TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
BLAVATNIK SCHOOL OF COMPUTER SCIENCE

Investigating Opportunistic Software Development Using Social Media Recommendation System

Master's Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Alexey Zagalsky

under the supervision of
Prof. Amiram Yehudai

August 2013

Abstract

Software developers face a constantly changing set of programming languages, platforms and technologies. Software projects may involve numerous technologies or platforms, while software developers are not able to master them all. In order to overcome these challenges software developers seek help on Question and Answer (Q&A) websites such as Stack Overflow. We believe that Q&A websites' success is related to the rapid pace of technology changes, where wikis and official documentation lag behind. By using social media, and specifically Q&A websites, developers might be able to mitigate these concerns. Stack Overflow embodies domain knowledge not only in the form of Q&A or comments but as source code as well, where developers post code snippets embedded in their question or answer. Whereas social media will play an increasingly important role in software engineering research and practice, there are currently few software development tools available that leverage social media, and even fewer code recommendation systems based on social media.

We begin our research with the aim of creating a recommendation system to support example usage while leveraging social media. But one can't design such a system without studying human-machine interactions, and most importantly human behavior in software development. Furthermore, designing useful tools for developers requires to identify the concerns and micro-activities involved in example usage when using social media. Our aim is then not the design of a tool or a recommendation system, but rather to investigate concerns involved in opportunistic software development with the use of the system we designed.

We choose to employ qualitative research methodology with design-based research method. Qualitative research methodology can deal with complexities that arise from not only technical issues in software development, but from human-machine interactions, and most importantly from human behavior in software development.

An example usage survey among professional developers regarding activities, provides the basis for the design and design decisions of a social media based recommendation system. We follow the design decisions and implement Example Overflow, a code search and recommendation system which brings together social media and code recommendation systems.

Based on Example Overflow, we conduct a user study, involving observations and interviews with professional developers asked to solve pre-determined tasks. We find that professional software developers have concerns related to example usage, concerns that govern how and when examples are used. However, developers do not avoid example usage altogether, but rather mitigate these concerns with micro-activities.

Acknowledgements

I wish to thank my advisor Amiram Yehudai. I am grateful for his guidance, constantly positive attitude, feedback and support. He allowed me to be independent and was willing to give me the time needed to learn from my own mistakes.

I wish also to thank Ohad Barzilay for helping me make progress, even when it was hard to do so. In addition to his academic advice, I appreciate him as a mentor and a friend.

I would like to thank my fellow students and office colleagues for their support, advice and friendship. In particular I wish to thank Mati Shomrat, Tamar Lavee, Liron Cohen, Ori Lahav, Lena Dankin, Yoni Zohar, and Udi Boker.

Finally, I want to thank the participants of the user study who were willing to spend their time, and agreed to be observed and questioned while they were working under pressure and out of their comfort zone.

This research was supported in part by The Israel Science Foundation (grant No.476/11).

"To raise new questions, new possibilities, to regard old problems from a new angle, requires creative imagination and marks real advance in science."

Albert Einstein

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the following thesis we describe our qualitative research investigating opportunistic software development using social media based recommendation system. We design a social media based code recommendation system, and investigate concerns involved in opportunistic software development with the use of the system we designed. We find that professional software developers have concerns related to example usage, concerns that govern how and when examples are used. However, developers do not avoid example usage altogether, but rather mitigate these concerns with micro-activities.

1.1 Background and Motivation

The success of social media has introduced new ways of exchanging knowledge via the Internet, and leveraging this knowledge is an important skill for a professional software developer[49]. Software developers face a constantly changing set of programming languages, platforms and technologies. Software projects may involve numerous technologies or platforms, while software developers are not able to master them all. In order to overcome these challenges software developers seek help on Question and Answer (Q&A) websites such as Stack Overflow[1]. Developers faced with a problem post questions on Q&A websites, while other members of the community may answer it, or add different answers to the existing ones. If the community is large enough, questions would be answered quickly, and common questions may have been answered previously. Stack Overflow, a popular Q&A website, uses social mechanisms to facilitate knowledge exchange between users and to create an information archive. In Stack Overflow, a programmer can ask a question about almost any programming related topic, and receive a detailed response within 10 minutes median [35]. The way Stack Overflow is designed allows each question and answer to be rated. Eventually for each question, the best answer is chosen to be "the accepted answer" for that question. In addition, members can edit each question and each answer to allow the information to constantly evolve and remain up to date. If we look at the publicly available Stack Overflow usage statistics[2]: 2.08M users, 5.18M questions, 9.53M answers and 5.47 questions per minute, we can see signs of its success[3]. We believe that Q&A websites' success is related to the rapid pace of technology changes, where

[1] http://stackoverflow.com/

[2] http://api.stackoverflow.com/1.1/usage/methods/stats

[3] http://blog.ninlabs.com/2013/03/api-documentation/

wikis and official documentation lag behind. By using social media, and specifically Q&A websites, developers might be able to mitigate these concerns. Stack Overflow embodies domain knowledge not only in the form of Q&A or comments but as source code as well, where developers post code snippets embedded in their question or answer. Whereas social media will play an increasingly important role in software engineering research and practice[49], there are currently few software development tools available that leverage social media, and even fewer code recommendation systems based on social media. Brandt *et al.* propose [10] that embedding a task-specific search engine in the development environment can significantly reduce the cost of finding information and thus enable programmers to write better code more easily. They developed Blueprint, a Web search interface integrated into the Adobe Flex Builder development environment that helps users locate example code. Ponzanelli *et al.*[43] present a recommendation system in the form of a plugin for Eclipse IDE allowing automatic query generation based on keywords extracted from the code, mining Stack Overflow knowledge base, and displaying the result inside the IDE, thus minimizing context switching.

We begin our research with the aim of creating a recommendation system to support example embedding Eco-system[6] while leveraging social media. But one can't design such a system without studying human-machine interactions, and most importantly human behavior in software development. Latoza and Myers[32] describe a design process intended to design useful tools for developers, tools that support software development work. They describe hierarchical decomposition of software development work into tasks through task analysis [14]. In each task at each level in the decomposition, developers have a goal, either a question to answer or something to accomplish. At the highest level tasks reflect activities, e.g. implementing features or refactoring. The activities can be further decomposed into sub-activities, e.g. understanding the code or debugging. At a lower level, developers formulate a specific plan for accomplishing a goal as a sequence of steps or a strategy. Useful tools are tools that support work by making a strategy faster or more successful. Similarly, we need to identify the concerns and micro-activities involved in example usage when using social media. Our aim is then not the design of a tool or a recommendation system, but rather to investigate concerns involved in opportunistic software development with the use of the system we designed.

We choose to employ qualitative research methodology with design-based research method. Qualitative research methodology can deal with complexities that arises from not only technical issues in software development, but from human-machine interactions, and most importantly from human behavior in software development. Qualitative research aims to study complexities of human behavior (e.g. motivation), and the reasons for that behavior. Furthermore, we are interested to investigate research questions with real practitioners, while collecting a broad set of data of various types. We use qualitative research tools, a survey, participant observations, and interviews to gather the data, and apply qualitative data analysis. Design-based research is one of the qualitative research methods, its goal is to design, create and study a single theoretically-inspired system or environment, as it systematically changed through multiple iterations, while simultaneously improving practices based on collaboration with practitioners in real-world settings, and leading to contextually-sensitive design principles and theories[56].

An example usage survey among professional developers regarding activities, provides the basis for the design and design decisions of a social media based recommendation system. Two main design decisions are formed - allowing comparison of multiple examples, and minimizing context switching. Secondary design decisions are also resolved by confirming with the literature, e.g. when deciding how many recommendation results are to be shown, we follow [21],

and confirm this with an initial benchmark of Example Overflow. They conduct an eye tracking analysis for Google search users, and show that there is a drop in viewing time and number of clicks at the 6/7 ranked results. We follow the design decisions and implement Example Overflow (EO), a code search and recommendation system which brings together social media and code recommendation systems. Example Overflow mines Stack Overflow's data, searching for accepted answers that have code snippets in them. We follow a conservative approach by choosing only accepted answers to ensure retrieval of high quality results. It analyzes these answers and extracts the code snippet and all the accompanying information: the question title, the question body, the answer body, the code snippet itself, the user rating of the answer from Stack Overflow, the view count of the question, the tags associated with the question and other relevant information. Example Overflow provides code recommendations by using a keyword search based on the term frequency-inverse document frequency (tf-idf) weight [58], while leveraging the social media provided by Stack Overflow - it uses both the code snippet and the additional metadata which accompanied the code snippet at Stack Overflow. This allows a developer to find code snippets that may not contain the search query keyword, but the keyword appears in the contextual data and indicates that it has been used in that context.

Based on Example Overflow, we conduct a user study, involving observations and interviews with professional developers asked to solve pre-determined tasks. We find that professional software developers have concerns related to example usage, concerns that govern how and when examples are used. However, developers do not avoid example usage altogether, but rather mitigate these concerns with micro-activities.

1.2 Findings and Contributions

The contribution of the research described in this thesis arises from the type of research questions we have chosen to follow and the complexities involved. We study real practitioners, professional software developers, and their behavior while investigating opportunistic software development using social media recommendation systems. We use qualitative research methodology, with qualitative research tools: surveys, participant observations and interviews. Large qualitative datasets of various types, represented as words and pictures, are collected by many different participants, introducing substantial problems of alignment, coordination, and analysis.

Furthermore, we design a social media based code recommendation system, Example Overflow, as part of the design-based research (DBR) method. However, social media is an emerging topic in the field of software engineering, and social media based recommendation systems are scarce. Employing DBR means that we need to correctly analyze the design decisions and form an effective design. DBR researchers sometimes compare this to the "egg drop" experiment[16], where students are given raw eggs and a few basic materials. They are asked to construct a "packaging" for an egg that will cushion it from breakage, even when dropped from a considerable height. This is not an easy task, oftentimes scholars fail to identify the key features that lead to an effective design. An important contribution of our research is the actual design and implementation of Example Overflow, a social media based code recommendation system.

Specifically, our research answers the following research questions:

RQ1: Is limiting software development in example driven manner helpful?

In economics and law fields, paternalism[18] is the interference of a state or an individual with a person or a group's liberty or autonomy for their own good. For example the compulsory wearing of seat-belts[13]. Looking at knowledge worker studies[28] and knowledge worker management guides[15], we see that autonomy is important to knowledge workers, however, some efforts to improve knowledge worker performance may involve limiting his autonomy. Can a similar approach be applied by limiting software development in example driven manner? One would expect that limiting may hinder or slow developers, prevent them from using certain resources. On the other hand, limiting developers in example driven manner, especially in opportunistic development, where time is of the essence, may benefit them: they'll be more focused with the task at hand, use more examples and create quick prototypes. We divide participants of the user study to two groups and compare their results, participants who are limited to searching with a social media example code recommendation system, and participants who are not limited. Our study shows that limiting professional developers in example driven manner did not improve their results.

RQ2: How do professional developers mitigate concerns related to example usage?

Services ,such as Stack Overflow, Github, Sourceforge, allow developers to access vast amount of source code on-line. Furthermore, some of the available code is reviewed, debugged and patched by other developers and may be considered as a high quality source for code snippets. On the other hand, as shown in our survey (see Section 3.4.1), example usage is a popular practice among developers, yet developers have concerns involved with example usage. Developers may hesitate to use or admit they use code examples. We identify the concerns professional developers have when using examples and see how they mitigate them in opportunistic development.

RQ3: What are the micro-activities involved in opportunistic development when using social media based recommendation system?

We use observations and interviews to identify micro-activities involved in opportunistic development when using social media based recommendation system. Specifically, we focus on activities that mitigate the concerns involved with example usage. We identified 5 micro-activities to mitigate concerns involved with example usage: task comprehension, forming and refining the query, browsing and examining results, reading additional context, and diversity in using the example.

RQ4: When searching for code examples, do developers refine their query or continue examining additional results?

When searching for examples, do developers use a single query to initiate a search and examine all results without refining their query, or do they work iteratively: form a query, initiate a search, examine results, refine query, and repeat the process until choosing a suitable example. We show the overall number of queries used by a developer per task is 5.95. Participants who used Example Overflow had an average of 7.7 queries per task, while participants who used Google search had an average of 4.2 queries per task.

RQ5: How many code examples are examined before choosing a suitable code example?

Granka *et al.*[21] conduct an eye tracking analysis for Google search users, and show that there is a drop in viewing time and number of clicks at the 6/7 ranked results, and a sharp drop occurs after result 10, as ten results are displayed per page. However, [21] investigated "regular" search with Google search, while we are interested with code example search based on social media recommendation systems. We speculate that a developer examines 3 code examples (in average) before choosing a suitable code example. Determining the average number of code examples examined by a developer before choosing a suitable code example, may influence the design of code recommendation tools, and specifically social media based recommendation tools. In our study, we found that the average number of code examples a developer would examine before doing copy/paste is 2.02.

RQ6: When searching for code examples, do developers use additional context?

If developers use or rely on additional context, it implies that offering a rich-verbose context in place, as an integral part of the code snippet, addresses a genuine need of the developers. "Complementary" tools that are lacking this functionality are missing an important requirement. If developers don't use additional context, it should be further investigated whether this additional context, though not used by many developers, is important - if so look for ways to encourage developers to use it. If not (not used, not important) - pragmatic tools such as Example Overflow could justify "hiding" additional context and offer clean code / improve code browsing.

1.3 Thesis Outline

In chapter 2 we review related work. This research involves multiple research fields in the area of software engineering. We discuss social media influence on software engineering, code recommendation systems and the combination of the two. We present related work involving the study of software developers, and overview the research tools used. We present work related to example usage among software developers, and we discuss work that may help with identification of example usage concerns among developers.

In chapter 3 we describe the methodology used for this research. The type of research questions we engage with, require the choice of an appropriate research methodology. A research methodology that can deal with complexities that arises from not only technical issues in software development, but from human-machine interactions, and most importantly from human behavior in software development. Additionally, we are interested to investigate research questions with real practitioners, while collecting a broad set of data. We choose to employ qualitative research methodology with design-based research method, we carefully follow the methodology practices and guidelines to ensure the validity of the methods used. In this chapter we elaborate on the research methodology, the research course and the research tools used.

In chapter 4 we describe the design, design decisions and implementation of a Social Media based Code Recommendation System (SMCRS). As part of the design-based research, we designed and implemented Example Overflow, a code search and recommendation system which brings together social media and code recommendation systems, that allowed us to test and analyze different design decisions of social media based recommendation systems. We conclude

this chapter by presenting an initial benchmark of Example Overflow.

In chapter 5 we provide the results of our research. We examine each research question, provide the relevant experiment data, and discuss possible implications.

In chapter 6 we conclude this thesis.

Chapter 2

Related Work

In this chapter we present the related work concerning this thesis. Our research lies in the fields of social media in software engineering and code search and recommendation systems. More importantly, our research involves not only the study of software development, but the software developers themselves and their behavior. We investigate professional software developers and characterize concerns and micro-activities related to example usage.

In Section 2.1 we discuss social media influence on software development. In Section 2.2 we focus our discussion to Stack Overflow, a specific source of social media in software engineering. In Section 2.3 we present code search engines and recommendation systems. In Section 2.4 we discuss the study of professional developers. In Section 2.5 we discuss example usage among professional developers and the programming skills involved with example usage. In Section 2.6 we discuss software developers' concerns related to example usage.

2.1 Social Media in Software Engineering

Social media provides useful recommendations for many areas of our lives. For example, when considering what movies to watch, one may use recommendations from his or her immediate social cycle (e.g. Facebook friends), or use the wisdom of the crowd [52], using, for instance, the ratings on imdb.com. This is part of a more general trend in which social recommendations (e.g. Facebook) have begun to replace search (e.g. Google Search). The Software Engineering (SE) domain is no different; social media has been shown to be beneficial in many areas of SE including feature prioritization [2], risk analysis [51], collaborative filtering [23], knowledge management [25], and documentation [9] [54][40]. The current adoption of social media in processes and integrated development environments is just scratching the surface of what can be done by incorporating social media approaches and technologies into software development.

