Seahawk: Stack Overflow in the IDE

Luca Ponzanelli, Alberto Bacchelli, Michele Lanza
REVEAL @ Faculty of Informatics – University of Lugano, Switzerland

***Abstract*—Services, such as Stack Overflow, offer a web platform to programmers for discussing technical issues, in form of Question and Answers (Q&A). Since Q&A services store the discussions, the generated "crowd knowledge" can be accessed and consumed by a large audience for a long time. Nevertheless, Q&A services are detached from the development environments used by programmers: Developers have to tap into this crowd knowledge through web browsers and cannot smoothly integrate it into their workflow. This situation hinders part of the benefits of Q&A services.**

To better leverage the crowd knowledge of Q&A services, we created Seahawk, an Eclipse plugin that supports an integrated and largely automated approach to assist programmers using Stack Overflow. Seahawk formulates queries automatically from the active context in the IDE, presents a ranked and interactive list of results, lets users import code samples in discussions through drag & drop and link Stack Overflow discussions and source code persistently as a support for team work.

Video Demo URL: http://youtu.be/DkqhiU9FYPI

I. Introduction

Developers spend most of their programming time on software maintenance, which is estimated to impact between 85% and 90% of the global cost of a software system [1], [2]. Up to 50-60% of this maintenance time is spent on program comprehension [3]. Clear, comprehensive, and updated software documentation would be an effective approach to reduce time spent in program comprehension. However, developers report how documentation is commonly inadequate, outdated, and hard to retrieve or link to actual source code entities [4] (open source development projects are similarly affected by documentation related issues [5]). Moreover, software developers are introduced and must remain updated on new technologies and ideas [6].

Trying to tackle this documentation and knowledge sharing issue, Q&A services, such as Stack Overflow, offer a web platform to programmers for discussing technical issues, so that they can share their knowledge and solve problems with undocumented public libraries, unclear programming tasks, or new technologies or frameworks to explore. In practice, developers pose questions and receive answers regarding issues from people that are not part of the same project, but might be more knowledgeable about a specific topic. Even though researchers pointed out that Q&A services could not provide high level technical answers [7] [8] [9], these services are "filling archives with millions of entries that contribute to the body of knowledge in software development" and they often become the substitute of the official product documentation [10] (*e.g.,* the developers of the open source project Aptana[1] store their official documentation as Q&A discussions).

[1]http://www.aptana.com/

Despite Q&A services being broadly used and deemed useful for practical programming tasks, they are currently isolated from the integrated development environments (IDEs) that programmers use in their daily activities, and where they spend most of their working time [4]). In fact, the web browser is the only gate to Q&A crowd knowledge: There is no integration with IDEs or programming and team workflow.

We claim that this status hinders the benefits brought by Q&A services for a number of reasons, such as:

(1) the quality of the results returned by Q&A service relies on the quality of queries manually formulated by developers [11], who must accurately phrase the meaning of their current programming task into useful terms;

(2) developers have no support for sharing functional Q&A discussions (along with the reference to the context where the discussion is valuable within the project they are developing) with other team members, and cannot archive these discussions for later reference and documentation;

(3) developers have to switch the context back and forth between the IDE and the web browser, while they should be focused only on their current task without interruptions or disturbance [12] to avoid wasting time.

To tackle these problems, we propose Seahawk[2], a recommendation system [6] (implemented as an Eclipse plugin) that integrates the crowd knowledge of Q&A services within the IDE. In particular, Seahawk mines the knowledge base of Stack Overflow[3], which is a notable example of technical Q&A service that gained popularity among developers and is an important venue for sharing knowledge on software development [9]. In Stack Overflow more than 92% of the questions on expert topics are answered in a median time of 11 minutes [9] and it is deemed to be very effective for "for code reviews, for conceptual question and for novices" [10].

Seahawk gives users the support to:

(1) formulate queries automatically from the active IDE context (by extracting keywords from the chosen code entities),

(2) view directly in the IDE a ranked list of related Q&A discussions and interact with them,

(3) import code samples in discussions through drag & drop,

(4) connect Stack Overflow discussions to code artifacts and store the link persistently.

