Holistic Recommender Systems for Software Engineering

Luca Ponzanelli
REVEAL @ Faculty of Informatics
University of Lugano, Switzerland
luca.ponzanelli@usi.ch

ABSTRACT

Software maintenance is a relevant and expensive phase of the software development process. Developers have to deal with legacy and undocumented code that hinders the comprehension of the software system at hand. Enhancing program comprehension by means of recommender systems in the Integrated Development Environment (IDE) is a solution to assist developers in these tasks. The recommender systems proposed so far generally share common weaknesses: they are not proactive, they consider a single type of data-source, and in case of multiple data-source, relevant items are suggested together without considering interactions among them.

We envision a future where recommender systems follow a holistic approach: They provide knowledge regarding a programming context by considering information beyond the one provided by single elements in the context of the software development. The recommender system should consider different elements such as development artifact (*e.g.,* bug reports, mailing lists), and online resources (*e.g.,* blogs, Q&A web sites, API documentation), developers activities, repository history *etc.* The provided information should be novel and emerge from the semantic links created by the analysis of the interactions among these elements.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments—*Interactive environments*

General Terms

Documentation

Keywords

Recommender Systems, Developer Support

1. MOTIVATION

In the early nineties, Corbi *et al.* pointed out how "*Software renewal tools are needed to reduce the costs of modifying and maintaining large programming systems, to improve our understanding of programs*" [5]. Research has shown that maintaining software systems can take up to 75% of the total costs of a software product [7, 24]. In the maintenance phase, developers to spend about 50% of their working time to understand the code before modifying it [15], and they have to deal with legacy and undocumented code that hinders the comprehension of a software system at hand.

The standard means adopted by developers both during development and maintenance tasks is the Integrated Development Environment (IDE). IDEs are used not only to write code, but also to understand it, and even to design new parts of a system [14]. Thus, enhancing program comprehension in the IDE to provide developers with better support to understand the system at hand is a viable solution. We see two ways of pursuing it: (1) by studying alternative types of interactions with the IDE [4, 8], or (2) by improving the available IDEs by extending their functionalities. In our research, we focus on the latter, and consider recommender systems as a means to enhance program comprehension.

2. STATE OF THE ART

According to Robillard *et al.*, a Recommender Systems for Software Engineering (RSSE) "*is a software application that provides information items estimated to be valuable for a software engineering task in a given context*" [22]. An RSSE is made up of three main components: (1) a data collection mechanism, (2) a recommendation engine to analyze the data and generate recommendations, and (3) a user interface to present recommendations. The data fed to the data collection mechanism defines the kind of recommendation produced. For example, HIPIKAT [6] and DEEPINTELLISENSE [10] suggest relevant software development artifacts according to the current code context, STRATHCONA [11] recommends relevant code example given a code fragment, and eROSE [30] suggests elements to be changed according to past changes extracted from the versioning system. These examples of recommender systems treat data produced in the context of the project under analysis. Research also focused on the web as data source for recommender systems [9, 12, 23, 27] to suggest artifacts that developers use to complement the information needed to complete a task. The weakness of seminal tools like eROSE, HIPIKAT and DEEPINTELLISENSE, concerns the interaction with the developer: She must proactively invoke them and they continuously display information that augments the complexity of what is displayed in the IDE. Moreover, the source of information fed to recommender systems is generally focused on a single type of data (*e.g.,* email, bug reports). In the rare case where multiple artifacts are taken into account as source of data (*i.e.,*DEEPINTELLISENSE), they are considered as single elements and proposed together to the developer. This way of generating recommendations overwhelms the developer and makes the suggestions ineffective. In contrast, all the artifacts should be considered together, analyzed from a holistic point of view and then proposed to the developer.