Storey et al. [49] discuss the impact of social media on software engineering practices and tools. They argue *how* and *why* social media will play an increasingly important role in software engineering research and practice. Treude *et al.*[55] discuss the opportunities and challenges for software developers that rely on web content curated by the crowd, and discuss the future of an industry where individual developers benefit from and contribute to a body of knowledge maintained by the crowd using social media.

2.2 Stack Overflow

Stack Overflow uses social mechanisms to facilitate knowledge exchange between users and to create an information archive. In Stack Overflow, a programmer can ask a question about almost any programming related topic, and receive a detailed response within 10 minutes median [35]. Answers on Stack Overflow often become a substitute for official product documentation, when the official documentation is sparse or currently non-existent[1]. Treude et al.[54] analyze data from Stack Overflow to categorize the kinds of questions that are asked, and to explore which questions are answered well and which ones remain unanswered. Their preliminary findings indicate that Q&A websites are particularly effective at code reviews and conceptual questions. The way Stack Overflow is designed allows each question and answer to be rated. Eventually for each question, the best answer is chosen to be "the accepted answer" for that question. In addition, members can edit each question and each answer to allow the information to constantly evolve and remain up to date. Finally, Stack Overflow has an enormous community of members, it is an already big knowledge base and it is constantly growing[2].

Parnin *et al.*[41] conduct an empirical study to investigate how Question and Answer (Q&A) websites, such as Stack Overflow, facilitate crowd documentation - knowledge that is written by many and read by many. They examine the crowd documentation for three popular APIs: Android, GWT, and the Java programming language. They collect usage data using Google Code Search, and analyze the coverage, quality, and dynamics of the Stack Overflow documentation for these APIs. They find that the crowd is capable of generating a rich source of content with code examples and discussion that is actively viewed and used by many more developers.

[1] https://stackoverflow.fogbugz.com/default.asp?W25450
[2] http://api.stackoverflow.com/1.1/usage/methods/stats

2.3 Code Recommendation Tools

Tools such as Strathcona [27] and PARSEWeb [53] provide developers with code fragment recommendations, taken from a central code repository, by generating queries based on code context and the structural details of the developer's activity. The quality of the code found by these tools is derived from the overall quality of the repositories they use. Code search engines, on the other hand, such as Krugle[3] and Koders[4], search in a large set of open source repositories, but do not provide explicit mechanisms to evaluate or improve the quality of the found snippets. Other tools like MICA [50], Exemplar [22] or [36] use API calls or API examples to recommend example code, but they are restricted to providing a limited set of examples based on the API only.

Brandt *et al.* propose [10] that embedding a task-specific search engine in the development environment can significantly reduce the cost of finding information and thus enable programmers to write better code more easily. They developed Blueprint, a Web search interface integrated into the Adobe Flex Builder development environment that helps users locate example code. Ponzanelli *et al.*[43] present a recommendation system in the form of a plugin for Eclipse IDE allowing automatic query generation based on keywords extracted from the code, mining Stack Overflow knowledge base, and displaying the result inside the IDE, thus minimizing context switching.

2.4 Study of Professional Developers

Sillito and Begel[47] studied software developers by using an interview and a diary study where developers shared their experience learning to develop Windows Phone applications. They characterize the learning strategies of the subjects as app-directed, and describe some of the particular challenges the subjects faced due to this strategy.

Miryung *et al.*[29] conducted a qualitative study in order to understand programmers' copy/paste programming practices and discover opportunities to assist common copy/paste usage patterns. They use observations and follow-up interviews to construct a taxonomy of copy/paste usage patterns.

Brandt *et al.*[11] describe two studies of how programmers use online resources. In one of the studies, conducted in the lab, they observed participants' Web use while building an online chat room. They found that programmers leverage online resources with a range of intentions: For just-in-time learning of new skills and approaches, to clarify and extend their existing knowledge, and to remind themselves of details deemed not worth remembering.

Latoza and Myers [32] discuss design of useful tools for software developers. They describe a design process intended to design useful tools for developers, tools that support software development work. They describe hierarchical decomposition of software development work into tasks through task analysis [14]. In each task at each level in the decomposition, developers have a goal, either a question to answer or something to accomplish. At the highest level tasks reflect activities, e.g. implementing features or refactoring. The activities can be further decomposed into sub-activities, e.g. understanding the code or debugging. At a lower level, developers formulate a specific plan for accomplishing a goal as a sequence of steps or a strategy.

[3] http://www.krugle.com/

[4] http://koders.com/

Useful tools are tools that support work by making a strategy faster or more successful. They advocate the use of exploratory study, involving interviews and observations, for evaluation of the design. They conclude that designing a useful tool requires more than finding a compelling motivating example, evaluating the tool's technical merits, and performing a carefully designed user study. Designing a useful tool requires understanding how a tool supports work and addresses an important problem that developers face.

LaToza *et al.*[33] investigates developers' typical tools, activities, and practices. Contrary to expectations that code duplication involves the copy and paste of code snippets, they found that developers reported several types of duplication. They used various research tools, a survey about activities, tools, and problems, a series of semi-structured interviews, and a follow-up survey of work practices. In our research, we use similar research tools to gather data, we use a survey to form our research questions and design. But the type of research questions we engage with, questions trying to understand developers' behavior, requires additional research tools. We use participant observations and structured interviews to investigate professional software developers' behavior related to example usage.

2.5 Example Usage

Programming by example was found to be intuitive to many developers, novices and experts alike [31]. Neal[38] presents an example-based programming approach. She implements an example-based programming environment, and conducts an experiment with students given a task to be solved with ability to use the example-based environment. She discusses the motivation for example-based programming, and reports on the results of the experiment to see how the system is used by programmers. Interestingly, she characterizes early signs of diversity in example usage. Barzilay[6] investigates example usage among professional software developers. He used various tools for data gathering that included field observations, interviews, surveys, reflective questionnaires, focus groups and virtual focus groups, to show different aspects of example usage diversity.

Nasehi *et al.*[37] conducted a qualitative analysis of the questions and answers posted to a Stack Overflow. By analyzing answers that were well-received, i.e. accepted answers or highly rated answers, they identified the characteristics of effective examples. They found that the explanations accompanying examples are as important as the examples themselves. We will investigate this further, by examining whether professional software developers use the additional context surrounding the example.

2.6 Concerns Related to Example Usage

Reimenschneider *et al.*[45] investigate why individual developers accept or resist methodologies deployed in order to improve software development processes. They conduct a field study with developers in a large organization that implemented a methodology, testing five theoretical models of individual intentions to accept information technology. They found, similarly to findings from the tool adoption context[19], that if a methodology is not regarded as useful by developers, its prospects for successful deployment may be severely undermined. But, in contrast to the typical pattern of findings in a tool context, they found that methodology adoption intentions are driven by: (1) the presence of an organizational mandate to use the methodology,

(2) the compatibility of the methodology with how developers perform their work, and (3) the opinions of developers' coworkers and supervisors toward using the methodology. In a similar manner, we are interested to investigate why developers accept or resist example usage, what concerns professional developers may have related to example usage when using social media. Furthermore, we examine the reasons developers choose a certain code example over another, and discuss the factors determining the acceptance of an example.

We are also interested whether limiting software development in an example driven manner is helpful. Janz *et al.*[28] conducted a study investigating how autonomy, interdependence, and team development were related to the effectiveness of teams of knowledge workers. Their results suggest that the positive relationship between team autonomy and team job motivation was reduced as teams worked under more interdependent conditions. This interaction effect also varied across the types of autonomy the team was given. In [15] the author suggests that although knowledge workers prefer autonomy, it doesn't mean they should always be given the maximum amount of it. Some efforts to improve knowledge worker performance may involve removing some discretion from the knowledge worker.

Chapter 3

Methodology

The type of research questions we engage with, require the choice of an appropriate research methodology. A research methodology that can deal with complexities that arises from not only technical issues in software development, but from human-machine interactions, and most importantly from human behavior in software development. Additionally, we are interested to investigate research questions with real practitioners, while collecting a broad set of data. We choose to employ qualitative research methodology with design-based research method, we carefully follow the methodology practices and guidelines to ensure the validity of the methods used. In the following we elaborate on the research methodology, the research course and the research tools used.

3.1 Research Questions

This thesis answers the following research questions:

RQ1: Is limiting software development in example driven manner helpful?

In economics and law fields, paternalism[18] is the interference of a state or an individual with a person or a group's liberty or autonomy for their own good. For example the compulsory wearing of seat-belts[13]. In knowledge worker studies[28] and knowledge worker management guides[15], we see that autonomy is important to knowledge workers, however, some efforts to improve knowledge worker performance may involve limiting their autonomy. Can a similar approach be applied by limiting software development in example driven manner? One would expect that limiting may hinder or slow developers, prevent them from using certain resources. On the other hand, limiting developers in example driven manner, especially in opportunistic development, where time is of the essence, may benefit them: they'll be more focused with the task at hand, use more examples and create quick prototypes. We divide participants of the user study to two groups and compare their results, participants who are limited to searching with a social media example code recommendation system, and participants who are not limited. Our study shows that limiting professional developers in example driven manner did not improve their results. We use observations to answer this question.

RQ2: How do professional developers mitigate concerns related to example usage?

Services ,such as Stack Overflow, Github, Sourceforge, allow developers to access vast amount of source code on-line. Furthermore, some of the available code is reviewed, debugged and patched by other developers and may be considered as a high quality source for code snippets. On the other hand, as shown in our survey (see Section 3.4.1), example usage is a popular practice among developers, yet developers have concerns involved with example usage. Developers may hesitate to use or admit they use code examples. We identify the concerns professional developers have when using examples and see how they mitigate them in opportunistic development.

RQ3: What are the micro-activities involved in opportunistic development when using social media based recommendation system?

We use observations and interviews to identify micro-activities involved in opportunistic development when using social media based recommendation system. Specifically, we focus on activities that mitigate the concerns involved with example usage.

RQ4: When searching for code examples, do developers refine their query or continue examining additional results?

When searching for examples, do developers use a single query to initiate a search and examine all results without refining their query, or do they work iteratively: form a query, initiate a search, examine results, refine query, and repeat the process until choosing a suitable example.

RQ5: How many code examples are examined before choosing a suitable code example?

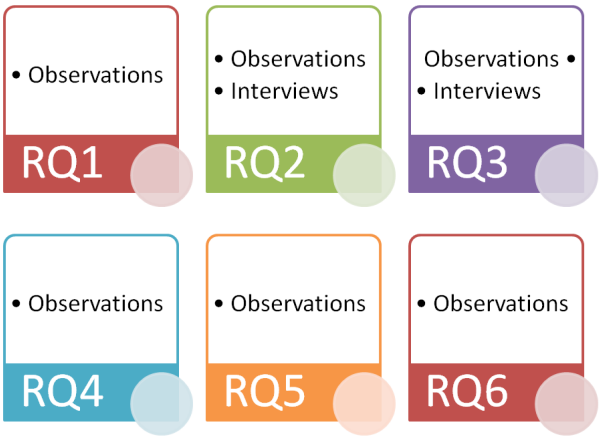
Granka *et al.*[21] conduct an eye tracking analysis for Google search users, and show that there is a drop in viewing time and number of clicks at the 6/7 ranked results, and a sharp drop occurs after result 10, as ten results are displayed per page. However, [21] investigated "regular" search with Google search, while we are interested with code example search based on social media recommendation systems. We speculate that a developer examines 3 code examples (in average) before choosing a suitable code example. Determining the average number of code examples examined by a developer before choosing a suitable code example, may influence the design of code recommendation tools, and specifically social media based recommendation tools.

RQ6: When searching for code examples, do developers use additional context?

If developers use or rely on additional context, it implies that offering a rich-verbose context in place, as an integral part of the code snippet, addresses a genuine need of the developers. "Complementary" tools that are lacking this functionality are missing an important requirement. If developers don't use additional context, it should be further investigated whether this additional context, though not used by many developers, is important - if so look for ways to encourage developers to use it. If not (not used, not important) - pragmatic tools such as Example Overflow could justify "hiding" additional context and offer clean code / improve code browsing.

Figure 3.1 illustrates what research tools are used to answer each research question.

Figure 3.1: Research tools used for each research question



3.2 Qualitative Research

Qualitative research aims to study complexities of human behavior (e.g. motivation), and the reasons for that behavior. Denzin & Lincoln[17] define qualitative research as: *A situated activity that locates the observer in the world. It consists of a set of interpretive, material practices that makes the world visible. These practices transform the world. They turn the world into a series of representations, including field notes, interviews, conversations, photographs, recordings, and memos to the self. At this level, qualitative research involves an interpretive, naturalistic approach to the world. This means that qualitative researchers study things in their natural settings, attempting to make sense of, or to interpret, phenomena in terms of the meanings people bring to them.*

Seaman[46] describes several qualitative methods for data collection and analysis, and describes how they might be incorporated into empirical studies of software engineering. She illustrates the use of qualitative methods with examples from real software engineering studies. In our research, we follow her data collection methods (participant observation, interviews), and data analysis methods.

Singer *et al.*[48] present work practice data of the daily activities of software engineers. They present four separate studies, discuss the advantages in considering work practices in designing tools for software engineers, and include some requirements for a tool they have developed as a result of their studies. Similarly, we conduct a study with professional developers, and look for practices and activities involved in example usage. Based on our findings, we design a social media based code recommendation system. Our aim is not the design of a tool or a recommendation system, but rather to investigate concerns involved in opportunistic software development with the use of the system we designed.

3.3 Design-Based Research

Design-based research(DBR)[3][57] is a type of research methodology commonly used by researchers in the Learning Sciences. It is one of several qualitative research methods. The goal of DBR is to design, create and study a single theoretically-inspired system or environment, as it systematically changed through multiple iterations, while simultaneously testing the validity of a dominant theory or generating new theories. The definition of DBR proposed by Wang and Hannafin[56]: *A systematic but flexible methodology aimed to improve educational practices through iterative analysis, design, development, and implementation, based on collaboration among researchers and practitioners in real-world settings, and leading to contextually-sensitive design principles and theories*.

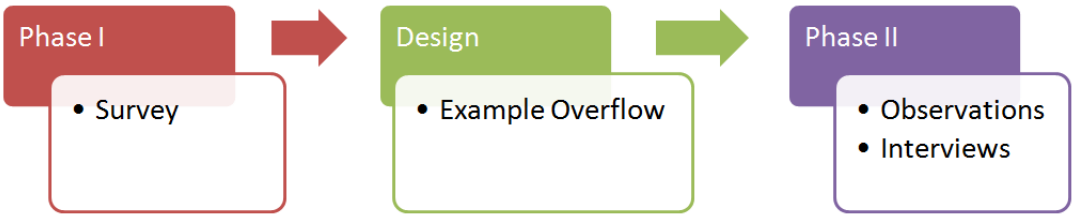
Anderson *et al.*[1] review the characteristics of DBR and analyze the five most cited DBR articles from each year of this past decade. They illustrate the context, publications, and most popular interventions utilized. They conclude that interest in DBR is increasing and their results provide limited evidence for guarded optimism that the methodology is meeting its promised benefits. Dede[16] discusses why DBR is both *important* and *hard*. He advocates that contrary to traditional research methods, in DBR studies many variables are deliberately and appropriately not controlled, the "treatment" may evolve considerably over time, and even the research methodologies utilized may shift to fit the morphing intervention. Further, to aid with interpretation under these difficult circumstances, in DBR large qualitative and quantitative datasets of various types are often collected by many different participants, introducing substantial problems of alignment, coordination, and analysis.

Reeves *et al.*[44] employ design-based research to explore the various incentives for conducting research on the impact of computing and other technologies in higher education, examine the social relevance of that research, and recommend DBR as a particularly appropriate approach to a socially responsible inquiry. Barab *et al.*[4] designed the Quest Atlantis (QA) project, a learning and teaching project that employs a multiuser, virtual environment to immerse children, ages 9–12, in educational tasks. It allows users at participating elementary schools and after-school centers to travel through virtual spaces to perform educational activities, talk with other users and mentors, and build virtual personae. They studied the impact of QA on learning, and showed that, when responding to personal narratives, students participating in QA offered character insights that were either deeper or better supported than did students in equivalent conditions; additionally, elementary students who used QA demonstrated statistically significant learning over time in the areas of science, social studies and sense of academic efficacy.

