Source Code Curation on StackOverflow: The Vesperin System

Huascar Sanchez
Computer Science Department UC Santa Cruz,
Santa Cruz, CA 95060
hsanchez@cs.ucsc.edu

Jim Whitehead
Computational Media Department UC Santa Cruz,
Santa Cruz, CA 95060
ejw@soe.ucsc.edu

***Abstract*—The past few years have witnessed the rise of software question and answer sites like StackOverflow, where developers can pose detailed coding questions and receive quality answers. Developers using these sites engage in a complex code foraging process of understanding and adapting the code snippets they encounter. We introduce the notion of source code curation to cover the act of discovering some source code of interest, cleaning and transforming (refining) it, and then presenting it in a meaningful and organized way. In this paper, we present Vesperin, a source code curation system geared towards curating Java code examples on StackOverflow.**

I. Introduction

Locating a useful code example on StackOverflow is laborious and challenging. It is laborious, because it often requires multiple rounds of specific steps (e.g., browsing, screening, filtering) to find the best code example [1]. It is challenging, because the quality of the content found in those examples is typically questionable. Some of it is poorly structured, incomplete, and contains ambiguous declarations [2], [3].

Despite any productivity gains in searching through code examples [4]–[6], developers still require time and effort to effectively digest the examples they encounter on question and answer (Q&A) sites. Consequently, if we are to be more effective at locating useful code examples, then we must also address their challenging nature upfront. We address this nature by introducing the notion of *source code curation*. Inspired by Stonebraker [7], this notion covers the act of discovering a source code of interest, cleaning and refining it, and then presenting it in a meaningful and organized way. This umbrella term aims at forging new relationships among activities such as code recommendation [8], code documentation generation [3], [9], and code retrieval [5]. They all play a role in curation: from recommending the best *ranked* answers to allowing frequent documentation and code edits that keep answers fresh and relevant.

Our goal is to help developers cope with code examples' challenging nature upfront, and to gain better understanding in curated examples. We accomplish this by allowing developers to explore code modification ideas in the web page of the Q&A system (in-place). By investing in such a process, developers can intuitively experiment with ideas hands-on before consumption, and thus unlock know-how, leading to more confidence in the examples.

To explore the extent and challenging nature of code examples on StackOverflow, we collected evidence from 50,000 Q&A pages accumulated over a 4 year span, and studied them in detail [1]. Our criteria for collecting these pages were simple: (1) each answer must contain *code examples*; (2) it should target a particular programming language (Java); and (3) it should be an accepted answer.

We found that *25,472 (51%)* of the studied Q&A pages were described by single snippets, which were typically short, poorly structured, and incomplete. Of these, the median number of lines of code is *8* and *75%* of them have less than *19* lines of code. Moreover, *99%* of single snippets were unparsable. Additionally, the remaining *49%* were described by multiple snippets. The average number of lines was *45*. Similar to the snippets in the (51%) group, they were typically incomplete, and thus unparsable. These characteristics raise challenges for developers to digest unfamiliar code on StackOverflow.

We address these characteristics in a single coherent source code curation system, called *Vesperin*. At a high level, *Vesperin* consists of two main components: a Chrome Extension (named *Violette*) for allowing developers to actually edit and change code examples in-place, and a RESTful service (named *Kiwi*) for managing curation and snippet parsing operations. Together, they provide a mechanism by which developers can examine code examples through a combination of manual and semi-automatic edits. By doing this, developers can (1) cope with the mentioned characteristics upfront, and (2) better understand the curated code.

A. Demonstration Structure

We present two scenarios that show how to use the *Vesperin* system. The first scenario familiarizes the audience with its main interface. It then delves into specific features of *Vesperin*: (1) application of code transformations, (2) adding notes to specific code sections, and (3) entering the preview mode. The last scenario demonstrates where and how the curated code examples are published and shared.

The remainder of the paper is organized as follows: Section 2 briefly describes the *use model* that *Vesperin* implements; Section 3 describes its architecture; Section 4 presents Vesperin's experimental validation. Section 5 concludes.

[1]Our data set, and results are available at http://goo.gl/6IbfVi

978-1-4799-1934-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICSE.2015.217

