Harnessing Stack Overflow for the IDE

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

***Abstract*—Developers often consult online tutorials and message boards to find solutions to their programming issues. Among the many online resources, Question & Answer websites are gaining popularity. This is no wonder if we consider a case like Stack Overflow, where more than 92% questions on expert topics are answered in a median time of 11 minutes. This new resource has scarcely been acknowledged by any Integrated Development Environment (IDE): Even though developers spend a large part of their working time in IDEs, and the usage of Q&A services has dramatically increased, developers can only use such resources using external applications.**

We introduce Seahawk, an Eclipse plugin to integrate Stack Overflow crowd knowledge in the IDE. It allows developers to seamlessly access Stack Overflow data, thus obtaining answers without switching the context. We present our preliminary work on Seahawk: It allows users to (1) retrieve Q&A from Stack Overflow, (2) link relevant discussions to any source code in Eclipse, and (3) attach explanative comments to the links.

***Keywords*-Q&A websites, Stack Overflow, Seahawk**

I. Introduction

Although the software development process outcome is heavily based on the knowledge and the creativity of software developers [17], writing code often requires knowledge beyond that which developers already possess [6]. To obtain the knowledge necessary to complete the task at hand, developers consult different sources of information, such as co-workers, project documentation, books, manuals, or computerized information systems. However, project documentation is commonly inadequate [7], manuals tend to be outdated, and books may be hard to retrieve or link to the actual task. For these reasons, often developers' knowledge needs can only be satisfied by posing questions to other programmers [6].

With the aim of leveraging on these circumstances and the success of social media, Question and Answers (Q&A) online services offer infrastructures to support knowledge exchange between programmers. Even though many studies (*e.g.,* [1], [10]) that investigated general purpose Q&A websites "suggest that [they] may be poorly suited to provide high quality technical answers" [8], in practice, Q&A sites for programmers and software engineers are filling "archives with millions of entries that contribute to the body of knowledge in software development" [14].

A. The Case of Stack Overflow

Among the available technical Q&A sites, Stack Overflow (http://stackoverflow.com/) in particular is becoming one of the most visible venues for sharing knowledge on software development [8].

By analyzing the data from the last public release (September 2011) of the entire Stack Overflow data dump [2], we measure approximately 750,000 registered users, 2 million posed questions, and 4 million answers, of which more than 1 million were accepted as resolutive from the person who posed the question. Figure 1 shows the growing trend of the number of questions and answers exchanged each month on the Stack Overflow website. Mamykina *et al.* reported that more than 92% of the questions on expert topics are answered in a median time of 11 minutes [8].

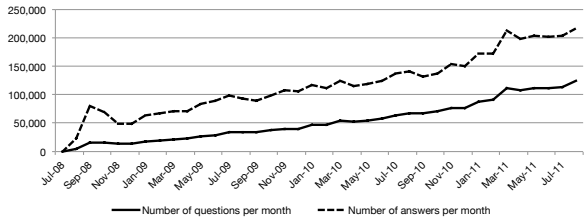

Figure 1. Number of questions and answers in Stack Overflow by month.

The Stack Overflow service aims to differentiate its method to access knowledge from those offered by web search engines [13]. This is mainly achieved through three of the specific design decisions that guided the creation of the service: *voting*, *tagging*, and *editing*. Voting addresses the critical issue that arises when reusing code samples found on the web: Before integrating a search result the developer has to assess its trustability to take a decision [4]. In the case of search engine results, developers have to take a decision that can only be based on their knowledge and experience. In Stack Overflow, on the contrary, developers can also rely on the result of the voting mechanism: Site members can vote to approve or disprove any answer, and the sum of the votes creates an overall quality score based on crowd knowledge.

Tagging is used to clarify the technology and the topic to which the question and answer applies. When retrieving explanations and examples from generic websites, in fact, the technology to which they refer might be unclear, *e.g.,* it is common to mistake VB6 for VB.NET. In Stack Overflow, questions can be filtered and retrieved by tag.