3. HOLISTIC RECOMMENDER SYSTEMS FOR SOFTWARE ENGINEERING

The term *Holism* derives from the Greek word *holos*, meaning *whole*. According to the idea of holism, a system and its functioning cannot be fully understood by means of a reduction to its components, but should rather be viewed as a whole. The same can be applied in the context of software development: The software development process leaves traces of the interaction of people in different artifacts either in in-project artifacts (*e.g.,* email, bug reports), and in the contents provided by the online resources (*e.g.,* API documentation, forums, blogs, tutorial, Q&A websites). These resources are accessed and used by developers whenever they need additional information to accomplish a programming task. The information is likely to be spread across different artifacts. For each consulted artifact, the developer would extract parts of the contents, and merge them together to get a better understanding of the programming context at hand. Therefore, all the resources represent small pieces of information that should be viewed as the whole information needed to accomplish the task. The whole is greater than the sum of its parts: By considering the interactions that those elements have with each other, we plan to derive semantic links among them that allow novel type of information to emerge.

Suggesting every artifact related to a programming context would overload the developer and make the recommender system ineffective. *Viceversa*, a holistic recommender system should collect all the involved artifacts, information about developers, and software history, to analyze their interactions (*e.g.,* what artifacts are accessed by developers, what part of the system relates to either a developer or an artifact), discover semantic links, and provide meaningful summaries. Summaries can be either *extractive* or *abstractive*. The former concerns the extraction of prominent sentences from a text, while the latter is "*generally sought to dig well below the source linguistic surface to identify important conceptual content*" and generally produces novel output [26].

According to our vision, a holistic recommender system can be approached with three milestones that would cope with the problems we pointed out about recommender systems:

(1) Multi-Level Summarization: Research already tackled the summary of development artifacts like bug reports [21], emails [13] and source code [25]. We want to provide *multi-level* extractive summary of development artifacts. For multi-level we mean that a developer would initially access the minimum information extracted to let her evaluate the usefulness of the discussion. Then, she could request more and access additional levels of information. The higher the accessed level, the larger the displayed contents of the discussion. This would require novel approaches to summarize artifacts according to a reference context.

(2) Multi-Source IDE Recommender: We need to retrieve information from different sources of information by taking into account both human-made artifacts (*e.g.,* bug reports, mailing lists) created during the development, and online resources (*e.g.,* blogs, forums, Q&A websites). To this aim, we should devise a set of models capable of evaluating these artifacts according to a given code context in the IDE.

(3) Multi-Source, Multi-Level IDE Recommender: By merging the previous milestones, we would provide a holistic recommender systems. We aim at generating multi-level abstractive summaries produced by considering multiple items concerning a code context in the IDE as data source for RSSEs. The summaries would not be a collage of extractive summaries, but rather they would provide a semantically connected overview of the information provided by the extractive summaries.

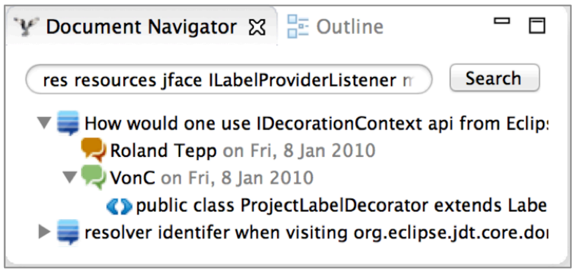

Document Navigator
Outline
res resources jface ILabelProviderListener r
Search
How would one use IDecorationContext api from Eclip
Roland Tepp on Fri, 8 Jan 2010
VonC on Fri, 8 Jan 2010
public class ProjectLabelDecorator extends Labe
resolver identifer when visiting org.eclipse.jdt.core.doi

