Documenting and Sharing Knowledge about Code

Anja Guzzi
Software Engineering Research Group
Delft University of Technology
Delft, The Netherlands
A.Guzzi@tudelft.nl

***Abstract*—Software engineers spend a considerable amount of time on program comprehension. Current research has primarily focused on assisting the developer trying to build up his understanding of the code. This knowledge remains only in the mind of the developer and, as time elapses, often "disappears". In this research, we shift the focus to the developer who is using her Integrated Development Environment (IDE) for writing, modifying, or reading the code, and who actually understands the code she is working with.**

The objective of this PhD research is to seek ways to support this developer to *document and share* her knowledge with the rest of the team. In particular, we investigate the full potential of *micro-blogging* integrated into the IDE for addressing the program comprehension problem.

***Keywords*-Program comprehension; micro-blogging; IDEs; recommender systems; CSCW**

I. Problem Statement

Trying to understand a given piece of code (most likely written by someone else) is a major challenge in software maintenance. Studies have shown that this *program comprehension* process may take over 60% of the software maintenance effort [1]. When developing long-lived software products, design decisions made in the past significantly affect the work of developers today [2]. Software developers faced with the task to modify an existing program, must first obtain an understanding of the program at hand. Studies have shown that developers frequently seek information from source code, documentation, and colleagues [3].

While good documentation may help developers to answer questions regarding dependencies, behavior, algorithms, *etc.*, in practice documentation is costly to write, often outdated, and not suitable for the task at hand. Other developers are often sought for information because much of the information remains in their heads [4]. Unfortunately, identifying someone to talk to, who is *relevant* to one's information and coordination needs, is a challenging task [5].

Research in program comprehension is primarily focusing on investigating methods and techniques that support developers in understanding software they need to work with (*e.g.,* via software visualization, feature localization and architecture reconstruction). However, once the maintenance task is completed, most of this knowledge built up during the process of conducting the task will disappear, especially when not articulated in code or documents. The only permanent result is the modified software, and, optionally, some updates made to the requirements or (UML) design documentation.

This is an unfortunate situation, since this knowledge may be valuable for future maintenance tasks, possibly being conducted by different developers.

II. Approach and Research Questions

In this PhD project, we point our attention to the developers who understand the code they are working with and on ways in which they can document and share this understanding with team members, who later may have to understand the same code. In this way, we try to benefit as much as possible from the precious knowledge a developer builds up when working on a maintenance task, knowledge that is lost in traditional ways of working.

The solution we aim to pursue in this research is inspired by Web 2.0 in general, and *micro-blogging* in particular: Communication via short, real-time message broadcast such as status updates in social networks like Facebook, and message-exchange services like Twitter.[1] We propose a tight integration of micro-blogging into the IDE, which is the tool developers use for writing, modifying, and understanding code. Micro-blogging has the potential to help with the program comprehension problem, in that it fosters team communication through light-weight messages.

The questions we seek to answer include the following. How can we encourage developers to *micro-blog*, *within the IDE* and in a lightweight manner? When developers write about their activities and findings in short messages, how can we capture their knowledge about the system? How can we combine information collected in such way with interaction data automatically collected from the IDE, providing "location awareness" to messages? Can the collected knowledge be useful to augment *recommender systems*, pointing developers to relevant colleagues, classes, test cases, and so on?

The proposed research consists of five tracks, including a longitudinal evaluation involving a qualitative as well as quantitative assessment of the entire approach. It also involves the construction of a prototype tool, codename

[1] http://facebook.com, http://twitter.com

JAMES, offering a tight integration of micro-blogging capabilities into an IDE. Figure 1 depicts an overview of the JAMES approach.

III. METHODOLOGY

This research is divided into five tracks, and follows a mixed-method design, as described below.

T1: How Do Developers Exchange Knowledge about Their Code? We conduct an empirical study (following a *grounded theory* approach [6]) on the way developers currently communicate knowledge about their understanding of programming artifacts. Since our goal is to improve program understanding, it is of particular interest to investigate how developers talk about and reference *development artifacts*: these can include classes, packages, methods, layers, test cases, test results, design documents, diagrams, bug reports, work items, and so on. The outcome of this study will be a theory describing the way in which communication channels (mailing lists, chat, IRC, blog posts, *etc.*) are currently used, the sort of information posted on them, and the way in which developers discuss and reference their code artifacts. We will use this theory to position micro-blogging in the overall collection of communication channels.