Editing clearly distinguishes this Q&A site from traditional web sites and web forums: Stack Overflow members can edit questions and answers to make them more refined and to the point over time, thus creating a reliable and correct knowledge base that can always be referenced and accessed.

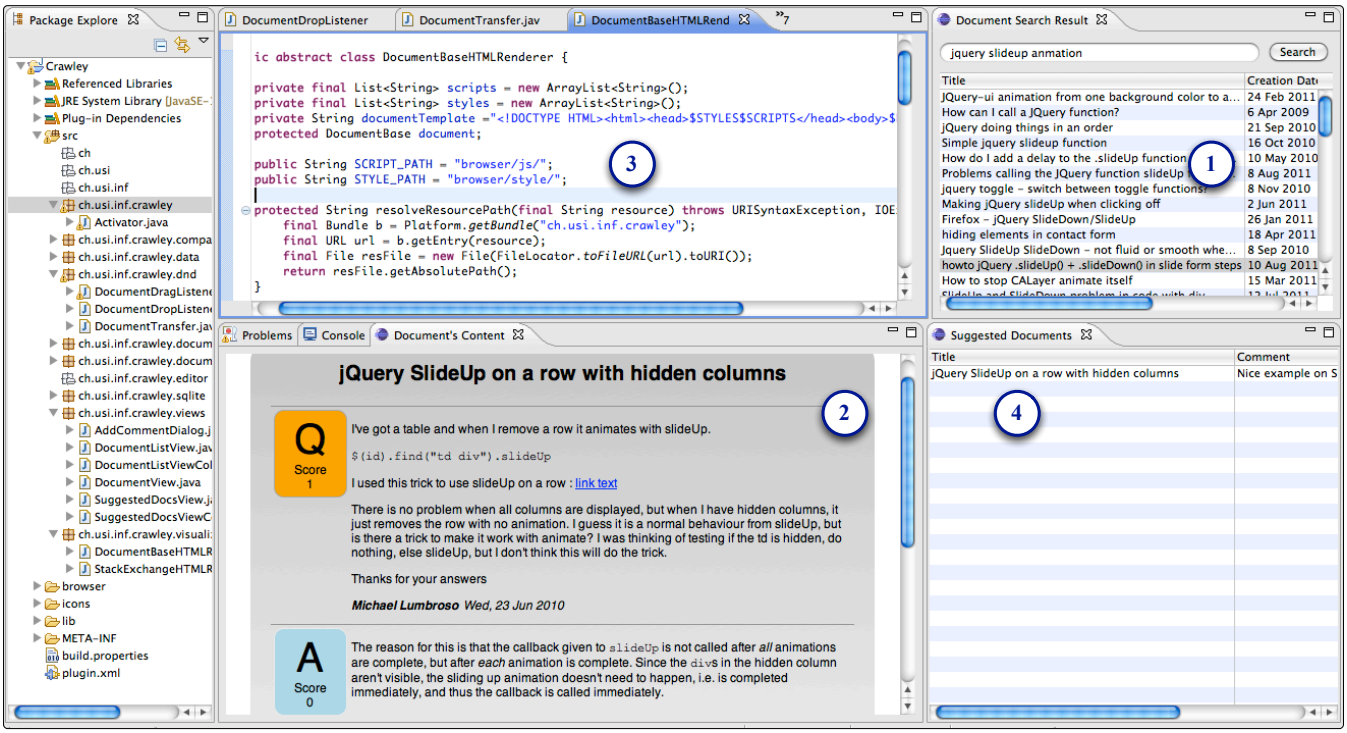

Figure 2. The Seahawk User Interface.

The Stack Overflow community encourages both editing contributions where the person who asked the question also gives the answer, thus creating a sort of mini-blog entries [14].