Figure 1: The SeaHawk Document View

3.1 Current Status

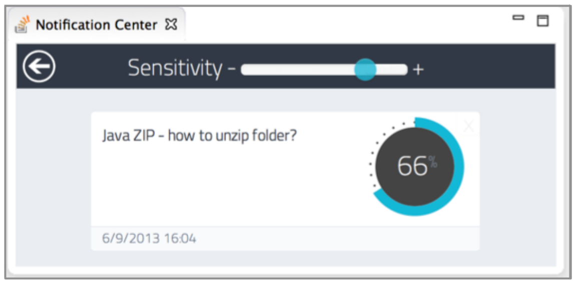
In the last years, Stack Overflow[1] has become a valuable venue where "*millions of entries that contribute to the body of knowledge in software development*" [28] are made by developers for developers. Stack Overflow is also effective: Mamykina *et al.* reported that more than 92% of the questions on expert topics are answered in a median time of 11 minutes [16]. Given these facts, Stack Overflow is a prominent data source for RSSEs that has scarcely been exploited by any Integrated Development Environment (IDE). We initially integrated this information inside the IDE to enhance program comprehension by means of recommender systems. In the second part of our research we focused on automating the recommendations and we introduced the concept of self-confidence. We devised a novel approach to establish a better a semantic connection between code contexts and Stack Overflow discussions by taking into account community related, textual, and code aspects.

3.1.1 Accessing Stack Overflow in the IDE

Due to its developer-oriented nature, Stack Overflow is most likely to be in the first results of a Google search when a developer searches for programming topics. However, accessing this source of information comes with a *caveat*: The developer needs to leave the IDE, access the web browser, retrieve the discussions, evaluate the contents, and bring the part of the needed information into the IDE. This process breaks the programming flow, and requires time and energy to formulate one's problem and peruse and process the results. The inception of our research concerns the integration of Stack Overflow inside the Eclipse IDE. We provided the capability of searching and displaying discussions without leaving the IDE [2]. In a second step we wanted to provide additional functionalities to enhance program comprehension in the IDE. We developed SeaHawk [19][20], a plugin for the Eclipse IDE that automatically formulates queries from the current context in the IDE, and presents a ranked and interactive list of results (see Figure 1). SeaHawk lets users identify individual discussion pieces (*i.e.,* single questions or answers) and import code samples through simple drag & drop. Users can also link Stack Overflow discussions and source code.

We evaluated SeaHawk, and collected insights about our approach. SeaHawk showed poor results on badly written methods (*e.g.,* long methods, abbreviated identifiers), while it provided good results when the code largely used libraries, or the code was implementing well known algorithms (*e.g.,* Fibonacci sequence, sorting algorithm).

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



Java ZIP - how to unzip folder?

66%

6/9/2013 16:04

Figure 2: The Prompter Notification Center

3.1.2 A Programming Prompter

A recommender system should ideally behave like a prompter in a theater: Ready to provide suggestions whenever the actor needs them, and ready to autonomously give suggestions if it feels something is going wrong. To this aim, we devised a novel approach that, given a context in the Integrated Development Environment (IDE), automatically retrieves potentially pertinent discussions from Stack Overflow, evaluates their relevance using a multi-faceted ranking model, and, if a given confidence threshold is surpassed, notifies the developer about the available help. We implemented our approach in Prompter, an Eclipse plug-in. Figure 2 shows the notification center through which Prompter notifies developers. Every notification reports the title of the Stack Overflow discussion and the percentage of confidence that Prompter has on the discussion in respect to the current context. Differently from SeaHawk, the notion of relevance of a discussion is based on a ranking model capable of capturing relations between Stack Overflow discussions and source code. The relations are evaluated trough different aspects concerning the code (*i.e.,* API methods usage), textual similarity, and community aspects (*i.e.,* user reputation). We evaluated the ranking model via a survey. We asked people from both industry and academia to evaluate the relevance between a code context and the top ranked discussion provided by Prompter. In general, people evaluated the discussion as related to the code context. We also conducted an experiment to evaluate the usefulness of Prompter in development and maintenance tasks. Results showed how Prompter helps developers in achieving a better completeness of the tasks with particular success for greenfield development.

The results of the research concerning Prompterhave been submitted to ICSE 2014 (36th ACM/IEEE International Conference on Software Engineering), and we are waiting for notification.

