Towards Integrating E-Mail Communication in the IDE

Alberto Bacchelli, Michele Lanza, Vitezslav Humpa

REVEAL @ Faculty of Informatics - University of Lugano, Switzerland

ABSTRACT

Software developers spend a large part of their working time using an Integrated Development Environment (IDE). However, IDEs are usually disconnected from the means of communication programmers use to interact and discuss with other co-workers. Because of this, many context switches are required and the existing connection between source code artifacts and artifacts generated from recorded communications, such as e-mails, cannot be effectively put to good use.

In this paper, we introduce *Remail*, an Eclipse plugin to integrate e-mail communication in the IDE. It allows developers to seamlessly handle source code entities and e-mails concerning the source code. We present our preliminary work on Remail that allows linking source code classes to the e-mails in which they are discussed, thus providing an updated and effective documentation.

Categories and Subject Descriptors

D.2.7 [**Distribution, Maintenance, and Enhancement**]: Restructuring, reverse engineering, and reengineering

1. MOTIVATION

In 2006 LaToza *et al.* reported the results of an extensive study on developers' work habits conducted at Microsoft Corporation [8]. They proposed two surveys to software developers to understand how their work is conducted. Analyzing hundreds of replies and interviewing eleven developers, they concluded that software documentation is commonly inadequate, outdated, and hard to retrieve or link to actual source code entities. This is due to the fact that developers create and maintain implicit mental models of the source code. For this reason, developers who need to understand source code entities (*e.g.,* to know the design rationale behind a certain implementation, which is the most common information need for a developer [7]) have to query other programmers. Unplanned face-to-face meetings are the favorite communication means in this kind of situation, preferred over e-mails or instant messaging (IM) [7].

SUITE '10, May 1 2010, Cape Town, South Africa

This way of communication has serious drawbacks: frequent disruption of developers' attention (workers are interrupted every 7 minutes [10]), concentration of the knowledge about a part of code in a few developers, and inapplicability to globally distributed development projects, such as open-source ones. In addition, the information exchange that takes place in face-to-face meetings can be neither recorded, nor traced, so it cannot be used to update and improve existing written documentation, which will therefore be increasingly outdated and unreliable, generating a vicious cycle.

In their research, LaToza *et al.* also analyzed how and *where* developers used their time to understand, design, and communicate about source code. To *understand* some code, most of developers' time (more than 60%) is spent on reading, comparing, and debugging. Developers perform *all* these actions inside an IDE. To *design* new parts of the system, the IDE is still the first used application (being at a considerable distance from the second means, *i.e.,* the whiteboard). When not communicating face-to-face, developers *communicate* through e-mails. However, no matter how much related these e-mails are to software development, they are managed by programs that are external to the IDE, sometimes even in a web browser, and are completely disconnected from the software development itself. This is a major problem in a serious and consistent adoption of e-mails as a means of communication in software development: Since e-mail communication is not integrated in modern IDEs, developers are forced to interrupt their programming activities and change context every time they need to deal with e-mails, and there is no possibility to easily have traceability links connecting e-mails with source code entities.

While e-mails cannot substitute other means of communication (*e.g.,* design documents, commit comments), and are less pleasant than face-to-face meetings, they offer an effective way to exchange software development related information: E-mails are less disruptive for developers' concentration and can be recorded and traced. An integration of e-mails in the IDEs, the environment in which developers spend the vast majority of their working time, can offer these advantages and improve e-mail usage.

In this paper, we present Remail, a plugin for Eclipse[1], an open source IDE, we are devising to integrate e-mail support in the programming environment. We report our preliminary work, and we show how -in the context of the IDE- source code entities can be linked to the related e-mails fast and effectively, using lightweight linking techniques we assessed in our previous work [3, 5].

