows the same information for all the three systems in a table to facilitate comparison. We used Ohloh to compare three systems searched by one of our participants, Oscar and Figure 7.1 shows the results of this comparison.

For each of the projects, Ohloh shows general information including how recently the repository was updated, the home page, and license. It also includes repository activity for the project, code analysis, and reputation of the project given by Ohloh users.

We believe that tools that allow comparison of systems will be useful for developers. These tools should be flexible enough to support comparisons of more than three open source projects. Currently, Ohloh supports only comparison of three open source projects, as previously mentioned. However, we observed that a developer compared 10 open source systems and he was planning on evaluating more before selecting one.

Compare Projects







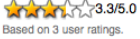
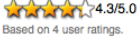
	 Solr Remove this project	 CloverETL Remove this project	 Talend Remove this project
General			
Ohloh Data Quality	✔ Updated 5 days ago	✔ Updated 2 days ago	✔ Updated about 22 hours ago
Homepage	lucene.apache.org	www.cloveretl.com	www.talend.com
Project License	Apache License 2.0	LGPL	GPL
Estimated Cost	\$2,394,852.00	\$5,748,075.00	\$80,460,445.00
All Time Activity			
Committers (All Time) View as graph	29 developers	47 developers	149 developers
Commits (All Time) View as graph	4,719 commits	8,705 commits	23,520 commits
Initial Commit	✔ almost 6 years ago	✔ almost 9 years ago	✔ about 5 years ago
Most Recent Commit	✔ 5 days ago	✔ 14 days ago	✔ about 24 hours ago
12 Month Activity			
Committers (Past 12 Months)	✔ 24 developers	✔ 18 developers	✔ 102 developers
Year-Over-Year Commits	✔ Stable	⚠ Decreasing	✔ Increasing
30 Day Activity			
Committers (Past 30 Days)	📌 15 committers	📌 6 committers	📌 55 committers
Commits (Past 30 Days)	71 commits	17 commits	694 commits
Files Modified	297 files	59 files	3,129 files
Lines Added	7,981 lines	3,492 lines	197,311 lines
Lines Removed	2,165 lines	1,821 lines	59,550 lines
Code Analysis			
Mostly Written In	Java	Java	Java
Comments	✔ Average	✔ Average	✔ High
Lines of Code View as graph	171,803 lines	400,607 lines	4,947,584 lines
People			
Managers	⚠ Position not yet claimed	 dpavlis	 ccarbhone
Ohloh Users	88 users	3 users	12 users
Ohloh User Rating	 Based on 34 user ratings.	 Based on 3 user ratings.	 Based on 4 user ratings.

Figure 7.1: Compare Projects Feature by Ohloh

Tools that help comparing projects should also include other criteria in addition to repository activity. This recommendation is based on the fact that we observed that developers evaluate open source projects having into account not only characteristics of their repository but also other characteristics such as the installation requirements and the architecture of the open source project.

7.2 Implications for Researchers

7.2.1 Extending Understanding of Source Code Search on the Web

Previous work [5] found that developers mainly had three intentions to look for source code on the Web: remember, clarify, and learn. We extended this work in two ways. One, we used the proposed Web intention classification for our analysis and we found the same three intentions, and also another that was not reported in that study. We found that developers also look on the Web to find tools to support programming activities or to find open source projects. Two, we focused on the judgments that developers make while evaluating search candidates and on how the information found on the Web is used. Brandt et al. found that developers use cosmetic features as a relevance cue. However, we found that they not only use cosmetic features but also they use other cues including the type of page (e.g. forums), source of page (e.g. official API documentation), and social cues (e.g. what other people think about a solution). We also provide details on the evaluation process followed by developers for each type of search. In addition, we also provide quantitative information on frequency of searches, effectiveness of searches, number of queries performed, number of results visited, reasons to give up searches, and use of information. Our results are

based on field studies of software developers in the workplace. Direct observations of developers on the workplace has not been reported before in the literature to study source code search behavior.

7.2.2 Searches for Code Snippets and Software Components are Different

Based on our findings using both an opportunistic problem solving approach and a naturalistic decision-making approach, we argue that looking for open source projects and looking for code snippets need to be studied as two different problems. This means that we recommend researchers to study these two types of searches separately because they have different motivations, different evaluation strategies, and different use of information found on the Web.

Using an opportunistic problem solving approach to understand the motivations of developers to perform Web searches, we found that developers perform opportunistic and non-opportunistic searches to solve software development problems. Opportunistic searches are done to find source code snippets to remember syntax details, clarify implementation details or fix bugs, and learn new concepts. On the other hand, non-opportunistic searches are done to find information to evaluate open source candidates. This opportunistic approach helped us realize that searches for code snippets and open source projects are two different problems.

In addition, using a naturalistic decision-making approach, we found that developers are using Recognition-Primed Decisions when they look for code snippets or source code examples. However, they do not use this type of unconscious and rapid decision-making when they look for open source projects or software tools. When developers need to choose an open source system to integrate into their projects, they

need to do an exhaustive and comparative evaluation of candidates. The comparison needs to be thorough because the decision is non-trivial for the project and also because developers need to justify why they chose a system over others. We also found that developers use different relevance cues or criteria to evaluate open source projects and snippets.

