Can Peer Code Reviews be Exploited for Later Information Needs?

Andrew Sutherland
University of Saskatchewan
andrew.sutherland@usask.ca

Gina Venolia
Microsoft Research
ginav@microsoft.com

Abstract

Code reviews have proven to be an effective means of improving overall software quality. During the review, there is an exchange of knowledge between the code author and reviewer that concerns the code being reviewed. We performed a study that looked at the code review practices of software product teams at Microsoft. The study results indicated that code reviews are a point at which design rationale is explicitly stated, but that retention and recovery of this information is not well supported in the current environment. The results also indicated that code reviews in collocated development environments such as Microsoft use a mix of face-to-face and electronic communication.

1. Introduction

Code review has been demonstrated as an effective way to reduce software defects and improve overall software quality [1, 2]. Sauer et al. characterize a code review as an "organizational device for detecting defects in software products" [12]. It has been shown that the cost of fixing errors becomes higher the later they are reworked [4], so frequent reviews should also decrease development time and cost in the long run. The formality of a review can vary widely, from the very formal code inspection, to the more lightweight desk-check review.

In both formal and lightweight reviews, the code author and the reviewer explicitly state knowledge about the code to one another, in service of the review at hand. This knowledge concerns the changes being reviewed as well as the code's current and future state. Previous research has indicated that this knowledge is usually maintained in the developers' mental models [9]. The fallibility and inaccessibility of this tacit knowledge made it difficult for developers to work effectively and efficiently [7].

We hypothesized that the knowledge exchanged during code reviews could be of great value to engineers later trying to understand or modify the code. (This value would be above and beyond the immediate benefits of the code review.) For this to be the case, however, the code review dialog would have to be captured systematically and readily available to the later engineer.

To validate our hypothesis we performed an investigation of the practices of software product teams at Microsoft. Goals of the study were to better understand the nature of the code review dialog and exchange of information, how the information was retained, and the nature of its later reuse. The study consisted of a preliminary email questionnaire, a survey, and an examination of email transcripts from reviews conducted over email.

2. Related Studies

There have been previous studies examining the practices of code inspection and code review. Stein et al. conducted a study focusing specifically on distributed, asynchronous code inspections [13]. The study included evaluation of a tool that allowed for identification and sharing of code faults or defects.

Laitenburger conducted an exhaustive survey of code inspection methods, and presented his taxonomy of code inspection techniques [8]. An investigation into code review practices in open source development and the effect they have on choices made by software project managers was conducted by Johnson [6]. Several of these studies have also resulted in collaborative tools to assist in review practices [3, 5, 10].

Rigby *et al.* performed a study that focused on code review practices in open source software development [11]. The authors data-mined the email archives from the Apache project, and found that reviews were typically small and frequent, and that contributions to a review were often brief and independent from one another. Not surprisingly, reviews were almost always performed asynchronously.

Ko *et al.* studied collocated development teams and identified a number of day-to-day information needs that the members of these teams encountered [7]. These needs included reasoning about design and maintaining awareness about the activities of other team members. When studying developer work habits, Latoza *et al.* found that many problems encountered by developers were related to understanding the rationale behind code changes, as well as gathering knowledge and expertise from other members of their team [9]. Studying code review dialogs and providing tool support for structuring and retaining code review knowledge may satisfy the needs identified in these two studies.

3. Study

While the nature of code reviews at Microsoft was not entirely clear prior to the study, there were certain facts about the development process that were well established [14]. Development typically occurs in a localized, bottom-up fashion with smaller teams or individuals contributing code to a larger project. This results in a strong sense of code ownership for developers. Development teams are usually localized in the same building, often allowing meetings to occur in an ad-hoc, opportunistic fashion.

3.1. Questionnaire

The study was initiated with an email questionnaire sent out to random sampling of product team members. The goal of this questionnaire was to gather preliminary data on the predominant format of code reviews, and also to determine the roles of developers, testers, and project managers in the review process[1].

It was found that developers and testers play the most prominent roles in reviewing code, while project managers do not usually participate. The general format of code reviews is best described by one of two models. The first model is the email-based approach where a package list of code changes is mailed between the author and reviewers. In the second model, the author and reviewers meet simultaneously to perform the review. These models are termed *asynchronous* and *synchronous*, respectively, in the remainder of the paper.

