An exploration of how comments are used for marking related code fragments

Annie T.T. Ying, James L. Wright, Steven Abrams
IBM Watson Research Center
19 Skyline Drive, Hawthorne, NY, 10532, USA
{aying,jimwr,sabrams}@us.ibm.com

ABSTRACT

A software developer performing a change task to a system very often has to examine a concern that is scattered across the source code of the system. Although many mechanisms attempt to alleviate the problem of dealing with scattered code, many software developers are still using more ad-hoc approaches to mark related code. In this paper, we explore how developers use comments to mark related code. We found that developers use two basic kinds of conventions to mark related code in comments: by explicitly stating relationships in the comment and by using similar comments in related code elements. These conventions have several major issues. However, we observe that using comments to mark related code fragments offers several benefits. We hope that our observations can give insights into building better tool support for scattered code fragments.

1. INTRODUCTION

A software developer performing a change task to a system very often has to examine a concern that is scattered across the source code of the system. Such scattered code that relates to a single concern may not necessarily result from poor design. Even if the system is well-designed, one cannot always anticipate all concerns over the lifetime of the system amongst ever-changing circumstances. Furthermore, some concerns are inherently difficult to modularize in the source code.

Many researchers have proposed a wide range of mechanisms to address this problem. For example, AspectJ provides language support for encapsulating concerns that crosscut a system's structure [4]. HyperJ provides support for decomposing concerns—usually overlapping and interacting with each other—along multiple dimensions simultaneously [6]. Concern Graph provides a light-weight representation to capture and persist concerns and relationships among program elements within a concern [5]. Aspect Browser supports visualization of concern related code fragments based on lexical searches on the program text [3].

Workshop on Modeling and Analysis of Concerns in Software (MACS 2005)
16 May 2005, St. Louis, MO, USA

Although such mechanisms offer many benefits in dealing with scattered code, many software developers are still using more ad-hoc approaches to mark related code fragments. In this paper, we explore how developers use comment conventions to mark related source code fragments. As a preliminary study, we investigated an IBM internal code base, the Architect's Workbench (AWB). We found that developers use two basic kinds of conventions to mark related code fragments with comments: by explicitly stating relationships in the comment and by using similar comments in related code elements. These conventions have several issues, such as similar comments denoting different concerns, inability to denote the scope of the concern, and inconsistent labeling of concerns. However, we observe that using comments to mark related code fragments offers several benefits: light-weight, flexible, and convenient to apply. We hope that our observations can give insights into building better tool support for scattered code fragments.

The rest of the paper is organized as follows: In Section 2, we presents our study of how comments are used to mark related code fragments in the AWB code base. In Sectiond 3, we describe issues with using comments for the purpose of marking related code fragments. In Section 4, we discuss some issues with our study. In Section 5, we conclude.

2. STUDYING CONCERN-RELATED COMMENTS

In this section, we present our study on how developers use comments to mark related code. Section 2.1 describes the settings of this study and Section 2.2 describes the categorization of the concern-related comments.

2.1 Study settings

To focus our attention on comments that are more likely to show how developers use comment conventions for marking related code, we chose to limit our investigation to Eclipse task comments [1]. Eclipse—a popular open-source integrated development environment—has provided support for comments that describe tasks to be performed on the source code, using a "task tag" mechanism. Using the Java perspective in Eclipse, Java programmers can embed predefined task tag strings, such as "TODO", in the comments on the source code, and then use the task view to browse a summary of the places in the code with a comment that contains a task tag. From the task view, a user can click on an entry and navigate to the corresponding source code. The intuition behind choosing to only focus on Eclipse task comments is that such comments tend to be more ad-hoc

and informal, encouraging developers to use them for whatever needs they may have. Other types of comments, such as JavaDoc, tend to be more structured, with well-understood conventions and goals. Thus, they do not tend to accommodate purposes other than the ones that are intended.

In this study, we chose to investigate task comments in the AWB code base. The AWB project consists of two major parts: a platform that provides customizable representations and tool support for models, and a particular instantiation of this platform in the system architecture domain, which embodies a tool that helps IT architects transform informal notes into various formal system architecture models. The source code of AWB is written primarily in Java and is implemented as an Eclipse plug-in.