T2: How Can Micro-Blogging Be Used to Exchange Information about Code? We investigate how new lightweight micro-blogging facilities can be integrated into an IDE, including a version-independent mechanism to reference code fragments. A number of technical challenges and research questions need to be addressed. The main technical challenge relates to the way in which *development artifacts* should be identified and referenced. How can we link a *word* in a message with an homonymous *class* in the code? Can we support some sort of code completion when entering class or method names? Furthermore, the references to artifacts should remain stable or meaningful as the underlying code evolves. Establishing a version-independent generic artifact referencing mechanism across work spaces is a significant challenge that needs to be addressed. Several additional questions arise with respect to the usage of a micro-blogging facility. What sort of messages are developers willing to share? How often would they write a message? What types of artifacts would they actually mention? Who should receive a particular message? How can we encourage developers to document their knowledge? Moreover, a number of social factors, as considered by Ye *et al.* in their framework supporting developers seeking information from their peers [7], need to be taken into account. To address these questions, we will develop a micro-blogging plugin for Eclipse that we will build upon the JAMES prototype [8], which we used to conduct some early stage experiments. We will use the JAMES server and plugin to conduct empirical studies on micro-blogging usage patterns. A set of guidelines for software-related communication tools, on how to encourage developers to share knowledge, will be among the outcomes of these studies.

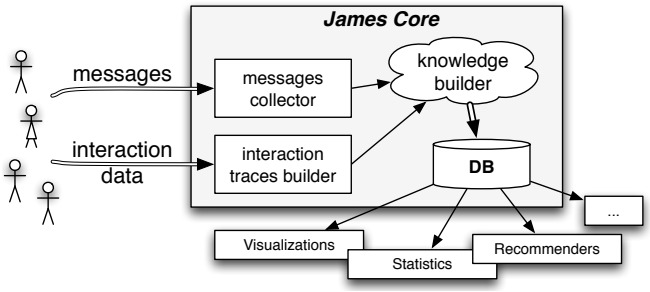

Figure 1. Overview of the JAMES approach

T3: Context-Aware Micro-Blogging in the IDE. We explore how the history of interactions (code artifacts inspected, executed, modified, and so on), may provide relevant *context* to a message. We will extend the JAMES plugin with automated interaction history collection facilities, as well as with mechanisms to automatically *link* artifacts to messages. We will then use this enhanced version in experiments in which developers use the plugin and assess the meaningfulness of the automatically established contexts for the program comprehension tasks.

T4: Recommender Systems Based On Micro-Blogs. In this track, we research how recommender systems could automatically suggest relevant code elements, co-developers, work items, *etc.*, taking micro-blog messages into account. Micro-blog messages written as part of a maintenance task, may be used to identify related messages written by others, which based on the linked code artifacts could lead to suggestions for code fragments to study. Likewise, the interaction trail followed by a developer, can lead to suggestions for other developers to contact who blogged about similar code elements. We will provide a prototype recommender implementation as part of the JAMES tool set, and evaluate the resulting recommendations in a number of open source projects. The recommender plugin will work continuously ("recommend-as-you-edit") as well as on-demand (entering questions in a JAMES search panel).

T5: Micro-Blogging Communication Patterns in the IDE. We analyze the use of the proposed micro-blogging techniques in longitudinal studies in which a number of teams use the approach for a period of several months, resulting in an enriched theory of program understanding exchange among developers. The study will focus on the way all proposed approaches, including automated context generation and recommendation mechanisms, will be used by the different teams. The study will involve the collection of qualitative data as well as quantitative data.

IV. Validation Strategy

Each of the tracks T1, T2, T3 and T4 includes an individual assessment. T5 concludes the research with a longitudinal evaluation in which we observe and analyze how teams (from industry as well as open source communities) use micro-blogging inside the IDE. Thus, the evaluation phase will take place throughout the project, with an emphasis on evaluating all proposed techniques together in the final stage.

Moreover, the JAMES plugin itself can be directly used by teams using Eclipse, the most popular Java IDE. It will be made freely available during the project, and we will adopt an active strategy to create a user base throughout the project. This user base will not only serve for evaluation purposes, but also to ensure immediate application of the research results, as the project is being conducted.