In our research, we follow the DBR guidelines: we use real-world practitioners, professional software developers, to form a design of a social media based recommendation system. Based on that design, we conduct a study among professional developers investigating opportunistic software development, involving participant observations and interviews, in each examining multiple dependent variables, and looking at multiple aspects of the design in practice.

3.4 Research Course

Figure 3.2: Research course



The research process can be divided into 2 phases, as shown in Figure 3.2. In phase I, we used an online survey to gain insights on activities involved in example usage among professional developers. We use our findings to design and implement, Example Overflow (EO), a code recommendation tool to based on social media. The design decisions for EO take into account the survey data of professional developers regarding example usage. In phase II, we use the social media based code recommendation system we designed, Example Overflow, with qualitative data gathering tools: participant observations, and interviews, to investigate opportunistic software development among professional developers.

3.4.1 Phase I: Example Usage Among Professional Developers Survey

To gain insights, we have conducted an online survey [42] among professional developers. The survey (available online[1]) contained no mandatory questions and was anonymous. Each subject was asked some professional details: years of experience as a developer, size of the company they work for, and what technologies they are using in their work. Following were seven questions regarding example usage, where we defined example usage as any use of an already existing code in the development process. The survey questions and activities mentioned here are based on [6]. The questions are:

- How often do you look for examples in your work? (All the time, Every few minutes, Once an hour, Once a day, Once a week, Less than once a week, I do not use examples in my work, or Other).
- For what kind of programming tasks do you use examples? (check all applicable options: Tasks involved with unfamiliar or new technology (API, programming language, platform, etc.), I have a set of specific tasks for which I use examples regularly, Complicated tasks (lot's of details and corner cases), Common tasks (popular API, standard operations), When starting a new task (new feature, new project, "hello world"), or Other).
- During what kind of activities do you use examples? (check all applicable options: Learning, Implementation, Problem solving, Design, Comprehension, Self improvement, Other).

[1] https://docs.google.com/spreadsheet/viewform?formkey=dFV2ZWhycGdFLUtjVVIyR1VMNUxhQnc6MQ#gid=0

Figure 3.3: Screen capture of part of the online survey

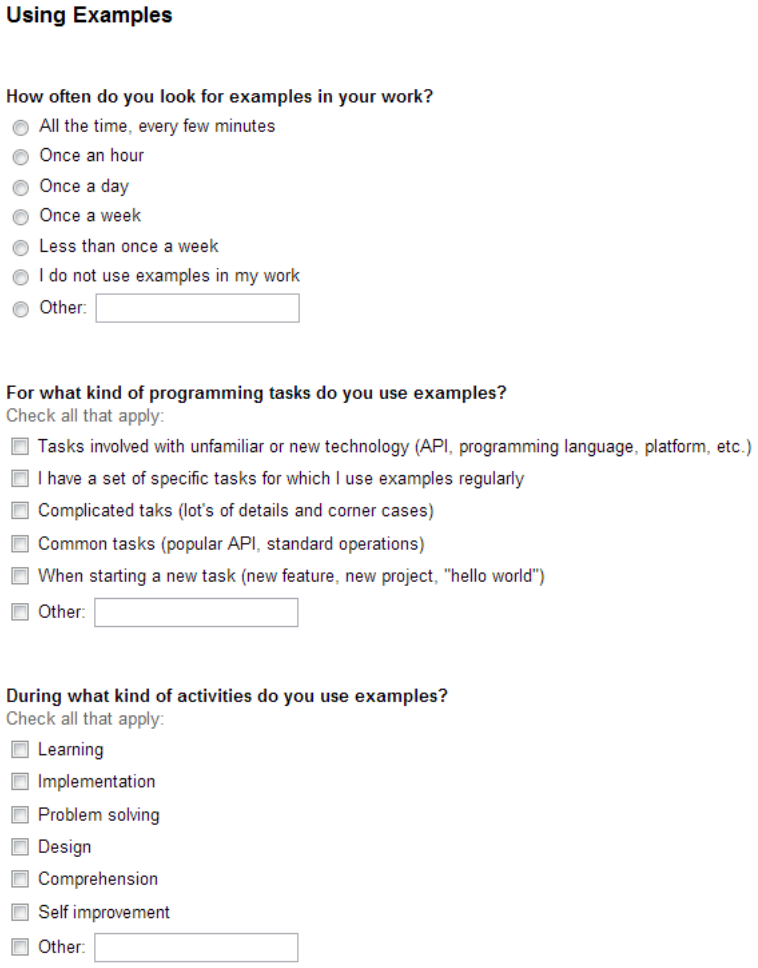
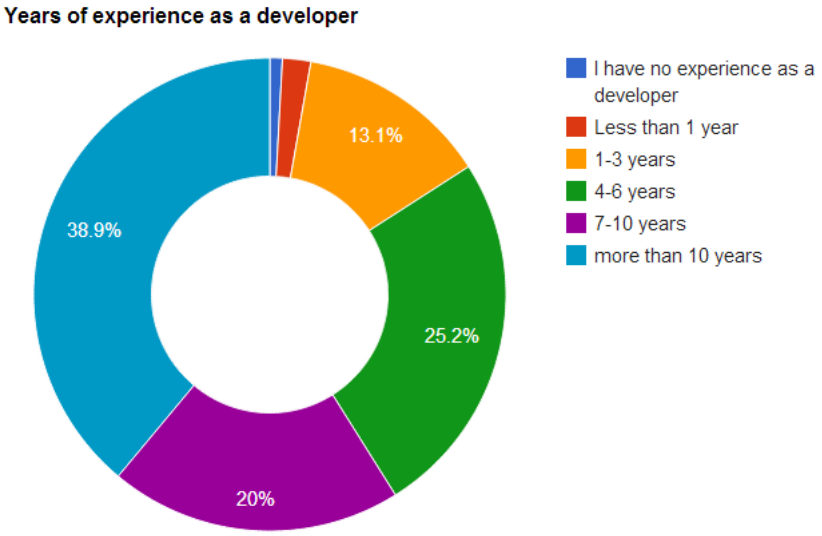


Figure 3.4: Professional developers survey: years of experience as a developer



- I am using examples in order to find... (check all applicable options: API usage, Feature implementation, Tool / framework / environment usage, Language syntax, Design decision, Algorithm, Inspiration / alternative ideas, Feedback about my own ideas, Programming techniques / coding conventions / style, Other).

- How many different activities do you think are involved in example usage? (choose a number from a given list of 1-20+).

- When you use examples, which of the following activities do you perform? (check all applicable options: Browse, Comment, Learn, Modify, Refactor, Run, Search, Share, Test, Understand, Wrap, Write). For each option check one of the following:

  - Yes, I do that
  - I don't do it, but I think it is important
  - Now that you mention it, I should do it more
  - Not relevant for example usage

- Do you use other activities not mentioned above? or have any additional comments? (open ended question).

The survey was published online using Google Forms technology[2]. We have solicited professional developers' participations by posting invitation in social and professional networks, such as LinkedIn[3], Facebook[4], and DZone[5]. In the period of 14 months over 480 forms were

[2]https://www.youtube.com/watch?v=IzgaUOW6GIs
[3]http://www.linkedin.com/
[4]https://www.facebook.com/
[5]http://www.dzone.com/

Table 3.1: How often do you look for examples in your work?

Frequency	No. of participants
All the time, every few minutes	33
Once an hour	90
Once a day	178
Once a week	98
Less than once a week	40
I do not use examples in my work	0
Other	26

submitted, and 465 submitted forms with at least one question answered (apart from the personal details). In 46 of them, the last (open ended) question, regarding additional activities used, was answered.

More than half of the survey participants have 7 years of experience or more (See Figure 3.4). Most of the participants, as it can be seen in Table 3.1, use examples at least once a day.

Our discussion regarding the survey is limited, as we use its result to derive the design and design decisions for a social media based code recommendation tool (as described in chapter 4). This is out of the scope of this thesis, a thorough analysis will be presented elsewhere.

3.4.2 Phase II: User Study

User Study Design

We designed our user study with the purpose of examining the validity of our research questions while following the guidelines in [46]. With that in mind we have chosen the following main use case: a professional developer is required to accomplish a set of coding tasks in an unfamiliar domain while working in opportunistic development manner. Our use case is not uncommon, on the contrary, web and mobile developers face this problem on a daily basis, as new programming languages or frameworks emerge, and developers can't master them all.

We decided to give programming tasks to our subjects to be accomplished in a predefined time frame, and to divide our subjects into two groups: (1) developers who are limited to searching in Example Overflow (but allowed to follow external links), and (2) developers who are not limited to using a specific tool or to using example code at all. In order to allow opportunistic programming we decided on a time frame for each task, but to avoid discouraging developers from participating we have limited ourselves to total time frame of 1 hour (or less) for the task solving phase.

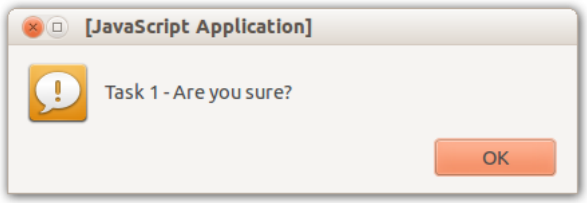
Tasks

Based on the main use case and the planned time frame we designed 4 tasks for participants to work on. The tasks are based on the jQuery domain, which we have chosen for Example Overflow. We visited various programming forums, Q&A web sites and jQuery related groups to look for common tasks a jQuery developer may be required to accomplish. In addition, we have approached the jQuery community on Linkedin and asked for suggestion of tasks. We have chosen the tasks while making sure they come from real world problems or questions of

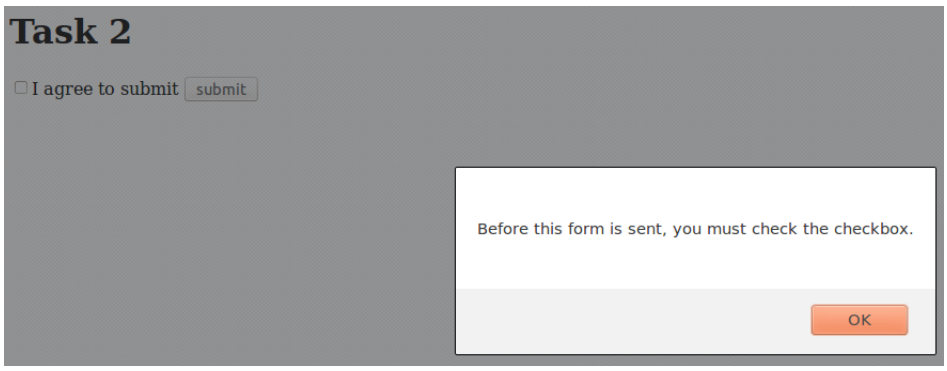
real jQuery developers, with priority to common tasks, and tasks that are not related to each other. Each task was solved by the author and measured for time in order to determine the time frame. The determined time frame for each task factors for developers who are unfamiliar with jQuery or JavaScript. The first three tasks were given 10 minutes and the fourth task 15 minutes for completion.

Following is the task description as it was given to the participants:

Task 1[6] *Write code that checks whether a user is leaving the current window (by closing the tab/window, going to a different url, pressing back or forward in the browser). If so, before leaving the window show a popup saying "Task 1 - Are you sure?"*



Task 2[7] *You are given with basic code that creates an unchecked checkbox and a submit button. You need to complete the code that will check if the checkbox is checked before submission. If the checkbox is unchecked, pop up an alert message saying "Before this form is sent, you must check the checkbox.", and prevent the submission. Otherwise, pop up an alert saying "form submitted".*

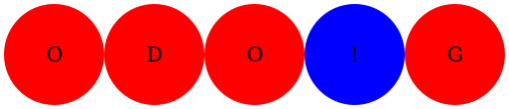


Task 3[8] *You are given with code that creates 5 bubbles on the screen with characters in them (D,G,O,O,!). The characters represent a game where the goal for a player is to line-up the letters so that it says "GOOD!". The game will allow the user to click on a bubble, thus moving that letter to the beginning (the most left side). You need to complete the code in order to make it work.*

[6]This task was chosen from the top of the official jQuery forum sorted by the "Most Voted" questions.

[7]This task was chosen from a blog post listing code snippets for common tasks in jQuery (http://www.joshuawinn.com/quick-jquery-code-snippets-for-common-tasks/).

[8]This task was chosen from a jQuery challenge (http://cfg.good.is/challenges/javascript-user-experience-with-jquery).



Task 4[9] *Create a text box that is able to auto suggest the names of a few major cities in the world. For simplicity you should use the local variable "cities" as your source for city names. Notice that the textbox needs to show only city names starting with the given input letters.*

Developer Recruiting

We have recruited 10 professional software developers to participate voluntarily in the user study. The participants were offered a token of appreciation for participation. We approached professional developers with at least 2 years of experience, and for this reason students were not considered for the user study. In fact the average years of experience among the participants was 7.05.

Data Collection

The user study began with a brief general questionnaire, followed by a short JavaScript & jQuery tutorial (all user study files are available online[10]). The tutorial includes 5 basic examples showing an HTML structure, JavaScript event, and a simple example of a single widget, the DatePicker, from jQuery.

Before each task, each participant was required to read the task instructions. Then it was followed by showing a working demo of the required feature and allowing time for questions if the task was unclear. It was explained to the participant that there will be no option to ask questions during the task and no ability to view the tutorial files.

The user study was both observed in real time and recorded, thus giving us the ability to analyze how a developer searches for code examples, how many code examples will they browse before refining the query, what parts of a code example and its surrounding context are they looking at, and other important information. The recording was capturing the participants screen in the exact way the participant saw the screen.

Nonetheless, capturing the screen and observing the participant is not enough, because we wanted to know exactly what parts of the screen the participant was looking at, and whether he or she are more interested in code examples or the surrounding context such as documentation, Q&A text, or comments. To overcome this, as suggested in the gidelines[46], participants were asked to use *Think Aloud protocol*[20], where they were asked to verbalize what were they doing and on which parts of the screen they were looking at. The audio was also recorded as part of the video capture of the screen.

During the observations field notes were taken, documenting the participants actions, as follows: for each action we documented the time, the query used and the action itself. We also wrote anything the participant may have said during the task, even though audio was recorded as well. Note that we, the observers, refrained from talking during the task. Figures 3.5 and 3.6 show an example of a field note used to collect data during observation of participant 4. To keep up with the participant, we would use abbreviations in our notes, i.e. instead of writing "the

[9] This task was chosen based on the official jQuery documentation.

[10] http://www.cs.tau.ac.il/~alexeyza/example/user_study_files.zip

participant uses Google to search for jquery element" we would write "google jquery element". In addition, the field notes were taken in Hebrew, the native language of the observer and of the participants.

At the end of the user study, participants were interviewed, and asked to reflect on how they worked during the tasks. The interview was recorded (audio) as well. Following are the interview questions:

1. What do you think of this experiment?
2. Did you use any particular strategy for solving the tasks? For all tasks? Why?
3. Were you familiar with Stack Overflow prior to this user study? Do you use code examples in your work ? Why ? If yes, from what source?
4. What difficulties did you have when you searched for code examples?
5. How did you choose your query?
6. Did you refine your query while searching? In what cases? If so how? Why/Why not?
7. Why did you choose that code snippet (e.g. it was first? Better compared to others)?
8. If you compared several code snippets, please explain how you compare between 2 code examples.
9. How many code snippets do you think you've examined (not browsed) in average before choosing?
10. Was it easy or difficult for you to find the code snippet you needed? Why?
11. Did you use only the code snippet itself to accomplish the tasks (without additional surrounding context such as comments, documentation, Q&A text) ? Why/Why not?
12. Would you like to add something that you did not mention? (share insights)

Research Participants

Of the 10 participants, 6 were male and 4 female. The average years of experience is 7.05. The most popular programming language a participant is familiar with was Java with 8 out of 10 participants. Only 4 of the participants were familiar with JavaScript, and only 1 was familiar with jQuery.