We studied the Eclipse task comments that were found in one version of AWB that was checked out from the AWB CVS repositories on February 9, 2005. The codebase consists of 2,213 files. This version of the code contains 221 task comments. Five developers contributed to these task comments.

2.2 Study results

To explore how task comments are used to mark related code, for each of the task comment we found in the version of AWB code we studied, we first determined whether the comment was used for marking a dependency with another program element. Of all such concern-related comments, we found that developers use two basic kinds of conventions to mark related code with comments: by explicitly stating relationships in the comment and by using similar comments in related code elements.

Table 1 gives some example comments. The first column labeled "Conventions" summarizes the comment conventions developers use to mark related code in the AWB code base. The second column labeled "Comment examples" shows examples of comments that use particular conventions to mark related code. For the rest of this section, we describe the examples presented in these examples.

2.2.1 Explicit relationship

The conventions prefixed "explicit relationship" in the first column of Table 1 describe cases where a comment explicitly states a relationship from the code close to the comment to other part(s) of the code.

- In the example labeled "explicit relationship: specified targets," the comment reveals a subtle dependency among duplicate code fragments scattered across four methods in two files.

- In the example labeled "explicit relationship: implied targets", the comment reveals a concern, tracing for finding a particular bug related to a NullPointerException. Although the code that relates to this concern is scattered throughout the method, the developer uses a single comment to mark this concern.

2.2.2 Similar comments marking related code

The conventions prefixed "similar comments marking related code" in the first column of Table 1 describe cases where two or more similar comments are used to mark related program elements. Griswold called this convention information transparency [2].

- The example labeled "similar comments marking related code: bug number" presents two comments, both containing the string "ECR 311," where ECR stands for Enhancement Change Report. These are used to denote code that are related to the same bug #311. Together with seven other comments that contains the string "ECR 311," the comments denote code fragments that are scattered across eight different files.

- The example labeled "similar comments marking related code: concern tag" presents two comments containing the string "richtext," denoting the code related to the "rich text" feature in AWB. Together with seven other comments that refer to the rich text feature, these comments indicate the scattered code fragments that are part of the rich text feature.

- The example labeled "similar comments marking related code: same comment" presents a comment that was repeated in four places in the code, each in a different file. The comments were used to mark a fix to a threading problem. The code that contains the fix is anticipated to be revisited because fixes to threading problems are tricky due to the difficulty in proving correctness and testing multi-threaded applications.

From this study, we observed several benefits of using comments as a convention to mark scattered code:

Light-weight: Using comments to mark related code requires minimal tool support – it only requires a grep-like tool for searching the related comments and code.

Flexible: Because a comment is free-from, it is flexible – it can be used to refer external entities, such as using a bug number to refer to external bug database. A comment can also be used to mark multiple concerns. For example, the comment "`// [..] notestonodes: ECR 311`" expresses a relationship of the code both to a higher level concept "notes-to-nodes" and a relationship to a bug report.

3. ISSUES WITH CONCERN-RELATED COMMENTS

In this section, we discuss a few issues with using comments to mark code fragments.

3.1 Similar comments denoting different concerns

Similar comments are not only used to denote the same concern, but may also denote different concerns. In the AWB code, there are many instances of the comment "`// TODO Auto-generated method stub`," which are generated by the Eclipse code generator. When using Eclipse to generate a Java class from a super-class or an interface, Eclipse automatically inserts this comment for the generated methods and constructor stubs. These kinds of comments may denote code fragments from completely different concern.

3.2 Scope of a comment

The scope of the comment is often not apparent because the comment only marks a single point in the code. Developers use different assumptions about what regions of code the comment applies to. For example, comments may not contain any explicit region information, but a developer may use a comment to refer to the statement immediately following the comment, may use a comment to refer to all the

Conventions	Comment examples
explicit relationship: specified target	`// [..] code duplication with IRelationTableModel: peer(), relation(), relationTypeName()`
explicit relationship: implied target	`// [..] remove tracery when NPE [NullPointerException] is solved`

similar comments marking related code: bug number	nine comments denoting ECR 311, two of which are:
	`// [..] ECR 311: get copy-text button to work`
	`// [..] ECR 311: handle the case of multiple Node-*types*`

similar comments marking related code: concern tag	nine comments denoting the "rich text" concern, two of which are
	`// [..] richtext: eliminate this once ECR 317 complete`
	`// [..] richtext: handle the URL-case`

similar comments marking related code: same comment	the same comment in four places in the code: `// [..] EXPERIMENTAL`

Table 1: Results of categorization how developers use comments to mark related code

statements until the next blank line occurs, or may use a comment to denote the code in the entire enclosing scope.