3.2. Survey

The answers obtained to the questionnaire informed the design of a 25-question survey, which we distributed to a random selection of 1000 Microsoft developers and testers. The survey was distributed using company email and the participants answered the questions using a web form. Offering a chance to win a $500US gift certificate motivated participation. All of the survey questions were optional. Nearly 30% of the individuals surveyed responded, with a total of 278 responses.

Participants were given a checklist-type question about the communication tools that they used for performing code reviews. Synchronous communication comprised 49% of all review-related communication, where asynchronous communication accounted for 51% of the responses.

We were particularly interested in determining the amount of review data retention and reuse. We suspected that change rationale information was articulated during a typical review, but that this data was not retained in an accessible manner. When asked whether review information was made accessible post-review, we found that only 18% of synchronous review data was retained so that it was accessible to those other than the reviewer and code author. Asynchronous data had a higher retention rate at 32%, but the majority of the data was still not easily accessible. We also found that when the retained data was used, it was primarily to understand change rationale, as well as to track code changes and verify bug fixes.

Further survey questions asked about aspects of code reviews that were inefficient or difficult. Participants also reported that with current tools it was easy to prepare for and carry out a review, but difficult to access the data that was retained.

In the final survey question, we asked the participants what features would be in their ideal CR tool. Many participants stated they would like to be able to use some type of enriched change list. Others included requests for the ability to annotate and share change lists in a more streamlined fashion, change lists that include context (i.e. the surrounding code), and an enriched user interface.

Other requests, such as the one below, were related to tools that structured review data so that it is easier to store and query.

"*Currently the biggest problem we have with our current code review process is we are building this huge archive of logs of each code review (in the form*

[1] The survey questions are available online at http://www.cs.usask.ca/~als153/code_review

of emails) and there is no easy way to query them. For example by bug #, devs, namespaces, issue types, regressions, etc. Basically all this data is unstructured so it gets very difficult to refer to it over time. A tool that gives structure to this data will be beneficial."

Additionally, participants who commonly performed reviews in a synchronous format noted that there was no easy way to record review data. They would like a tool that allowed them record notes or review sessions for synchronous reviews.

3.3. Email-based Code Review Dialogs

To further aid in understanding asynchronous code reviews, we asked some of the survey respondents to send the email threads of recent reviews conducted over email. We received 31 threads from 12 participants, and examined them to better understand asynchronous review conversations.

The initial email review included a list of changes, usually packaged up as a diff file. The changes were always listed in file order. It was typical for the author to include a description of the changes and of the tasks that resulted in the changes being made. In this preamble the author explicitly stated the rationale behind each task he performed. The middle section of the email threads consisted of questions asked by the reviewers, along with corresponding answers provided by the author. In some cases, a debate concerning a specific task would occur, resulting in many back-and-forth comments between the reviewers and author. The email threads conclude with a resolution agreed upon by the reviewers and author. This resolution was to accept, reject, or postpone the proposed tasks.

It was also a common for the code author to revise the change list as the review progressed. When this occurred, the author would provide a new change list file or hyperlink to the reviewers.

3.4. Discussion

Several conclusions are supported by the data collected during this study. These conclusions suggest latent needs for tool support.

Developers and testers perform code review using synchronous and asynchronous communication in roughly equal amounts. This suggests that a tool that supported only one or the other would not succeed in this particular environment. The results presented here do not rule out that developers mix these models within a single review; if they do then a single, integrated tool would be needed. Stein *et al.* identified requirements for CR that are vital to both synchronous and asynchronous tools [13]. These requirements may be useful when designing a tool to support both types of communication.

Engineers think about code reviews in a task-oriented way. (It is probable that they think about change lists this way, whether or not they are code reviewed.) This suggests that making tasks an explicit part of the code review tools could improve their utility and usability. This has not been reported in the previous literature to our knowledge.

The meat of the code review dialog, no matter what the medium, is the articulation of design rationale. Design rationale is the reasoning behind the decisions made when designing and implementing software. Johnson investigated the link between CR activities and design rationale, and prototyped a tool to create design documents from articulated CR data [6]. Previous work has also identified other circumstances where design rationale is externalized: during specification, during post-hoc documentation and during ad-hoc conversations [11, 14].

At Microsoft there is little systematic retention of code review dialogs, so that valuable articulation of design rationale may be lost. This is in contrast to the code review processes in the open-source environment, which are more likely to be retained in the project's email archive [5, 15]. This suggests that a suite of tools for retaining and accessing the dialogs could have a significant positive impact.