Structure of the paper. In Section II we detail Seahawk and its user interface, we present a use case scenario in Section III, and describe its data-collection mechanism and recommendation engine in Section IV. In Section V we sum up our contributions.

[2]http://seahawk.inf.usi.ch
[3]http://stackoverflow.com

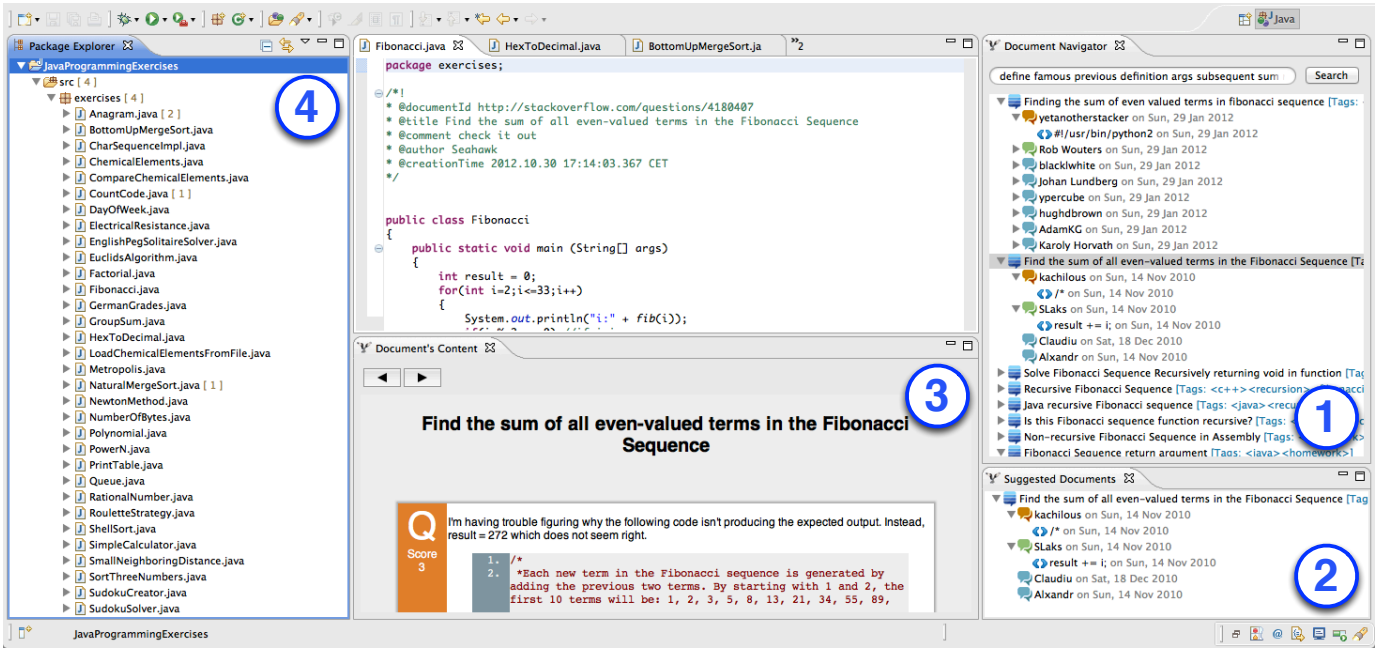

Fig. 1: Seahawk User Interface

II. Seahawk

Figure 1 shows the user interface (UI) of Seahawk. Users can interact with Seahawk through four main components:

1) **Document Navigator View:** In this view (Point 1 in Figure 1) developers compose queries—in the text field—and retrieve documents, which are displayed in a tree view. Developers can navigate nodes of discussion (*i.e.*, question or answers), and drag&drop documents or code snippets into the code editor. Once a document is dropped in the editor, Seahawk shows a dialog (Figure 2) to let users put a comment to explain the connection between the document and the code. Subsequently it generates the annotation in the code editor to support coordination and for later reference.