V. Progress

My PhD research started in November 2009,[2] under the supervision of Prof. Dr. Arie van Deursen and Prof. Dr. Martin Pinzger. We initially focused on defining a research outline, including a study of the literature. Additionally, in Summer 2010, I visited the CHISEL[3] group, where Prof. Dr. Margaret-Anne Storey gave valuable feedback on this research. Afterward, we mainly focused on track T2.

A. Micro-Blogging within the IDE - First Steps with JAMES

We developed a JAMES prototype tool and performed a first evaluation in an exploratory study with 7 developers. Our initial results provide strong indication of the great potential in the combination of micro-blog messages and interaction history automatically collected from the IDE.

We found that developers are generally willing to communicate their activities through micro-blog messages. We noticed that more than one fourth (28%) of all the collected messages contained an explicit reference to a code element, such as a class or method name. We also grouped messages into five categories: future intention, past activity, ongoing activity, comments, todos. We observed that connections between messages and interactions are in principle meaningful: messages, and the associated traces, contained information valuable to other developers working on similar tasks.

From the exploratory evaluation, we concluded that knowledge about the software being changed can be captured in the form of association of messages and IDE interactions. The details of this evaluation, and of our initial approach, can be found in [8].

B. Linking Code to Messages - POLLICINO

In October 2010, we collaborated with Lile Hattori and Prof. Dr. Michele Lanza from the REVEAL group[4] on documenting and sharing locations in the code.

We investigated (by means of online surveys and interviews) how developers exchange information about locations in the code and we explored code bookmarking as a lightweight manner to mark, annotate and share code locations. We developed a prototype tool, POLLICINO, to enhance bookmarks within the Eclipse IDE and we conducted a pre-test/post-test experiment to assess its usefulness [9].

The experiment results show that POLLICINO can be used to effectively (micro-)document a developer's findings, and that those can be used by others in his team. The tool did not always fit in the developers' workflow, especially during active development. This augments the need to explore ways to encourage developers to document their findings

C. Facilitating Communication between Engineers - CARES

Last summer (2011), I spent three months at Microsoft Research in Redmond as a Research Intern. During the internship in the Human Interactions in Programming (HIP) Group,[5] I was mentored by Dr. Andrew Begel. We mainly focused on facilitating communication between engineers who collaborate, but are not in the same team.

We studied, by means of surveys and interviews, how and why contact between developers on different teams occurs. Then, to facilitate communication between engineers, we built and deployed a tool called CARES: Colleagues and Relevant Engineers Support, which is an IDE-based tool that enables engineers to easily discover and communicate with the people who have contributed to the source code. A video demonstrating CARES' features, is available at *http://research.microsoft.com/~abegel/cares/demo.mp4*.

Our study (not yet published) has the first empirical data that reveals how engineers choose someone to speak with, in situ. Two academic publications about this work have been submitted and are currently under review.

D. Currently - Further Steps with JAMES

We focus on extending the JAMES tool to support version-independent referencing, automated interaction history collection, as well as automatic linking of artifacts to messages.

We are also interested in investigating which design elements can foster active knowledge sharing among colleagues. We will experiment with, *e.g.,* proposing developers to share information with different sets of people, and different ways of representing people within the IDE. We will then integrate these elements into the design of JAMES.

VI. Related Work

Software-related discussions over Internet Relay Chat (IRC) have been analyzed by Shihab [10] and an analysis of how wikis are used for documenting frameworks is provided by Dagenais [11]. While both are valuable starting points for ***T1***, an overarching conceptual framework is not available.

[2] A comprehensive list of publications from this PhD research can be found at *http://www.st.ewi.tudelft.nl/~guzzi/*

[3] www.thechiselgroup.org/, University of Victoria, CA

[4] www.inf.usi.ch/faculty/lanza/reveal.html, University of Lugano, CH

[5] http://research.microsoft.com/en-us/groups/hip/, Redmond, USA

An overview of collaborative software engineering is provided by Whitehead [12]. Both formal and informal forms of communication are needed, where micro-blogging can be included in the category of informal communication. Informal communication is particularly important for *task articulation work*, involving coordination with team members [13]. Web 2.0 techniques, including micro-blogging, provide new ways of collaboration and informal communication, and their tight integration in the software engineering field is attracting substantial interest around the world [14]. An example of the integration of Web 2.0 into the software development process is the work by Begel *et al.* on Codebook (inspired by Facebook) [5]. While micro-blogging is an active topic of research itself,[6] we know of no direct study exploring its potential during software development ***(T2)***.

