Prompter: A Self-confident Recommender System

Luca Ponzanelli[1], Gabriele Bavota[2], Massimiliano Di Penta[2], Rocco Oliveto[3], Michele Lanza[1]
1: REVEAL @ Faculty of Informatics – University of Lugano, Switzerland
2: University of Sannio, Benevento, Italy *3: University of Molise, Pesche (IS), Italy*

***Abstract*—Developers often consult different sources of information like Application Programming Interfaces (API) documentation, forums, Q&A websites, etc. with the aim of gathering additional knowledge for the programming task at hand. The process of searching and identifying valuable pieces of information requires developers to spend time and energy in formulating the right queries, assessing the returned results, and integrating the obtained knowledge into the code base. All of this is often done manually.**

We present PROMPTER, a plug-in for the Eclipse IDE which automatically searches and identifies relevant Stack Overflow discussions, evaluates their relevance given the code context in the IDE, and notifies the developer if and only if a user-defined confidence threshold is surpassed.

I. INTRODUCTION

Developers have to cope with large and complex systems. Even though they are probably active developers in the community, they do not know the complete system by heart. The knowledge they possess probably concerns some parts of the system they are mostly working with. Sooner or later, for example to fix a bug, they have to deal with the unknown or partially known parts of the system. When this situation happens, developers need to access additional sources of information to go beyond the knowledge of the system they already posseses [1]. This happens by asking teammates [2], through pair programming sessions [3], or by searching and perusing the vast amount of information available on the internet [4].

However, people are not always available. To overcome this situation, developers harness information retrieved from different sources, such as forums, mailing lists [5], blogs, Q&A websites, bug trackers [6], *etc.* Among all the online resource available, Q&A websites have become a prominent venue among developers to share programming knowledge. A prominent example is Stack Overflow[1], a vast Q&A website for developers that has hundreds of thousands of users, and millions of questions, answers, and comments [7].

Developers cannot access the information within the Integrated Development Environment (IDE), but they have to interrupt their work flow, leave the IDE, and use a web browser to perform and refine searches. The process of searching and identifying valuable pieces of information requires developers to spend time and energy in formulating the right queries, assessing the returned results, and transfer the obtained knowledge to the problem context in the IDE.

Recommender systems [8] represent a possible solution to this problem. A recommender system gathers and analyzes data, identifies useful artifacts, and suggests them to the developer. Seminal tools, such as ERose [9], HIPIKAT [10] and DEEPINTELLISENSE [11], suggest project artifacts in the IDE aiming at providing developers with additional information on specific parts of the system. They come however with a caveat: the developer must proactively invoke them, and, once invoked, they continuously display information. This may defeat their purpose, as they augment the complexity of what is displayed in the IDE. *Ideally, a recommender system should behave like a prompter in a theatre: Ready to provide suggestions whenever the actor needs them, and ready to autonomously give suggestions if it feels something is going wrong*.

The interaction between the theatre prompter and the actor is similar to the interaction between two developers doing pair programming, working side by side to write code. These developers have different roles, *i.e.,* the driver, who is in charge of writing code, and the observer, who observes the work of the driver [12], tries to understand the context, and, if she has enough confidence, interrupts the driver by giving suggestions. In addition, the driver can consult the observer whenever she needs it, making the observer the programming prompter of the programming actor.

This interaction is what we propose in PROMPTER[2], a tool that automatically retrieves and recommends, with push notifications, relevant Stack Overflow discussions to the developer [13]. PROMPTER makes the IDE a programming prompter that silently observes and analyzes the code context in the IDE, automatically searches for Stack Overflow discussions on the Web, evaluates their relevance by taking into consideration *code aspects* (*e.g.,* code clones, type matching), *conceptual aspects* (*e.g.,* textual similarity), and Stack Overflow *community aspects* (*e.g.,* user reputation) to decide, given a certain amount of self-confidence (encoded in a threshold the user can change through a slider, to make the recommender quiet or talkative) when to suggest discussions.