7.2.3 Applying Naturalistic Decision Making Theories

For our analysis of judgements made to evaluate search results, we turned to the decision-making literature, which does consider performance in complex domains. To our knowledge, our work is the first to use Naturalistic Decision Making Theory and the Recognition-Primed Decision Model to understand code search on the Web. We used the theory and model to bring a different perspective on judgments and decision-making. We believe that these could also be used to better understand decision-making when knowledge workers look for information on the Web. Also, these theories could be applied to any high stakes setting with multiple actors where decisions are made to solve ill-structured problems under time pressure and stress.

When applying this theory, we recommend researchers to carefully interpret the explanations given by participants about their rapid judgments. When we ask people to explain their thinking, we need to be careful in how we interpret their answers. Sometimes we request an explanation when an explanation is not possible. Researchers have found that often people know little about how they think and how they solve problems [35]. In our case, our results related to the evaluation cues used while reviewing search result candidates are based on a mix of observations and the answers given by developers which minimizes the impact of imprecise explanations.

7.3 Implications for Developers

Developers can also benefit from knowing that they are making unconscious and rapid judgments when they evaluate search results. By knowing that developers use first impressions when choosing candidates can help them realize what are the first impressions they are using and why they could not even consider some good candidates. It is possible to change our first impressions by changing the experience associated with them. Also, by being aware of these unconscious decisions developers can feel more comfortable answering “I do not know” when asked why they made those decisions.

Our research findings show how the Web is changing the way developers program and also raise questions about whether or not developers should be trained and hired differently. We provide empirical evidence that shows that looking for source code on the Web is a common activity for software developers and it is done more often than what is reported by them. Also, we observed that developers rarely use books or printed manuals to find solutions to their problems, or they less often memorize syntax details of commands. Instead, developers are using the Web as a source of information to find solutions and also as an external memory aid to remember syntax details.

Regarding the training of developers, our findings raise the questions: should developers be trained on the kinds of software problems that can be effectively solved using the Web?, should they be trained to evaluate code snippets and open source projects on the Web?, should they be trained to remix code snippets from the Web and to integrate code found on the Web? Regarding the hiring process of developers, currently this process often includes technical interviews where developers are asked to code on a white board without using the Web as a source of information. Based

on our observations, we believe that it would be more realistic to ask developers to solve a coding problem using the Web during a technical interview. In this way, interviewers could assess developers' skills on using the Web to program efficiently, a technique that developers will use almost everyday at work.

7.4 Discussion of Research Methods Used

We conducted empirical studies using four research methods. Each of them provided valuable data to understand the motivations that developers have to look for source code on the Web, how developers evaluate search results, and how they use the information they find on the Web. Each research method has strengths and weaknesses but these methods complement to each other because the strengths of some cover the weaknesses of others.

Online questionnaires were effective to gather personal opinions of a large number of professional developers working in diverse domains. Data collected also helped us quantify our findings from the literature review. For example, from the literature review we learned that developers have six different motivations to look for source code on the Web, but we did not know how frequent these motivations were. The online questionnaire helped us answer this question. We were aware that using online questionnaires involves collecting self-reporting data and that we had to trust that developers were actually reporting what they do. In addition, it is possible that participants misunderstood any of the questions or categories that we included in the questionnaire. To mitigate this issue, we conducted pilot surveys to verify that we were getting the information we were looking for. Another shortcoming of online questionnaires is the potential presence of the non-response bias. It is possible that the opinions of developers who decided to answer the questionnaire differ from the

opinions of those who did not answer the questionnaire.

Focus groups were useful to gather conversations and opinions of software developers talking in groups about source code search on the Web. This empirical study helped us collect anecdotes about Web searches and also discussions between participants. Similar to online questionnaires, data collected using this research method relies on self-reporting information, but in this case, we were able to interact with our participants to ask follow-up questions or clarifications. Also, the opinions of developers were influenced by their partners and possibly by the presence of the researcher running the focus groups.

Laboratory experiments were helpful to observe developers selecting and using information from the Web to complete software development tasks. In this case, the data collected does not rely on self-reported information but instead on direct observation. Also, laboratory experiments allowed us to control the environment for all participants, including the task assigned, the user interface, the number of results shown, and the maximum time to complete a task. Controlling these variables was useful to compare the behaviour of developers. Although laboratory experiments allowed us to collect data for specific aspects of source code search on the Web such as the evaluation of search results and the use of information from the Web, this type of empirical study has some limitations. The environment in the laboratory is different from the workplace where variables are not controlled, developers often work in teams, and developers have multiple interruptions while working on tasks. Also, we asked our participants in the laboratory experiments to think aloud while completing the tasks and a researcher was observing behind the participant, which might have affected the behaviour of developers.

