The Journal of Systems and Software 85 (2012) 2293–2304

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Controversy Corner

On the relationship between comment update practices and Software Bugs

Walid M. Ibrahim, Nicolas Bettenburg, Bram Adams*, Ahmed E. Hassan

Software Analysis and Intelligence Lab (SAIL), School of Computing, Queen's University, Ontario, Canada

ARTICLE INFO

Article history:
Received 4 December 2010
Received in revised form 22 May 2011
Accepted 11 September 2011
Available online 17 September 2011

Keywords:
Code quality
Software bugs
Software evolution
Source code comments
Empirical studies

ABSTRACT

When changing source code, developers sometimes update the associated comments of the code (a consistent update), while at other times they do not (an inconsistent update). Similarly, developers sometimes only update a comment without its associated code (an inconsistent update). The relationship of such comment update practices and software bugs has never been explored empirically. While some (in)consistent updates might be harmless, software engineering folklore warns of the risks of inconsistent updates between code and comments, because these updates are likely to lead to out-of-date comments, which in turn might mislead developers and cause the introduction of bugs in the future. In this paper, we study comment update practices in three large open-source systems written in C (FreeBSD and PostgreSQL) and Java (Eclipse). We find that these practices can better explain and predict future bugs than other indicators like the number of prior bugs or changes. Our findings suggest that inconsistent changes are not necessarily correlated with more bugs. Instead, a change in which a function and its comment are suddenly updated inconsistently, whereas they are usually updated consistently (or vice versa), is risky (high probability of introducing a bug) and should be reviewed carefully by practitioners.

1. Introduction

Source code comments play a central and important role in understanding legacy systems. Comments are often the only form of documentation available for a system, explaining algorithms, informally specifying constraints like pre- and post-conditions, or warning developers of the peculiarities of complex code (Woodfield et al., 1981). Such documentation is crucial to avoid that developers lose grasp of the system as it ages and evolves (Brooks, 1995; Parnas, 1994).

When changing the source code, developers either update the associated comments (a "consistent update") or not (an "inconsistent update"). Updating a comment without the associated code (up to 50% of the comment changes; Fluri et al., 2007) can also be considered as an inconsistent update. The ill-effects of inconsistent updates between code and comments are often noted as anecdotes by researchers and practitioners. An example of such anecdotes is the change comment attached to change #27068 on October 15, 2007 in the PostgreSQL project (we highlight in bold the most relevant part):

> *"Fix pg_wchar_table[] to match revised ordering of the encoding ID enum. Add some comments so hopefully the next poor sod doesn't fall into the same trap. (**Wrong comments are worse than none at all**...)"*

Siy and Votta (2001) made the same observation during one of their case studies, and noted:

> *"According to most developers we talked to, once they encounter an inconsistent comment, they lose confidence in the reliability of the rest of the comments* [...] *and they ignore the remainder of the comments"*

Indeed, inconsistent updates can be critical, as they likely introduce out-of-date comments, which in turn might mislead developers and lead to bugs in the future. For example, a recent manual analysis of the bug reports for the FreeBSD project found sixty bugs that are due to out-of-date comments (Tan et al., 2007). However, Fluri et al. (2007) found that API (Application Programming Interface) changes, although they impact countless other developers, are typically not commented until later revisions.

While prior research has empirically demonstrated the negative impact of code churn and prior bugs on future code quality, for example in the form of bugs, little is known about the impact of comment update practices. Sundbakken (2001) found that the average number of comment lines per C++ class is a good indicator of source code maintainability, yet it is not clear how this relates to comment update practices. While some (in)consistent updates might be harmless (or even expected), others might lead to out-of-date comments, and hence possibly to bugs.

* Corresponding author. Tel.: +1 613 533 6802; fax: +1 613 533 6513.
E-mail addresses: walid@cs.queensu.ca (W.M. Ibrahim), nicbet@cs.queensu.ca (N. Bettenburg), bram@cs.queensu.ca (B. Adams), ahmed@cs.queensu.ca (A.E. Hassan).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.09.019

In this paper, we study comment update practices in three large open-source systems written in C (FreeBSD and PostgreSQL) and Java (Eclipse) to determine the impact of comment update practices on future bugs. We refine two traditional bug prediction models with information about the comment update practices of the three systems (mined from the source code repositories) to study whether comment update practices are able to explain and predict future bugs. We indeed find a strong relation between comment update practices and future bugs. Closer analysis of our bug models shows that a deviation in the common update practices for a particular function (i.e., the function and its comment are always consistently updated, until suddenly an inconsistent update occurs, or vice versa) is a risk that practitioners must review carefully.