II. PROMTPER

Figure 1 shows the user interface of PROMPTER. It provides two views through which the user can (i) receive and track notifications, and (ii) read the suggested Stack Overflow discussions. The notification center (1) is the main view of PROMPTER and it is used to notify the developer whenever a relevant result is available. Whenever PROMPTER considers a discussion as relevant for the current context, it opens the notification center and plays a sound. If a Stack Overflow discussion is notified

[1]http://stackoverflow.com

[2]http://prompter.inf.usi.ch

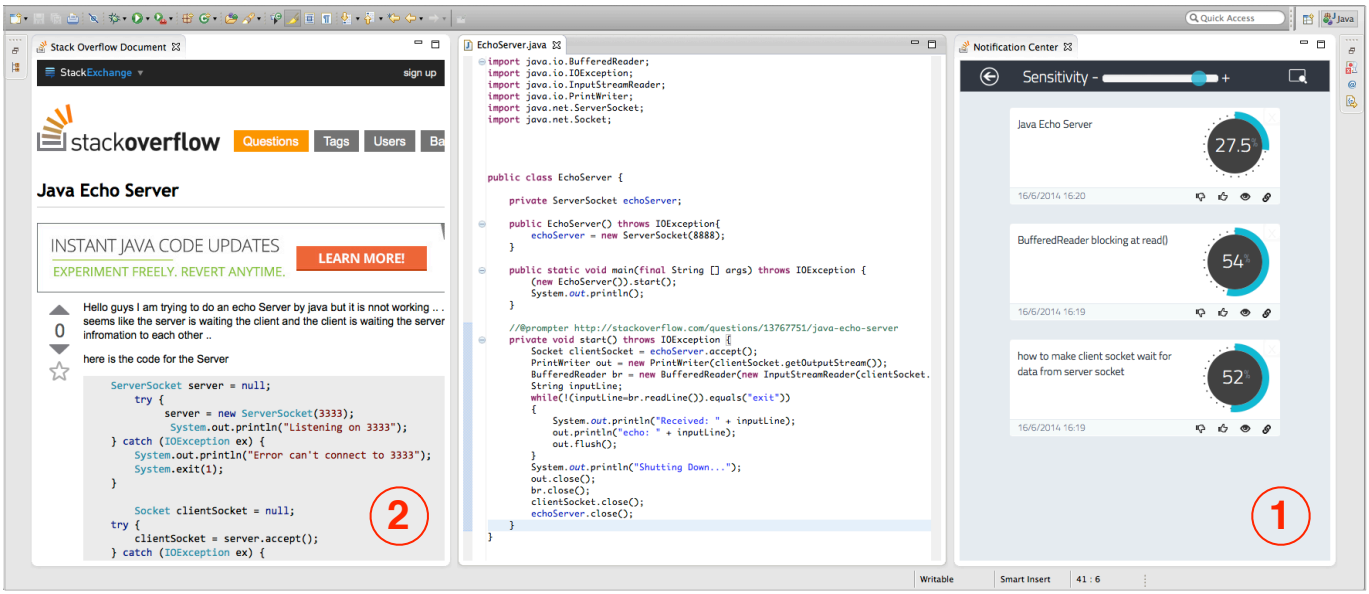

Fig. 1. The Prompter User Interface.

more than once, it is pushed to the top of the list for visibility. Figure 2 shows an example of notification. The developer is provided with some information regarding (a) the title of the Stack Overflow discussion, (b) the notification date and time, so that the developer can know how old the notification is, (c) the confidence level of Prompter on the Stack Overflow discussion against the related code context, and (d) some feedback, tracking and linking functionalities in the bottom-right corner. By clicking on the thumb up (down) icon, the developer can rate the discussion as useful (useless) with respect to the coding activity she is performing in the IDE. The other icons on the notification allows the developer to backtrack to the code entity associated with a specific notification (eye icon), or to link the suggested discussion to its code entity (chain icon). If the developer clicks on the former, Prompter opens up a code editor and highlights the portion of code related to the notification. If the developer clicks on the latter, a simple annotation reporting the URL of the discussions is created in the code in form of a comment.