Participants were randomly assigned a group: (1) participants who were limited to search for examples at Example Overflow (EO) only, but were allowed to follow any external link from within the EO results. (2) participants who were not limited and were allowed to use any search tool they wanted. Participants 1, 3, 5, 7, and 10 were assigned to group (1), while participants 2,4,6,8, and 9 were assigned group (2). We have noticed that all the participants from group (2) have used Google search.

In the interview when asked about example usage at work, all 10 participants have said they were using code examples at work.

Study Limitations

Our user study was limited to professional developers (no novice developers) and focused on opportunistic development in an unfamiliar domain. However, the participants had some variance with the programming languages they are familiar with, and their past experience. Four out of ten participants were familiar with JavaScript, and one participant was familiar with jQuery.

The interviews were conducted in Hebrew, the participants native language, and translated to English by the author.

Figure 3.5: Form used to collect data during observation of participant 4, task 3

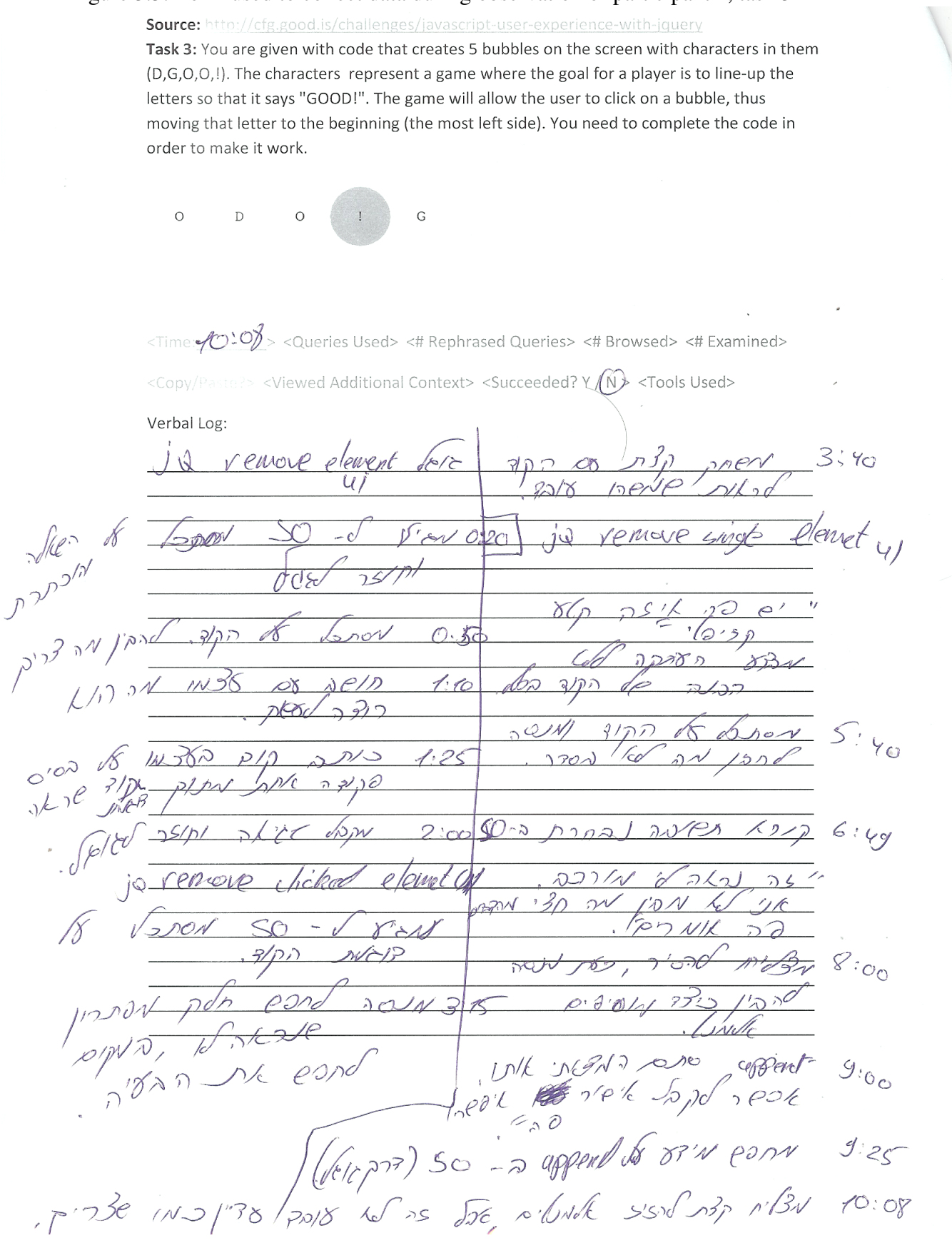


Figure 3.6: Form used to collect data during observation of participant 4, task 4

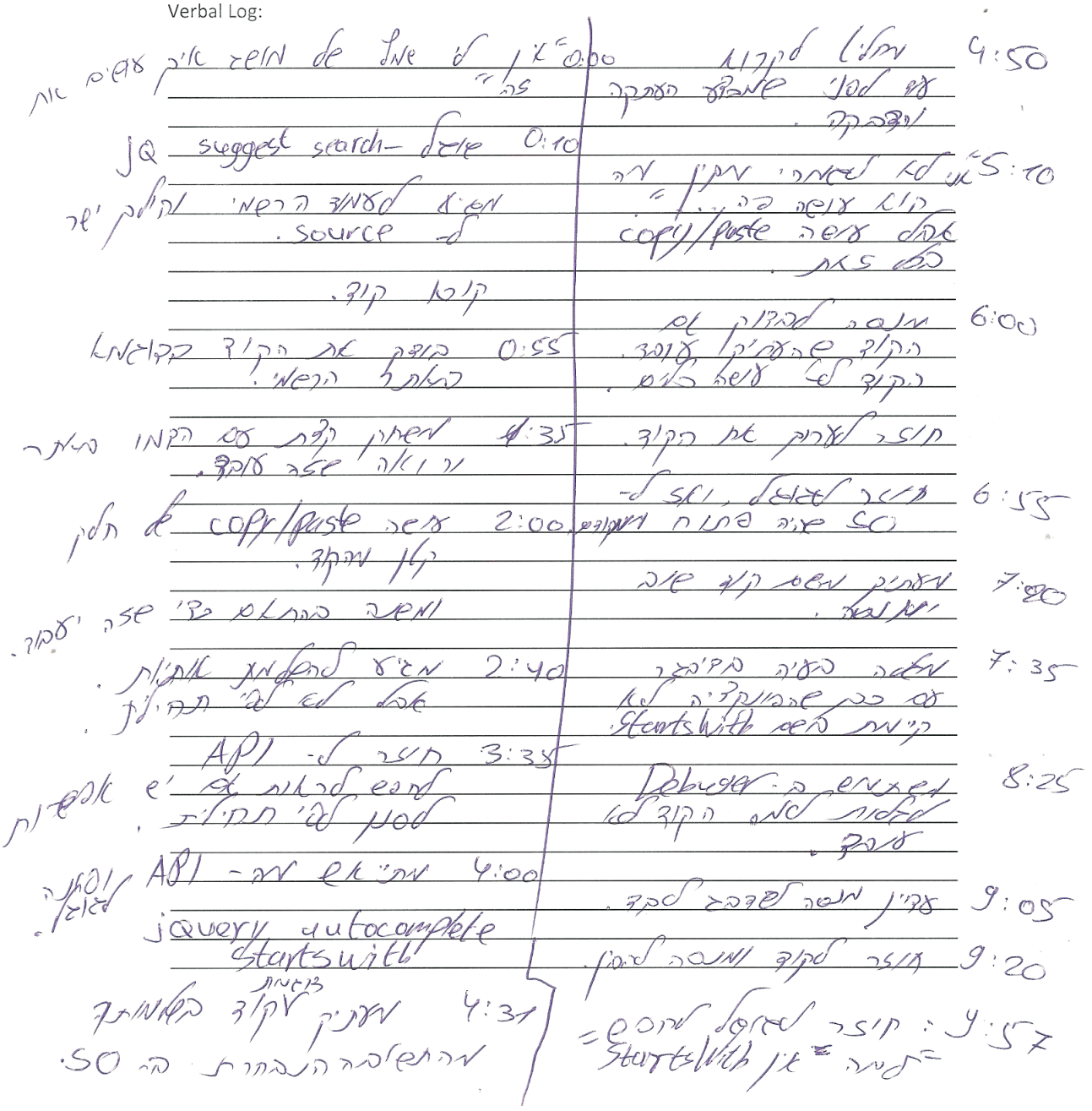
Source: jQuery official documentation.

Task 4: Create a text box that is able to auto suggest the names of a few major cities in the world. For simplicity you should use the local variable "cities" as your source for city names. Notice that the textbox needs to show only city names starting with the given input letters.

<Time 12:45> <Queries Used> <# Rephrased Queries> <# Browsed> <# Examined>

<Copy/Paste?> <Viewed Additional Context> <Succeeded? Y / N > <Tools Used>

Verbal Log:



3.5 Qualitative Data Analysis

Qualitative data analysis [30] has several stages: (1) transcription of the data recorded, (2) organizing the data into easily retrievable sections, (3) familiarization with the data by reading and re-reading the data, making memos and summaries, (4) reading the data and labeling segments, i.e. coding, and (5) identifying themes or emergent concepts, and engaging in re-coding to develop more well defined categories.

We analyze the data gathered in the participant observations and interviews in 2 phases, while following the analysis stages mentioned above. In phase I the qualitative data was transcribed, organized in a single data file, and coded. At the coding stage, we identified several concerns and micro-activities involved in example usage. In phase II we continued to analyze the data, identifying further concerns, micro-activities and the connection between them. We have repeated phase II iteratively, involving the coding stage and the identifying themes stage until we refined our findings. As stated in the qualitative analysis guidelines[46], one of the most important ways to help confirm a qualitatively generated proposition is to ensure the validity of the methods used to generate it. In the following we address the challenges in maintaining objectivity in this research on how we address them.

3.5.1 Phase I: Coding

We began the initial analysis with forming a data analysis form (see Table 3.2). We replayed the recordings of each participant, while looking at the field notes taken during the observations, and filled the data analysis form. This required multiple replays as it was hard to capture all the data with a single replay. The interviews were transcribed and all the data was saved electronically in a single data file.

The user study (observation, interview) was conducted by a single researcher (the author of this thesis), as part of our measures to ensure ensure that those being observed are not constantly thinking about being observed[46]. This is to help ensure that the observed behavior is "normal". This might challenge the objectivity of the research findings by introducing bias, or preconceptions. We address this with the following measures:

- The observations and interviews were recorded (audio,screen capture) and allow other researchers to replay the recordings, thus preserving the raw data from being influenced by bias or agenda.
- The field notes (observation and interview notes) were written in real-time during the user study, before the focus of the research was determined. This ensured that the researcher would interpret the data objectively.
- The analysis was conducted in iterations between the author and the research supervisors: reviewing the results, raising questions and reservations and reexamining the study recordings.

3.5.2 Phase II: Identifying Themes

In phase I we identified several concerns and micro-activities involved with example usage, in phase II we focus on looking for supportive evidence, while refining and modifying the

Table 3.2: Initial data analysis example: table for phase I analysis of participant 4, task 3

Task 3	Data
All Queries used by participant	jquery remove element ul, jquery remove clicked element ui, jquery remove single element UL, jquery append to ul
How many queries used?	4
How many results browsed (per query)?	1,2,5,1
How many results examined (per query)?	1,1,1,1
How many queries rephrased?	3
How many snippets browsed or examined before rephrasing the first query?	1
How many examined (not browsed) before the first copy/paste of a code example?	1
How many times viewed additional context?	3
How many times gave up on a code example because it looked too long?	0
Was he/she able to find and copy/paste a "good" code example?	no
If yes to above, what query and result rank was used?	-
Actual time (max = 0:10:00)	0:10:00
Did he/she succeed in task (part 1 - move any one of the bubbles to any side, i.e. left or right)?	yes
Did he/she succeed in task (part 2 - move the clicked bubble to the left most side)?	no
Tools used directly	Google search (with built-in auto-complete)
Tools used indirectly	stack overflow
Were the examples used as a reference only or as copy/paste snippets?	code snippet
If used Google search, did rely on auto-complete ?	yes

list of concerns and micro-activities. We differentiate this phase from phase I, because it is repeated iteratively until we refined the list of concerns and micro-activities. The end result is a proposition that insightfully and richly describes how developers mitigate concerns involved in example usage.

We conducted this step in a similar manner to the analysis in phase I, we replayed the recorded observations and interviews while looking on the field notes as well. Table 3.4 shows the data analysis form used in phase II.

Analysis Example

We'll use the field note of participant 4 from task 3 (see Figure 3.5) to show as an analysis example. Note that this is an example of the analysis of the field note alone, and is only a partial analysis that may miss some of the context. We applied a similar analysis to the observation recordings and interviews and to give the full context needed for the analysis. Table 3.3 shows the field note transcribed and translated to English. Table 3.4 shows an example analysis of the raw data from Table 3.3.

Table 3.3: Data analysis example: transcribed data form of participant 4, task 3

0:00	Searches with Google the query “jquery remove element ui”
0:20	The result points him to a Stack Overflow question, participant looks on the question and it’s title. Returns to Google.
0:50	Looks at his source code, to understand what needs to be done.
1:10	Thinks to himself what he wants to do.
1:25	Writes code manually based on a single keyword he saw in an example.
2:00	He receives an error, and returns to search in Google with query “jquery remove clicked element ui”. The result brings him to Stack Overflow, and he examines the code example there.
3:15	Tries to search with keywords based on a possible solution he has in mind, instead of searching for the problem (keywords based on the task).
3:40	Plays with his own code just to make something work. Searches Google with “jquery remove single element ui”. Says out-loud “There is some cryptic thing here”. Copies the example without understanding the example code at all.
5:40	Looks at the his code (with the previously example embedded) and tries to understand what is wrong.
6:49	Reads an accepted answer from a question on Stack Overflow. Says out-loud “It looks complicated to me. I don’t understand the meaning of half of the things here”.
8:00	Successfully able to remove UI elements (as part of the required task). Now tries to understand how to add elements.
9:00	Wants to use the “append” keyword. Says out-loud while starting to search Google - “I just made it up (the append keyword), can I have some confirmation here”.
9:25	Looks for information on the “append” keyword on Stack Overflow (by using Google to search).
10:08	Able to move some elements on the screen (as required in the task), but it doesn’t fully accomplish the task.

Table 3.4: Data analysis example: analysis of the data from Table 3.3

Time	Raw Data	Micro-activity Identified	Concern(s) mitigated
0:50	Looks at his code to understand what needs to be done.	Task comprehension	Confidence in the example
1:25	Writes code manually based on a single keyword from example.	Diversity in using the example	Confidence in the example
3:15	Tries to search with keywords based on a possible solution.	Forming and refining the query	Confidence in the example
3:40	Forms a query based on the task.	Forming and refining the query	Lack of knowledge
3:40	Copies the example without understanding.	Diversity in using the example	Confidence in the example, Time
6:49	Reads an accepted answer from a question on Stack Overflow.	Reading additional context	Confidence in the example

Chapter 4

Example Overflow

In order to conduct our research, as part of designed based research, we designed and implemented a Social Media based Code Recommendation System (SMCRS), Example Overflow, that allowed us to test and analyze different design decisions of social media based recommendation systems.

In this chapter we describe our design, design decisions and implementation of a SMCRS, and our initial benchmark.

4.1 Social Media Based Recommendation System Design Decisions

In order to implement a crowd sourced software recommendation system, one needs to explicitly foresee the division of labor between the developer and the machine. The system should facilitate the core practices[50] involved in Example Embedding[7][5] namely enable browsing and comparing multiple code examples and reducing the developer's context switch as elaborated below.

4.1.1 Comparing Multiple Examples

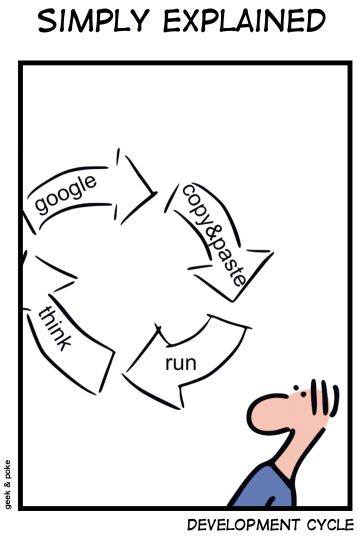
Professional developers, participants of the example usage survey from section 3.4.1, reported browsing multiple examples, comparing them side by side, and eventually choosing the examples most suitable for the developer's needs (sometimes merging multiple examples).

