Source code that talks: an exploration of Eclipse task comments and their implication to repository mining

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 programmer performing a change task to a system can benefit from accurate comments on the source code. As part of good programming practice described by Kernighan and Pike in the book *The Practice of Programming*, comments should "aid the understanding of a program by briefly pointing out salient details or by providing a larger-scale view of the proceedings." In this paper, we explore the widely varying uses of comments in source code. We find that programmers not only use comments for describing the actual source code, but also use comments for many other purposes, such as "talking" to colleagues through the source code using a comment "Joan, please fix this method." This kind of comments can complicate the mining of project information because such team communication is often perceived to reside in separate archives, such as emails or newsgroup postings, rather than in the source code. Nevertheless, these and other types of comments can be very useful inputs for mining project information.

1. INTRODUCTION

Accurate comments on source code can be useful to a programmer performing a change task. As Knuth suggested in the literate programming technique, programs should not only be intended to be executed by computers, but also intended to be read by human [4]. As part of good programming practice, Kernighan and Pike suggested that programmers should write comments that "aid the understanding of a program by briefly pointing out salient details or by providing a larger-scale view of the proceedings" [3].

Many programmers use comments for purposes other than describing source code, but yet these comments contain very useful information to a programmer performing a change task. One example of such a kind of comments is the Eclipse task comments [1]. Since March 2003, Eclipse—a popular open-source integrated development environment—has provided support for comments that describe tasks to be performed on the source code through the task tag mechanism. Using the Java perspective in Eclipse, Java programmers can embed pre-defined task tag strings, such as "TODO", in the comments on the source code and 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.

In this paper, we perform an informal empirical study on the use Eclipse task comments in Java source code. As a preliminary study, we look at an IBM internal codebase, the Architect's Workbench (AWB). We found that although many of these comments do not describe the actual source code, they describe other interesting development aspects, such as communication and changes that were performed or to be performed to the source code. For example, some developers "talk" to colleagues through the source code using a comment such as "Joan, please fix this method." Such kinds of comments can complicate the mining of project information because such team communication and task-oriented information is often perceived to reside in separate archives, such as emails or change request management systems, rather than in the source code. In addition, these comments typically contains ad-hoc meta-data, depends on the context of the code, have a implied scope, and are informal.

The rest of the paper is organized as follows: first, in Section 2, we present a categorization of Eclipse task comments from our study on the AWB codebase. In Section 3, we describe the challenges of analyzing task comments in the context of mining project information. In Section 4, we discuss some issues with our study. Finally, in Section 5, we conclude and outline future work.

2. TASK COMMENTS CATEGORIZATION

To explore what information Eclipse task comments contains and what they are intended for, we studied the Eclipse task comments that were in the AWB code checked out from the AWB CVS repositories on February 9, 2005. The codebase consists of 2,213 files. The code contains 221 task comments[1].

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

[1]We define the number of task comments as the number of lines of Java comments that contain the string "TODO".

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.

Five developers contributed to the task comments in the version of the AWB code we studied. To preserve the privacy of the developers, whenever we paraphrase a comment from the AWB codebase, we have substituted the name of a developer in a comment with a made-up name–Beth, Joan, Pam, Rea, or Sue.

In the AWB codebase, we found different uses of Eclipse task comments. We categorized these different uses, as shown in Table 1. The first column shows the categories, each of whose cell belongs to one of the seven main groups: communication, past tasks, current tasks, future tasks, pointers to a change request, location markers, and concern tags. The second column presents an example of Eclipse task comment found in the AWB code. Some comments belong to multiple groups, for example, a comment that is both for communication and for describing a task. For the rest of this section, we describe the seven categories of comments and present an examples of comment from each categories.

Each of the sub-sections in the rest of this section describes a main category and the examples listed in Table 1.

2.1 Communication

We found some cases where developers use the source code as a medium to communicate to each other.

- In the example labelled "communication: point-to-point" in Table 1, Sue wrote an Eclipse task comment dedicated to Joan. Prior to this message, Sue and Joan had actually discussed the error that was fixed by the hack referred in the comment. In their discussion, Sue suggested the hack. Although Joan was not satisfied with the hack, Joan could not come up with a better fix. Because of the urgency to get the bug fixed, Sue just temporarily implemented the hack. To remind Joan to better fix the error, Sue wrote this comment.

- In the example labelled "communication: multi-cast/broadcast" in Table 1, Joan may have intended to only direct this question to Sue, the implementer of the method referred in the comment. However, this question may worth directing to other team members who may be thinking to call this method and thus may advocate against making the method non-public.

- In the example labelled "communication: self-communication" in Table 1, the example serves as a reminder to Pam herself to clean up the tracing statements in the code.