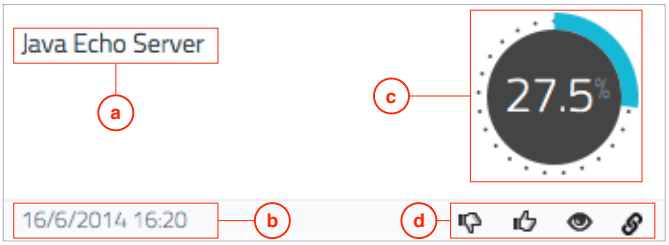

Fig. 2. Prompter notification details.

Whenever a developer clicks on a notification, a Stack Overflow document view (Figure 1 (2)) is opened, which shows the contents of the Stack Overflow discussion.

Fig. 3. Prompter sensitivity bar

At the top of the notification center, the developer can change the sensitivity of the notification system (Figure 3, (2)): by sliding to the right Prompter is more talkative and produces more notifications, by sliding to the left it becomes more taciturn and requires a higher level of confidence to notify the developer. Moreover, by clicking on the arrow in the top-left corner (Figure 3 (1)), the developer can access the full result set of Stack Overflow discussions related to the last notification.

A. *Explicit Query Writing*

Fig. 4. Prompter manual search bar

Sometimes Prompter is not able to point out the right Stack Overflow discussion or probably it has not enough information to generate a notification. For example, a similar situation can happen at the very beginning of the development, where there are few lines of code (*e.g.,* a class stub). For this reason, we implemented an additional manual interaction where we provide the developer with the capability to perform manual searches. Whenever the developer wants to search for Stack Overflow discussions on her own, she can click on the manual search button at the top right corner (Figure 3 (3)). The

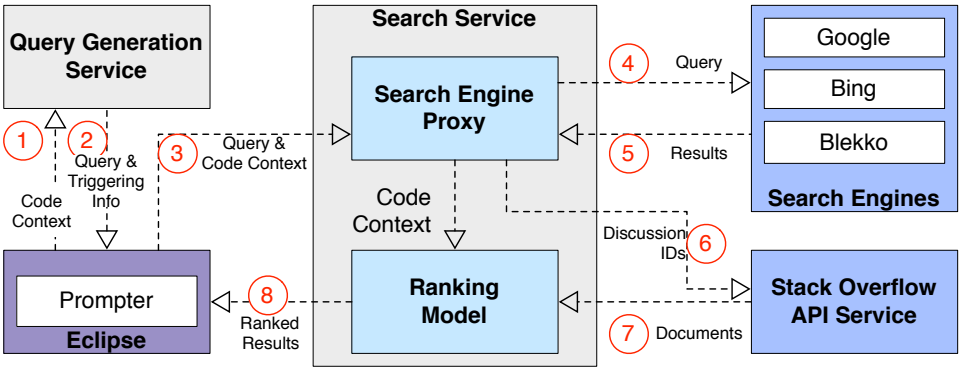

Fig. 5. PROMPTER architecture.

notification centers disappears and a manual search bar becomes available (Figure 4). There, the developer can manually type a query (Figure 4 (4)) and search for Stack Overflow discussions. The results are presented in form of notification, where each of them presents a confidence value according to the code context obtained from the code editor on top. While the developer is interacting with the manual search view, he can continue modifying and writing code. If PROMPTER notifies a discussion in the meanwhile, the developer is notified anyway: a counter of the unseen notifications will popup on top of the notification center icon—Figure 4 (5), and it resets as soon as the developer accesses the notification center by clicking on the icon.

B. Explicit Invocation

A prompter in a theater does not only prompt the right sentence to the actors on the stage, but he also provides support on demand. Indeed, an actor can always ask the prompter for a cue in order to go on with the show. In PROMPTER we implemented the same interaction: the developer can always ask PROMPTER to perform a search on a specific code entity (*i.e.,* method or class), by accessing the contextual menu in the code editor, or on the package explorer. In the first case, PROMPTER searches discussions for the code entity identified by cursor in the editor, while in the second case it searches according to the code entity selected.