Field studies were effective in gathering data from direct observations of developers in the workplace performing Web searches spontaneously to help them complete

their development tasks. During observation, we paid particular attention to what happened before and after the Web search which gave us the context of the task and the search. This context was useful to understand the motivations behind Web searches and how effective they were to help developers solve software development problems. In the field studies, developers were in their own work space, interacting with co-workers, and being interrupted for multiple reasons. We, researchers, sat behind a developers for a day. It is possible that the behaviour of developers could have been influenced by the fact that they were being observed. However, we tried to mitigate this effect by explaining that our observations were for research purposes and not to be reported to their managers in the company. We also interacted with our participants by going for lunch with them so they will be more comfortable around us. We also observed them from a place that was not very intrusive to them. We sat behind them, not next to them. At the end of the observation day, we asked our participants if we disturbed them during the day. All of them indicated that we did not disturb their work and some of them mentioned that they even forgot they were being observed at some point during the day. We also collected Web searches using the browser plug-in when developers were not observed. As part of our future work, we plan to compare the Web searches that developers performed when they were observed and when they were not, to study the potential effect that the presence of a researcher might have on the Web search behaviour of developers.

Gathering data using a combination of research methods allowed us to triangulate the phenomenon of source code search on the Web. This triangulation strengthens the validity of our research findings and deepens our understanding of how, when, and why developers look for source code on the Web. The benefit of combining multiple research methods is that the shortcomings of some methods are covered by other methods. Both online questionnaires and focus groups rely on self-reported data given by developers about source code search on the Web. In contrast, laboratory experi-

ments and field studies helped us collect data on developers actually performing Web searches without depending on self-reported information from developers. Another shortcoming of online questionnaires was the potential presence of non-response bias, that is, that developers who answered the questionnaire might have different opinions from the ones who did not answer the questionnaire. This shortcoming was covered by field studies where most of observed developers were volunteered by their managers. In addition, the limitation of laboratory experiments in allowing us to observe developers in a simulated environment was covered by field studies where we observed developers performing Web searches in their own working environment.

Chapter 8

Conclusion and Future Work

In this chapter, we present the conclusions for this dissertation as well as the work we would like to do in the future in the area of source code search on the Web.

8.1 Conclusion

The increased availability and quality of open source code on the Web is changing the way software developers write source code. Developers are using the Web as a huge source code repository to look for source code they could reuse to solve a software development problem. It is important to understand how developers look for source code on the Web so that tools and approaches can be suggested to better support their needs.

We used a set of complementary empirical studies to understand the phenomenon of looking for source code on the Web from different but interrelated perspectives. We first performed a survey of the literature in order to have an overview of what we know so far about source code search on the Web. Next, we used two comple-

mentary approaches to study source code search on the Web. The first was to gather developers opinions on this activity by collecting developers reflections via an online questionnaire and team reflections via a focus group. The second approach was to observe developers not only talking about code search but also actually performing code search on the Web. For that purpose, we conducted field sites in companies in the US and abroad, and also laboratory experiments to test specific hypothesis where we needed to control some variables. We present the results of our analysis by answering the research questions that were posed in the introduction.

What motivates developers to search the Web to find source code to complete their software development tasks?

We analyzed Web searches using an opportunistic problem solving approach to find out what motivates developers to look for information on the Web. We found that developers mainly perform searches to opportunistically solve software development problems (82% of Web searches). Opportunistic searches are ad hoc and are done to remember syntax details, clarify implementation details or fix bugs, and learn new concepts. The software problem that developers want to solve define the search target. We found that developers mainly look for examples, code snippets, syntax, or API documentation when they want to remember or find a fact. When developers want to clarify implementation details or find a solution to a bug, they mainly look for API documentation, examples, code snippets, and error related information. If developers need to learn new concepts, they usually look for tutorials or documentation for APIs. On the other hand, non-opportunistic searches (only 18% of Web searches) follow a systematic process and are performed to find open source projects.

We also found that looking for source code on the Web is a common technique used by developers to solve problems. We learned that developers perform searches more often than what they report doing. In a survey, only 45% of developers re-

ported performing Web searches daily, but in our field observations we found that 83% of developers performed at least one Web search during their work day and they performed on average 3.6 searches per day.

What information and strategies do developers use to evaluate search results when they perform Web searches to find source code?

We learned that developers evaluate source code search results differently when they are performing opportunistic searches and non-opportunistic searches. Developers use different information and strategies to evaluate results for these two types of searches.

Using a naturalistic decision-making approach, we found that when developers evaluate results for opportunistic searches they make quick and unconscious judgments using Recognition-Primed Decisions. Developers first recognize the situation by identifying the goal, relevance cues, expectations, and actions that matches with a previous similar experience. When trying to remember detailed syntax, developers often perform only one query and evaluate one result, but when clarifying implementation details or learning new concepts they perform multiple queries and evaluate multiple results one after the other. Developers use two or three relevance cues including the result order, the presence of examples or code snippets, and the Web host domain. When they want to remember or clarify syntax details, developers evaluate the results through actual coding or testing the source code in an IDE. When developers find information to learn new concepts, they often read the information and understand it. They do not usually test it in an IDE.

On the other hand, when developers perform non-opportunistic searches to look for open source projects, they often perform only one query and evaluate one result to find information about a candidate system. After they collect information for sev-

eral candidate systems, they compare the systems based on the system architecture, installation requirements, cost, and the opinion of other people about the system. Developers often mentally process the information they gather for each candidate they find in a Web search session. Selecting an open source candidate requires multiple Web search sessions.

What strategies do developers use to reuse/use source code found on the Web?