Subject 435 answered *"Compare a few different examples with one another to find the best"* in his answer to the open ended question "(When using examples) Do you use other activities not mentioned above?". Subject 444 answered *"Sometimes I compare different code examples to get a better understanding of the feature. Then I refactor and modify the code to fit my needs and implement it in my own code, sometimes as part of a library (wrapping)."* Subjects 316 and 467 mention *"Compare"* as one of the additional activities not mentioned in the survey. Subject 77 explains the advantage of multiple example authors *"Get a general idea about technology used in example; Multiple example authors can give you a more detailed explanation about the technology used in example"*. Subject 143 answered *"Many times the results of a search includes many examples that fit in terms of programming language and license, so choosing the "best" one to try to reuse is a very important task. Searching is easy but I think making the*

selection is one of the more difficult tasks. Maybe you could separate it from the more general "browse" task or split the browse task into (1) search - deciding how to define the search query - sometimes this is iterative and includes refinements (2) selection of most suitable example to use among all relevant ones returned by the search".

This also conforms to the literature suggesting that it is easy to extract the repetitive example structure from a specific context, and to reuse the repetitive part for new tasks [39][26]. Traditional code search tools (e.g. Google Code Search, Krugle) allow searching for code, where a developer inputs a query and then he or she is displayed with the search results consisting of the filename or the first few lines of the source code. The developer is then forced to click on each result, open it in a new view, inspect it separately and decide whether it is the best example to be found. Using these tools, there is no way for the developer to compare the current code example with the ones viewed previously or the one to be inspected next. We believe the ideal solution should allow the developer to be able to browse and compare easily and quickly through the top results. This will allow the developer to find the most suitable code snippet or realize as soon as possible he or she is searching in the wrong direction. In order to achieve this the developer should be able to compare the few top results in the same view, without the need to change views in the working environment. This can be further enhanced with visual comparison tools such as diff[1].

4.1.2 Reducing Context Switching



We believe that modern software development and web search are part of the same activity, thus should be conducted in the same context. The developer should not be in a different *state-of-mind* when coding and when searching for code examples. The lack of IDE support for built-in web search, forces developers to work directly with web search tools, such as Google

[1] http://en.wikipedia.org/wiki/Diff

search, but these tools aren't intended solely for software developers, and create unnecessary distractions.

Barzilay *et al.*[7] argue that example search is an integral part of modern software development (Example Embedding Ecosystem [5]). Ponzanelli *et al.*[43] and Brandt *et al.* [10] support this approach as well, by allowing developers to search for code examples from within the IDE.

We aim to allow the developer to find example code with minimal context switching as possible, ideally without leaving the IDE.

4.2 Example Overflow

In this section we describe our implementation of a social media based code recommendation system, Example Overflow. Example Overflow is a live system, and is currently deployed as a public and free website[2]. Our implementation contains all code snippets that appear in accepted jQuery related answers (more than 47,000 code snippets) on Stack Overflow[3]. jQuery[4] is a popular JavaScript library, initially released in 2006 and is ranked sixth in its popularity on Stack Overflow (with over 330,000 related questions). We chose it as our case study since we assume that Web developers would find it easier to adopt an example centric programming approach. Our decision is also supported by the following: (1) Parnin and Truede [40] found that Stack Overflow covers 84.4% of the jQuery API, and (2) Our study shows that 20% of the jQuery related questions have a code snippet embedded in their accepted answer. Example Overflow is developed as part of a comprehensive effort to create an Example Embedding Ecosystem [5] – an example centric development method in which example related concerns are weaved in the development process, software tools, practices, training, organization culture and more.

4.2.1 Following Our Design Decisions

To support comparison of multiple code examples, when searching in Example Overflow the developer is presented with the code of the 5 most suitable results. We base our decision on [21], they conduct an eye tracking analysis for Google search users, and show that there is a drop in viewing time and number of clicks at the 6/7 ranked results. They give a possible explanation of the fact that typically only the first 5-6 links were visible without scrolling. A sharp drop occurs after result 10, as ten results are displayed per page. Our preliminary benchmark also supports this decision as can be seen in section 4.3. This allows the developer to see all the code examples in the same view, where they are not isolated from each other, compare them and choose the one that suites him or her best. If none of the results are suitable, then automatically the next 5 most suitable code examples are displayed as well. This way the developer will be presented with the minimum amount of code examples that are needed to find the most suited one(s).

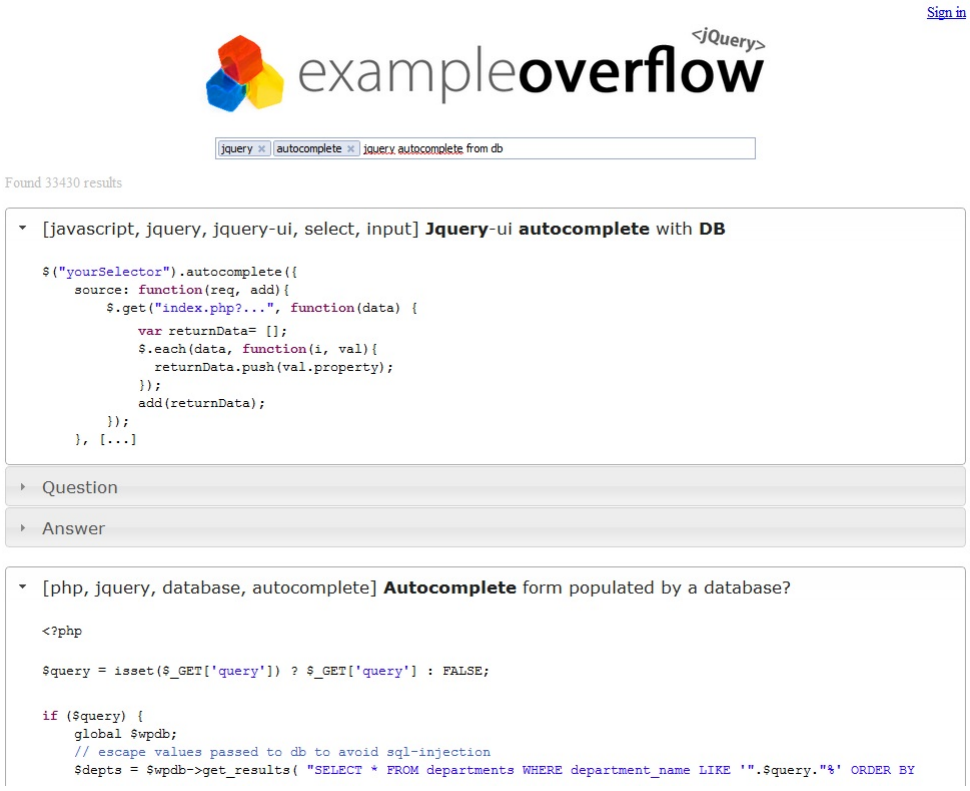
To support reduction of context switching, our design has a single search window, as can be seen in Figure 4.1. The developer is presented with the most relevant search results, where each result shows only the code snippet itself, thus allowing the developer to see all the code

[2]http://www.exampleoverflow.net/

[3]http://stackoverflow.com/

[4]http://jquery.com/

Figure 4.1: Example Overflow screenshot



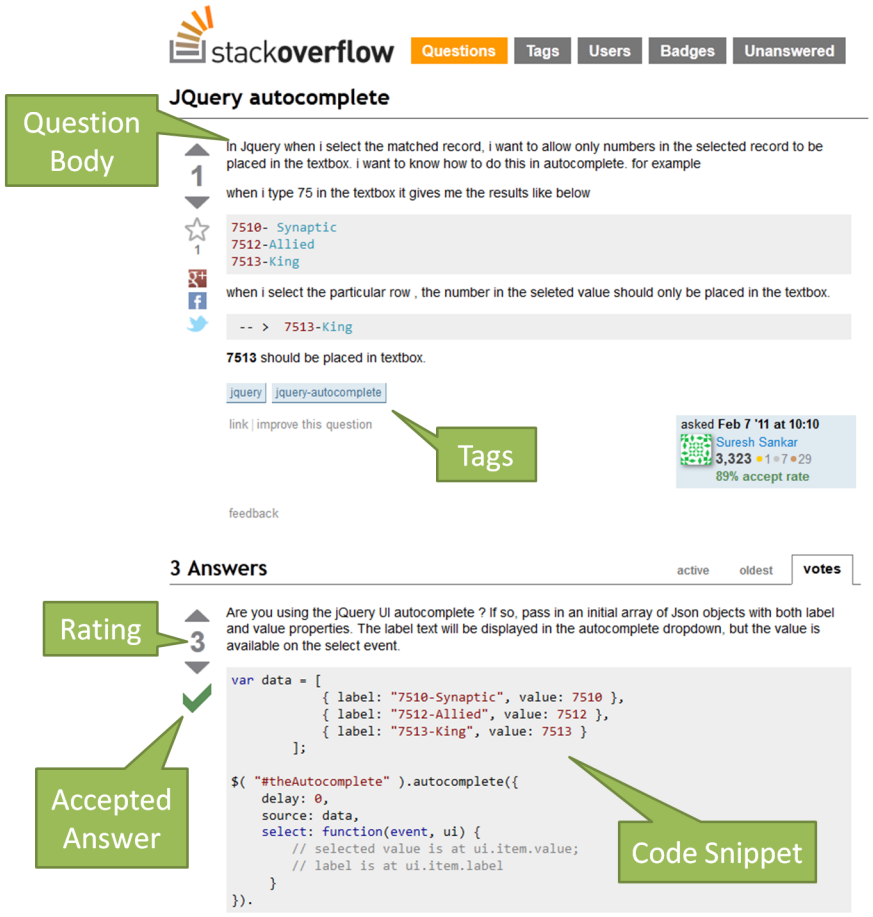
snippets at the same view without opening new views. If the developer needs more context for the code snippet, all he or she has to do is hover (without even clicking) over the example with the mouse, and choose either "Question" or "Answer" to see that context inside the same view.

4.2.2 Populating the Repository

We use Stack Overflow's API[5] to request all the questions relevant to our current domain, jQuery tagged questions[6], where we filter out all the questions without an accepted answer. We follow a conservative approach by choosing only accepted answers to ensure retrieval of high quality results. The next step is to check whether each of these questions has a code snippet inside the accepted answer. If so, that code snippet is extracted and saved to our database with all the accompanying information: the question title, the question body, the answer body, the code snippet itself, the user rating of the answer from Stack Overflow, the view count of the question, the tags associated with the question and other relevant information (see Figure 4.2). If that question is already in our database we only update the changed information. This process can be executed as a scheduled task to allow us to keep the data in sync with the data at Stack Overflow.

[5]https://api.stackexchange.com/
[6]http://stackoverflow.com/questions/tagged/jquery

Figure 4.2: Stack Overflow user interface analysis



4.2.3 Searching

Example Overflow uses keyword search based on the Apache Lucene [24] library, which internally uses the term frequency-inverse document frequency (tf-idf) weight [58]. In order for Apache Lucene to search, one needs to define which parameters are to be analyzed and indexed. For keyword search index we use both the code snippet and the additional metadata which accompanied the code snippet at Stack Overflow. This allows a developer to find code snippets that may not contain the search query keyword, but the keyword appears in the contextual data and indicates that it has been used in that context. Each code example is represented as a document with several parts: title, tag, answer, question, code, and social metadata. We use the following formula to calculate the score of each document representing a code example:

$$S_{doc} = [W_{title}S_{title} + W_{tag}S_{tag} + W_{answer}S_{answer} + W_{question}S_{question} + W_{code}S_{code}]\, S_{metadata} \tag{4.1}$$

Where each S_{part} represents the individual score of the respective part of the document, and W_{part} represents the weights that may be chosen to tune the tool for the best results possible. The weights would be computed based on a set of experiments, but for the initial benchmark presented here, they were chosen heuristically to give higher priority to results with matching keywords in the title or tag, over matches in the other parts, and are $W_{title} = 4$, $W_{tag} = 4$, $W_{answer} = 1$, $W_{question} = 1$, $W_{code} = 2$.

4.3 Preliminary Benchmark

As a preliminary benchmark, we used a jQuery benchmark to compare the characteristics of Example Overflow with other existing code recommendation systems, as elaborated below.

4.3.1 Benchmark Setup

We used the code assignments from the book jQuery in Action [8] to define a benchmark of ten frequent programming tasks shown in Table 4.1. For each task we have manually decided on a concise query to be used by a potential developer in order to find the desired code snippet. We have used the same query in each of the following tools, and have examined the first 20 results returned for each query.

We used the following existing tools in the evaluation: Google Search, Stack Overflow, Krugle, and Koders. We also used Google Code Search in our preliminary benchmark, where it had similar results to Krugle, but this service has been shut down by Google. We have not included Strathcona [27], Blueprint [10] or PARSEWeb [53] in the benchmark, because they are domain specific and would not work for the jQuery domain. We have not used SEAHawk[43] as it was not available at the time of the benchmark.

4.3.2 Benchmark Methodology

For each query and each tool we have received a list of results. These results were manually examined by the author (see section 4.3.4). We have used the actual code from the book jQuery in Action as a point of reference.

Table 4.1: Search queries used for the evaluation benchmark

Data Point	Search Query
Dynamic Dimension	“jquery dynamic dimension”
Hover	“jquery hover div”
Position	“jquery position”
Rounded Corners	“jquery rounded corner”
Draggable	“jquery draggable”
Droppable	“jquery droppable”
Autocomplete	“jquery autocomplete from db”
Accordion	“jquery accordion”
Date Picker	“jquery datepicker”
Image Scale	“jquery image scale effect”

Table 4.2: Search result comparison: rank of a suitable example at the returned search results.

Data Point	Code Repository Tools				
	Google Search	Krugle	Koders	Stack Overflow	Example Overflow
Dynamic Dimension	4	Not found	Not found	1	3
Hover	1	2	1	1	2
Position	3	Not found	Not found	4	1
Rounded Corners	2	Not found	3	3	1
Draggable	1	Not found	3	2	1
Droppable	1	Not found	3	1	2
Autocomplete	1	Not found	Not found	1	1
Accordion	1	Not found	12	3	1
Date Picker	1	Not found	3	1	1
Image Scale	2	Not found	Not found	Not found	3
Avg. Rank	1.7	19.1	9.7778	3.8	1.6

We examined the list of results retrieved from each tool, and determined whether it accomplished the programming task. If no matching result was found at the top 20 results, it was marked as "not found" and received a rank of 21 for the average calculation.

4.3.3 Benchmark Results

Table 4.2 shows the rank location of suitable example code in the search results returned from each tool.

It can be seen that our tool has overall the best results with an average result rank of 1.6, where Google Search and Stack Overflow show similar results with 1.7 and 3.8 respectively, but Krugle and Koders have poor results. The reason for the poor results may be that both of them search for keywords to match the search query, without taking into account the context of the found keyword or any additional metadata. On the other hand our tool gives different weights to keywords based on their origin (code, title, tag, question, and answer). With this approach we obtain better results, we present suitable examples at the top of the recommendations. Another

Table 4.3: Context switching comparison: the number of mouse clicks required by the developer to see the actual code example.

Data Point	Code Repository Tools				
	Google Search	Krugle	Koders	Stack Overflow	Example Overflow
Dynamic Dimension	7	-	-	1	0
Hover	1	3	1	1	0
Position	5	-	-	7	0
Rounded Corners	3	-	5	5	0
Draggable	1	-	5	3	0
Droppable	1	-	5	2	0
Autocomplete	3	-	-	1	0
Accordion	1	-	23	6	0
Date Picker	1	-	5	1	0
Image Scale	3	-	-	-	0
Avg. Mouse Clicks	2.6	3	7.3333	3	0

possible cause for the poor results might be because both Krugle and Koders are limited to open source projects, where recent domains such as jQuery are not currently present. In addition it can be seen that our tool didn't require loading additional results (by scrolling down) and managed to show a suitable code example in the top 5 results.