C. A Use Case Scenario

Alice is a developer who wants to implement an echo server in Java. She has little knowledge about the Java SDK, but she knows how the echo server should be implemented. An echo server is a simple program that waits for incoming connections and, whenever a client connects and sends messages, it replies back to the connected client with the same message. Alice decides to take advantage of PROMPTER to help her to implement the server. Due to her lack of Java SDK knowledge, Alice wants PROMPTER to be more talkative at the very beginning, thus she moves the sensitivity bar to the right (see Figure 3). Alice begins developing the server by creating a new Java class in the Eclipse IDE, she starts by naming the class *EchoServer*. As soon as she starts typing some code, PROMPTER begins to silently gather the code context in the current code editor and searches for relevant Stack Overflow discussions. Given the low threshold set by Alice, PROMPTER immediately notifies a couple of discussions. The last notified discussion, titled *Java Echo Server*[3], seems to tackle what Alice is trying to implement. She clicks on the notification, and another view pops-up in the IDE, showing the Stack Overflow web page. As the title suggested, the discussion tackles the implementation of an echo server, and Alice can start copying and pasting the parts of the code she needs by also taking into account the corrections proposed by the answers in the discussion. In the meanwhile, PROMPTER continues to observe what Alice is writing in the code editor and to search for other Stack Overflow discussions relevant for the modified code. Since the current discussion also provides the code needed to accomplish her tasks, she decides to lower the number of notification she is receiving, and she slides the sensitivity bar more on the left side. In doing so, she allows PROMPTER to interrupt her in case of higher quality discussions.

When she finished implementing the server, she decides to test its functionalities. Unfortunately, when testing her implementation, she discovers a bug: the server does not reply back to the client. However, while she was implementing the server, PROMPTER continued to silently search and notify discussion (with a higher level of relevance). She spots a discussion titled *Sockets, BufferedReader.readLine() – why the stream is not ready?*[4], and she opens it. By reading the discussion, she discovers that other people are dealing with the same problem. The solution lies in a single statement needed to flush the stream towards the client. She adds the statement, tests the fixed implementations, and verifies that everything works well. Once she has completed the assigned programming task, Alice can take advantage of the feedback system of PROMPTER, and give a "thumbs up" (see Figure 2, (d)) for the two discussions she used. PROMPTER collects these feedbacks and stores them for future improvements of its ranking model.

[3]http://stackoverflow.com/questions/13767751
[4]http://stackoverflow.com/questions/8563529

D. Architecture

Figure 5 depicts the architecture of PROMPTER, which is composed of (i) the *Eclipse plug-in*, (ii) the *Search Service*, and (iii) the *Query Generation Service*. The numbers in the picture represent the sequence of actions that PROMPTER performs when it retrieves documents from Stack Overflow.

PROMPTER is meant to be a silent observer that looks at what the developer is writing. Every time a change in the source occurs, PROMPTER extracts the code context (*i.e.,* source code, and API information) from the code element under observation (*i.e.,* a method or a class), and sends it to the *Query Generation Service* (1). This service generates a query starting from the code context received, and sends it back, together with the triggering information, to the plug-in (2). If the quality of the query is good enough, the plug-in forwards the query and the code context to the *Search Service*. The query is used to search for Stack Overflow discussions on the web by means of different search engines (*e.g.,* Google, Bing, Blekko). All resulting URLs are collected, and duplicates are removed (5). Every URL that refers to a question from Stack Overflow must match the form *stackoverflow.com/questions/⟨id⟩/⟨title⟩*, otherwise it is discarded. By matching the aforementioned form, Stack Overflow discussion ids are extracted from URLs. (6) The ids are used to query the Stack Overflow API[5] to obtain a fresh and up-to-date copy of each discussion. (7) The collected discussions, given the code context in the IDE, are evaluated and reranked by means of a ranking model we devised [13]. (8) When all the discussions are evaluated, their URLs, together with the related relevance value, are sent back to the plugin. If the top ranked discussion surpasses a user-defined threshold of relevance, PROMPTER fires a new notification in the IDE, or the results are discarded otherwise.