From our empirical studies, we learned that developers use the information they found on the Web differently for opportunistic and non-opportunistic searches. Developers often use the information they found for opportunistic searches by reading information from the Web and then understanding the information or using it as a reference for coding. When developers perform non-opportunistic searches to find open source projects, they mainly read and understand the information they found for each candidate system. In few cases, they also download the software and try it.

Are source code searches on the Web efficient to complete software development tasks?

We define efficiency of Web searches in terms of the success of a Web search to help solve a software development problem and in terms of the time it takes to solve a problem using a Web search. We found that developers were able to successfully solve 63% of their opportunistic software problems in 4.9 minutes on average by using the information they found on the Web. Problems were solved between 1 second and 38 minutes. We found that developers are highly efficient when they are trying to remember syntax details or to learn new concepts. Web searches are not very efficient to find information to clarify implementation details or to fix bugs. Less than half of this type of search helped developers to find a solution to their problems. In these

cases, developers often did not find anything useful on the Web or they found only a starting point for a solution. After developers gave up on a Web search, they solved the problem by coding a solution by themselves or asking for advice to a co-worker. In other cases, developers decided to postpone dealing with the problem for the moment.

In the case of non-opportunistic searches, developers were very successful when they looked on the Web to find information related to open source projects they were evaluating. Developers successfully found the information they were looking for for 88% of this type of searches and they did it in 11.5 minutes on average. From our empirical studies, we have information on how successful developers were finding information to evaluate open source candidates. However, we do not have information on how successful developers were by actually using the open source project they selected.

What are the implications of our results for tool designers, researchers, and developers?

From our empirical studies, we have learned that developers are not highly successful when they search on the Web for code snippets to clarify implementation details or fix errors, which is the most common motivation for searching on the Web. We believe that improving tools to help developers to be more successful in this type of searches can have a positive impact on their effectiveness in solving software problems. The following recommendations can help improve tools that support opportunistic problem solving. First, we suggest to make examples and source code snippets more visible in search results by showing in the result list which results contains code and presenting a summary of source code characteristics for each result. Second, we recommend that IDEs should show error related information gathered from the Web when an exception occurs. This could include information about the reason of the error and what other people did to solve it. Third, we recommend

to show Web search results in the same environment where developers are coding, which could facilitate the integration of results and use of the programming context to perform a Web search.

Our results also have implications for software engineering researchers. One, our results extend the understanding of source code on the Web and also present a model to characterize this phenomenon. Two, our results show empirical evidence that looking for code snippets and looking for open source projects are two different problems that should be studied separately. Three, we have applied the Naturalistic Decision Making Theory and the Recognition-Primed Decision Model to understand how developers evaluate search results. This theory and model bring a different perspective on judgments and decision-making. We believe that this theory and model could also be used to better understand decision-making when knowledge workers look for information on the Web. Also, this theory and model could be applied to any high stakes setting with multiple actors where decisions are made to solve ill-structured problems under time pressure and stress.

Regarding the implications for developers, we believe that developers can also benefit from knowing that they are making unconscious and rapid judgments when they evaluate search results. By being aware of these unconscious decisions and that they use first impressions when choosing candidates can help them to realize what are the first impressions they are using and why they could not even consider some good candidates. It is possible to change our first impressions by changing the experience associated with them. In addition, our research findings show how the Web is changing the way developers program and also raise questions about whether or not developers should be trained and hired differently.

8.2 Future Work

Although the results of this dissertation have extended the understanding of source code search on the Web, there are still questions that could be explored in the data we collected. Also, there are some studies that could be conducted to broaden our understanding of source code search on the Web.

During our field studies, we collected data for the Web searches we observed, but we also collected self-reported searches that developers performed for a period of three weeks. These Web searches were collected in a paper format in Peru and using the browser plug-in in the US. We collected around 300 searches only in Peru. This data can help validate our observations on the goal of the search, motivation, duration of the search, number of queries, results visited, and success. Analysis of this data can also help showing if there is a difference between searches that we observed and searches that were self-reported.

In this dissertation, we briefly explored few cultural differences among developers in the US and Peru, such as the prevalence of collaborative searches in the US and the fact that queries are written in English in Peru where the native language is Spanish. However, more cultural differences could be analyzed from our data including: differences in motivations to perform searches, in queries performed, in evaluation of results, use of information, and success of searches.

In both our focus groups and field studies, we observed that developers had a preference for using question-answering Web sites, such as Stack Overflow, to find source code snippets and find solutions to program errors. Stack Overflow is a Web site where anyone can post questions and receive answers from the general community. The original poster can select the best comment and “accept” it as the answer to the question. There are currently over 2.5 million questions on the site. Good questions

and comments can receive “up votes” and earn reputation for the author. There have been some studies on Stack Overflow about how developers ask and answer questions on the Web [64]. But it would be beneficial to conduct studies on Stack Overflow to also understand how programmers use snippets in their conversations with each other. Some questions include: what kinds of questions elicit a snippet as part of the accepted answer?, how do code snippets function as a part of speech?, how are they used to ask questions?, how are they used to answer questions? and do snippets provoke new questions or requests for clarification or are they self-explanatory?