2.2 Pointers to change requests

Some Eclipse task comments denote a task that is part of a bigger change logged in the change tracking system. AWB uses their own change tracking system called the ECR (Enhancement Change Request) system.

- In the example labelled "pointer to a change request" in Table 1, Pam wrote the two comments to redirect further details to the change report ECR with ID 327. Because an Eclipse task comment is in a particular location in the code, it often denotes a finer-grained task that a task logged into the change tracking system.

2.3 Bookmarks on past tasks

We found in the AWB that some comments describe changes that had been completed. These comments often denote places where changes to the code may require further work.

- In the example labelled "bookmark: hack" in Table 1, which is the same example as an example we described in Section 2.1, Sue indicated that she had performed a code modification which was a hack.

- Another example shows that Eclipse task comments are used to mark places in the code where the developer is uncertain about whether the change actually fixed the defeat. In the task comment labelled as "bookmark: experimental fix" in Table 1, Joan wrote this same comment in several places in the code. Although she has completed a fix to a threading problem, she is not totally confident that fix actually solves the problem until the system has been used for a while. Therefore, she marked use this comment to mark the places that contributed to the fix.

2.4 Current tasks

Most of the comments in the AWB code denotes outstanding tasks that need to be done currently.

- In the example labelled "current task: refactoring" in Table 1, Pam uses a comment to suggest refactoring, briefly outlining the current strategy and the suggested strategy.

- Another example of a current task is a task comment generated by the Eclipse code generator. When using Eclipse to generate a Java class from a super-class or an interface, Eclipse automatically inserts a "TODO" comment for the generated methods and constructor stubs, as demonstrated in the example labelled "current task: from automatically generated code" in Table 1. Eclipse also generates a "TODO" comment for an empty Java `catch` black when Eclipse "Encode try-catch block" functionality is used to generate a `catch` block.

2.5 Future tasks

Some tasks cannot be done currently because those tasks depend on something to be available in the future:

- In the example labelled "future task: once the library is available..." in Table 1, Pam cannot proceed with the task of using the "Eclipse-icon-Decorator" mechanism in the code depends on the availability of that mechanism.

Categories	**Example**
communication: point-to-point	`// TODO an ugly hack for now -sue. Joan, please fix it`
communication: multi-cast/broadcast	`// TODO joan: explain why this [method] is public, since it is used only internally`
communication: self-communication	`// TODO [..] remove tracery if cell-editing is ever stable`
pointer to a change request	RichAttributeComparison.java: `// TODO pam: ECR 311: get copy-text button to work` AttributeViewerImpl.java: `// TODO pam: ECR 311: handle the case of multiple Node-*types*`
bookmark: hack	`// TODO an ugly hack for now -sue. Joan, please fix it`
bookmark: experimental fix	`// TODO joan EXPERIMENTAL`
current task: refactoring	`// TODO [..] make this work using subtyping, not parsing the String type-name!`
current task: from automatically generated code	`// TODO Auto-generated method stub`
future task: once the library is available...	`// TODO pam: once we have the Eclipse-icon-Decorator mechanism, use it here`
future task: once some code modification is complete ...	`// TODO [..] eliminate this once ECR 317 complete`
location marker: point location	`// TODO`

location marker:
range location
	`// TODO Workaround for [..]`
	`[..]`
	`// [..]` End Workaround

concern tag	in 12 places in the code: `TODO pam: null-guard case of [..] [input] corruption`

Table 1: Eclipse task comment categorization

- Similarly, in the example labelled "future task: once some code modification is complete" in Table 1, the developer cannot perform the task until ECR 317 is complete.

2.6 Location markers

All tasks comments are location markers – the Eclipse task view enables a developer to easy view and navigate to the places in the code with task comments:

- For example, the empty comment labelled "location marker: point location" in Table 1 serves as a location marker. Considering the context, such a comment can serve as a reminder that something needs to be done to the code around the comment.
- Another example, the example labelled "location marker: range location" in Table 1, precisely denotes a range of source code that the task comment applies to.

2.7 Concern tags

To mark the places in the code that are related to a single concern [5], developers place the same identifying tag—which we call concern tag—in the task comments. This is concern tagging approach is an example of Griswold's information transparency techniques, which aim to capture related parts of the code—especially the ones that are not well-modularized—by non-programming language constructs, such as naming convention, formatting style, or tags embedded in comments [2].

- In the example labelled "concern tag" in Table 1, the developer used the same comment to denote 13 places in the code that relates to an input corruption.

3. ANALYZING COMMENTS

Having investigated the task comments in the AWB codebase, we see some challenges in using Eclipse task comments as inputs in repository mining, which are discussed in the rest of this section.