E. The Ranking Model

PROMPTER uses a ranking model [13] that, given a code context in the IDE, evaluates the relevance of a set of retrieved Stack Overflow discussions. The evaluation of the discussions is performed by taking in consideration both code-related aspects (*e.g.,* API types, API methods, code similarity), and community-related aspects (*e.g.,* user popularity, question score). The code-related aspects are focused on revealing similarities between the code context in the IDE and the technical aspect of a Stack Overflow discussion, while the community-related aspects are focused on including the judgment of the crowd inside our confidence value.

F. Evaluation Summary

PROMPTER has been evaluated through two studies [13]. The first was aimed at evaluating the devised ranking model, while the second was conducted to evaluate the usefulness of PROMPTER. In the first study we asked people to evaluate how much a code sample and the top ranked PROMPTER recommendation were related. We showed how the agreement of the users involved pointed out good results.

[5]http://api.stackexchange.com/

In the second study we carried out a controlled experiment with developers asking them to perform coding tasks. Then, we measured the completeness of the assigned tasks achieved by participants (with and without using PROMPTER). We showed how PROMPTER is effective for development tasks and how, from a qualitative point of view, developers were really satisfied of the tool.

III. CONCLUSION AND FUTURE WORK

We have presented PROMPTER, a plugin for the Eclipse Integrated Development Environment (IDE) that silently observers the developer, captures a code context in the IDE, retrieves Stack Overflow discussions in the background, ranks them according to a ranking model we have developed, and suggests discussions if and only if a developer-defined threshold of relevance is surpassed. PROMPTER implements the ideal behavior of a recommender system: Like a prompter in a theatre, PROMPTER is a silent observer that interrupts the developer only when it has enough confidence about the results, but it is always at disposal of the developer if she needs to explicitly invoke it. We also presented a summary of the evaluation of PROMPTER where we showed how it has been proven to be effective for development tasks.

Acknowledgments. Ponzanelli and Lanza thank the Swiss National Science foundation (SNF) for the financial support through SNF Project "ESSENTIALS", No. 153129.

REFERENCES

[1] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proceedings of ICSE 2007*. IEEE CS Press, 2007, pp. 344–353.

[2] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of ICSE 2006*. ACM, 2006, pp. 492–501.

[3] L. Constantine, *Constantine on Peopleware*. Yourdon, 1995.

[4] M. Umarji, S. Sim, and C. Lopes, "Archetypal internet-scale source code searching," in *Proceedings of OSS 2008*, 2008, pp. 257–263.

[5] A. Bacchelli, T. dal Sasso, M. D'Ambros, and M. Lanza, "Content classification of development emails," in *Proceedings of ICSE 2012*, 2012, pp. 375–385.

[6] J. Anvik, L. Hiew, and G. Murphy, "Who should fix this bug?" in *Proceedings of ICSE 2006*. ACM, 2006, pp. 361–370.

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

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

[9] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of ICSE 2004*. IEEE, 2004, pp. 563–572. [Online]. Available: http://www.st.cs.uni-sb.de/papers/icse2004/icse.pdf

[10] D. Cubranic and G. Murphy, "Hipikat: recommending pertinent software development artifacts," in *Proceedings of ICSE 2003*. IEEE Press, 2003, pp. 408–418.

[11] R. Holmes and A. Begel, "Deep intellisense: a tool for rehydrating evaporated information," in *Proceedings of MSR 2008*. ACM, 2008, pp. 23–26.

[12] L. Williams, "Integrating pair programming into a software development process," in *Proceedings of CSEET 2001*. IEEE, 2001, pp. 27–36.

[13] L. Ponzanelli, "Mining StackOverflow to Turn the IDE into a Self-confident Programming Prompter," in *In Proceedings of MSR 2014 (11th Working Conference on Mining Software Repositories)*. ACM, 2014, pp. 102–111.