We found that developers are looking for code snippets for 82% of Web searches and we suggested in our implications that it could be useful for developers to have a code snippet repository. We envision this code snippet repository to help developers make their opportunistic searches more efficient. We already have a prototype of this kind of repository [61]. To build the repository, we crawled code snippet from different Web pages, such as tutorials. We would like to also include code snippet from Stack Overflow. For indexing code snippets, we used text around them in Web pages and also properties of the code based on static analysis. We still need to run some studies to determine which is the best way to rank code snippets. We currently show the snippets accompanied by a summary of characteristics but also need to conduct usability studies to improve the presentation of code snippets. We would also like to conduct some comparative studies between our source code snippet search engine and other code snippet Web pages such as Sniplr and Smipple.

Bibliography

- [1] A. Aiken and B. R. Murphy. Implementing regular tree expressions. In *Proceedings of the 1991 Conference on Functional Programming Languages and Computer Architecture*, pages 427–447. Springer-Verlag, 1991.
- [2] S. Bajracharya and C. Lopes. Mining search topics from a code search engine usage log. In *Proceedings of the 6th IEEE Working Conference on Mining Software Repositories*, pages 111–120, Vancouver, Canada, 2009. IEEE Computer Society.
- [3] S. Bajracharya, J. Ossher, and C. Lopes. Searching api usage examples in code repositories with sourcerer api search. In *Proceedings of 2010 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, pages 5–8, Cape Town, South Africa, 2010. ACM.
- [4] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-centric programming: Integrating web search into the development environment. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, pages 513–522, Atlanta, Georgia, USA, 2010. ACM.
- [5] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1589–1598, Boston, MA, USA, 2009. ACM.
- [6] L. D. Catledge and J. E. Pitkow. Characterizing browsing strategies in the world-wide web. In *Proceedings of the Third International World-Wide Web conference on Technology, tools and applications*, pages 1065–1073, New York, NY, USA, 1995. Elsevier North-Holland, Inc.
- [7] S. Chatterjee, S. Juvekar, and K. Sen. Sniff: A search engine for java using free-form queries. In *Proceedings of the 12th International Conference on Fundamental Approaches to Software Engineering*, Lecture Notes in Computer Science, pages 385–400, Berlin/Heidelberg, 2009. Springer.
- [8] W. Chen, J. Li, J. Ma, R. Conradi, J. Ji, and C. Liu. An empirical study on software development with open source components in the chinese software industry. *Software Process*, 13(1):12, 2008.

- [9] Y.-F. Chen, M. Nishimoto, and C. Ramamoorthy. The c information abstraction system. *IEEE Transactions on Software Engineering*, 16(3):325–334, mar 1990.
- [10] R. Cottrell, R. J. Walker, and J. Denzinger. Jigsaw: a tool for the small-scale reuse of source code. In *Companion of the 30th international conference on Software engineering*, pages 933–934, Leipzig, Germany, 2008. ACM.
- [11] Coverity. Coverity scan open source report, 2009.
- [12] B. Dagenais and H. Ossher. Automatically locating framework extension examples. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 203–213, Atlanta, Georgia, 2008. ACM.
- [13] R. E. Gallardo-Valencia and S. E. Sim. Internet-scale code search. In *Proceedings of 2009 ICSE Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation*, pages 49–52, Vancouver, Canada, 2009. IEEE.
- [14] R. E. Gallardo-Valencia and S. E. Sim. Information used and perceived usefulness in evaluating web source code search results. In *Proceedings of the 29th International Conference on Human Factors in Computing Systems*, Vancouver, Canada, 2011. ACM.
- [15] R. E. Gallardo-Valencia and S. E. Sim. What kinds of development problems can be solved by searching the web?: A field study. In *Proceedings of the 2011 ICSE Workshop on Search-Driven Development-Users, Infrastructure, Tools and Evaluation*, pages 41–44, Honolulu, Hawaii, USA, 2011. ACM.
- [16] R. E. Gallardo-Valencia, P. Tantikul, and S. E. Sim. Searching for reputable source code on the web. In *Proceedings of the Group Conference 2010*, Florida, USA, 2010. ACM.
- [17] M. Goldman and R. C. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages and Computing. Special Issue on Best Papers from VL/HCC 2008*, 20(4):223–235, 2009.
- [18] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. A search engine for finding highly relevant applications. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, pages 475–484, Cape Town, South Africa, 2010. ACM.
- [19] F. Gysin. Improved social trustability of code search results. In *Proceedings of the International Conference on Software Engineering*, Cape Town, South Africa, 2010.
- [20] R. Hoffmann, J. Fogarty, and D. S. Weld. Assieme: finding and leveraging implicit references in a web search interface for programmers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, Newport, Rhode Island, USA, 2007. ACM.