The use of navigation history to assist developers is also an active area of research ***(T3)***. Fritz *et al.* conducted an empirical study assessing the relationship between a programmers activity and what she knows about the code base [15]. DeLine *et al.* conducted two studies which demonstrate that sharing navigation data can improve program comprehension "*and is subjectively preferred by users*" [16]. The identification of relevant code elements in the context of development tasks from navigation history was explored by Kersten *et al.* [17]. Their Mylyn tool includes a task-focused user interface and the automatic identification of relevant context. We focus on short messages, their potential to capture developers knowledge and their use in recommender systems. For the implementation of JAMES we will investigate the possibility of using parts of the Mylyn API.

Recommender systems are a topic of active research [18]. Among it, a study by Robbes on recommender systems based on recorded interactions [19]. We seek to augment such recommender systems based on interaction histories with messages directly provided by the developers (***T4***). We focus on *gathering* firsthand knowledge about the software system, rather than *reconstructing* it later.

VII. CONTRIBUTIONS

In summary, we expect this research will lead to the following innovations: (1) A theory on how developers exchange their understanding of programs; (2) Guidelines on fostering developers to document and share knowledge; (3) An approach for version-independent micro-blogging about artifacts in the IDE; (4) An approach for automatically deriving contexts relevant to messages; (5) Recommendation algorithms taking advantage of developer-provided messages; (6) A qualitative evaluation of the role of developers micro-blogging.

REFERENCES

[1] T. A. Corbi, "Program understanding: Challenge for the 1990s," *IBM Systems Journal*, vol. 28, no. 2, 1989.

[2] J. E. Burge, J. M. Carroll, R. McCall, and I. Mistrk, *Rationale-Based Sw. Engineering*, 1st ed. Springer, 2008.

[3] A. J. Ko, R. DeLine, and G. Venolia, "Information Needs in Collocated Software Development Teams," in *ICSE '07*, Minneapolis, MN, USA, 2007, pp. 344–353.

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

[5] A. Begel, Y. P. Khoo, and T. Zimmerman, "Codebook: Discovering and exploiting relationships in software repositories," in *ICSE'10*, Cape Town, South Africa, 2010.

[6] B. Glaser and A. Strauss, *The discovery of Grounded Theory: strategies for qualitative research*. Aldine Transaction, 1967.

[7] Y. Ye, Y. Yamamoto, and K. Nakakoji, "A socio-technical framework for supporting programmers," in *Proceedings of ESEC-FSE '07*, New York, NY, USA, 2007.

[8] A. Guzzi, M. Pinzger, and A. van Deursen, "Combining micro-blogging with IDE interaction to support developers in their quests," in *ICSM ERA*, Washington, DC, USA, 2010.

[9] A. Guzzi, L. Hattori, M. Lanza, M. Pinzger, and A. van Deursen, "Collective code bookmarks for program comprehension," *ICPC 2011*, pp. 101–110, 2011.

[10] E. Shihab, Z. M. Jiang, and A. E. Hassan, "On the use of Internet Relay Chat (IRC) meetings by developers of the GNOME GTK+ project," in *MSR '09*. IEEE, 2009.

[11] B. Dagenais, H. Ossher, R. K. E. Bellamy, M. P. Robillard, and J. P. de Vries, "Moving into a new software project landscape," in *ICSE '10*, New York, NY, USA, 2010.

[12] J. Whitehead, "Collaboration in software engineering: A roadmap," in *FOSE 2007*, Washington, DC, USA, 2007.

[13] R. E. Kraut and L. A. Streeter, "Coordination in software development," *Commun. ACM*, vol. 38, no. 3, 1995.

[14] C. Treude, M.-A. Storey, K. Ehrlich, and A. van Deursen, "Web2SE: First workshop on web 2.0 for software engineering," in *Companion to the Proceedings of ICSE*. ACM, 2010.

[15] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *ESEC/FSE '07*. New York, NY, USA: ACM, 2007, pp. 341–350.

[16] R. DeLine, M. Czerwinski, and G. Robertson, "Easing program comprehension by sharing navigation data," in *VLHCC '05*, Washington, DC, USA, 2005.

[17] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *FSE '06*. New York, NY, USA: ACM, 2006, pp. 1–11.

[18] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, July 2010.

[19] R. Robbes, "On the evaluation of recommender systems with recorded interactions," in *ICSE SUITE '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 45–48.

[6] See bibliography at http://www.danah.org/researchBibs/twitter.html