Tools for light-weight knowledge sharing in open-source software development

Davor Čubranić, Reid Holmes, Annie T.T. Ying, and Gail. C. Murphy

Department of Computer Science
University of British Columbia
201-2366 Main Mall, Vancouver BC
Canada V6T 1Z4
E-mail: {cubranic, rtholmes, aying, murphy}@cs.ubc.ca

1. Introduction

Open-source projects are almost by definition developed by *virtual teams*: "groups of people with a common purpose who carry out interdependent tasks across locations and time, using technology to communicate much more than they use face-to-face meetings" ([4], p346).

In such settings, communication and diffusion of information becomes more difficult [10]. As a result, maintaining awareness and coordination of team's activities becomes a challenge [7] and sharing the expertise with one's colleagues requires more time and effort.

In this paper we present our ongoing work on using existing open-source processes to provide developers with ways to learn and benefit from the past experiences of others working on the same project. Our aim is to enable *light-weight* knowledge sharing: without requiring additional system infrastructure or additional work to create those knowledge repositories or disseminate the experiences.

2. Knowledge Portals

One of the solutions being proposed for the problem of sharing knowledge in distributed organizations are knowledge portals [11]. Knowledge portals are single-point-access software systems intended to provide easy and timely access to information and to support communities of knowledge workers who share common goals. The vision for knowledge portals is for them to support knowledge workers in their "gathering of information relevant to a task, organiz[ing] it, search[ing] it, and analyz[ing] it, sythesiz[ing] solutions with respect to specific task goals, and then shar[ing] and distribut[ing] what has been learned to other knowledge workers" ([11], p. 926).

Open-source software projects, of course, already use a variety of computer-mediated communication and coordination mechanisms: mailing lists and Usenet newsgroups, CVS source management repositories, and Bugzilla issue-tracking systems are the most-common examples of such tools. These tools already are, in effect, repositories of knowledge about a project, and we could arguably call web sites unifying access to those tools (for example, SourceForge [1]) a rudimentary knowledge portals, especially when coupled with ability to search the archives.

The problem is that such sites still do not offer much support to developers engaged in the kinds of knowledge tasks described above, especially in the last step: sharing and distributing what has been learned to other members of the team. The developer wanting to use the "portal" has to rely on, at best, keyword searching to find bits of previously-collected information that may be of use in her current task. Furthermore, once this information has been collected—from questions answered in the past on the newsgroup, for example—there is little or no support to organize it (or "distill" it, for easier reference in the future [1]).

3. Our Projects

One of the main barriers to providing effective knowledge portals is getting a sufficient amount of useful information into the portal to attract a critical mass of users, thereby further increasing the information available in the portal.

Getting useful information is complicated by the well-known fact about groupware systems that asking people to do extra work when it will only benefit others tends to turn users away [8]. For this reason, we are focusing our research efforts on ways of taking advantage of existing social and work processes in open-source software to build repositories of useful project knowledge. In this section we describe three such ongoing projects, each taking a different tack at the problem and targeting a different audience.

[1] www.sourceforge.org

3.1. Hipikat

Hipikat [6] is a tool that forms an implicit group memory from the information stored in a project's archives, and that recommends artifacts from the archives that are relevant to a task that the developer is trying to perform. Therefore, Hipikat's target user is a new developer who joined an open-source project and, because of temporal and geographic separation, does not have the level of mentoring support from more experienced colleagues that is normally present in co-located software teams. When using Hipikat, these newcomers would not have to find their way amongst the huge amount of archived, electronic information that is maintained as part of the project, but would instead be recommended a small, relevant subset on which to focus their understanding effort.

The artifacts used to build the group memory are drawn from a number of sources: newsgroup and mailing list archives, items in the Bugzilla database, CVS repository, and the project Web site. Hipikat then infers links between the artifacts that may have been apparent at one time to members of the development team but that were not recorded. For instance, Hipikat infers which source revisions correspond to which Bugzilla items, and infers similarity between Bugzilla items. Later, using these links the tool, in a role similar to that of a mentor, suggests possibly relevant parts of the group memory given information about a task a newcomer is trying to perform.

Hipikat exploits the information and artifacts that are produced in the normal course of a project's development activities. Its group memory is built transparently to the developers and does not require any additional work on their part.

The Hipikat prototype is a client-server system. Hipikat is currently instantiated for the Eclipse project,[2] but has been designed to be adapted easily to other open-source projects that follow the general model of open-source software development [5]. Eclipse is an extensible integrated development environment platform that was originally developed by IBM, and was subsequently released under an open-source license. The platform can be extended through *plug-ins*. Basic Eclipse distribution includes plug-ins for Java development and for communication with CVS.