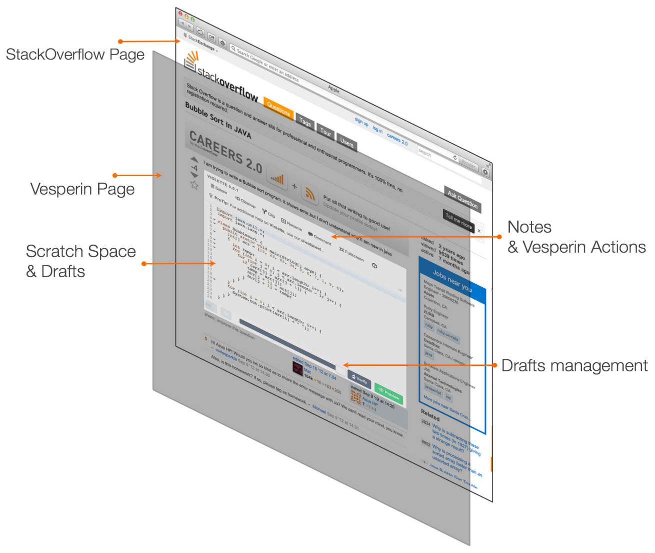

Fig. 1. *Vesperin* Use Model and the connection of a *Vesperin* page to a StackOverflow page. A *Vesperin* page is pictured as a layer (i.e., scratch space) placed on top of a Q&A page containing Java code examples. On that layer, developers can curate the enclosed examples.

II. Vesperin Use Model

Figure 1 shows the *Vesperin Use Model* and the connection of a *Vesperin* page to a StackOverflow page. This model is described in this section.

A. Scratch Space and Drafts

Vesperin assures that code blocks containing Java code have their own scratch space. A *scratch space* is the space where developers make all the edits; either manually via direct editing or semi-automatically via *Vesperin*'s built-in operations. An auxiliary interface controls the version structure of a code example affected by these edits.

When performing edits on the scratch space, major versions (or drafts) of code examples are automatically marked. We mark these drafts to provide a view of how the example has changed over time. A draft is a named set of edits. Edits include insertions or deletions of text. Drafts are final; i.e., they can't be deleted.

B. Other elements

Additionally, as the developer is curating the example, there are times when his/her edits need more context. The developer can give this context via notes. A note is used to enter comments, and could be used either to summarize some code or outline its intention. Ultimately, notes provide meaningful feedback to those trying to understand a draft. For all these reasons, *Vesperin* supports notes.

Using notes in the above manner may present a fundamental tension. Either we consider keeping notes connected to a specific global point in the text (which can change in future drafts), or we consider keeping notes local to their corresponding draft; i.e., not available to other drafts. The latter turned out to be simpler and is our preferred solution. In future versions, we may explore the former strategy.

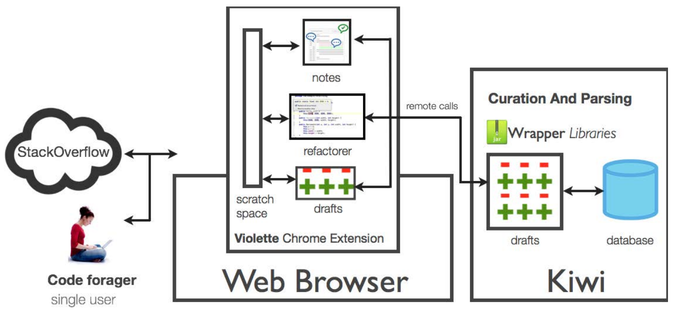

Fig. 2. *Vesperin*'s architecture.

C. Vesperin Actions

When using the scratch space, the developer modifies a code example directly by editing it. At any time, during the editing step, this individual may trigger a semi-automatic code transformation, or refactoring. *Vesperin* follows this action by issuing a remote call to *Kiwi* requesting an update to the current draft. *Kiwi*'s reply contains either a new draft (updated code), or a message describing a failed updating attempt. At any time during editing, the developer can verify if the current draft is syntactically correct (i.e., no compiler errors).

Lastly, as previously discussed, the scratch space keeps a history of all created drafts. At any time, the developer may view any previous point in the curation process.

D. D.R.Y.ING Source Code Curation

In some cases, the code examples of a set of Q&A pages may have been curated by other developers, and continue to be viewed by others. In this case, *Vesperin* can present revisions of a curated code example to a new developer. We keep unique records of all curated code examples, including their revisions, in a database. The main identifier for a curated code example is created by combining the StackOverflow Q&A page URL with the ID of the accepted answer that provided the original code snippet. In the current version of *Vesperin*, revisions are tracked but not displayed to the developer. Adding this is planned future work.

III. Vesperin's Architecture

Figure 2 shows *Vesperin*'s architecture. Indicated in the figure are *Violette* and *Kiwi* components. *Kiwi* provides a RESTful interface to *Vesperin* back end services. We describe *Violette* in this section.

A. Violette Chrome Extension

Violette is *Vesperin*'s companion Chrome extension. When installed, *Violette* detects Java code blocks on StackOverflow and then transforms them into a form that allows developers to actually edit and change the code in-place. We implemented Violette using the Javascript programming language.

Out of the box, *Violette* supports the following operations.