During our benchmark we have also examined the amount of view/context switches by counting the number of mouse clicks required by the developer between the search request and until the developer was able to see the actual suitable example code. It can be seen at Table 4.3 that our approach has an average of 0 mouse clicks hence it doesn't require the developer to switch views or open new views, but instead we immediately show the developer the actual relevant code snippets. During our preliminary benchmark process we noticed that both Krugle and Koders returned results which linked to actual project files, without guiding the developer to the location of the required example code inside the project. This requires the developer to read that file as a whole, and search for the possible match for his query, thus forcing the developer to context switch from his actual task. In addition, most of their returned search results are only partial and have context in other files of that project, which requires the developer to further switch context and start looking at the other possibly relevant files.

4.3.4 Preliminary Benchmark Discussion and Limitations

Searching for code examples is possible using Stack Overflow directly. However Example Overflow is better optimized for this use case, as our preliminary benchmark suggests. Although our approach uses data taken from Stack Overflow, we show different results, since we analyze the data differently and we use our own example-targeted search formula as shown in (4.1).

The preliminary benchmark provided above is limited; we examined only a small subset of programming tasks, with mostly popular tasks. The queries were phrased by the authors, who also determined the relevance of the results.

Chapter 5

Investigating Opportunistic Software Development Using Social Media Recommendation System

Opportunistic software development has been studied before, [12, 11]. In our research we focus on understanding software developers' behavior and activities, when using social media in opportunistic software development.

Following are the research questions, results of our study, and implications of each question.

5.1 Is Limiting Software Development in Example Driven Manner Helpful?

In economics and law fields, paternalism[18] is the interference of a state or an individual with a person or a group's liberty or autonomy for their own good. For example the compulsory wearing of seat-belts[13]. Based on knowledge worker studies[28] and knowledge worker management guides[15], we see that autonomy is important to knowledge workers, however, some efforts to improve knowledge worker performance may involve limiting his autonomy. Can a similar approach be applied by limiting software development in example driven manner? One would expect that limiting may hinder or slow developers, prevent them from using certain resources. On the other hand, limiting developers in example driven manner, especially in opportunistic development where time is of the essence, may benefit them: they'll be more focused with the task at hand, use more examples and create quick prototypes. In the following, we discuss whether limiting professional developers in example driven manner is helpful, we support our discussion with data from the user study (see section 3.4.2).

Research participants were assigned to one of two groups: (1) developers who are limited to searching in Example Overflow (but allowed to follow external links), and (2) developers who are not limited to using a specific tool or to using example code at all. Research participants of group (1) were given a brief tutorial on how to use Example Overflow (EO), and how it displays the search results. Specifically they were told that EO shows results as example code snippets. Participants of group (2), were not told to use or search for code examples, and were free to use any tool and search for anything in order to accomplish the programming tasks. This creates a division between research participants who were in-fact limited in example driven manner, and

research participants who were not limited.

For each task, we have defined two parts needed to be accomplished in order for the task to be considered solved. Prior to each task we explained to each participant how each task was divided into parts and advised them to focus first on solving the first part, and only after accomplishing it, they should move to solving the second part of the task.

Table 5.1 shows the score of each participant per task, based on how much he or she has accomplished. For each part (out of 2) of a task we have given 0.5 points. Table 5.2 shows a comparison of score per task between participants who were limited in example driven manner, and participants who were not limited. We have noticed in the user study that participants were actually able to find suitable code examples, but were having trouble embedding them correctly into their code, thus failing to accomplish a task. We define the ability to find a suitable code example, as the participant's success to find a code example, and copy/paste it, thus indicating he actually have chosen it. Furthermore, a code example was considered suitable if it only required changing the variable names, and would accomplish the given task. Table 5.3 shows the ability of each research participant to find a suitable example code per task, by marking 'yes' as 1 and 'no' as 0. Note that some participants were able to solve a task by themselves, without using code examples, thus were marked as 0. Table 5.4 shows a comparison of ability to find a suitable code example between participants who were limited in example driven manner, and participants who were not limited.

We can see that limiting professional developers in example driven manner did not improve their results.

Table 5.1: RQ1: Score of each research participant per task

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Task 1	0	1	0	1	0	1	0	1	1	1	0.6
Task 2	0	0	0	0	0	0	0	0.5	0.5	1	0.2
Task 3	0	0	1	0.5	0	0	0.5	0	1	1	0.4
Task 4	0.5	0	0	1	0	0	0	1	0.5	0	0.3
Overall Tasks	0.5	1	1	2.5	0	1	0.5	2.5	3	3	1.5

Table 5.2: RQ1: Comparison between the groups for average score per task

	Avg.	Avg. for Limited to EO	Avg. for Non-limited
Task 1	0.6	0.2	1
Task 2	0.2	0.2	0.2
Task 3	0.4	0.5	0.3
Task 4	0.3	0.1	0.5
Overall Tasks	1.5	1	2

Table 5.3: RQ1: Ability of each participant to find a suitable example per task

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Task 1	1	1	0	1	0	1	0	1	1	1	0.7
Task 2	1	0	0	1	0	0	1	1	1	0	0.5
Task 3	0	0	1	0	0	0	1	0	1	1	0.4
Task 4	1	0	0	1	0	0	1	1	1	0	0.5
Overall Tasks	3	1	1	3	0	1	3	3	4	2	2.1

Table 5.4: RQ1: Comparison between the groups for ability of each participant to find a suitable example per task

	Avg.	Avg. for Limited to EO	Avg. for Non-limited
Task 1	0.7	0.4	1
Task 2	0.5	0.4	0.6
Task 3	0.4	0.6	0.2
Task 4	0.5	0.4	0.6
Overall Tasks	2.1	1.8	2.4

5.2 How Do Professional Developers Mitigate Concerns Related to Example Usage?

Services ,such as Stack Overflow, Github, Sourceforge, allow developers to access vast amount of source code on-line. Furthermore, some of the available code is reviewed, debugged and patched by other developers and may be considered as a high quality source for code snippets. On the other hand, as shown in our survey (see Section 3.4.1), example usage is a popular practice among developers, yet developers have concerns involved with example usage. Developers may hesitate to use or admit they use code examples. Some developers may say *"If you use copy and paste while you're coding, you're probably committing a design error"*, while others may go as far as advocate disabling copy and paste entirely[1]. While most developers won't ban example usage altogether, they have concerns involved with example usage.

We wish to identify the concerns professional developers have when using examples and see how they mitigate them in opportunistic development.

5.2.1 Concerns

In the following, we outline professional developers' concerns involved in opportunistic development, as we identified them in our user study (see section 3.4.2).

Confidence in the example found (acceptance of result) - Developers have a measure of confidence for each one of the results. Developers mention several factors that may influence the measure of confidence (in no particular order):

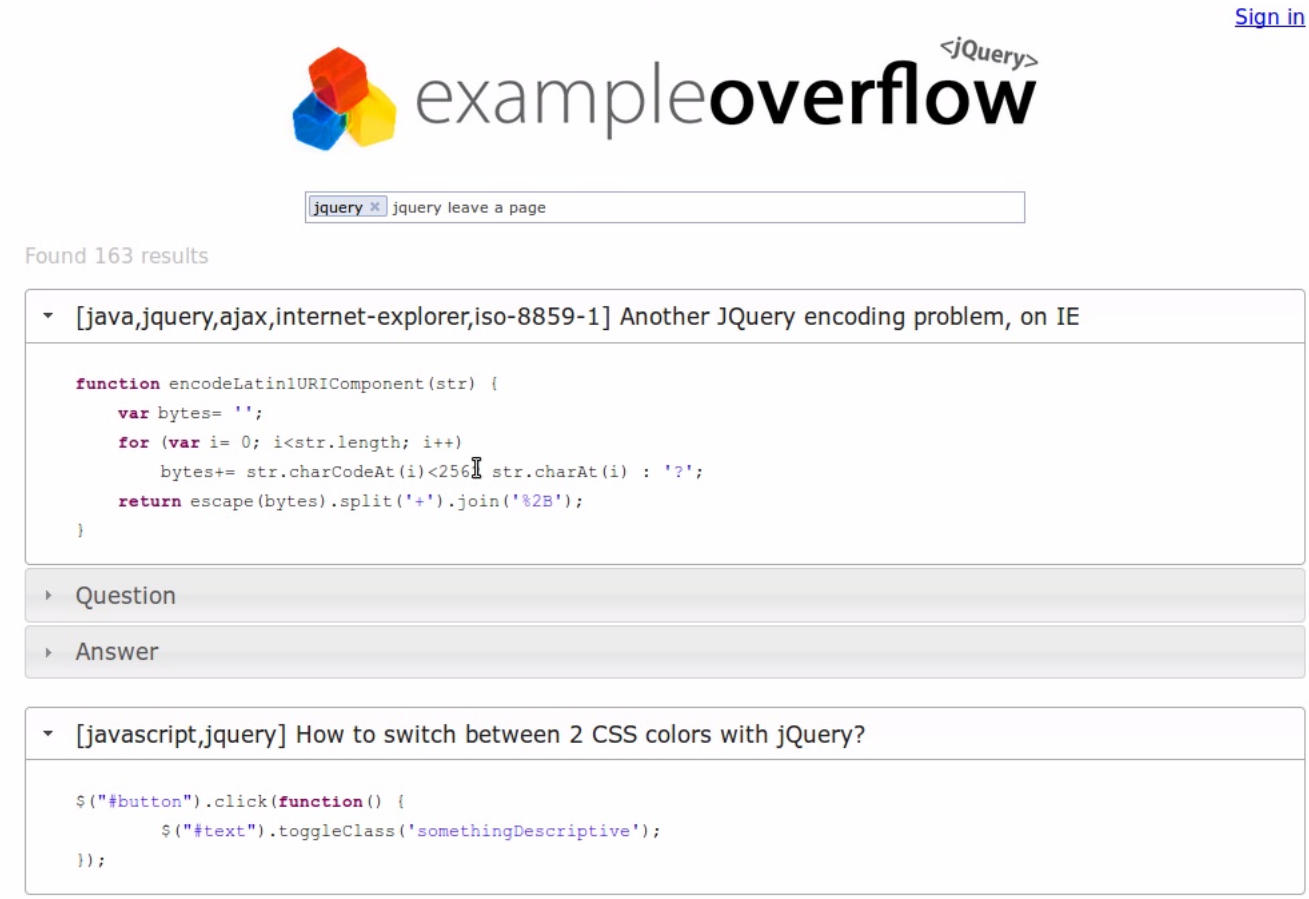
- Source of example code - Developers examine the source of the example code, whether it came from the official documentation, a popular Q&A website like Stack Overflow, a developers blog, or other sources. When asked in the interview why a specific code snippet was chosen, participant 4 has answered *"If it came from an accepted answer on Stack Overflow, someone has tested it and it is working, or if it comes from a documentation. If there is a rating, and a proper forum with good comments then I'll choose it"*.

- Author of example code - In some cases developers find code examples published by other developers whom they familiar with, or used his or her code examples before. In our observations, participant 6, who was using Google search, has recognized the author of one of the results and chosen to examine that code example over the others. This is part of a more general trend where social media allows developers to "follow" other developers, see their code and comments. Developers use it to learn from other developers, and this process is supported by the software development communities like GitHub and Stack Overflow.

- Simplicity and length of example - Developers tend to prioritize code examples based on their simplicity and length. In our observations, we have noticed several cases where developers skipped over code examples because of their length compared to the other results. In other cases, this was the factor to determine in what order they would browse or examine the results, where shorter and simpler code examples were examined first.

[1] http://www.secretgeek.net/copy_paste_dont_do_it.asp

Figure 5.1: "The first example seems irrelevant to me. It's too complicated" - participant 7, during task 1 of user study



During the tasks, participants have explicitly stated 6 times that they "give up" on a code example because of it's length. Participant 7, during task 1, said *"The first example seems irrelevant to me. It's too complicated"*. We can see evidence for this in the interviews as well: participant 1 said *"I was looking for short code snippets (it would be harder to understand a long code example)"*. Participant 5 said *"I've chosen code snippets that seemed easy. The tasks were easy, and it didn't make sense to copy 50 lines of code. In a familiar domain, I would prefer shorter code (examples) as well"*. Participant 6 mentioned simplicity as well and said *"Tried to choose the simpler one, or the one more suitable for the case. Usually the simpler one (to understand)"*. Participant 7 and participant 9 mention in the interview simplicity and similarity to the task at hand as well.

- Similarity of content or keywords - Developers searching for code examples in an unfamiliar domain, look for similar keywords or similarity in content to their own task. In the interview, when asked why a specific code snippet was chosen, participant 9 replied *"The one that seemed easier to understand, simpler. I've stayed away from the complicated ones. If there was a explanation/story similar to mine, or if some of the code in the example was similar to mine."*
- Rank of an example in search results - Developers rely on the recommendation system

or search tool ranking of the results, and give high priority to the top results. If it is a tool they are familiar with and trust, then they may choose a highly ranked code examples that contradicts other factors mentioned here. We noticed in our observations that developers who were using EO, were less lenient to trust the result ranking, until proved to be trustworthy. We see evidence for this in the interviews as well: participant 3, who used EO, mentions in the interview *"I would have accomplished more if I was able to search in Google - I would get better results. It (EO) was lacking query auto-completion (feature), which would have encouraged me to see that others have searched this query as well"*. Participant 2 has said *"In most cases I went for the first result. Greedy, the first that seemed relevant"*. Participant 8 explains Stack Overflow ranking importance - *"When looking at code examples in a question or answer on Stack Overflow, I go from the second post and downwards, and try to choose based on comments by other people. The higher the example on Stack Overflow the more chance there will be more answers (comments by other people)"*.

- Social rating - Developers rely on social ratings (including comments and reviews) of other developers when searching for code examples. We see evidence for this in the interviews: participant 4, participant 6, and participant 8 talk about using comments or relying on reviews of other developers when choosing a code example.

- Comprehension of code - Developers used code examples which they do not understand, in some cases it worked and in some cases it didn't. But they would prefer a code example they do understand (at least some of it). In the interview participants 1,5,6, and 9 mention "understanding" and comprehension when asked why a specific code snippet was chosen. Participant 5 said "I try/prefer to copy code I understand 80% of it".

- Past experience - Developers rely on past experience when choosing a code example. This is also true even if that code example is much less suitable that other examples returned by the recommendation tool. We saw evidence for this with participant 10, the only participant to have prior experience with jQuery: at tasks 1 and 2 his past experience has helped him to choose a suitable code example, but in task 4 he has chosen an unnecessarily complicated example based on his past experience (while not choosing a more suitable example that was shown to him), and failed to accomplish the task.

All the above factors go into account when a developer's measure of confidence in a code example is determined, while each developer may give different priorities to different factors.

Sense of responsibility - Participants have stated, regarding example usage, that in the wild (at their job) they would have acted differently. They would have been more cautious with the examples they copy/paste, and put more effort into understanding them. In some cases, they would have used less examples. This behavior can also be interpreted as how the developer sees the expectations of him by the organization, similar to the findings of Reimenschneider *et al.*[45] regarding methodology acceptance. They investigate why individual developers accept or resist methodologies deployed in order to improve software development processes. They found that methodology adoption intentions are driven by: (1) the presence of an organizational mandate to use the methodology, (2) the compatibility of the methodology with how developers perform their work, and (3) the opinions of developers' coworkers and supervisors toward

using the methodology.

Lack of (domain) knowledge - In the user study the developers were faced with tasks in an unfamiliar domain. Some developers tried to approach the tasks as if it was a familiar domain, they were trying to look for similarities in keywords, or code structure to other familiar domains. We saw evidence for this when participants were trying to solve tasks without using any code examples, and they were guessing the jQuery keywords based on similar keywords in Java. Other participants quickly have realized that by using code examples they are able to accomplish the tasks. We believe that lack of domain knowledge contributed to example usage: some of the participants were looking for examples to learn the jQuery syntax or find the required keyword, while others were searching for a fully working example code to solve the task.