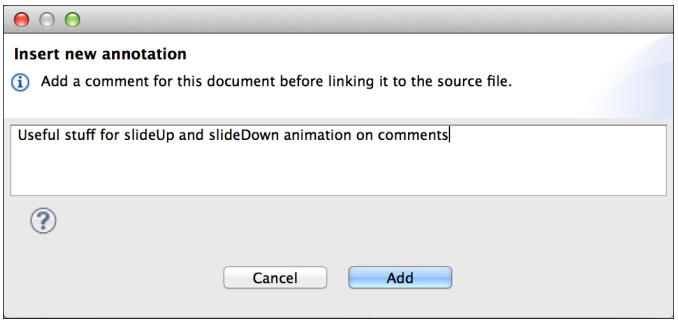

Fig. 2: Seahawk dialog for annotation's comment

2) **Suggested Documents View:** This view (Point 2) shows the documents linked to the code editor currently active. Whenever a code editor tab becomes active, the view asks the annotation engine to parse the code and retrieve the documents. Similarly to the *Document Navigator View*, this view shows a tree view to navigate the documents. Documents removed from the search engine (because removed from Stack Overflow) are not traversable (Figure 3) and are prefixed by the message "Not Available" in the document's title. Users can modify comments of annotation, or delete them, through a contextual menu.

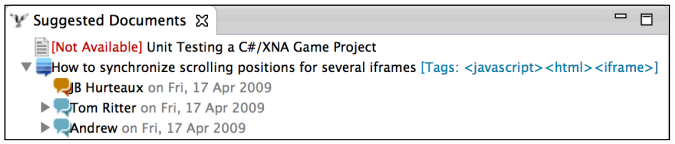

Fig. 3: Not available document in Seahawk's view

3) **Document Contents View:** When a document or a node is selected in one of the aforementioned views, this view displays its contents by using a custom layout in an embedded web-browser widget. This widget allows developers to navigate the links contained in the document and to get additional information. We use a Javascript library[4] for multi-language syntax highlighting of the text in <*code*> tags. Questions are in orange, the accepted answer is in green, and other answers are in blue.
4) **Notification System:** To quickly spot new annotations in the project, we implemented a notification system in the package explorer (Point 4). Seahawk decorates files in the package explorer by putting the number of new annotations between square brackets.

[4]http://code.google.com/p/google-code-prettify/

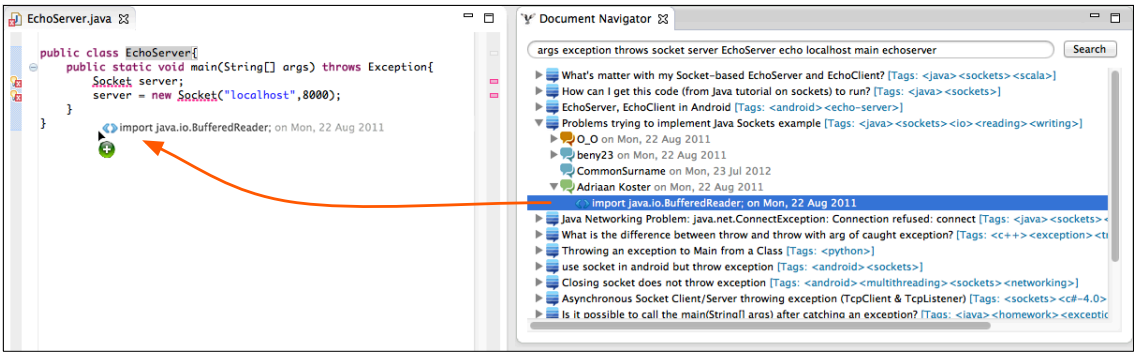

Fig. 4: Alice imports the code snippet in the code editor.

III. A Use Case Scenario

By means of a simple scenario, we illustrate how Seahawk can help developers solving programming problems by leveraging Stack Overflow from within the Eclipse IDE.