The main contributions of this paper are as follows:

- We empirically study the evolution of comment update practices in three large, long-lived, open-source systems (FreeBSD, PostgreSQL and Eclipse).
- We establish an empirical link between comment update practices and future bugs.

Overview of the paper: Section 2 provides the necessary background and related work on comments and bug prediction. Section 3 introduces the comment update practices considered in our study, whereas Section 4 explains how we extract the comment update practices from source code repositories. Section 5 studies the distribution over time of comment update practices and future bugs, followed by an exploration of the relation between the comment update practices and bugs in Section 6. Section 7 discusses our findings. Section 8 elaborates on possible threats to the validity of our findings, and Section 9 concludes this work.

2. Background and related work

This section discusses typical use case scenarios of source code comments, and presents related work on the evolution of source code comments and on code quality analysis.

2.1. The use of comments in source code

The most widely-known use of source code comments is to document developer knowledge and assumptions about the source code. A survey of software maintainers done by de Souza et al. (2005) finds that developers use comments as a key element to understand source code. Similarly, Nurvitadhi et al. (2003) report on the significant impact of code comments on improving program understanding among students. Both studies highlight the important and critical role of comments for software development and maintenance.

Recent studies show that comments are also used for other purposes beyond documenting the knowledge of developers. Ying et al. (2005) observed that commercial developers use comments to communicate with colleagues through messages such as "TODO" or "FIXME" and to address code-related questions from specific team members. Storey et al. (2008) report similar findings through an online survey on a larger, more varied population of developers working on different types of projects. These findings demonstrate the extensive use of source code comments for collaboration and communication throughout the software development process.

A work that is closer to ours is by Tan et al. (2007), who use natural language processing techniques to automatically identify and extract locking-related programming rules from comments. Tan et al. then analyze the source code to locate deviations from these extracted rules. This analysis locates *current* bugs in the Linux kernel code based on inconsistencies between source code and comments, whereas we study the impact of inconsistent comment updates on *future* bugs in the whole software system, not just the commented code snippet.

Sundbakken (2001) performed an empirical study to identify the major indicators of maintainability of open-source C++ software systems. Although the total number of comment lines is not a good indicator, the average number of comment lines per C++ class is a major indicator of maintainability, because it includes information about the spread of comments across all classes. This work differs from our work in multiple ways. Instead of measuring comments in software *releases* to analyze their role in the ability of developers to *maintain* source code, we consider the *changes* in between releases to identify whether (in)consistent updates to comments are related with *future bugs*.

2.2. Updating code comments

The work most closely related to this paper is that on co-evolution of comments and source code by Fluri et al. (2007). The authors perform a fine-grained, syntax tree-level analysis of how comments evolve over time. Based on empirical analysis of eight open-source and commercial systems, the authors find that comments and source code grow similarly (normalized for size) and that 51–69% of all comment changes were driven by source code changes, usually in the same transaction to the version control system. Our study builds on the latter finding, yet analyzes comments at the function-level, not statement-level. Instead of focusing on the quality of *comments*, we try to find an empirical link between comment update practices and the quality of a *software system*, in terms of future bugs.

Marin (2005) shows that developers are more likely to update the comments of well-commented code. Developers are also more likely to update comments for large and complex changes. Similar results were found by Malik et al. (2008), who built a model to predict the likelihood of updating a comment. Their model is influenced in particular by the complexity of the performed change, the time of the change, and developer characteristics. The fact that the complexity of a function is a good indicator for the appearance of bugs (Herraiz et al., 2007) suggests that updating a comment (or missing to do so) is likely to lead to future bugs.

Arafat and Riehle (2009) found that commenting source code is an important practice in open-source development. They studied the density of comments in source code changes, i.e., the proportion of new comment lines in a source code change, in more than 5000 open-source systems. Small changes turn out to have the highest comment density (>60% for 1-line changes). Larger changes have ever smaller density, asymptotically approaching 19% (1 comment line per 5 code lines). Comment density is independent of project and team size, but depends on the programming language.

Finally, Siy and Votta (2001) found that updating comments is often a major priority for companies. More than 28% of all issues identified during code inspection are related to documentation problems. In particular, missing or outdated comments, together with incorrect comment layout and typos in the comments, represent the main source of documentation issues. Furthermore, almost half of the source code lines that are changed or added in response to the inspection results are comments.

2.3. Bug prediction

Bug prediction models play an important role in the prioritization of testing and code inspection efforts. For example, if a prediction model determines that a particular component will see a significant increase in error-proneness, managers can react by allocating more testing and inspection efforts, instead of having to wait for bug reports from clients after release (Arisholm and Briand,