Time - The tasks were time-framed and participants had a limited time to accomplish each task. Participants have stated that the lack of time has influenced the way they were searching for code examples - "If I had more time, I would have read more context". But does lack of time discourages example usage or can it be a motivator for example usage ? Barzilay[6] investigated motivation for example usage, and has identified three axes that affect example usage motivation (a) the propreties of a task, (b) the development activity, and (c) software engineering aspects, such as: speeding up development time and learning time. This means time-framed tasks may have contributed to example usage.

5.3 What are the Micro-Activities Involved in Opportunistic Development When Using Social Media Based Recommendation System?

In the previous section we identified concerns involved in example usage with opportunistic development. In this section, we characterize micro-activities, as identified in the user study (see section 3.4.2), involved in example usage and how they are used by professional developers to mitigate the concerns. Figure 5.2 illustrates how each concern is mitigated by each micro-activity.

Task comprehension - Developers try to comprehend the task at hand, the domain involved, and the challenges they may be faced with. This affects the developer's approach to completing the task, and the other micro-activities. Participant 10 said he decided that the task is considered "*a simple task*", thus he was searching for a code example that was able to fully accomplish the task. Code examples which may have had a partial solution to the task were disqualified immediately and combining several code snippets was not an option. Task comprehension mitigates the lack of (domain) knowledge concern - by understanding the task at hand, it's requirements, and main keywords, developers are more motivated to look for examples. In addition, task comprehension mitigates confidence in the example found concern, as it improves comprehension of examples found, and allows for better comparison between different examples and also between the example and the developer's code or task.

Forming and refining the query - During the observations participants have given several parameters taken into consideration when they are forming their queries: The task (keywords), technical keywords, similar tasks from a different domain, Google's auto-complete (collaborative/social recommendations), keywords based on a possible solution, or keywords that they believe other developers may have used to solve this task. Participants refined their query when the top few results (the first batch of results) were not suitable to what they were searching, or if their confidence in the results was low. In most cases, refinement of the query involved addition, removal or changing of a single keyword. Participant 7, during task 1 when using the query "jquery leave a page", said *"The first example seems irrelevant to me. It's too complicated. The second one as well, it is css. I think something in my search (query) was not good."* The query was refined to "jquery leave a url". By refining their query, developers try to improve the confidence in the results to a certain satisfactory level, at which point they will choose a suitable example and try it, either by copy/paste or by using it as a reference to write their own code.

Browsing and examining results - Participants browsed over the results while trying to identify if there are results that seems suitable, or acceptable to them. Participants looked at the title, tags, author, source, and the length of the example. They would also examine the social rating (if present), and look for presence of main keywords that may appear in their task. During task 2, participant 4 has said *"The first result seems promising, it has all the keywords (from the query)"*. This process doesn't require developers to understand the example and is mainly based on technical mechanisms, but browsing affects most of the factors determining the confidence of the example found concern, thus able to mitigate this concern.

Figure 5.2: Overview diagram of concerns related to example usage and how they are mitigated by each micro-activity

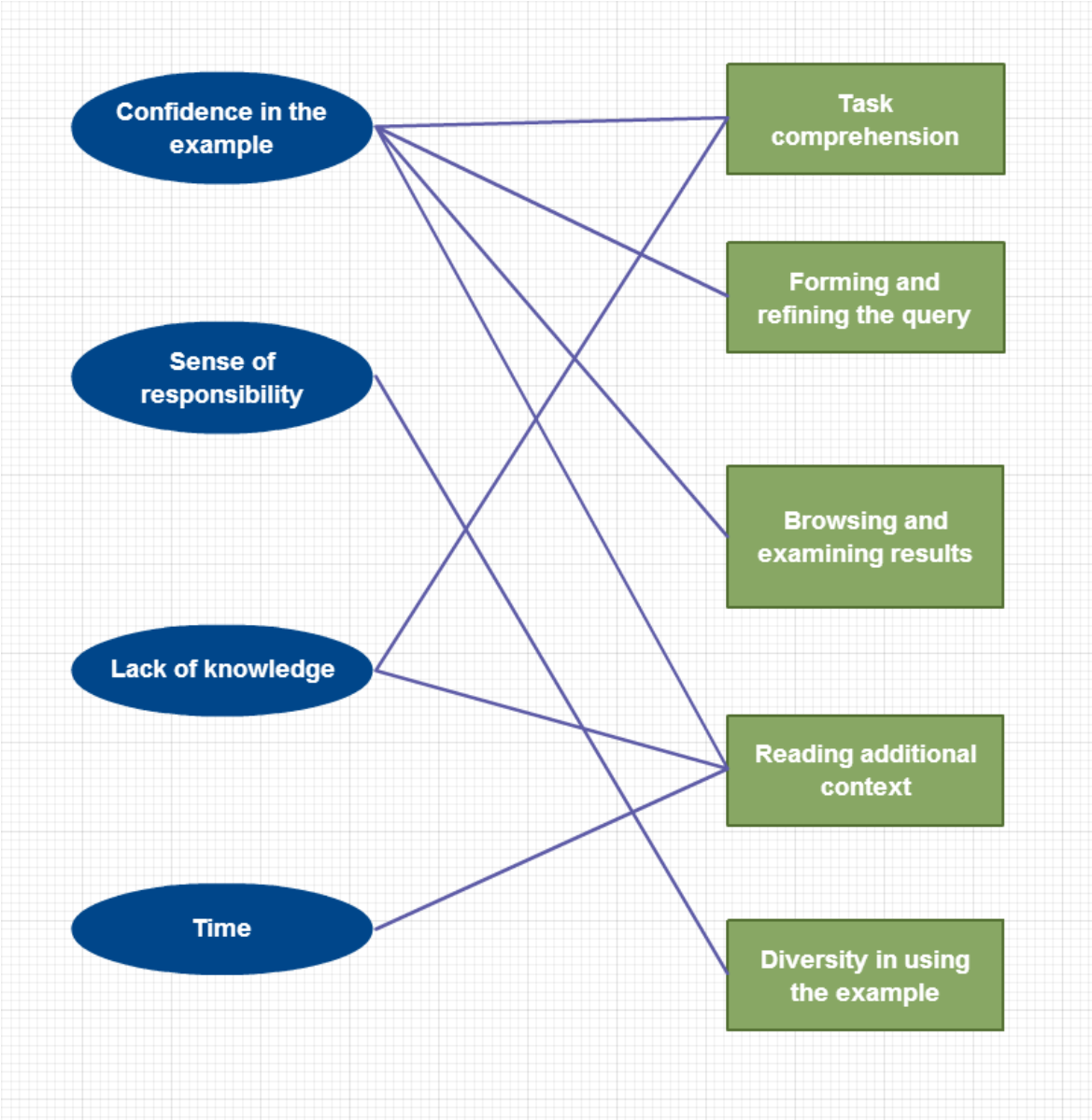


Table 5.5: Example usage: as a code snippet (copy/paste) vs. as a reference

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
Task 1	c/p	c/p	ref	c/p	ref	c/p	c/p	c/p	c/p	c/p
Task 2	c/p,ref	ref	ref	ref	ref	c/p	ref	c/p	c/p	c/p
Task 3	ref	c/p	c/p	c/p	c/p	ref	c/p	c/p	c/p	ref
Task 4	c/p	c/p	ref	c/p	c/p	ref	c/p	c/p	c/p	c/p,ref

Results that have passed the browsing stage, are being examined more deeply and this time the developer reads the actual code snippet or the surrounding context, including comments and documentation.

Reading additional context - Developers read additional context (other than the code itself) in order to determine the example code relevance and their confidence in the example they are examining. We have observed that some participants read additional context for every example code browsed or examined, but others may do it only on rare occasions or not at all. The amount of context viewed also is influenced by various factors. Some partipants have mentioned that in a familiar domain (in work) they may use much less additional context, whereas in a non familiar domain (user study) they will opt to use it more.

Diversity in using the example - Developers may use a code example in various ways, starting from copying and pasting it as is, to copying some of it, or to just using it as a reference while coding manually their solution. We have observed that a single developer may use examples differently for similar tasks, while trying to mitigate some of the concerns. Table 5.5 shows in which way each participant used examples per task, whether he or she copy/pasted it (partially or fully), or whether he just used it as a reference while manually writing the code. Developers try to mitigate the sense of responsibility concern, by avoiding a direct copy/paste, and either copying small part of the example or by using it as reference. Though this may be more time consuming, this allows developers to learn from the example and gives them a higher sense of responsibility.

Developers are required not only to find a suitable code example, but also to successfully embed it into their existing code. In our study (see section 5.1), participants were able to find and copy/paste a suitable example in 21 tasks (out of 40 combined), but only 13 of those were successfully embedded.

5.3.1 Implications

By identifying and characterizing the concerns related to example usage and the micro-activities used to mitigate them, we can improve example usage and support the example embedding Eco-system[5]. Understanding the micro-activities performed by real practitioners, we (as tool developers) can design tools and recommendation systems with better support for these micro-activities, and help developers mitigate the concerns related to example usage.

5.4 When Searching for Code Examples, Do Developers Refine Their Query or Continue Examining Additional Results?

Search and recommendation tools based on keyword search, such as Example Overflow and Google search, are keyword dependent. A developer needs to use "the right" keywords in his query in order to find the most suitable example. But what happens when the returned results are irrelevant or not accepted by the developer? We want to examine whether developers who search for examples would refine their query or continue reading additional results without refining the query.

5.4.1 Implication

We hypothesize that when developers would use Example Overflow they would apply the following pattern: the developer would form a query, initiate a search, and examine all results until choosing one of them without refining the query.

The implication from this research question may affect design of example code recommendation systems, and improve example usage support tools and practices. Additionally, by comparing between participants who are limited to searching with EO and participants who are not limited, we can examine if there is a difference between their search pattern. If there is a difference between the groups, this implies the design of search tools influences developer behavior when searching for examples, and we (as tool developers) should reuse our design to support the process of example searching better.

5.4.2 User Study Results

In order to examine this we have measured the number of different queries used per task by each participant from the beginning of the task and until the task was accomplished or the time frame was exceeded. We define query refinement as a any change of the original query, as opposed to [34] who abstracted search actions into a set of mutually exclusive refinement classes, where the refinement class of a query represents a user's intent relative to his prior query. Table 5.6 shows the number of queries used per task by each participant. Table 5.7 shows the comparison between participants who were using EO and participants who were allowed to use other tools.

We can see that the average number of queries used by a developer per task is 5.95. Participants who used EO had an average of 7.7 queries per task, while participants who used Google search had an average of 4.2 queries per task. A possible cause for this can be a combination of (1) participants were working in an unfamiliar domain and had difficulties forming their queries, (2) Google search, as opposed to Example Overflow, has query auto-complete feature based on similar popular queries.

Table 5.6: RQ4: Number of queries per task

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Task 1	7	1	8	1	14	1	9	4	3	1	4.9
Task 2	5	6	8	4	8	2	6	3	2	5	4.9
Task 3	4	3	10	4	9	6	3	4	3	5	5.1
Task 4	3	10	16	4	14	5	13	13	5	6	8.9
Overall Avg.	4.75	5	10.5	3.25	11.25	3.5	7.75	6	3.25	4.25	5.95

Table 5.7: RQ4: Comparison between the groups for number of queries per task

	Avg.	Avg. for EO	Avg. for Other
Task 1	4.9	7.8	2
Task 2	4.9	6.4	3.4
Task 3	5.1	6.2	4
Task 4	8.9	10.4	7.4
Overall Avg.	5.95	7.7	4.2

5.5 How Many Code Examples are Examined Before Choosing a Suitable Code Example?

When designing Example Overflow (EO), one of the design questions we were interested with was - how many recommendations should be presented to a developer. Granka *et al.*[21] conduct an eye tracking analysis for Google search users, and show that there is a drop in viewing time and number of clicks at the 6/7 ranked results, and a sharp drop occurs after result 10, as ten results are displayed per page. We decided to follow their findings and display 5 results in a page on EO, with pagination (see Section 4.2.1). However, [21] investigated "regular" search with Google search without focusing on software developers, while we are interested with code example search based on social media recommendation systems. In addition, we want to differentiate between browsing and examining an example.

5.5.1 Implications

The implication from this research question may affect design of example code recommendation systems, and improve example usage support tools and practices. If the number of examined results is high, it supports the traditional "Google search" design, with 10 results per page and pagination. But if the number of examined results is very low, it may imply that example code recommendation tools should display less results, 1-2 recommendations, and focus on supporting developer comprehension and providing example embedding support.

Based on our preliminary benchmark, we hypothesize that a developer examines 3 code examples before choosing a suitable code example.

5.5.2 User Study Results

We distinguish between browsing and careful examination of the search results. Browsing involves reading the title or the question, while careful examination involves delving into an examination phase and may include understanding, learning, and comparing. A participant would choose a suitable code example by copying and pasting any part of it or by copying it by hand.

Table 5.8 shows the number of code examples examined by a participant per task before choosing a suitable one. Table 5.9 shows a comparison between participants who were limited to searching with EO and participants who were not limited. Note that some participant didn't examine any code examples during a task. This happened for two reasons: (1) they were unable to find a suitable code example in the given time frame, (2) they wrote all their code without using code examples at all. We marked these cases with as "-" in the table.

We can see the overall average number of code examples a developer would examine before doing copy/paste is 2.02, and there was no significant difference between participants in either group in the overall average. But, the participants who used EO show a larger variance in the results between tasks.

Table 5.8: RQ5: Number of code examples examined before copy/paste

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Task 1	1	1	2	2	0	1	1	2	2	1	1.3
Task 2	6	4	-	1	3	2	0	1	1	1	2.37
Task 3	-	2	4	1	3	-	1	1	2	-	2
Task 4	1	5	-	1	1	1	4	7	1	1	2.44
Overall Avg.	2.67	3	3	1.25	1.75	1.33	2	2.75	1.5	1	2.02

Table 5.9: RQ5: Comparison between the groups for number of code examples examined before copy/paste

	Avg.	Avg. for EO	Avg. for Other
Task 1	1.3	1	1.6
Task 2	2.37	3.33	1.8
Task 3	2	2.667	1.5
Task 4	2.44	1.75	3
Overall Avg.	2.02	2.18	1.975

Table 5.10: RQ6: Number of times a participant viewed additional context per task

Participant	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	Avg.
Task 1	3	3	1	1	7	2	3	2	3	1	2.6
Task 2	0	5	1	5	0	3	2	2	2	1	2.1
Task 3	0	3	3	3	1	6	3	3	2	1	2.5
Task 4	3	10	5	3	0	6	6	11	13	3	6
Overall Avg.	1.5	5.25	2.5	3	2	4.25	3.5	4.5	5	1.5	3.3

5.6 When Searching for Code Examples, Do Developers Use Additional Context ?

Nasehi *et al.*[37] conducted a qualitative analysis of the questions and answers posted to a Stack Overflow. They found that the explanations accompanying examples are as important as the examples themselves. We are interested to examine this further, whether the additional context surrounding a code example is being used more than the example itself when used in an opportunistic software development setting. In addition, we want to examine the design decision of Example Overflow to hide the additional context, and display it "upon request" to the developer.

5.6.1 Implications

We hypothesize that developers need additional context for code examples, and may even use it more than the code snippet itself. If our hypothesis holds it implies that offering a rich-verbose context in place, as an integral part of the code snippet, addresses a genuine need of the developers. "Complementary" tools that are lacking this functionality are missing an important requirement. If our hypothesis doesn't hold, it should be further investigated whether this additional context, though not used by many developers, is important - if so look for ways to encourage developers to use it. If not used or not important - pragmatic tools such as EO could justify "hiding" additional context and offer clean code / improve code browsing.

5.6.2 User Study Results

Table 5.10 shows the number of times the participant viewed additional context, such as comments, documentation, body of question and answer (if it was from a Q&A website), per task. Table 5.11 shows a comparison between participants who were limited to searching with EO and participants who were not limited.

We can see that the average number of times a participant viewed additional context per task was 3.3 for all participants, 2.2 for participants who were limited to searching with EO, and 4.4 to participants who were not limited.