- [21] R. Holmes and R. J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *Proceedings of the 29th International Conference on Software Engineering*, pages 447–457, Minneapolis, MN, USA, 2007. IEEE Computer Society.
- [22] R. Holmes, R. J. Walker, and G. C. Murphy. Approximate structural context matching: An approach to recommend relevant examples. *IEEE Transactions on Software Engineering*, 32(12):952–970, 2006.
- [23] O. Hummel and C. Atkinson. Extreme harvesting: test driven discovery and reuse of software components. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, pages 66–72, Las Vegas, Nevada, USA, 2004. IEEE.
- [24] O. Hummel, W. Janjic, and C. Atkinson. Code conjurer: Pulling reusable software out of thin air. *IEEE Software*, 25(5):45–52, 2008.
- [25] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, and S. Kusumoto. Ranking significance of software components based on use relations. *IEEE Transactions on Software Engineering*, 31(3):213–225, 2005.
- [26] G. A. Klein, J. Orasanu, R. Calderwood, and C. E. Zsombok. *Decision Making in Action: Models and Methods*. Ablex Publishing Corporation, Norwood, New Jersey, 1993.
- [27] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, pages 971–987, 2006.
- [28] J. Lawrance, C. Bogart, M. Burnett, R. Bellamy, K. Rector, and S. Fleming. How programmers debug, revisited: An information foraging theory perspective. *IEEE Transactions on Software Engineering*, 2011.
- [29] J. Lawrance, M. Burnett, R. Bellamy, C. Bogart, and C. Swart. Reactive information foraging for evolving goals. In *Proceedings of the 28th international conference on Human factors in computing systems*, Atlanta, Georgia, USA, 2010. ACM.
- [30] O. A. L. Lemos, S. Bajracharya, J. Ossher, P. C. Masiero, and C. Lopes. Applying test-driven code search to the reuse of auxiliary functionality. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 476–482, Honolulu, Hawaii, 2009. ACM.
- [31] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336, 2009.

- [32] M. A. Linton. Implementing relational views of programs. *ACM SIGSOFT Software Engineering Notes*, 9:132–140, April 1984.
- [33] J. Lofland, D. Snow, L. Anderson, and L. Lofland. *Analyzing Social Settings: A Guide to Qualitative Observation and Analysis*. Wadsworth/Thomson Learning, Belmont, CA, 2006.
- [34] T. R. Madanmohan and R. De'. Open source reuse in commercial firms. *IEEE Software*, 21(6):62–69, 2004.
- [35] N. R. F. Maier. Reasoning in humans. ii. the solution of a problem and its appearance in consciousness. *Journal of Comparative Psychology*, 12:181–194, August 1931.
- [36] D. Mandelin, L. Xu, R. Bodik, and D. Kimelman. Jungloid mining: helping to navigate the api jungle. *ACM SIGPLAN Notices*, 40(6):48–61, 2005.
- [37] G. Marchionini. *Information seeking in electronic environments*. Cambridge University Press, New York, NY, USA, 1997.
- [38] F. McCarey, M. Cinnide, and N. Kushmerick. Knowledge reuse for software reuse. *Web Intelligence and Agent Systems*, 6(1):59–81, 2008.
- [39] G. C. Murphy and D. Notkin. Lightweight lexical source model extraction. *ACM Transactions on Software Engineering and Methodology*, 5:262–292, July 1996.
- [40] N. Niu, A. Mahmoud, and G. Bradshaw. Information foraging as a foundation for code navigation (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, Waikiki, Honolulu, HI, USA, 2011. ACM.
- [41] M. P. O'Brien and J. Buckley. Modelling the information-seeking behaviour of programmers - an empirical approach. In *Proceedings of the 13th International Workshop on Program Comprehension*, pages 125–134, St. Louis, MO, USA, 2005. IEEE.
- [42] J. J. Ockerman and C. M. Mitchell. Case-based design browser to support software reuse: theoretical structure and empirical evaluation. *International Journal of Human-Computer Studies*, 51:865–893, 1999.
- [43] S. Paul and A. Prakash. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering*, 20(6):463–475, 1994.
- [44] P. Pirolli. Rational analyses of information foraging on the web. *Cognitive science*, 29(3):343–373, 2005.
- [45] P. Pirolli. *Information foraging theory: adaptive interaction with information*. Oxford Univ. Press, 2007.

- [46] P. Pirolli and W.-T. Fu. Snif-act: A model of information foraging on the world wide web. In *Proceedings of the 9th international conference on User modeling*. ACM, 2003.
- [47] D. Poshyvanyk, A. Marcus, and Y. Dong. Jiriss - an eclipse plug-in for source code exploration. In *Proceedings of the 14th IEEE International Conference on Program Comprehension*, pages 252–255, Athens, Greece, 2006. IEEE Computer Society.
- [48] S. P. Reiss. Semantics-based code search. In *Proceedings of the 2009 IEEE 31st International Conference on Software Engineering*, pages 243–253, Vancouver, Canada, 2009. IEEE Computer Society.
- [49] P. Robillard. Opportunistic problem solving in software engineering. *IEEE Software*, 22(6):60–67, 2005.
- [50] N. Sahavechaphan and K. Claypool. Xsnippet: mining for sample code. *ACM SIGPLAN Notices*, 41(10):413–430, 2006.
- [51] E. Salas and G. Klein. *Linking expertise and naturalistic decision making*. Lawrence Erlbaum Associates, Mahwah, NJ, 2001.
- [52] R. C. Seacord, S. A. Hissam, and K. C. Wallnau. Agora: A search engine for software components. *IEEE Internet Computing*, 2(6):62–70, 1998.
- [53] C. Silverstein, H. Marais, M. Henzinger, and M. Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33:6–12, September 1999.
- [54] S. E. Sim, C. L. A. Clarke, and R. C. Holt. Archetypal source code searches: A survey of software developers and maintainers. In *Proceedings of the 6th International Workshop on Program Comprehension*, pages 180–187, Ischia, Italy, 1998. IEEE Computer Society.
- [55] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How well do internet code search engines support open source reuse strategies?, To appear 2012.
- [56] R. Sindhgatta. Using an information retrieval system to retrieve source code samples. In *Proceedings of the 28th international conference on Software engineering*, pages 905–908, Shanghai, China, 2006. ACM.
- [57] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, page 21. IBM Press, 1997.
- [58] J. D. Sinnott. *Everyday Problem Solving: Theory and Applications*. Praeger, New York, 1989.