3.1 Inferring meta-data from a task comment

An Eclipse task tag only provides two pieces of meta-data than a Java comment, tag creation time and tag severity: Eclipse logs the time when the task tag is first saved, and also supports users in defining a severity value for each task tag type (not for each instance of task tag).

From our study, we see that developers employ common convention to encode additional meta-data that is not explicitly supported by a Eclipse task tag. However, some of types of meta data can still be hard to infer from comments.

Author

Many comments contain the name of the author of the comment. This author information can be helpful for searching all the comments written by the author. However, parsing the author information from the comment may require some care because the format of the format of the author information can vary. For example, Pam tends to put her name preceding a colon, as in "`// TODO pam: [..]`." Sue sometimes types her name all in letters followed by a dash, as in "`// TODO [..] -sue`." If the source code is kept in a code repository, an alternative way to infer the author information is to associate the author information in the change log with the comment.

Change request identifiers

An Eclipse task comment sometimes represents a task that a developer needs to perform as part of the change described in the change tracking system, as shown in Section 2.2. In such a case, the developer usually put the change request number in the comment. For example, in AWB, a change request is denoted as an ECR (Enhancement Change Request) and a particular ECR is referred to by its ID, such as in "`// TODO pam: see ECR 327`." The convention for specifying an ECR is pretty standard, with the ECR number followed by the string "ECR".

3.2 Implied context in a task comment

Because the tags are embedded in the code, task comments tend to depend a lot on the context of the surrounding code. For example, some task comments tend to use context-sensitive words which need to be interpreted with the surrounding code. For example, in the task comment we have shown in Section 2.1, "`// TODO an ugly hack for now -sue. Joan, please fix it`," the word "it" requires the previous discussion between Sue and Joan and the code context to make sense.

Some task comments may not even have words at all, but the meaning of the task may be apparent to a human. We demonstrate by an example not from the AWB code, an empty task comment "`// TODO`." Such a comment does not mean much on its own. However, if we notice that the task comment is enclosed by a method with no statements, it is apparent to us that the task is to implement the method. Such a case can pose challenges to mining algorithms.

3.3 Inferring the scope of a task comment

The scope of the comment is often not apparent because the comment only denotes a single point in the code. Developers use different assumptions on what region of code the comment applies. For example, comments may not contain any region information, but a developer sometimes uses a comment that refers to the statement immediately following the comment, sometimes uses a comment to refer to all the statements until a blank line is encountered, and sometimes uses a comment to denote the code in the whole enclosing scope, such as the empty comment denoting an unimplemented method we describe in this section. Although we have shown in Section 2.6 of one example where the developer have precisely denote a region that the comment applies to, that is the only such example from the whole study.

In addition, the task comment may apply to multiple non-contiguous places in code. For example, the task comment "`// TODO pam: 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 developers implied can be challenging for a mining tool.

Furthermore, even the developer may only have a fuzzy idea of all the places the comment denotes. For example, the task comment "`// TODO -- Beth changed these at some point, to something Eclipse 3.0 compliant`" denote a fuzzy area of code that was to be changed, as porting the code to work with Eclipse 3.0 is not a trivial task and requires changes to many places in the code. Thus, a developer cannot easily specifies all the places in the code that need to be changed in complete when planning the change. Therefore, it is very hard for a mining tool to infer such information.

3.4 Informality in a task comment

In the study, we see that the task comments are typically more informal and shorter than description from the bug report or JavaDoc comments. For example, many of the comments only contain one single word or incomplete sentences. This is not surprising because many task comments are meant for personal reminders and for temporarily use. Also, because the task comments are embedded in the code, the fear of clutterness in code may have prevented developers on elaborating a comment to make it formal. However, this informality in the task comments can make the mining tools that use natural language processing techniques challenging to apply.

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[2] and 13,457 non-JavaDoc comments[3] , 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.

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 AND FUTURE WORK

In this paper, we have described our preliminary study on Eclipse task comments on the AWB codebase. We have found that these task comments contain rich and a wide range information, as shown in the categorization of task comments we have presented. Many task comments from study illustrate some challenges in treating task comments as input for analysis.

Although the conclusion drawn from our study is not generalizable to all projects, our study has shown some examples of task comments being a promising input to analyze. As future work, one direction of research is to infer the meta-data and contextual information of task comments, as such information is not captured by the current Eclipse task mechanism. Another direction of research is to come up with novels ways to analyze the inferred meta-data and contextual information, together with the content of the task comments.

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

[3]We defined the number of non-JavaDoc comments as the number of Java tokens "//", plus the number of Java tokens "/*" in the source 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] 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] B. W. Kernighan and R. Pike. *The practice of programming.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[4] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

[5] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communication of ACM*, 15(12):1053–1058, 1972.