- Programming language detection. It combines disambiguation heuristics[2] with other heuristics used by syntax highlighters to guess the language of a code example.
- Code transformations and refactorings. It supports a few code transformations and refactoring strategies including Rename refactoring, deduplication, etc. The full list of strategies can be found at http://goo.gl/Rzslhz. With the exception of deduplication, which analyses the entire code in search for clones [10] to remove, each of them analyze the scope of a code selection, and use this information to update the structure of a code example.
- Content annotation via notes. Notes can highlight specific code within the scratch space. They are in context.
- Lightweight drafts management. It provides an auxiliary interface (i.e., Edit Tracker) to control the version structure of an edited example. This interface tracks marked drafts; produced when using the scratch space.
- Syntax correctness verification. It provides a mechanism for verifying if code examples are syntactically correct.
- Preview mode. It provides the option of scrubbing the edited Q&A page of distractions and provides a clean view for examining the curated code example.

IV. Experimental Validation

Our initial evaluation sought to establish evidence for the feasibility of the *Vesperin* end-to-end approach for curating code examples. We asked people from both academia and industry to participate in this study. Our evaluation considered the following three concrete questions:

1) How will developers use these source code curation interfaces?
2) Will the set of provided facilities be sufficient for curating code snippets?
3) Will developers be able to better understand the unfamiliar snippets via curation?

A. Method: User Study

Our study consisted of two parts: (1) a pre-study questionnaire, to measure the development experience of the participants, and (2) a pretest-posttest design [11], to measure both the participants' expectations prior to using *Vesperin* and the experience subsequent to using it. In this section, we describe the study and then summarize the study results.

1) Pre-study Questionnaire: We recruited 15 participants, 6 from the computer science department of UC Santa Cruz, 5 from local software companies, and 4 independent consultants. We required participants to have a college degree in computer science or related discipline. Figure 3 summarizes participants' background information.

2) Pretest-Posttest Design: The second part of the study consisted of having participants work on a set of source code curation tasks for 50 minutes (including a 10 minute break). A single task consisted of four parts:

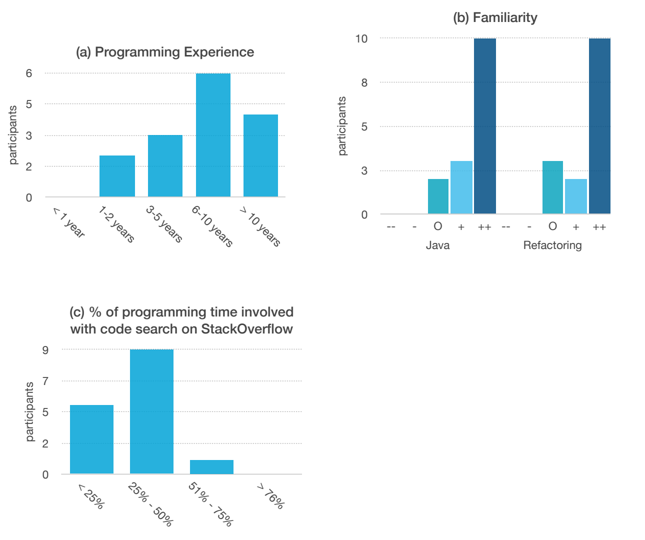

Fig. 3. Summary of Participants' Background Information. On Figure 3b, the horizontal axes are 5-point Likert scale, ranging from "Not at all familiar" ("–") to "Extremely familiar" ("++"). The vertical axes are the number of participants.

1) Use a search application developed for the experiment to find Java code examples of sorting algorithms.
2) Select a result from the set of returned results.
3) Examine the content of selected result; with and without source code curation.
4) Consume its information by describing its intent.

The focus of this part of the study was twofold. First, to observe participants as they work on the tasks. This would answer questions 1 and 2. Second, to collect data on the participant's perception of the benefits of *Vesperin* before and after using it via a survey[3]. Particularly, we focused on evaluating two attributes: (1) Better understanding; and (2) minimal value. This would answer question 3.

B. Findings

Our findings cover how the participants used *Vesperin*, including the evaluation of the attributes mentioned in the previous subsection.

1) How will developers use these source code curation interfaces?: We posed this question in order discover what usage strategies participants employed when using our system and evaluating their mental model representations.

Participants relied on a hybrid comprehension strategy; mixing bottom-up and top-down comprehension strategies [12]. As participants were trying to comprehend the code, they often tried to verify whether it was syntactically correct. For some participants, such feedback gave them some reassurance of the validity of the code. For others, however, the effect of such feedback was the opposite. During editing and verifying, new and old compilation errors sometimes kept occurring.