Since Eclipse is self-hosted, we wrote the client as a plug-in that works within the IDE. This approach permits the Hipikat client to integrate seamlessly into a full-featured work environment, and to thus be used in combination with other software engineering tools plugged into Eclipse. For example, an Eclipse developer can use both Hipikat and the Java search feature that comes bundled with the default Eclipse distribution.

[2] `www.eclipse.org`

We are currently evaluating Hipikat on a group of newcomers to Eclipse development working on enhancements requested for Eclipse in the past. The client is also available for download at `www.cs.ubc.ca/labs/spl/project/hipikat`.

3.2. Code Example Queries

A new approach on which we are currently working is making recommendations on proper API usage based on established practices in the existing code base. This approach works orthogonally to Hipikat, which currently ignores the content of source artifacts in the CVS repository in favour of their metadata—i.e., author, creation time, and the check-in comment. Identifying which APIs the developer is using by looking at the code and making suggestions based upon correlations between the current API use with similar usage patterns used by other developers may provide useful hints for the developer in accomplishing the current task. This could be extended to looking at every API used in the current method or class and making searches based on the intersection of this API usage.

As in Hipikat, we are applying this technique in the context of the Eclipse project. This project is attractive because it involves two separate communities of developers: that working on the core Eclipse platform with the core set of standard plug-ins (e.g., for Java editing and debugging), and a much larger community of developers working on third-party Eclipse plug-ins that use the extension points provided by the Platform and the core plug-ins to integrate into the IDE. While Hipikat is targeting the first community, third-party plug-in developers are clearly more interested in good API usage than in the history of changes in the Eclipse platform code, and it is they for whom we expect code-example queries to be the most useful.

The information on which recommendations would be based would be populated with statistics from core Eclipse plug-ins. Each core plug-in will have all of its API calls identified and cataloged. These calls will be analyzed for relationships between one and other. Patterns will be examined between these calls in the plug-ins indexed in the database. The relationships discovered from the plug-in usage of the API will form the basis of the results for code-example queries.

Similarly to Hipikat, the client would detect what the user is trying to do based on the context of their current work and then make recommendations based on this information. This contextual data could be derived from methods and classes being used by the code the developer is working on, as well as comments in it. Query responses should consist of a simple list of examples which are each weighted by the amount of similarity to the current work being performed. A textual description of the method or class being

referenced as the example should be provided based on the comments that exist of that class or method, if they exist. Further examination of each of these examples should show the example with the areas used to match the example to the current context highlighted so the developer can easily determine whether or not the example actually fits the current task. Future API calls that the developer may want to investigate based upon the example code should also be highlighted so the developer can quickly see calls that may be relevant to further work on the task. Examples should be queriable as well, as the current context is, to find more examples similar to the one being currently examined.

3.3. Source Code Change Patterns Extraction

Another approach currently being implemented is the extraction of source code change patterns from a versioning repository. Our belief is that information on the parts of source code that tended to be modified together by developers in the past is helpful in the implementation a current task, and is especially useful in providing an indication of possible couplings that may not be immediately obvious from the code itself, but need to be taken into account when making the change.

Again, this is the kind of knowledge that would be easy to share in geographically, or even just temporally, collocated teams, when a developer can see which files in the repository are being updated by his or her co-workers. It is also the kind of knowledge that will not necessarily be recorded in the documentation or explicitly shared through an online knowledge repository as those mentioned in Sections 4.1 and 4.2, but is relatively easy to automatically extract from the project history recorded in the source repository.

The patterns of source code changes would be given by a list of rules of the form "when this part of the source code is changed in the implementation of a task, those other parts of the source code are likely to be changed in the same task." These rules can be found using association rule mining [2] or correlation rule mining [3]. The idea of association rule mining is to find the rules satisfying a predetermined support and confidence levels, which ensure that a rule covers a certain number of cases (support) and has a certain predictive strength (confidence). Correlation rule mining uses chi-square statistic to measure the significance of the rules by comparing the observed frequency counts of different source code changes with the expected frequency counts assuming the changes are independent.

Several issues deserve further investigation. First, determining what parts of source code belonging to one semantic change from CVS logs is tricky. Developers can check in source code for multiple changes together or check in a single semantic change over multiple commits. Second, because the number of association rules discovered depends on the user-specified support threshold (and confidence threshold), choosing a good support threshold is tricky. Setting the support threshold too high would eliminate potentially interesting rules; setting the support threshold value too low would kill the efficiency and would likely return many uninteresting rules, which would cause problems with the acceptance of the approach by developers. We are investigating whether alternative mechanisms other than support and confidence to measure the interestingness of the rules are more appropriate.

Our tool uses information from the CVS repository and the Bugzilla database of the Eclipse project, since we already have this information recorded in our Hipikat artifact database.