B. Limitations of Q&A Sites

Despite the aforementioned valuable features and its exceptional growth in usage and interest, both in academia and industry, we currently see two limitations in Stack Overflow and Q&A sites in general that hinder their full potential for software engineering.

In the first place, Q&A sites suffers from a disconnection from the IDEs (Integrated Development Environments): The environments where developers spend most of their working time. Through the investigation conducted by LaToza *et al.* on developers' work habits [7], we know that programmers spend most of their time in the IDE not only when *writing* code, but also when *understanding* it, and even when *designing* new parts of a system. Despite this, Q&A websites are currently only accessible from web browsers, in a way that is disconnected from the development process. This might limit a consistent adoption of Q&A sites as a mean to acquire knowledge: Since Q&A sites are not integrated in modern IDEs, developers are forced to interrupt their flow and change context every time they need to deal with them. Moreover there is no possibility to easily retrieve answers directly related to the current programming context.

In the second place, by their nature, Q&A websites provide a platform for questions aimed at "a general audience that is not part of the same project" [14]. We do not see this as a limitation of the interest that a whole team working on the same project can have toward certain valuable discussions. In fact, we can easily imagine a developer who bases her implementation of a part of a software system on a meaningful discussion that took place on a Q&A site. Knowing the existence of such a productive discussion would be a valuable resource to recover the rationale, design, or implementation details of that particular portion of the system. Nevertheless, Q&A websites currently only offer web links to refer their data and do not offer any resource to collaboratively and privately exploit valuable questions and answers in the context of a team working on the same project. This reduces the context of usage of Q&A sites.

C. Our Contribution

We claim that an integration in the IDE—the environment where developers spend most of their working time—of Q&A sites' data would be a viable solution to remove the aforementioned limitations suffered by these services. To this aim, we present Seahawk, a plugin for the open-source IDE Eclipse [1], we are devising to integrate Stack Overflow

[1]http://www.eclipse.org

data support in the programming environment. We report our preliminary work, and we show that questions can be (1) easily and effectively accessed within IDE, (2) linked to specific source code entities, and (3) commented within the context of the project being developed.

II. Seahawk

We are devising Seahawk to obtain a recommendation system for Stack Overflow data, integrated in the IDE. We chose Eclipse as our target IDE because of its modular and pluggable structure, its significant amount of users, and its support for multiple languages. The current implementation of Seahawk works for any language recognized by Eclipse.

The current implementation of Seahawk[2] (Figure 2) lets the user query Stack Overflow data, list relevant questions and answers, read each single discussion, link any discussion to any code file, and provide a comment for justifying the links, which will be recommended again when the file will be opened again. We detail Seahawk by following the division described by Robillard *et al.* [11]: A recommender system involves: A *data-collection mechanism* to store development-process data and artifacts in a model, a *recommendation engine* to analyze data and generate recommendations, and a *user interface* to trigger recommendations and show results.

A. Data-collection Mechanism

Figure 3 shows the current architecture of the Seahawk recommendation system. On the left side, the background shape delimits the data-collection mechanism.

To obtain more flexibility in the recommendation engine (see Section II-B), instead of relying on the official Stack Overflow API [3] to obtain question data, we took advantage of the entire data dump of the service database publicly provided under a Creative Commons license [2]. This data dump is made of several XML files, which map the tables of the official database. The biggest file has a size of more than 7GB and maps the "posts", *i.e.,* both answers and questions, treated as rows of the same table, relying on intra-table relations to build complete discussion documents. The size of the dump did not allow us to rebuild original documents in memory, thus we created a *XML dump importer* module to first reproduce the database locally. The *Document builder* reconstructs the documents by restoring the dependencies between questions and answers and comments from the database (*i.e.,* additional discussion concerning only the specific answer, or question), and includes users' data and votes to recover the scores. This complete data is then given as input to an Apache Solr [4] instance, which is the basis of our recommendation engine.