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

2. COMMUNICATION

According to Tom De Marco, the principal work of software developers is "human communication to organize the users' expressions of needs into formal procedure" [9]. We analyze the communication means used by developers to exchange source code related information. We present their advantages and drawbacks comparing them to e-mail, to give the grounds to our decision to improve e-mail adoption.

Face-to-face is considered the most effective communication form by collocated software developers [8]. It promotes camaraderie and no effort is wasted recording design information that might be outdated before being read [7]. However, if we consider a project in the long-term, face-to-face communication is not optimal. It is not scalable and is disruptive for developers: programmers are often interrupted by colleagues asking questions and must spend much time both to answer them and to regain the lost concentration necessary to complete the task they were doing before they were interrupted. These conversations are not recorded, thus they cannot contribute to the project documentation. Finally, this communication form cannot be applied in a globally distributed development environment.

Design documents span up to high level concerns and design rationales, which are among the most popular information needs of developers [7]. However, since design documents are written before the actual source code implementation and they usually present the software design at a high level of abstraction, they are more difficult than e-mails to be linked to the source code entities they are related to [1, 5]. In addition, developers often regard design documents as "write-only media" [8], as they are difficult and cumbersome to evolve along with the rest of the system, and not all the developers have the right to modify them.

Code comments have the advantage of being already linked with source code entities they are referring to. However, source code comments have a very definite and narrow focus (*e.g.,* they cannot be used to describe design rationale), and in the case they refer to entities in the source code other than those which include them, these are linked only in developer's mind. Moreover, developers perceive source code comments as extremely "authoritative" [8]: Adding or modifying code comments means actually committing a change in the source code repository, and this automatically labels the modifier as an "expert" of that piece of code. E-mails, on the contrary, can be more informal as they report the individual point of view of the author. Finally, source code comments cannot be used as a place to conduct discussions.

Commit comments are linked to specific files and versions. Over code comments, commit comments can have a broader focus, because they are linked to more than a single source code file (*i.e.,* all the committed files). However, in practice, they are only used for short descriptions about code changes or fixes. As commit comments, it is not possible to "reply" to them and start a discussion. In order to overcome this limitation, modern source code management systems provide facilities to forward commit notifications and comments to development mailing lists.

Bug reports: Broadly used issue/bug tracking systems (e.g., Bugzilla[2], Jira[3], Mantis[4]) offer to developers the possibility to not only create and manage issue reports (*e.g.,* trough insertion, modification, deletion), but also to communicate with the reports in a threaded fashion. Through pattern matching and heuristics, it is possible to find the traceability links between bug reports and source code files with a good precision [6]. However, the communication in bug reports must be focused to the issue itself and it hardly crosses these borders: As opposed to e-mails, it is uncommon to find high level concerns or design rationale discussed in this media. As a confirmation about the reduced focus of bug reports, new developers are commonly employed to work on bug fixing, as it requires less design knowledge than implementing new features [8].

Instant Messaging: Developers exchange synchronous messages using real-time communication protocols (*e.g.,* Jabber, Internet Relay Client (IRC)). Although this genre of communication is not much used in collocated industrial settings [7, 8], it is widely employed by communities of open source developers, for rapid coordination and as a means to conduct online meeting that discuss development issues [11]. The advantage of IM consists in almost immediate responses, the parallel participation of many people, and the availability of light and effective protocols. Unfortunately, the synchronous aspect of IM, while improving the dynamics of interactions, brings also disadvantages when compared to asynchronous means, such as e-mails: IM causes interruptions to developer concentration (similarly to face-to-face meeting), it can create difficulties for developers residing in different time-zones, and it does not allow later-replies (*e.g.,* replies to finished online meetings). Moreover, the kind of jargon used in IM is generally terse, implicit, and noisy, having an impact on traceability.