4. Related Work

4.1. Weblogs

Weblogs are frequently updated sites, usually consisting of many relatively short posts of commentary on recent events or articles elsewhere on the web. The posts are typically time-stamped and organized in reverse chronology so that a reader will always see the most recent post first. [13]

While weblogs are not commonly associated with open-source software engineering, they are certainly a significant and influential presence in the open-source community. Slashdot,[3] for example, is an extremely popular technology-oriented weblog, that is often used to quickly disseminate news about computing in general and open-source software in particular—sometimes resulting in a phenomenon known as *slashdotting*, where a mention of a new project or site on Slashdot results in its readers rushing to view the referenced site, overwhelming its Web server.

More recently, weblogs have started being used in open-source projects to keep a sort of a diary of developments in the project for the wider audience of non-core developers and potential users. This is typically used in very large projects where there is a significant user community (e.g., MozillaZine, a weekly update for the Mozilla project [4]) or where a project that is still in the early stages has generated a significant interest because of the caliber of its core developers (e.g., Chandler personal information manager, which is led by Mitch Kapor, author of Lotus 1-2-3 and founder of the Electronic Frontier Foundation,[5] and includes on its team Andy Hertzfeld, author of Macintosh's operating system and graphical user interface). When such Weblogs allow posting of readers' comments, they often turn into repositories of design discussions, links to similar projects,

[3] `www.slashdot.org`

[4] `www.mozillazine.org`

[5] Kapor's weblog: `blogs.osafoundation.org/mitch/`

and stories sharing lessons learned from past experiences. Ultimately, however, weblogs belong to and are driven by a single author, and if the maintenance of the weblog becomes too much of a time burden, and the weblog is not regularly updated with new postings, its audience will leave and the weblog will die.

Furthermore, Weblogs are fundamentally focused on sharing information with audiences in near temporal proximity. The way weblog sites are organized—with older content being slowly pushed off the page, and little or no facilities for organizing it and browsing or retrieving it at a later time—makes them far more suitable to those simply keeping up to date than to someone looking for information needed to execute a specific task. For this reason, we see weblogs and our work as complementary, and indeed Hipikat could easily use project weblogs as its information sources when those exist in a project.

4.2. Wiki

A *wiki* [6] is a tool for collaborative development of documents and web pages, although the term is also used to describe such web sites themselves. A wiki page can be easily edited by any reader and new content added without prior review. Since the content can be easily added, refined, and hyper-linked with related topics in the wiki and links to external sites, the result is attracting an active community of users/authors with a stake in further development of the site as the knowledge of its topic grows.

Wikis have been used in software engineering from the very beginning: the original Wiki-Wiki Web [7] was created in 1995 as a companion site to the Portland Pattern Repository. More recently, wikis have found use in open-source software development as knowledge- and tip-sharing repositories, often independent of the developer group (e.g., the Gnome Wiki [8] or the Eclipse Wiki [9]), although there are also projects using them in the design stage. For example, the Chandler, already mentioned above, is using its Wiki [10] to provide a forum where design issues can be discussed and organized in a collaborative fashion by both the core developers and a wider community interested in the project.

At their best, wikis are excellent examples of organizational memory systems grown organically by an active community with a stake in the product. Surprisingly, although the editing is open to all, the resulting product can be of very high quality (although some technical mechanisms had to be introduced to guard against occasional vandals defacing the site or users persistently trying to impose their controversial point of view on a topic in opposition to the rest of the community). However, since authoring on wiki is entirely manual (i.e., hand-editing a kind of simplified HTML form of its pages), maintenance and growth of the site are the main problem, otherwise—just as with weblogs—the user (and author) community will leave.

4.3. Code reuse

Similarly to Hipikat, Ye and Fischer's CodeBroker [16] uses information retrieval methods to determine software artifacts to suggest in the context of a developer's current task. However, CodeBroker is tailored to helping a developer on small-scale reuse tasks: The tool monitors a developer's use of a text editor watching for the method declarations and the descriptions of those methods in comments, the tool uses that information as a query to a library to find potential components that could be reused instead of a new component being created. In contrast, when used as a reuse tool, Hipikat works at the granularity of a task, providing such information as documents describing how a component is to be used with other components. The CodeBroker approach also relies on a developer properly formatting documentation in the component being defined, and on the presence of properly formatted documentation in the components in the reuse library. Hipikat avoids placing any additional requirements on the developers, making use of information that is potentially more informal. In this regard, Hipikat is more similar to the *Remembrance Agent* [15], which used information sources, such as user's email folders and text notes, to present documents relevant, or similar, to the one currently being edited.