3.2 Improving Defect Prediction

Many researches performed defect prediction by relying on the historical and structural information provided by past defects [29], source code metrics [18][3] and repositories [17]. Bacchelli *et al.* [1] obtained relevant results in using information regarding the popularity of classes, in the contents of development emails, as a complement to approaches based on code metrics. Following a holistic approach, additional information regarding part of the software used (*i.e.,* libraries) can be exploited from other artifacts in other to analyze different aspects of a software component. Stack Overflow can be used as source of information to study the bugginess of an API: We can collect statistics about bugginess of specific classes in common used libraries and use this data as predictor to extend actual defect prediction models. We need to devise an approach to understand if a discussion regards a problem with a library and we need to identify the classes involved. A possible approach would be to extract stack traces and analyze them to identify such information. Moreover, natural language analysis is needed in the analysis of the discussion to gather additional information about the problem.

4. CONCLUSIONS

Program comprehension is an important aspect of software development, in particular during the maintenance phase. Providing recommender systems in the IDE is a possible solution to provide developers with additional information about the system at hand and helpful for the current programming task. Developers use information spread across development artifacts (*e.g.,* bug reports, emails) and online resources (*e.g.,* Q&A websites, blogs, tutorials) while performing a programming task. We propose a research plan to develop a holistic recommender systems that should consider all of these artifacts and discover semantic links.

So far we focused on harnessing a Stack Overflow as data source for recommender systems. We implemented SeaHawk to provide developers with capabilities of generating queries from code, retrieving discussions, linking them to the code, and importing code samples. On the other hand, recommender systems should be also proactive. Following this philosophy, we developed Prompter, a plugin for the Eclipse IDE that automatically retrieves, evaluates, and suggests Stack Overflow discussions to the developer if a certain confidence threshold is surpassed.

We are at the beginning of the second year of the Ph.D., and so far we obtained two tools, SeaHawk and Prompter, as contributions and the following publications:

1. *Harnessing Stack Overflow for the IDE*
   Alberto Bacchelli, Luca Ponzanelli, Michele Lanza. In Proceedings of RSSE 2012 (3rd International Workshop on Recommendation Systems for Software Engineering), pp. 26-30, IEEE CS Press, 2012.
2. *Leveraging Crowd Knowledge for Software Comprehension and Development*
   Luca Ponzanelli, Alberto Bacchelli, Michele Lanza. In Proceedings of CSMR 2013 (17th IEEE European Conference on Software Maintenance and Reengineering), pp. 57-66. IEEE CS Press, 2013.
3. *SeaHawk: Stack Overflow in the IDE*
   Luca Ponzanelli, Alberto Bacchelli, Michele Lanza. In Proceedings of ICSE 2013 (35th ACM/IEEE International Conference on Software Engineering), pp. 1295-1298. IEEE CS Press, 2013.

Reaching the concept of holistic recommender system requires further research investigation. The source of information cannot be limited to Stack Overflow. The information should arise from the semantic links among all the artifacts relevant for a code context and proposed to the developer in a multi-level summarized form without overwhelming her.

5. ACKNOWLEDGEMENTS

Luca Ponzanelli thanks the Swiss National Science foundation for the financial support through SNF Project "SOSYA", No. 132175.

6. REFERENCES

[1] A. Bacchelli, M. D'Ambros, and M. Lanza. Are popular classes more defect prone? In *Proceedings of FASE 2010 (13th international conference on Fundamental Approaches to Software Engineering)*, pages 59–73. Springer-Verlag, 2010.

[2] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing stack overflow for the ide. In *Proceedings of RSSE 2012 (3rd International Workshop on Recommendation Systems for Software Engineering)*, pages 26–30. IEEE CS Press, 2012.

[3] V. R. Basili, L. C. Briand, and W. L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct. 1996.

[4] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. Laviola. Code bubbles: Rethinking the user interface paradigm of integrated development environments. In *Proceedings of ICSE 2010 (32nd ACM/IEEE International Conference on Software Engineering)*, pages 293–296. ACM, 2010.

[5] T. Corbi. Program Understanding: Challenge for the 1990s. *IBM Systems Journal*, 28(2):294–306, 1989.

[6] D. Cubranic and G. Murphy. Hipikat: recommending pertinent software development artifacts. In *Proceedings of ICSE 2003 (25th IEEE International Conference on Software Engineering)*, pages 408–418. IEEE CS Press, 2003.