E-Mails: In the surveys by LaToza *et al.*, developers reported that e-mail questions take hours, if not days, to receive a response, that the e-mail meaning is often misunderstood by the reader, and that e-mails are tedious to write [8]. These problems are also common in other means such as design documents or bug reports, and partially in IM. However, e-mails offer advantages that make them one of the optimal choices between all the communication means: They can be used to discuss issues ranging from low-level decisions (e.g., implementation, bug fixing) up to high-level considerations (e.g., design rationales), they can be written and read by both software system developers and beta-testers or end-users, they always come with additional information (e.g., time-stamp, thread, author) that can be taken into account, they can be linked to any source code entity they are related to, they can be used to study the evolution of a system, and they can be employed in the prediction of defect-prone entities [2].

For all these reasons, we consider e-mail as one of the best candidates to improve the communication between developers. Our goal is to support in a seamless way e-mail communication within integrated development environments.

3. REMAIL

In this section we present the architecture of Remail, the Eclipse plugin we are developing to integrate e-mail communication in the IDE. Remail allows a developer, without ever exiting from the Eclipse IDE, to select a Java class, automatically retrieve all the related messages from a specified e-mail archive, and read them in a convenient way.

[2] `http://www.bugzilla.org`
[3] `http://atlassian.com/software/jira`
[4] `http://www.mantisbt.org`

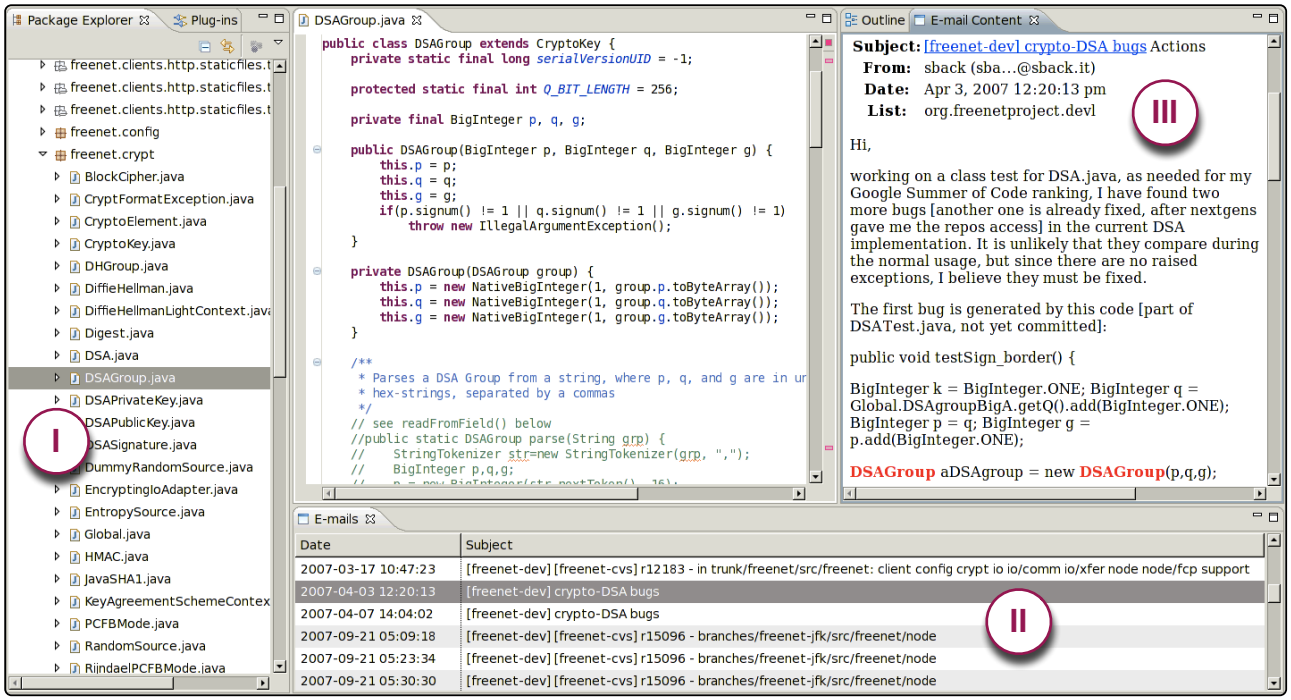

Figure 1: The Remail Eclipse plugin

3.1 E-Mail archive

The current implementation of Remail relies on an external database managed by the DBMS PostgreSQL[5], which stores all the e-mails pertaining to the chosen software system, and it is used by Remail to retrieve the e-mails related to the chosen class with a specific technique.