[2]available at http://seahawk.inf.usi.ch
[3]http://api.stackoverflow.com/1.1
[4]http://lucene.apache.org/solr

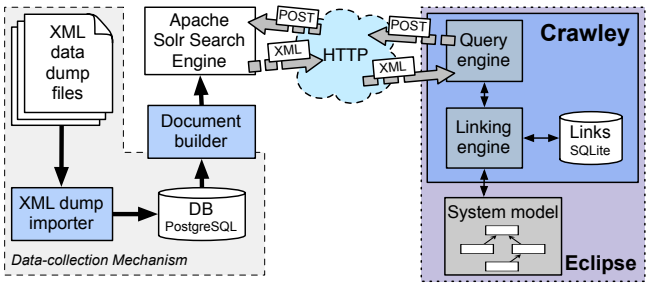

Figure 3. The Architecture of the Seahawk Recommendation System.

B. Recommendation Engine

The recommendation engine of Seahawk is made of three components: The *Apache Solr engine*, the *Query engine*, and the *Linking engine* plus its SQLite database. The choice of using the entire data dump gives us the freedom to decide on the technology for retrieving relevant discussion from Stack Overflow data, instead of relying on the "black-box" service that the website offers. In the current version of Seahawk, we work with Solr, which offers good flexibility and effective search performances. In particular the data in input is stored and indexed in a vector space model [9] using *tf-idf* as weighting. To improve the effectiveness of the indexing mechanism, we applied widely used information retrieval pre-processing: We removed natural language stop words, we filtered out possessive words, we lowered the case of all the characters, and we applied a stemming process [9].

Once the indexing is complete, the Solr engine can be queried via HTTP: It answers with the relevant documents in XML format. The dialog with Solr is implemented by the *Query engine* component, which, as seen in Figure 3, is part of the Eclipse plugin itself. In addition, the *linking engine* is used to store and retrieve the manual links that users provide between Stack Overflow discussion and code files in the project being developed in Eclipse.

C. User Interface

Figure 2 presents the Seahawk plugin as seen during a development session in the Eclipse IDE. The starting point for interacting with Seahawk is the *Development Search Result* view (Point 1, in Figure 2): The user inputs terms to form a query in the text field, then hits the *Search* button to trigger the retrieval. The *Query engine* sends the request to Solr and, in a few seconds, displays the results in the table. The user, then, is able to sort the result by relevance (as computed by Solr), author, date, or title.

Subsequently, when the user double clicks on any entry in the list of results, the *Document's Content* view is accordingly updated and shows the content of the discussion (Point 2). The complete information that would be available in the Stack

Overflow website is presented through a HTML page displayed by the internal web browser of the Eclipse IDE. We decided to create a page that is more compact than the original Stack Overflow version, thus allowing an easier integration within Eclipse. For example, comments are displayed only on request by clicking on a link under each post (not visible in the figure for space reason).

The first two views provide the first contribution of our work: An approach to allow programmers to seamlessly access Stack Overflow data within the IDE with no context switches. To have access to the second contribution (*i.e.,* linking discussions to project's code files, and commenting them), the user interacts with the standard *Code Editor* view (Point 3) and with the *Suggested Documents* view (Point 4).

The user selects any entry in the *Development Search Result* view, and drags and drops it to the code editor containing the code file to be linked. After this, an interaction dialog appears (see Figure 4), in which the comment explaining the link can be entered. The *Linking engine*, in the background, creates a new link by storing the information (path to the code file, id and title of the question, and linking comments) in a SQLite database in the Eclipse directory. Every time any file with links to discussions is opened again, the *Linking engine* uses the stored information to query Solr and receive the documents to display.