Alice is a student required to build an *echo server* in Java. The server handles one client at a time and terminates itself whenever a client sends the "`quit`" string. To start, Alice opens up the Eclipse IDE, with the Seahawk plugin installed, and begins creating the class *EchoServer*. She first creates a socket by using the *Socket* class:

Listing 1: Initial Implementation of an Echo Server

```
public class EchoServer{
    public static void main(String[] args){
        Socket server;
        server = new Socket("localhost",8000);
    }
}
```

Alice looks at the methods trying to understand how to accept incoming connections. Since she does not find any method to accomplish this task, she invokes Seahawk, which analyzes the existing code, builds a query, retrieves a set of documents related to what is written in the *EchoServer* class, and visualizes them in the *Document Navigator View*. Alice inspects the documents by reading questions and answers. Every time she moves to a specific node of a document, the content is visualized inside the *Document Content View*. Among the documents, Alice finds a question titled "Problems trying to implement Java Sockets". She reads the document and finds an accepted answer that proposes the implementation of a simple echo server. She understands that the right class to be used is *ServerSocket*, instead of *Socket*. Thanks to the document navigation system of Seahawk, Alice locates the code snippet and drags it into the code editor, importing it (see Figure 4). Alice can now modify the code in the editor to achieve the desired outcome. With minor modifications she adapts the imported snippet and makes the server able to terminate when receiving a `quit` string from a connected client.

To conclude, Alice wants to bookmark the original solution directly in the code, so that she can retrieve it later and show it to her classmates. To this aim, she drags the document in the editor. When Alice drops the document, Seahawk asks her to put a comment by means of a dialog box. After Alice types the comment and confirms, the annotation becomes visible in the code editor, as a special comment. Subsequently, the *Suggested Documents View* shows that a document is linked to the source code. Moreover, every other person opening the file with the Seahawk plugin installed will be notified about the bookmark in the *Suggested Documents View*.

IV. Behind the Scenes

According to the definition given by Robillard *et al.* [6], we present the components forming recommendation systems: the *data collection mechanism* and the *recommendation engine*.

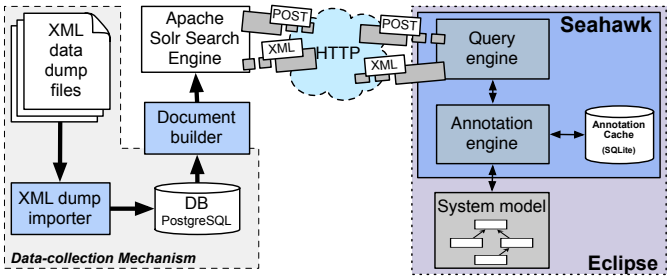

Fig. 5: The architecture of Seahawk

Data Collection Mechanism. The Data Collection Mechanism component is responsible for gathering Q&A data from Stack Overflow. We import Stack Overflow documents from the public data dump provided as a set of XML file[5]. The data is extracted through a *XML dump importer* and stored in a relational database for performance reasons. We built a tool to query the database and build a JSON representation of each document (thus making it available to any programming language). The representation is then included in an additional document schema, as required by the Apache Solr[6] search engine. Documents are indexed by a variation of the standard *tf-idf* and are available for queries through a RESTful interface as soon as the indexing phase is finished.

[5]blog.stackoverflow.com/2009/06/stack-overflow-creative-commons-data-dump/
[6]http://lucene.apache.org/solr/

The Recommendation Engine. The recommendation engine provides manual and automatic interactions. The core is composed of a query engine and an annotation engine.

The Query Engine: This component is responsible of building queries and communicating with Apache Solr. The engine builds the query, according to the syntax of Apache Solr, in a way that every token must be present in the document field or at least one of those is contained in the title field. The overall relevance of a document is determined by the relevance of its body and its title. Documents whose title is interesting for the given query are retrieved even if the document's body does not match any of the tokens. **Automation of Queries:** Our query engine also provides an automatic keyword extraction feature to build queries. The extraction does not rely on the Abstract Syntax Tree (AST) to analyze the code, but it relies on island parsing, which can recognize structural information (*i.e.,* classes and methods) even though the code does not compile. The identified code entities are treated as text and analyzed as natural language. The target entity is defined by the cursor position in the text editor: The nearest entity is picked as target entity. Once the entity is selected, the query is built by merging the obtained keywords in two ways:

Processing the entity's body: After we apply some basic information retrieval techniques (*e.g.,* removing stop-words, split on change case), we extract the ten most frequent keywords in the body. To this set, we add the name of the entity.