A Remail compatible database, containing mailing lists from thousands of open source projects, can be generated through Miler, a tool infrastructure we devised to analyze mailing lists [4]. Using Miler, it is possible to import any mailing list offered by the free service MarkMail[6] and store it in a database as requested by Remail. For example, while developing our plugin, we used the free-software project Freenet[7] as a case study. The complete source code repository for this project is easy to access and to import, but the mailing list archives are only accessible from web pages that do not offer any concrete importing functionality. Using Miler, we imported and stored, in a Remail-compliant database, the `org.freenetproject.devl` mailing list, which contains discussions concerning the Freenet development.

3.2 Eclipse integration

Figure 1 shows how Remail is integrated in the Eclipse IDE. First of all, the menu in the "Package Explorer", a panel that is commonly used to navigate through the classes of a Java project (Figure 1, Point I), has been extended with a new entry: When a developer selects and right-click on a class she is offered the option "E-mail Recommender". As shown in Figure 2, it contains a sub-menu that allows the user to select the preferred class-to-mails linking technique.

[5] `http://www.postgresql.org/`
[6] `http://markmail.org/`
[7] `http://freenetproject.org/`

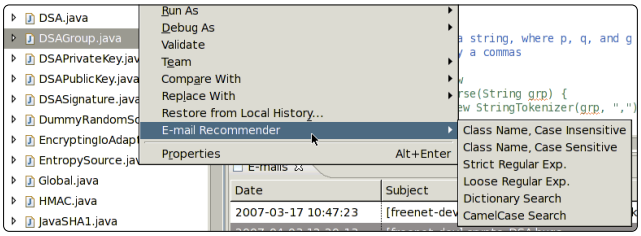

Figure 2: Linking Technique Selection

The six lightweight approaches proposed to find the class-to-mails link were assessed in previous work [3]. The substantial difference between the methods is the average *precision* and *recall* when finding the links they recommend. Precision and recall are widely-known metrics used in the Information Retrieval field: the latter specifies how many links between all those present are retrieved by the technique, while the former measures how many links are correct over all the links recommended by the technique. For example, the "Class name, Case Insensitive" method offers, on average, a high recall, but a low precision: this technique finds more links compared to the others, but many of them can be not relevant. The method "Strict Regular Exp.", instead, recommends less links (*i.e.,* with a low recall), but more relevant (*i.e.,* with a high precision).

Once a technique is clicked, after a few seconds, the users is notified with the number of mails retrieved, and the "E-mails" panel is filled as shown in Figure 1, Point II. E-mails are sorted by date and their subject is visible. As opposed to what we did in our previous work [3], we did not remove from

the e-mail archive all the messages automatically generated by the source code management system, in this way e-mails containing commits and commit comments related to the chosen class can be retrieved.

From the "E-mails" panel, the user can double-click on any e-mail and see the content in the "E-mail content" panel (Figure 1, Point III). The panel also contains the e-mail subject, the author's e-mail address, the date, and the list in which it was submitted. Thanks to the format in which e-mails are imported by Miler, it is also possible to show the e-mail content using different colors according to the quotation level (as usually done in popular mail clients). Finally, as shown in the figure, the name of the class from which the searching started is red colored, making it easier to find parts of the content specifically referring it.