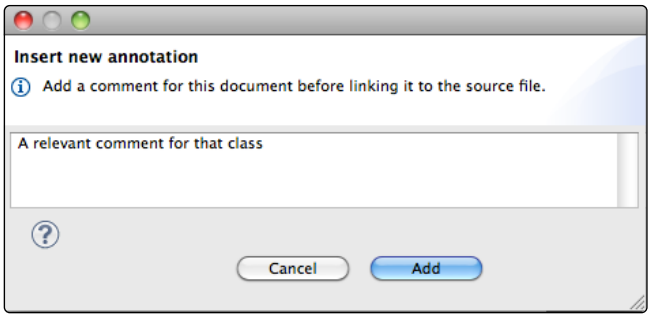

Figure 4. Dialog to comment a link to a Stack Overflow discussion.

III. Discussion

The current implementation of Seahawk reflects our preliminary work on the topic: It helped us to determine the practical feasibility of our approach and delineate future working directions. Nevertheless, Seahawk is affected by a number of limitations.

Concerning the data-collection mechanism, even though using the Stack Overflow data dump gives us both more flexibility and the possibility to use Seahawk without Internet access, this approach poses a few challenges: First, the availability of the data dump relies on the benevolence of Stack Overflow maintainers, will they continue to provide this data in the future? If not, how can we get the data in a reasonable time? Second, currently data dumps are available at a quarterly schedule, meaning that updated and valuable data can be not available for as long as three months. Even though this can be reasonable for well established technologies, such as Java, it might pose a problem for emerging technologies, whose popularity critically grows from one month to another.

Concerning the recommendation engine, Solr offers high quality time performances (*i.e.,* less than three seconds to provide run-time results from a database with more than 2 million documents), but experimenting new information retrieval techniques and implementing them in Solr can become a daunting task. Thus, we might need to consider other approaches for investigating further information retrieval techniques for Q&A data. In addition, we base our retrieval approach on user generated queries, however, a recommendation system should be able to automatically suggest relevant documents for the task at hand. As a first step we might consider to automatically link questions related to popular libraries used in the system being developed.

Currently the linking engine for Stack Overflow discussions works only locally, since the SQLite database is stored in the local Eclipse directory. Moving it to a global database would simply require a form of authentication to recognize who created and commented the links. Currently, links are created at file level, while it would be more useful to have links at the code entity level, such as class, method, or even statement level. However, this poses a challenge: What happens when the entity is removed from the code base? Should we remove the linking, or should we store it at another level? We are considering to integrate the linking engine with the versioning system, to keep track of the history of links.

Finally, currently Stack Overflow offers read-only access to its data outside of its website: Even the official API does not allow to write new questions or answers from external applications. This partially limits the usefulness of any external approach, in particular if we want to avoid context switches from the IDE. We do not see a clean solution to this problem, unless Stack Overflow changes its policy.

IV. Related Work

Q&A answers websites have recently gained popularity not only among users, but also among researchers. In software engineering, researchers are currently exploring how this new source of information can be leveraged for supporting how individuals and teams develop software, and for improving the research process itself: Treude *et al.* presented an exploration of the Stack Overflow service and data [14]. They identified five categories of tags (programming language, framework, environment, domain, non-functional, homework) and eleven categories of questions, among which questions asking about instructions, about unexpected behavior, about the environment, and about error messages were the most frequently posed and answered. In general, they found that

Stack Overflow is "particularly effective at code reviews, for conceptual questions, and for novices" [14]. In a subsequent work, Treude *et al.* discussed the impact of web content curated by the crowd for software developers and their working practice [15]. They raised questions about whether and how a large valid amount of programming knowledge could redefine the attributes of good programmers, about who owns the intellectual property of shared code, and about the impact of social media programming knowledge on software engineering education and career planning.

Our work also lies in the context of integrating additional sources of information within the IDE, to improve software understanding and development. In this context, we find many examples, such as Holmes and Begel's *Deep Intellisense* [5], an IDE plugin that links bug reports, emails, code changes to source code entities, or the Eclipse plugin *Hipikat* [16], intended to assist newcomers by recommending items from problem reports, newsgroup articles, *etc.*.