In addition, the task comment may apply to multiple non-contiguous places in code. For example, the task comment "`// [..] remove tracery when NPE [NullPointerException] is solved.`" refers to tracing statements in many places, not just the statement immediately below the comment. Finding all the places the developer had in mind can be challenging.

3.3 Inconsistent concern labeling

Using comment conventions to denote related code is a very light-weight and convenient approach, but sophisticated support is lacking. For example, there were nine comments related to the "rich text" concern in AWB and a developer describes this concept using a concern tag. However, the developer used three different formats to write "rich text": "rich-text", "richtext", "RichText". This inconsistent labeling can hinder subsequent efforts by a developer to locate these comments through search tools.

3.4 Consistency on comments and code

Comments can easily go stale because maintaining the consistency on comments and code is a manual process. For example, the comment "`// [..] code duplication with IRelationTableModel: peer(), relation(), relationTypeName()`" explicitly states the names of the method in the comment. When one of these methods is renamed, the comment needs to get updated with the new method name manually in order to maintain the accurarcy of the comments. Another example is when similar comments are used to mark a related concern and one of the comments changes, all such comments must be updated manually.

4. DISCUSSION

In this section, we discuss some issues with our study.

4.1 Significance of Eclipse task comments

To "talk" to other team members through source code, a developer may use a Java comment, not necessarily a task comment. However, we did not investigate all the Java comments: The codebase contains 15,748 JavaDoc comments[1] and 13,457 non-JavaDoc comments[2] , and it was impossible to analyze all of them manually. Although task comments only accounts for a small fraction of all the comments in the AWB codebase, we still chose to examine task comments. Task comments are likely to be good candidates to contain information that is relevant to the current development context, as task comments are intended to be more transient—created and deleted more often—than other comments.

[1]We define the number JavaDoc comments as the number Java tokens "/**" in the source code.

4.2 Generalizability of the results

In this preliminary study, we examined the task comments of one project. We cannot draw general conclusions about our task comment categorization from only one project. In addition, the results from this study may not be generalizable to other projects. The AWB is a small team of less than ten developers. Programming practices that are peculiar to a particular developer can dramatically affect the results.

5. CONCLUSION

In this paper, we have described uses of comments to mark related code fragments in the AWB code base. We have found that developers use an explicit reference to relate to other scattered code fragments and use similar comments to mark related code fragments. The results of this study is not surprising, but it points out some benefits and issues of using comments as a convention for marking related code. These may provide insights for building future mechanisms to better support scattered code.

6. ACKNOWLEDGMENT

We are grateful to the AWB team for lending their codebase for this study, as well as the prompt and useful help in understanding the intention of the comments. We would also like to thank Mark Chu-Carroll and Martin Robillard for many inspirational discussions. Moreover, we would like to thank anonymous reviewers for the useful feedback.

7. REFERENCES

[1] Eclipse task tags website. http://127.0.0.1:55317/help/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-preferences-task-tags.htm.

[2]We define the number of non-JavaDoc comments as the number of Java tokens "//", plus the number of Java tokens "/*" in the source code.

[2] W. G. Griswold. Coping with crosscutting software changes using information transparency. In *Reflection 2001: International Conference on Metalevel Architectures and Separation of Crosscutting*, pages 250–265, 2001.

[3] W. G. Griswold, J. J. Yuan, and Y. Kato. Exploiting the map metaphor in a tool for software evolution. In *International Conference on Software Engineering*, pages 265–274. IEEE Computer Society, 2001.

[4] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *European Conference on Object-Oriented Programming*, pages 220–242, 1997.

[5] M. P. Robillard and G. C. Murphy. Concern graphs: finding and describing concerns using structural program dependencies. In *International Conference on Software Engineering*, pages 406–416. ACM Press, 2002.

[6] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton. N degrees of separation: Multi-dimensional separation of concerns. In *International Conference on Software Engineering*, pages 107–119, 1999.