[7] A. Davis. *201 Principles of Software Development*. McGraw-Hill, 1995.

[8] R. DeLine and K. Rowan. Code canvas: zooming towards better development environments. In *Proceedings of ICSE 2010 (32nd ACM/IEEE International Conference on Software Engineering)*, pages 207–210. ACM, 2010.

[9] M. Goldman and R. Miller. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing*, pages 223–235, 2009.

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

[11] R. Holmes, R. Walker, and G. Murphy. Strathcona example recommendation tool. *SIGSOFT Software Engineering Notes*, 30:237–240, 2005.

[12] O. Kononenko, D. Dietrich, R. Sharma, and R. Holmes. Automatically locating relevant programming help online. In *Proceedings of VL/HCC 2012 (The IEEE Symposium on Visual Languages and Human-Centric Computing)*, pages 127–134. IEEE, 2012.

[13] D. Lam, S. L. Rohall, C. Schmandt, and M. K. Stern. Exploiting e-mail structure to improve summarization. In *Proceeding of CSCW 2002 (ACM Conference on Computer Supported Cooperative)*. ACM, 2002.

[14] 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)*, pages 492–501. ACM, 2006.

[15] B. Lientz and B. Swanson. Problems in Application Software Maintenance. *Communications of the ACM*, 24(11):763–769, 1981.

[16] 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 (29th Conference on Human factors in computing systems)*, pages 2857–2866. ACM, 2011.

[17] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Proceedings of ICSE 2008 (30th International Conference on Software Engineering)*, pages 181–190. ACM, 2008.

[18] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proceedings of ICSE 2006 (28th International Conference on Software Engineering)*, pages 452–461. ACM, 2006.

[19] L. Ponzanelli, A. Bacchelli, and M. Lanza. Leveraging crowd knowledge for software comprehension and development. In *Proceedings of CSMR 2013 (17th IEEE European Conference on Software Maintenance and Reengineering)*, pages 59–66. IEEE, 2013.

[20] L. Ponzanelli, A. Bacchelli, and M. Lanza. Seahawk: Stack Overflow in the IDE. In *Proceedings of ICSE 2013 (35th ACM/IEEE International Conference on Software Engineering)*, pages 1295–1298. IEEE CS Press, 2013.

[21] S. Rastkar, G. Murphy, and G. Murray. Summarizing software artifacts: a case study of bug reports. In *Proceedings of ICSE 2010 (32nd ACM/IEEE International Conference on Software Engineering)*, pages 505–514. ACM, 2010.

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

[23] N. Sawadsky and G. Murphy. Fishtail: from task context to source code examples. In *Proceedings of TOPI 2011 (1st Workshop on Developing Tools as Plug-ins)*, pages 48–51. ACM, 2011.

[24] I. Sommerville. *Software Engineering*. Addison-Wesley, 7th edition, 2004.

[25] A. M. Sonia Haiduc, Jairo Aponte. Supporting program comprehension with source code summarization. In *Proceedings of ICSE 2010 (32nd ACM/IEEE International Conference on Software Engineering)*, pages 223–226. ACM, 2010.

[26] K. Spärck Jones. Automatic summarising: The state of the art. *Information Processing and Management*, 43(6):1449–1481, Nov. 2007.

[27] J. Stylos and B. A. Myers. Mica: A web-search tool for finding api components and examples. In *Proceedings of VL/HCC (The IEEE Symposium on Visual Languages and Human-Centric Computing)*, pages 195–202, 2006.

[28] 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)*, pages 804–807. ACM, 2011.

[29] T. Zimmermann, R. Premraj, and A. Zeller. Predicting defects for eclipse. In *Proceedings of PROMISE 2007 (3rd International Workshop on Predictor Models in Software Engineering)*, page 9. IEEE Computer Society, 2007.

[30] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *26th International Conference on Software Engineering (ICSE 2004)*, pages 563–572. IEEE CS Press, 2004.