Table 5.11: RQ6: Comparison between the groups for number of times viewed additional context

	Avg.	Avg. for EO	Avg. for Other
Task 1	2.6	3	2.2
Task 2	2.1	0.8	3.4
Task 3	2.5	1.6	3.4
Task 4	6	3.4	8.6
Overall Avg.	3.3	2.2	4.4

Chapter 6

Summary

Social media will play an increasingly important role in software engineering research and practice, as it raises new questions, new possibilities. In this thesis we examined the use of social media based recommendations system in opportunistic software development. We did not focus only on the technical side of the design, but rather on the human-machine interactions, and the human behavior involved. Designing useful tools for developers requires to identify the activities and sub-activities involved in example usage when using social media.

We found that professional software developers have concerns related to example usage, concern that govern how and when examples are used. However, developers do not avoid example usage altogether, but rather mitigate these concerns with micro-activities. By identifying the concerns and micro-activities we have done the first step towards improving the design of useful social media based tools and recommendation systems.

Furthermore, we found that being part of the Example Embedding Ecosystem is a double-edged sword - tools and recommendation systems, such as Example Overflow, are not only enabling the ecosystem, but are also being enabled by it. Without proper training, the developer is not able to critically evaluate the various examples, browse them and merge them. Without proper practices, systems which are developed using examples extensively may end up as Frankenstein code, and bugs may find their way in because the examples used were not properly tested.

"The easy, conversational tone of good writing comes only on the eighth rewrite."

Paul Graham

Bibliography

[1] Terry Anderson and Julie Shattuck. Design-based research a decade of progress in education research? *Educational Researcher*, 41(1):16–25, 2012. 3.3

[2] Dejana Bajic and Kelly Lyons. Leveraging social media to gather user feedback for software development. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 1–6, New York, NY, USA, 2011. ACM. 2.1

[3] Sasha Barab and Kurt Squire. Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences*, 13(1):1–14, 2004. 3.3

[4] Sasha Barab, Michael Thomas, Tyler Dodge, Robert Carteaux, and Hakan Tuzun. Making learning fun: Quest atlantis, a game without guns. *Educational Technology Research and Development*, 53(1):86–107, 2005. 3.3

[5] Ohad Barzilay. Example embedding. In *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, ONWARD '11, pages 137–144, New York, NY, USA, 2011. ACM. 4.1, 4.1.2, 4.2, 5.3.1

[6] Ohad Barzilay. *Example embedding: On the diversity of example usage in professional software development*. PhD thesis, Tel Aviv University, 2011. 1.1, 2.5, 3.4.1, 5.2.1

[7] Ohad Barzilay, Orit Hazzan, and Amiram Yehudai. Characterizing example embedding as a software activity. In *SUITE 2009: Proceedings of the 1st international workshop on Search-Driven Development - Users, Infrastructure, Tools and Evaluation at ICSE '09*, pages 9–12. IEEE Computer Society, 2009. 4.1, 4.1.2

[8] Bear Bibeault and Yehuda Katz. *jQuery in Action*. Manning Publications, 2nd edition, 2010. 4.3.1

[9] Gargi Bougie, Jamie Starke, Margaret-Anne Storey, and Daniel M. German. Towards understanding twitter use in software engineering: preliminary findings, ongoing challenges and future questions. In *Proceeding of the 2nd international workshop on Web 2.0 for software engineering*, Web2SE '11, pages 31–36, New York, NY, USA, 2011. ACM. 2.1

[10] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott 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*, CHI '10, pages 513–522, New York, NY, USA, 2010. ACM. 1.1, 2.3, 4.1.2, 4.3.1

[11] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott 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*, CHI '09, pages 1589–1598, New York, NY, USA, 2009. ACM. 2.4, 5

[12] Joel Brandt, Philip J. Guo, Joel Lewenstein, and Scott R. Klemmer. Opportunistic programming: how rapid ideation and prototyping occur in practice. In *Proceedings of the 4th international workshop on End-user software engineering*, WEUSE '08, pages 1–5, New York, NY, USA, 2008. ACM. 5

[13] Richard J Budd, Derek North, and Christopher Spencer. Understanding seat-belt use: A test of bentler and speckart's extension of the 'theory of reasoned action'. *European Journal of Social Psychology*, 14(1):69–78, 1984. 1.2, 3.1, 5.1

[14] Abe Crystal and Beth Ellington. Task analysis and human-computer interaction: approaches, techniques, and levels of analysis. In *AMCIS*, page 391. Citeseer, 2004. 1.1, 2.4

[15] Thomas H Davenport. *Thinking for a living: how to get better performances and results from knowledge workers*. Harvard Business Press, 2005. 1.2, 2.6, 3.1, 5.1

[16] Chris Dede. Why design-based research is both important and difficult. *Educational Technology*, 45(1):5–8, 2005. 1.2, 3.3

[17] Norman Kent Denzin and Yvonna S. Lincoln. *The Sage Handbook of Qualitative Research*. Sage Publications, 3rd edition, 2005. 3.2

[18] Gerald Dworkin. Paternalism. *the Monist*, 56(1):64–84, 1972. 1.2, 3.1, 5.1

[19] Jean-Marie Favre, Jacky Estublier, and Remy Sanlaville. Tool adoption issues in a very large software company. In *Proceedings of 3rd International Workshop on Adoption-Centric Software Engineering (ACSE'03), Portland, Oregon, USA*, pages 81–89, 2003. 2.6

[20] Marsha E. Fonteyn, Benjamin Kuipers, and Susan J. Grobe. A Description of Think Aloud Method and Protocol Analysis. *Qual Health Res*, 3(4):430–441, November 1993. 3.4.2

[21] Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '04, pages 478–479, New York, NY, USA, 2004. ACM. 1.1, 1.2, 3.1, 4.2.1, 5.5

[22] Mark Grechanik, Chen Fu, Qing Xie, Collin McMillan, Denys Poshyvanyk, and Chad M. Cumby. A search engine for finding highly relevant applications. In *ICSE (1)*, pages 475–484, 2010. 2.3

[23] Ido Guy, Naama Zwerdling, David Carmel, Inbal Ronen, Erel Uziel, Sivan Yogev, and Shila Ofek-Koifman. Personalized recommendation of social software items based on social relations. In *Proceedings of the third ACM conference on Recommender systems*, RecSys '09, pages 53–60, New York, NY, USA, 2009. ACM. 2.1

[24] Erik Hatcher, Otis Gospodnetic, and Mike McCandless. *Lucene in Action*. Manning, 2nd revised edition. edition, 8 2010. 4.2.3

[25] Takashi Hattori. Wikigramming: a wiki-based training environment for programming. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 7–12, New York, NY, USA, 2011. ACM. 2.1

[26] Douglas R. Hofstadter. Analogy as the core of cognition. In Dedre Gentner, Keith J. Holyoak, and Boicho N. Kokinov, editors, *The Analogical Mind: Perspectives from Cognitive Science*, pages 499–538. MIT Press/Bradford Books, 2001. 4.1.1

[27] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 117–125. ACM, 2005. 2.3, 4.3.1

[28] Brian D Janz, Jason A Colquitt, and Raymond A Noe. Knowledge worker team effectiveness: The role of autonomy, interdependence, team development, and contextual support variables. *Personnel psychology*, 50(4):877–904, 1997. 1.2, 2.6, 3.1, 5.1

[29] Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. An ethnographic study of copy and paste programming practices in OOPL. In *ISESE '04: Proceedings of the 2004 International Symposium on Empirical Software Engineering*, pages 83–92. IEEE Computer Society, 2004. 2.4

[30] Anne Lacey and Donna Luff. *Qualitative data analysis*. Trent Focus Sheffield, 2001. 3.5

[31] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. A study of the difficulties of novice programmers. *SIGCSE Bull.*, 37:14–18, June 2005. 2.5

[32] Thomas D LaToza and Brad A Myers. Designing useful tools for developers. In *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*, pages 45–50. ACM, 2011. 1.1, 2.4

[33] Thomas D. LaToza, Gina Venolia, and Robert DeLine. Maintaining mental models: a study of developer work habits. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 492–501. ACM, 2006. 2.4

[34] Tessa Lau and Eric Horvitz. Patterns of search: analyzing and modeling web query refinement. *COURSES AND LECTURES-INTERNATIONAL CENTRE FOR MECHANICAL SCIENCES*, pages 119–128, 1999. 5.4.2

[35] Lena Mamykina, Bella Manoim, Manas Mittal, George Hripcsak, and Björn Hartmann. Design lessons from the fastest Q&A site in the west. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2857–2866, New York, NY, USA, 2011. ACM. 1.1, 2.2

[36] Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. Recommending source code examples via api call usages and documentation. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, RSSE '10, pages 21–25, New York, NY, USA, 2010. ACM. 2.3

[37] S.M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming q amp;a in stackoverflow. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 25–34, 2012. 2.5, 5.6

[38] L. R. Neal. A system for example-based programming. *SIGCHI Bull.*, 20(SI):63–68, 1989. 2.5

[39] L Novik. Analogical transfer, problem similarity, and expertise. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14:510–520, 1988. 4.1.1

[40] Chris Parnin and Christoph Treude. Measuring api documentation on the web. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 25–30, New York, NY, USA, 2011. ACM. 2.1, 4.2

[41] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. Technical report, Georgia Institute of Technology, 2012. 2.2

[42] Shari Lawrence Pfleeger and Barbara A. Kitchenham. Principles of survey research: part 1: turning lemons into lemonade. *SIGSOFT Softw. Eng. Notes*, 26:16–18, November 2001. 3.4.1

[43] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 57–66, 2013. 1.1, 2.3, 4.1.2, 4.3.1

[44] Thomas C Reeves, Jan Herrington, and Ron Oliver. Design research: A socially responsible approach to instructional technology research in higher education. *Journal of Computing in Higher Education*, 16(2):96–115, 2005. 3.3

[45] C.K. Riemenschneider, B.C. Hardgrave, and F.D. Davis. Explaining software developer acceptance of methodologies: a comparison of five theoretical models. *Software Engineering, IEEE Transactions on*, 28(12):1135–1145, 2002. 2.6, 5.2.1

[46] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *Software Engineering*, 25(4):557–572, 1999. 3.2, 3.4.2, 3.4.2, 3.5, 3.5.1

[47] Jonathan Sillito and Andrew Begel. App-directed learning: An exploratory study. 2.4

[48] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 21. IBM Press, 1997. 3.2

[49] Margaret-Anne Storey, Christoph Treude, Arie van Deursen, and Li-Te Cheng. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*, FoSER '10, pages 359–364, New York, NY, USA, 2010. ACM. 1.1, 2.1

[50] J. Stylos and B.A. Myers. Mica: A web-search tool for finding api components and examples. In *Visual Languages and Human-Centric Computing, 2006. VL/HCC 2006. IEEE Symposium on*, pages 195 –202, 2006. 2.3, 4.1

[51] Ashish Sureka, Atul Goyal, and Ayushi Rastogi. Using social network analysis for mining collaboration data in a defect tracking system for risk and vulnerability analysis. In *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, pages 195–204, New York, NY, USA, 2011. ACM. 2.1

[52] James Surowiecki. *The Wisdom of Crowds*. Anchor, 2005. 2.1

[53] Suresh Thummalapenta and Tao Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, ASE '07, pages 204–213, New York, NY, USA, 2007. ACM. 2.3, 4.3.1

[54] Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. How do programmers ask and answer questions on the web? (nier track). In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 804–807, New York, NY, USA, 2011. ACM. 2.1, 2.2

[55] Christoph Treude, Fernando Figueira Filho, Brendan Cleary, and Margaret-Anne Storey. Programming in a socially networked world: the evolution of the social programmer, 2012. 2.1

[56] Feng Wang and MichaelJ. Hannafin. Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, 53(4):5–23, 2005. 1.1, 3.3

[57] Feng Wang and MichaelJ. Hannafin. Design-based research and technology-enhanced learning environments. *Educational Technology Research and Development*, 53(4):5–23, 2005. 3.3

[58] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26:13:1–13:37, June 2008. 1.1, 4.2.3

אוניברסיטת תל-אביב
הפקולטה למדעים מדויקים ע״ש ריימונד ובברלי סאקלר
ביה״ס למדעי המחשב ע״ש בלבטניק

על פיתוח תוכנה אופורטוניסטי בעזרת מערכת המלצה מבוססת מדיה חברתית

חיבור זה הוגש כחלק מהדרישות לתואר מוסמך אוניברסיטה (M.Sc.)
באוניברסיטת תל-אביב, ביה״ס למדעי המחשב

מאת

אלכסיי זגלסקי

העבודה הוכנה בהדרכתו של פרופ׳ עמירם יהודאי

תשרי תשע״ד

תקציר

מפתחי תוכנה מתמודדים עם סט משתנה ללא הרף של שפות תכנות, פלטפורמות וטכנולוגיות. פרויקטי תוכנה עשויים להשתמש במספר רב של טכנולוגיות או פלטפורמות, בעוד שמפתחי תוכנה לא מסוגלים להתמקצע בכולן. על מנת להתגבר על אתגרים אלה מפתחי תוכנה מחפשים עזרה באתרי שאלה ותשובה כגון Stack Overflow. אנו מאמינים כי ההצלחה של אתרים אלו נובעת מקצב השינויים המהיר בטכנולוגיה, בעוד אתרי ויקי והתיעוד הרשמי נגררים מאחור. על ידי שימוש במדיה חברתית, ובמיוחד באתרי שאלה ותשובה, ייתכן והמפתחים יוכלו לצמצם פערים אלו. האתר Stack Overflow מגלם בתוכו ידע לא רק בצורה של שאלות ותשובות או הערות אלא גם כקוד מקור, שמפתחי תוכנה נוהגים לשלב כחלק מהשאלה או התשובה שלהם. בעוד שתפקיד המדיה החברתית במחקר להנדסת תוכנה ובתעשיה יגדל משמעותית, ישנם כיום מעט כלי תוכנה זמינים המשתמשים במדיה חברתית, ובפרט מעט מאוד מערכות להמלצה על קוד המבוססות על מדיה החברתית.

התחלנו את המחקר שלנו במטרה לפתח מערכת המלצת קוד התומכת בשימוש בדוגמאות תוך שימוש במדיה חברתית. אבל לא ניתן לתכנן מערכת זו מבלי לחקור אינטראקציות בין האדם למכונה, ובעיקר מבלי ללמוד על ההתנהגות האנושית בעת פיתוח תוכנה. יתרה מכך, על מנת לעצב כלים שימושיים למפתחי תוכנה יש לזהות את החששות ומיקרו-פעילויות הכרוכים בשימוש בדוגמאות בעזרת מדיה חברתית. המטרה שלנו עם כך, היא לא עיצוב של כלי או מערכת להמלצת קוד, אלא לחקור את החששות המעורבים בפיתוח תוכנה אופורטוניסטי בעזרת שימוש במערכת שעיצבנו.

אנחנו בוחרים להשתמש במתודולוגיית מחקר איכותני ובשיטת מחקר מבוססת עיצוב. מתודולוגיית מחקר איכותני יכולה להתמודד עם מורכבויות שנובעות לא רק מבעיות טכניות בפיתוח תוכנה, אלא מאינטראקציות אדם ומכונה, והכי חשוב מהתנהגות אנושית בפיתוח תוכנה.

סקר בנושא פעילויות בעת שימוש בדוגמאות בקרב מפתחים מקצועיים, מהווה את הבסיס לתכנון ולעיצוב עבור מערכת המלצת קוד מבוססת מדיה חברתית. אנו עוקבים אחר החלטות העיצוב שלנו ומפתחים את Example Overflow, מערכת חיפוש והמלצת קוד המשלבת מדיה חברתית ומערכות המלצת קוד.

בעזרת Example Overflow, ערכנו ניסוי, הכולל תצפיות וראיונות עם מפתחים מקצועיים אשר התבקשו לפתור משימות שנקבעו מראש. המחקר מגלה כי למפתחי תוכנה מקצועיים ישנם חששות הקשורים לשימוש בדוגמאות, חששות אשר קובעים איך ומתי הם ישתמשו בדוגמאות. עם זאת, מפתחים לא נמנעים משימוש בדוגמאות, אלא מנסים למזער חששות אלה בעזרת מיקרו-פעילויות.