Analyzing the import statements: We take all the imports statement used by an entity and we extract keywords from them by splitting on the "." character. The set of tokens obtained, together with the set obtained from the entity's body, becomes part of the query.

The Annotation Engine: The annotation engine allows the creation of links between source code and documents. Links are represented through annotations in the code, so that we automatically take advantage of versioning systems already in place without requiring additional infrastructure. We use a structure for the annotations similar to Doxygen[7], but developers can define custom delimiters (so that it can be language independent), and to avoid conflicts with Doxygen or JavaDoc annotations, we decided to put an exclamation mark as last character for the opening delimiter. Listing 2 shows an example of an annotation generated by Seahawk.

Listing 2: Example of an annotation generated by Seahawk

```
/*!
* @documentId <Document's Id>
* @title <Document's title>
* @comment <Author's comment>
* @author <Autor's name>
* @creationTime <creation date>
*/
```

The annotation engine also provides a notification system to keep track of annotations already seen by developers. To this aim, we use two different ways of parsing code: (1) We take advantage of Eclipse's partitioning system, and (2) we implemented our own parser for annotations.

[7]www.doxygen.org/

The partitioning system identifies code blocks (partitions) that match specific delimiters in the code editor (*e.g.,* comment, classes, methods *etc.*) every time a source file is opened or modified in the code editor. The identified links are shown in the *Suggested Documents View* and put in a cache. If there are new annotations for file not already opened, they are notified to the UI notification system (see Section II).

V. Conclusion

We presented Seahawk, an Eclipse plugin to leverage the crowd knowledge provided by Q&A services. We detailed how it lets users interact with Stack Overflow documents in a novel way. Users can import code snippets and create links between documents and source code by means of language-independent annotations, and they can use annotations to take advantage of the versioning system to collaborate and suggest documents to teammates. Seahawk automatically generates queries from code entities, and we explained how it deals with import statements and uncompilable code to extract keywords from Java entities.

Acknowledgements

We thank the Swiss National Science foundation for the financial support through SNF Project "SOSYA", No. 132175.

References

[1] L. Erlikh, "Leveraging legacy system dollars for e-business," *IT Professional*, vol. 2, no. 3, pp. 17–23, 2000.

[2] R. C. Seacord, D. Plakosh, , and G. A. Lewis, *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley, 2003.

[3] T. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal ()*, pp. 294–306, 1989.

[4] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of ICSE 2006 (28th ACM International Conference on Software Engineering)*. ACM, 2006, pp. 492–501.

[5] T. Gleixner, "The realtime preemption patch: Pragmatic ignorance or a chance to collaborate?" in *Keynote of ECRTS 2010 (22nd Euromicro Conference on Real-Time Systems)*, 2010, http://lwn.net/Articles/397422/.

[6] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, pp. 80–86, 2010.

[7] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman, "Knowledge sharing and yahoo answers: everyone knows something," in *In Proceedings of WWW 2008 (17th international conference on World Wide Web)*. ACM, 2008.

[8] K. K. Nam, M. Ackerman, and L. Adamic, "Questions in, knowledge in?: a study of naver's question answering community," in *In Proceedings of CHI 2009 (27th international conference on Human factors in computing systems)*. ACM, 2009, pp. 779–788.

[9] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann, "Design lessons from the fastest q&a site in the west," pp. 2857–2866, 2011.

[10] C. Treude, O. Barzilay, and M.-A. Storey, "How do programmers ask and answer questions on the web? (nier track)," in *Proceedings of ICSE 2011 (33rd International Conference on Software Engineering)*. ACM, 2011, pp. 804–807.

[11] S. Haiduc, G. Bavota, R. Oliveto, A. Marcus, and A. D. Lucia, "Evaluating the specificity of text retrieval queries to support software engineering tasks," in *Proceedings of ICSE 2012 (34nd International Conference on Software Engineering)*, 2012, pp. 1273–1276.

[12] J. Raskin, *The Humane Interface - New Directions for Designing Interactive Systems*. Addison-Wesley, 2000.