Engineers expressed a desire to systematically retain code review dialogs, suggesting that they would not avoid a tool because it captured this information. This data can be used for understanding change rationale, or for other purposes. For instance, the collection of inspection data can be used for immediate process control as well as process improvement [2].

Engineers find code review dialogs useful for a variety of purposes, but for understanding design rationale more than any other, in those situations where they are retained. This reinforces the value of retention and access. Other stakeholders in the review process are project managers, who care about code quality, and later viewers of the CR, who are trying to excavate design rationale.

4. Conclusion

Code review practices at Microsoft are varied. Despite the variety, a number of overall properties of reviews have been identified. The questionnaire and survey provided basic knowledge of the individuals most involved with reviews, as well the communication mediums and formats of reviews. Issues related to review data retention and retrieval were identified. Supporting data gathered from the survey and email transcripts indicate that this issue may stem from inadequate tool support.

The investigation presented here expands the understanding of the code review process, reinforcing previous work and adding several new insights, including:

- Code reviews are a enticing opportunity for capturing design rationale,
- Reviews are conducted using a mix of synchronous and asynchronous communication
- Developers think about change lists in a task-oriented way

These observations suggest tool support that allows task-oriented structuring of change lists that are to be reviewed, as well as features for retaining and querying these structures lists and the discussions that are centered around them.

8. References

[1] A.F. Ackerman, P.J. Fowler and R.J. Ebenau, "Software Inspections and the Industrial Production of Software", *Proceedings of a Symposium on Software Validation*, Elsevier North Holland, New York, NY, USA, 1984, pp. 13-40.

[2] A.F. Ackerman, L.S. Buchwalk and F.H. Lewski, "Software Inspections: an Effective Verification Process", *IEEE Software*, Volume 6(3), May 1989, pp. 31-36.

[3] L. Brothers, V. Sembugamoorthy, and M. Muller, "ICICLE: Groupware for Code Inspection", *Proceedings of the 1990 ACM Conference on Computer-supported Cooperative Work*, ACM, Los Angeles, California, USA, 1990, pp. 169-181.

[4] M. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", *IBM Systems Journal*, Volume 15(3), 1976, pp. 258-287.

[5] J. Gintell, J. Arnold, J. Houde, J. Kruszelnicki, R. McKenney, and G. Memmi, "Scrutiny: A Collaborative Inspection and Review System", *Proceedings of the 4th European Software Engineering Conference*, Springer-Verlag, London, UK, 1993, pp. 344-360.

[6] J.P. Johnson, "Collaboration, Peer Review, and Open Source Software", *Information Economics and Policy*, Volume 18, Elsevier, 2006, pp. 477-497.

[7] A.J. Ko, R. DeLine, G.Venolia, "Information Needs in Collocated Software Development Teams", *International Conference on Software Engineering*, IEEE Computer Society, Washington, DC, USA, 2007, pp. 344-353.

[8] O. Laitenberger, "A Survey of Software Inspection Technologies", *Handbook on Software Engineering and Knowledge Engineering*, Volume 2, 2002, pp. 517-555.

[9] T.D. LaToza, G. Venolia, and R. DeLine, "Maintaining Mental Models: A Study of Developer Work Habits", *International Conference on Software Engineering*, ACM Press, New York, NY, USA, 2006, pp. 492-501.

[10] V. Mashayekhi, C. Feulner, and J. Riedl, "CAIS: Collaborative Asynchronous Inspection of Software", *SIGSOFT Software Engineering Notes*, Volume 19(5), ACM, New York, NY, USA, 1994, pp. 21-34.

[11] P. Rigby, D. M. German, and M. A. Storey, "Open source software peer review practices: A case study of the Apache server". *In ACM/IEEE International Conference on Software Engineering (ICSE'08),* 2008.

[12] C. Sauer, D.R. Jeffrey, L. Land, and P. Yetton, "The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research", *IEEE Transactions on Software Engineering*, Volume 26, 2000, pp. 1-14.

[13] M. Stein, J. Riedl, S.J. Harner, and V. Mashayekhi, "A Case Study of Distributed Asynchronous Software Inspections", *Proceedings of the 19th International Conference on Software Engineering*, ACM Press, New York, NY, USA, 1997, pp. 107-117.

[14] G. Venolia, R.DeLine, T. LaToza, "Software Development at Microsoft Observed", Tech. Report MSR-TR-2005-140, Microsoft Research, Redmond, WA, 2005.