4. RELATED WORK

Holmes and Begel devised *Deep Intellisense*, a Visual Studio IDE plugin that links a number of artifacts (*e.g.,* bug reports, e-mails, code changes) to source code entities. It uses an implicit query system [12] to find links and offer them in an inverted chronological order. Our work focus on e-mail communication, and offers different linking techniques to be used according to developers' needs. In addition, we plan to completely integrate e-mail communication in the IDE to avoid any context switch, while Deep Intellisense relies on external applications to handle the different artifacts.

Official support to communication has started to be featured by different IDEs: Eclipse, NetBeans[8], and IntelliJ IDEA[9] offer an integration to issue tracking systems and IM communication. NetBeans and IntelliJ offer new IM protocols with features thought for software development, such as "code pointers" –references to a particular point in a file, and visualization of differences between files opened by two developers that are messaging each other.

IBM Jazz[10] offers a framework, built on top of Eclipse, to support collaborative software development. It features IM communication in the IDE, and uses the concept of "work items" to track and coordinate development tasks and workflows. Each work item is connected to other artifacts (*e.g.,* builds, defect reports, change sets, or source code entities). While Jazz advises the use of a brand new technology, we decided on harnessing the power of e-mails, a pre-existing popular communication means successfully adopted by developers of a vast majority of software projects.

5. CONCLUSION

We introduced Remail, an Eclipse plugin we are developing to integrate e-mail communication in the Eclipse IDE. We motivated our work through an analysis of developers' communication habits. We showed how we integrated Remail in Eclipse and how it offers class-to-emails traceability. Finally, we presented how current work towards communication in IDE is related to ours. As our future work, we plan to extend Remail (1) allowing developers to use e-mail archives in different formats, (2) offering mail-to-classes traceability (*i.e.,* access the source code of classes mentioned in e-mails), (3) enabling developers to write e-mails from the IDE, making it easy to annotate them with references to selected classes, (4) adding a not-intrusive notifier that tracks new e-mails about classes in which a developer is interested.

Acknowledgments We gratefully acknowledge the financial support of the Swiss National Science foundation for the project "DiCoSA" (SNF Project No. 118063).

[8] `http://netbeans.org/`

[9] `http://www.jetbrains.com/idea/`

[10] `http://jazz.net/`

6. REFERENCES

[1] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[2] 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 xxx–xxx, 2010.

[3] A. Bacchelli, M. D'Ambros, M. Lanza, and R. Robbes. Benchmarking lightweight techniques to link e-mails and source code. In *Proceedings of WCRE 2009 (16th IEEE Working Conference on Reverse Engineering)*, pages 205–214. IEEE CS Press, 2009.

[4] A. Bacchelli, M. Lanza, and M. D'Ambros. Miler - a tool infrastructure to analyze mailing lists. In *Proceedings of FAMOOSr 2009 (3rd International Workshop on FAMIX and Moose in Reengineering)*, 2009.

[5] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of ICSE 2010 (32nd International Conference on Software Engineering)*, pages xxx–xxx, 2010.

[6] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proceedings of ICSM 2003 (19th IEEE International Conference on Software Maintenance)*, pages 23–32. IEEE CS, 2003.

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

[8] 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.

[9] T. D. Marco. *Peopleware - Productive Projects and Teams*. Dorset House, 1999.

[10] G. Mark, V. M. Gonzalez, and J. Harris. No task left behind?: examining the nature of fragmented work. In *Proceedings of CHI 2005 (SIGCHI Conference on Human Factors in Computing Systems)*, pages 321–330. ACM, 2005.

[11] E. Shihab, Z. M. Jiang, and A. E. Hassan. Studying the use of developer irc meetings in open source projects. In *Proceedings of ICSM 2009 (25th IEEE International Conference on Software Maintenance)*, pages 147–156. IEEE, 2009.

[12] G. Venolia. Textual allusions to artifacts in software-related repositories. In *Proceedings of MSR 2006 (International workshop on Mining software repositories)*, pages 151–154. ACM, 2006.