Research Statement

Huascar A. Sanchez

February 07, 2015

The Web has changed the dynamics of programming. We are in an era where copying and pasting code from the Internet is no longer a bad thing; just an observation of how much of modern programming gets done. In this era, source code volumes on the Internet are increasing exponentially. The spread of Web technologies magnifies this growth by allowing millions of people to share their work (code examples) with the world; e.g., blogs, forums, etc. Consequently, the Web has become the largest repository of software development knowledge in human history.

As such, locating useful code examples in such a large repository is *laborious and challenging*. It requires some effort and some elbow grease to scavenge the right code example for the required task. This is particularly true when the means and methods programmers employ to scavenge code on the Internet are informal and ad-hoc. So, despite the progress in searching through code examples, programmers still have trouble digesting the ones they find. This is caused particularly by the examples' questionable quality [3]. Consequently, if we are to be more effective in locating useful code examples, then we must also address their questionable quality upfront. It is until then we'll be able to make sense of them properly and then decide whether to reuse them.

My research focuses on such proposition by introducing the notion of source code curation. I introduced this notion to cover the act of discovering a code of interest, cleaning and transforming (refining) it, and then presenting it in a meaningful and organized way. My dissertation explores the development of source code curation tools. These tools aim at enabling code foragers to cope with code examples' questionable quality upfront, and to gain better understanding in the curated examples.

My dissertation addresses three research questions: *How should interfaces*

for curating source code work? How will programmers use these interfaces? Will programmers be able to understand the found code more effectively and efficiently, via source code curation?

1 Source Code Curation Interfaces

Programmers typically turn to examples during software development, but finding relevant examples for a particular task can be difficult. Although questions and answer sites like StackOverflow dominate code example searches and retrieval, these sites offer little support for curation. As a result, programmers are relegated to copying and pasting code examples and then trying them in their own environments as a way to understand them. This can be a painful task; especially if one didn't comprehend *in advance* the tradeoffs of using any foreign code.

To tackle this problem, I developed *Vesperin*, a source code curation system for Java code examples [3]. At a high level, *Vesperin* consists of two main components: a Chrome Extension (named Violette) for allowing developers to edit and change the code examples in the web page of the Q & A system (in-place), and a RESTful service (named Kiwi) for managing curation and code example parsing operations. With *Vesperin*, code foragers will be able to comprehend in advance some of the tradeoffs of using code examples.

Vesperin provides a mechanism by which a developer can examine Java code examples, on StackOverflow, through a combination of manual and semi-automatic edits. By doing this, developers can (1) cope with examples' questionable quality upfront, and (2) better understand the ones that go through a curation process.

2 Distilling Code Examples' Essence

The premise behind example-driven reuse is that code foragers reviewing code examples are mentally "anxious". They want to get their work done immediately; and they want to grasp the example's main abstractions as soon as possible, so they can be on their way [2]. Code examples, therefore, must not obstruct the natural impulses of code foragers. Whenever possible, their essence should be distilled quickly, so that code foragers can consume

their abstractions.

While such distilling code examples' essence can be invaluable to code foragers, it is still cumbersome because currently distilling decisions are still performed manually. To tackle this problem, I developed a code example multi-stager. A new feature of the *Vesperin* System. This multi-stager uses static analysis and code transformations [2] to distill the essence of the example.

The multi-stager divides code examples into a series of intermediate stages or concrete chunks (e.g., pieces of functionality). Whenever possible, each stage builds upon, and in relation to, preceding stages; therefore, each stage may increase the complexity of the overall example. Multi-staging enables *out of order access and exploration of the example*, helping code foragers complete their code understanding tasks quickly and more accurately.

Behind the scenes, the multi-stager uses an algorithm (called Method Slicing) which identifies the minimal set of declarations (e.g., used methods, classes, fields), while resolving unknown referents that are required for each stage to compile properly. Additionally, the algorithm takes into consideration whether the code example is partial or complete and adapts its processing accordingly.

3 Code Example Summarization Model

Often during code foraging, developers wish to skim large code examples rather than read them in depth. This dynamic raises new opportunities for summarizing code. While summarization can be invaluable to code foragers, it is also difficult to automate, and must be done manually [1].

Inspired by the state of the art in document summarization, I cast the problem of code example summarization as an optimization problem. More specifically, I formulated it as a Precedence Constrained Knapsack optimization problem and made use of a dynamic programming algorithm to solve it [1]. I call this algorithm 'SIngle Code Example Summarization' (*SICES*).

SICES generalizes the code example summarization problem, taking into account a precedence relation between code blocks in a code example. A 'Directed Acyclic Graph' represents such a relation, where nodes correspond to code blocks in a one-to-one fashion. Each code block has a profit (e.g., the usage frequency, throughout the example, of the set of statements contained in the code block over the number of elements in the block) and a weight

(e.g., the number of lines of code in the code block). SICES' Knapsack has a fixed capacity (i.e., total number of lines of code to be displayed in the summary). So, given a code example, and a budget (e.g., lines of code to display), *SICES* automatically creates a code summary. It does it by identifying the location of non essential code blocks; i.e., those code blocks that if added to the solution would exceed the fixed Knapsack's capacity.

SICES aims at helping with rapid understanding of code examples; provided that summary decisions are now done automatically. Its functionality is available in the *Vesperin* system and uses Violette's scratchspace autofolding feature to visualize the produced summaries.

4 Research Agenda

In future research, I will continue my focus on software engineering in general and on tools for search driven development in particular.

My research agenda comprises two main themes: exploring different ways of interweaving code retrieved through search and/or through curation into a programmer's partially completed program, and analyzing the resulting code to make sure the retrieved code did not introduce any bugs or security flaws.

4.1 Interweaving Retrieved Code

Just like graphic designers, using graphics editors like Photoshop, appreciate simple and usable techniques for representing design alternatives, and work in progress. Code foragers would enjoy techniques for creating and maintaining development alternatives during code retrieval. These techniques aim at allowing code foragers to work with multiple code alternatives in their development environment before settling on any one in particular.

4.2 Verifying Completed Code

Leveraging latest techniques in data mining and crowdsourced program verification to analyze repositories of open source code. I will detect and record common practices (e.g., idioms) used in popular object oriented languages. Through this knowledge, I will be able to identify code that is unlikely to occur in practice and may constitute bugs or security flaws.

References

[1] H. Sanchez and J. Whitehead. Code Example Summarization Model based on The Precedence-constrained Knapsack Problem. Work in progress.

[2] H. Sanchez and J. Whitehead. Multi-staging Stackoverflow Code Examples via Method Slicing. Work in progress.

[3] H. Sanchez and J. Whitehead. Source Code Curation on Stackoverflow: The Vesperin System. In *The 37th International Conference on Software Engineering*, ICSE '15, 2015. (to appear).