CodeBroker's approach to implicitly determining the query context from JavaDoc comments and method declarations is similar to our code-example querying. The difference, however, is in our focus on patterns of API usage, rather than descriptions of components or their names and parameter signatures. Additionally, our approach is less reliant on component's documentation and names to find similarities and make recommendations, making it more robust in the environment of open-source software development.

4.4. Example-Based Programming

Several tools exist to match developers with existing examples of API usage. Henninger's CodeFinder and PEEL [9], Michail's CodeWeb [12], and Neal's example-based programming system [14]all tackle the problem of providing examples to developers in different ways. Neal's approach couples a syntax-directed editor to a repository of example programs. The editor is augmented with an additional example code pane. The contents of this pane

[6] also called *wiki-wiki*, meaning "quick" in Hawaiian, and *Wiki-Wiki Web*

[7] `c2.com/ci/wiki`

[8] `gnomesupport.org/wiki/`

[9] `eclipsewiki.swiki.net/1`

[10] `wiki.osafoundation.org/bin/view/Main/WikiHome`

are selected by choosing examples from a list of available routines. Henninger's CodeFinder and PEEL are two complimentary tools which facilitate providing examples to developers. PEEL identifies reusable components in a semi-automatic fashion, while CodeFinder allows developers to search for, and display, examples and their representations. CodeFinder represents examples in a hierarchy, and allows the developer to construct specific queries or browse the hierarchy manually to locate relevant artifacts. CodeWeb compares example systems against software libraries and displays parts of the examples which are relevant to the library. This allows developers to quickly identify the major library components, the components specific to the example being examined, and the quick location of example code using the library. Each of these tools provides a different method of indexing examples and locating them for developers. Our approach hopes to automate the task of locating examples by taking advantage of the context of the developers current work. By reducing the overhead required to to identify relevant examples, and by providing additional information about the relevance of different parts of each example, we hope to better match developers with artifacts which would otherwise be extremely difficult to locate and use.

5. Conclusion

In this paper we have presented three ongoing projects that aim to enable sharing of knowledge and experiences in open-source projects that is light-weight and does not require any additional work on the part of project members, as is usually needed to build knowledge repositories. While we do not intend our tools to be the perfect oracles or entirely replace the help given by a developer's colleagues, we do expect that they will alleviate the problems with knowledge-sharing and mentoring introduced by the open-source development's environment.

References

[1] M. S. Ackerman and D. W. McDonald. Collaborative support for informal information in collective memory systems. *Information Systems Frontiers*, 2:333–347, 2000.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., 26–28 1993.

[3] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlatio ns. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 265–276, 1997.

[4] C. D. Cramton. The mutual knowledge problem and its consequences for dispersed collaboration. *Organizational Science*, 12:246–371, May–June 2001.

[5] D. Čubranić and K. S. Booth. Coordinating open-source software development. In *Eighth IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 61–65, Stanford, CA, USA, 16–18 June 1999. IEEE Computer Society Press.

[6] D. Čubranić and G. C. Murphy. Hipikat: Recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, 2003.

[7] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: dealing with distance in r&d work. In *Proceedings of the International ACM SIGGROUP conference on Supporting group work", year = 1999, pages = 306–315, location = "Phoenix, AZ", publisher = ACM Press,*.

[8] J. Grudin. Groupware and social dynamics: eight challenges for developers. *Communications of the ACM*, 37(1):92–105, Jan. 1994.

[9] S. Henninger. An evolutionary approach to constructing effective software reuse repositories. *ACM Transactions on Software Engineering and Methodology*, 6(2):111–140, 1997.

[10] S. Kiesler and J. N. Cummings. What we know about proximity and distance in work groups? In P. Hinds and S. Kiesler, editors, *Distributed Work*, pages 57–80. MIT Press, Cambridge, MA, 2002.

[11] R. Mack, Y. Ravin, and R. J. Byrd. Knowledge portals and the emerging digital knowledge workspace. *IBM Systems Journal*, 40:925–955.

[12] A. Michail. Learning to use a software library through user-selected examples.

[13] T. Mortensen and J. Walker. Blogging thoughts: Personal publication as an online research tool. In A. Morrison, editor, *Researching ICTs in Context*, pages 249–279. InterMedia Report, Oslo, Norway, 2002.

[14] L. R. Neal. A system for example-based programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 63–68. ACM Press, 1989.

[15] B. J. Rhodes and T. Starner. Remembrance agent. In *The Proceedings of The First International Conference on The Practical Application Of Intelligent Agents and Multi Agent Technology (PAAM '96)*, pages 487–495, 1996.

[16] Y. Ye and G. Fischer. Information delivery in support of learning reusable software components on demand. In Y. Gil and D. B. Leake, editors, *Proceedings of the 2002 International Conference on Intelligent User Interfaces (IUI-02)*, pages 159–166, New York, Jan. 13–16 2002. ACM Press.