[2]As seen on https://github.com/github/linguist/

[3]Survey can be accessed online at http://goo.gl/XB5281

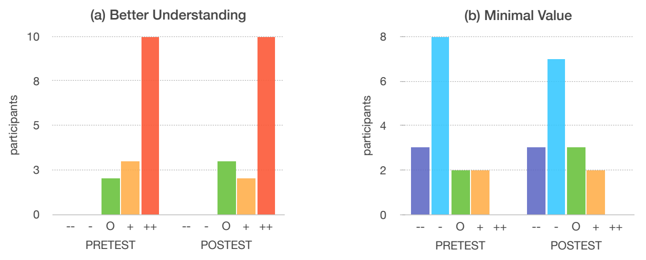

Fig. 4. Distribution of participants' expectations before (Pretest), and experiences after (Posttest) using our tools. Horizontal axes: 5-point Likert scale, ranging from strongly disagree ("–") to strongly agree ("++"). Vertical axes: number of participants.

Consequently, they felt frustrated for not making sufficient progress after a few edits.

Such frustration caused some participants to change their answers to "neither agree nor disagree" to the questions on better understanding and minimal value (See Figure 4). Rather than focusing on trying understand the example, they were consumed with verifying a series of impromptu changes. Consequently, they derailed from the asked task; making their answers outliers (not representative of the overall collected data). To prevent any future obstruction caused by the code verification feature, we decided to relegate it to Violette's backend logic.

We also observed participants exploring control flow relationships via text find and replace actions, followed by adding notes and cleanup operations. We later found out that the mechanics used by some of these participants matched the mechanics of some of Violette's supported refactorings. When asked why they did not use this functionality, they answered that Violette looked like an editor. Consequently, it was natural to them to edit code this way.

2) Will the set of provided facilities be sufficient for curating code snippets?: The principal realization from this experiment is that participants employed a set of *Violette* operations to curate examples. With those operations, they were able to cope with some of the characteristics we described in the introduction of this paper. However, given the other unused operations, we concluded that a small set of operations were necessary, but not sufficient for curating different types of code examples. For example, we lacked the ability to cope with multiple independent classes simultaneously in a single scratch space. Or linking multiple scratch spaces that should be considered in concert (e.g., tutorials). These are limitations observed by a few participants.

3) Will developers be able to better understand the unfamiliar snippets via curation?: Figure 4 shows the results of the pretest and posttest. Excluding the case where participants lost focus on the task (Section IV-B1), the pretest and posttest results are similar. Despite all issues, they came in with a positive expectation and *Vesperin* did not disappoint. It added value and better understanding as predicted.

The majority of participants indicated that *Violette* could help them understand the snippets more effectively (See Figure 4a) with the current operations. When asked whether the system added minimal value, they disagreed (See Figure 4b). They felt it did provide value in better understanding of unfamiliar code examples.

V. Conclusions

The main contributions of this paper are (a) introducing the notion of *source code curation*, (b) a system (*Vesperin*) that provides curation support for developers, and (c) preliminary evaluation of this system and the core idea of curation. The majority of participants felt they were able to better understand the unfamiliar code examples via curation. Furthermore, they felt the system provided value.

Acknowledgment

We would like to thank the members of both Software Introspection and Augmented Design labs at UC Santa Cruz for their valuable feedback.

References

[1] G. Marchionini, "Exploratory search: from finding to understanding." *Communications of the ACM*, vol. 49, no. 4, pp. 41–46, 2006.

[2] S. Subramanian and R. Holmes, "Making sense of online code snippets," in *MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 85–88.

[3] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014, 2014, pp. 643–652.

[4] O. Barzilay, "Example embedding," in *Proceedings of the 10th SIGPLAN Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2011, 2011, pp. 137–144.

[5] D. Wightman, Z. Ye, J. Brandt, and R. Vertegaal, "Snipmatch: Using source code context to enhance snippet retrieval and parameterization," in *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '12, 2012, pp. 219–228.

[6] L. Ponzanelli, A. Bacchelli, and M. Lanza, "Seahawk: Stack overflow in the ide," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13, 2013, pp. 1295–1298.

[7] M. Stonebraker, D. Bruckner, I. Ilyas, G. Beskales, M. Cherniack, S. Zdonik, A. Pagan, and S. Xu, "Data curation at scale: The data tamer system," in *Proceedings of CIDR'13*, 2013.

[8] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza, "Mining StackOverflow to turn the IDE into a self-confident programming prompter," in *MSR 2014: Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 102–111.

[9] C. Parnin, C. Treude, L. Grammel, and M.-A. Storey, "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow," http://www.cc.gatech.edu/~vector/papers/CrowdDoc-GIT-CS-12-05.pdf, Georgia Institute of Technology, Tech. Rep., 2012.

[10] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '98, 1998, pp. 368–.

[11] E. Babbie, *The Practice of Social Research*. Cengage Learning, 2012.

[12] C. L. Corritore and S. Wiedenbeck, "An exploratory study of program comprehension strategies of procedural and object-oriented programmers," *Int. J. Hum.-Comput. Stud.*, vol. 54, no. 1, pp. 1–23, 2001.