- [59] J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In *IEEE Symposium on Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006*, pages 195–202, Brighton, United Kingdom, 2006. IEEE.
- [60] A. Sutcliffe and M. Ennis. Towards a cognitive theory of information retrieval. *Interacting with Computers*, 10(3):321–351, 1998.
- [61] P. Tantikul. Jcsse: Java code snippet search engine. Master’s thesis, Information and Computer Sciences, University of California, Irvine, 2011.
- [62] K. Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.
- [63] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of 22nd IEEE/ACM international conference on Automated software engineering*, pages 204–213, Atlanta, Georgia, USA, 2007. ACM.
- [64] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web?: Nier track. In *Proceedings of 33rd International Conference on Software Engineering*, pages 804 –807, Honolulu, Hawaii, USA, 2011.
- [65] M. Umarji, S. E. Sim, and C. Lopes. Archetypal internet-scale source code searching. In B. Russo, E. Damiani, S. Hissam, B. Lundell, and G. Succi, editors, *IFIP International Federation for Information Processing 275: Open Source Development, Communities and Quality*, pages 257–263. Springer, 2008.
- [66] T. D. Wilson. Information behaviour: An interdisciplinary perspective. *Information Processing and Management*, 33(4):551–572, 1997.
- [67] T. Xie and J. Pei. Mapo: mining api usages from open source repositories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, pages 54–57, Shanghai, China, 2006. ACM.
- [68] Y. Ye and G. Fischer. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th International Conference on Software Engineering*, pages 513–523, Orlando, Florida, 2002. ACM.
- [69] Y. Ye, Y. Yamamoto, K. Nakakoji, Y. Nishinaka, and M. Asada. Searching the library and asking the peers: Learning to use java apis on demand. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*, pages 41–50, Lisboa, Portugal, 2007. ACM.

Appendices

A Glossary

This appendix defines the concepts we use in this dissertation.

Efficiency We define efficiency of Web searches in terms of the success of a Web search to help solve a software development problem and in terms of the time it takes to solve a problem using a Web search. We consider that a search is successful when developers found what they were looking for and the information found helped them to solve the problem that motivated the Web search. To measure the time a Web search takes, we consider that the Web search started when a developer entered a query in a Web browser or when a developer accessed a bookmark. We consider the search ended when a developer solved the problem using the information found on the Web or gave up on finding information on the Web to find a solution.

Search Session A Search Session represents a period of continuous Web usage to fill a single information need in the same day.

Some definitions of sessions used by other researches indicate that Web usage must be continuous with no breaks longer than 5 minutes [53], 6 minutes [5], or 25.5 minutes [6]. However, we decided not to include a cutoff in the longest time

a break could take in the same day. Our decision was based on the fact that we observed that developers had interruptions longer than 25.5 minutes while using the Web and when they came back they continued reviewing information on the Web or refining queries to fill the same information need. For that reason, the only time constraint we consider is that Web usage should happen the same day. Instead, we put more emphasis on the intention of the search. If Web usage was done to meet the same information need, we consider all those intervals as part of the same Search Session. The three longest breaks that we observed in our field sites were of 3 hours 25 minutes 46 seconds, 1 hour 34 minutes 19 seconds, and 1 hour 16 minutes and 18 seconds. In these cases, developers interrupted their search to chat with co-workers, answer calls from customers, code, write documentation, and have personal breaks. When they used the Web again, they read the information they found before and in some cases they also copied and pasted lines of source code and used them even though more than 3 hours passed since their first query.

Problem Solving Sequences Often developers only need one Search Session to solve their software problems or to decide to give up on finding information on the Web to find a solution. This is mainly because developers only need to meet an information need to solve a problem. However, in other cases, developers need to meet more than one information need in order to solve their software problem and for that reason they had to perform more than one Search Session. We call Problem Solving Sequences to the set of Search Sessions done to solve a single software problem.

Problem Solving Sequences were common when developers were looking for Open Source Projects. Developers performed individual Search Sessions to find information about individual system candidates. For example, the developer we observed, who was looking for an open source project to do data mining of logs

and to manage alerts, performed 10 separate Search Sessions to find information of 10 different open source project candidates. The information need for each search session was to find the architecture, installation requirements and other information of a single system. Then, he compared the results from different Search Sessions to evaluate the different systems and choose one open source project to use.