Other tools tackled the challenge of reducing the context-switching from IDE to web browser. For example, Fishtail [12], an Eclipse plugin, harnesses programmer's interactions history to get relevant web resources, and Blueprint [3], an Adobe Flex Builder plugin, integrates web researches in the code editor to provide example snippets.

V. Conclusion

Q&A websites for technical questions about software development provide programmers with an extremely large, updated, and curated knowledge. Even though these sites have important features to foster usage, referencing, and active participation, we observed two limitations that might hinder their full potential for software engineering: Disconnection from the IDEs and lack of support for team collaboration.

We argued that an Eclipse plugin would be a solution to overcome these two issues, thus providing a more complete usage of this form of information. We presented the Eclipse plugin we devised: Seahawk, a recommending system for Stack Overflow data, that integrates Stack Overflow within the IDE, adding support for linking code files to discussions and create specific comments on the links. Through Seahawk we verified the practical feasibility of our approach and realized limitations that must be addressed in the future to continue our work on the topic.

Acknowledgment

Bacchelli gratefully acknowledges the Swiss National Science foundation's support for the project "SOSYA" (SNF Project No. 132175).

References

[1] L. A. Adamic, J. Zhang, E. Bakshy, and M. S. Ackerman. Knowledge sharing and yahoo answers: everyone knows something. In *Proc. of WWW (17th International Conference on World Wide Web)*, pages 665–674. ACM, 2008.

[2] J. Atwood. Creative commons data dump september '11. http://blog.stackoverflow.com/2011/09/creative-commons-data-dump-sep-11/, September 2011.

[3] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-centric programming: integrating web search into the development environment. In *Proc. of CHI 2010 (28th International Conference on Human factors in computing systems)*, pages 513–522. ACM, 2010.

[4] F. S. Gysin and A. Kuhn. A trustability metric for code search based on developer karma. In *Proc. of SUITE 2010 (2nd International Workshop on Search-driven Development: Users, Infrastructure, Tools and Evaluation)*, pages 41–44, 2010.

[5] R. Holmes and A. Begel. Deep Intellisense: a tool for rehydrating evaporated information. In *Proc. of MSR 2008 (International working conference on Mining software repositories)*, pages 23–26. ACM, 2008.

[6] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *Proc. of ICSE 2007 (29th International Conference on Software Engineering)*, pages 344–353. IEEE CS, 2007.

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

[8] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest Q&A site in the west. In *Proc. of CHI 2011 (29th Conference on Human factors in computing systems)*, pages 2857–2866. ACM, 2011.

[9] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[10] K. K. Nam, M. S. Ackerman, and L. A. Adamic. Questions in, knowledge in?: a study of naver's question answering community. In *Proc. of CHI 2009 (27th International Conference on Human factors in computing systems)*, pages 779–788. ACM, 2009.

[11] M. P. Robillard, R. J. Walker, and T. Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, 2010.

[12] N. Sawadsky and G. C. Murphy. Fishtail: from task context to source code examples. In *Proc. of TOPI 2011 (Workshop on Developing Tools as Plug-ins)*, pages 48–51, 2011.

[13] J. Spolsky. Learning from stackoverflow.com. http://www.youtube.com/watch?v=NWHfY_lvKIQ, April 2009.

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

[15] C. Treude, F. F. Filho, B. Cleary, and M.-A. Storey. Programming in a socially networked world: the evolution of the social programmer. In *Proc. of FutureCSD 2012 (International Workshop on The Future of Collaborative Software Development)*, 2012.

[16] D. ČubraniĆ, G. C. Murphy, J. Singer, and K. S. Booth. Learning from project history: a case study for software development. In *Proc. of CSCW 2004 (19th Conference on Computer Supported Cooperative Work)*, pages 82–91, 2004.

[17] Y. Ye. Supporting software development as knowledge-intensive and collaborative activity. In *Proc. of WISER 2006 (International Workshop on Interdisciplinary Software Engineering Research)*, pages 15–22. ACM, 2006.