B Online Questionnaire

Introduction

We are interested in how people look for source code on the web, in any language, for any purpose. By source code, we mean anything written in a programming language. You might be looking for a few lines, a complete system, or anything in between.

Instructions

Please take a few moments to answer the questions below.

Looking for Source Code

We'll begin by asking you about how you search for source code on the Web in general. For now, we're just interested in your habits and tendencies, rather than any particular search.

1. How often do you search for source code on the Web?

- Never
- Once per month
- Once per week
- Everyday
- Many times a day

Other (Please specify):

2. What are you trying to accomplish by looking for source code on the Web?

- Reuse source code
- Find examples
- Remember syntactic programming language details or frequently used functionality
- Learn unfamiliar concepts
- Fix a bug
- Get ideas

Other (Please specify):

3. What are you looking for when you are searching for source code on the Web?

- Some lines of source code
- Implementation of data structures, algorithms, or parsers
- Libraries or APIs
- Frameworks
- Open source projects
- Tutorials

Other (Please specify):

4. What resources do you use when searching for source code on the Web?

- General purpose search engines (such as Google or Yahoo!)
- Source code specific search engines (such as Koders or Krugle)
- Specific websites
- Domain knowledge
- References from peers
- Mailing lists
- Forums
- Blogs

Other (Please specify):

5. If you do not use code-specific search engines, such as Koders and Krugle, why not?

- I was not aware that they existed
- They do not offer good support for my search
- I am more used to general-purpose search engines

Other (Please specify):

Looking for Code Snippets

Now we are going to ask you about how you search for code snippets. Specifically, we are interested in the resources that you use to locate snippets and the criteria that you use for selecting them. When we refer to code snippets, we mean a few lines of source code.

6. How often do you use these sites to look for source code snippets?

	Never	Once per month	Once per week	Everyday	Many times a day	Never heard of it
Google	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Google Code Search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Koders	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Krugle	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Smipple	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sniplr	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sourceforge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Stackoverflow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

7. Other (Please specify):

8. How important are the following criteria when evaluating candidate code snippets?

	1 - Not at all important	2 - Slightly important	3 - Neutral	4 - Very important	5 - Extremely important
Architectural compatibility	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Cost/effort required to adapt or integrate	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Documentation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Frequency of updates	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Functionality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
License	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Quality based on appearance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of source code	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Responsiveness to feature requests	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for users	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Time to close bugs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

9. Other (Please specify):

Looking for Software Components

When working on a development project, sometimes you want a software component that already implements some functionality that you do not want to code by yourself.

Now, we are going to ask you about how you search for these software components. Specifically, we are interested in the resources that you use to locate software components and the criteria that you use for selecting them.

10. How often do you use these sites to look for software components?

	Never	Once per month	Once per week	Everyday	Many times a day	Never heard of it
Google	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Google Code Search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Koders	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Krugle	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Smipple	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sniplr	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sourceforge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Stackoverflow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

11. Other (Please specify):

12. How important are the following criteria when evaluating candidate software components?

	1 - Not at all important	2 - Slightly important	3 - Neutral	4 - Very important	5 - Extremely important
Architectural compatibility	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Cost/effort required to adapt or integrate	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Documentation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Frequency of updates	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Functionality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
License	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Quality based on appearance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of source code	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Responsiveness to feature requests	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for users	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Time to close bugs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

13. Other (Please specify):

Looking for Open Source Projects

Now we are going to ask you about how you search for open source projects. Specifically, we are interested in the resources that you use to locate open source projects and the criteria that you use for selecting them. When we refer to open source projects, we mean a stand alone project that can be used on its own and whose source code is available.

14. How often do you use these sites to look for open source projects?

	Never	Once per month	Once per week	Everyday	Many times a day	Never heard of it
Google	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Google Code Search	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Koders	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Krugle	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Smipple	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sniplr	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Sourceforge	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Stackoverflow	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

15. Other (Please specify):

16. How important are the following criteria when evaluating candidate open source projects?

	1 - Not at all important	2 - Slightly important	3 - Neutral	4 - Very important	5 - Extremely important
Architectural compatibility	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Cost/effort required to adapt or integrate	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Documentation	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Frequency of updates	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Functionality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
License	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Quality based on appearance	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Reputation of source code	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Responsiveness to feature requests	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Software quality	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for users	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Support for developers	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Time to close bugs	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Other	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

17. Other (Please specify):

Background Information

Finally, we will ask you some questions about your occupation, your programming experience, and your skill level with Web search engines.

18. What is your current occupation?

- Undergraduate Student
- Graduate Student
- Industrial Developer

Other (Please specify):

19. If you are a student, have you ever been employed as a developer in industry?

- Yes
- No

If Yes, for how long?

20. Indicate how many years of development experience you have

21. How often do you use Web search engines such as Google or Yahoo!?

- Once per week
- Once per day
- 2-5 times per day
- 6-10 times per day
- 11+ times per day

Other (Please specify):

22. Rate your skills in using Web search engines such as Google or Yahoo!

- 1 - Beginner
- 2 - Intermediate
- 3 - Advanced
- 4 - Expert

Closing Text

Thank you for helping us with our research. By taking the time to complete this survey, you have contributed to our understanding of how everyday programmers search for code on the web.