Natural Language Processing and Program Analysis for Supporting Todo Comments as Software Evolves

Pengyu Nie, Junyi Jessy Li, Sarfraz Khurshid, Raymond Mooney, and Milos Gligoric
The University of Texas at Austin
{pynie@, jessy@utexas., khurshid@ece., mooney@cs., gligoric@}utexas.edu

Abstract
Natural language elements (e.g., API comments, todo comments) form a substantial part of software repositories. While developers routinely use many natural language elements (e.g., todo comments) for communication, the semantic content of these elements is often neglected by software engineering techniques and tools. Additionally, as software evolves and development teams re-organize, these natural language elements are frequently forgotten, or just become outdated, imprecise and irrelevant.

We envision several techniques, which combine natural language processing and program analysis, to help developers maintain their todo comments. Specifically, we propose techniques to synthesize code from comments, make comments executable, answer questions in comments, improve comment quality, and detect dangling comments.

Introduction
Natural language elements form a substantial part of software repositories. These elements are used to communicate between users and developers (e.g., API comments, bug reports, and feature requests), and among developers (e.g., todo comments). Todo comments contain invaluable data that describe changes to code that can increase software maintenance, reliability, and quality. Despite occurring frequently in practice and containing valuable information, these elements, because of their informal nature, are largely not exploited by existing software engineering tools.

Research on combining program analysis and natural language processing (NLP), which recently started to gain some traction, is in its infancy (Ernst 2017; Arnaoudova et al. 2015; Hindle et al. 2012; Oda et al. 2015; Allamanis, Peng, and Sutton 2016; Vasilescu, Casalnuovo, and Devanbu 2017; Raychev, Vechev, and Krause 2015; Nguyen et al. 2012), and the existing work, although novel, mostly neglected comments that are used to communicate among the developers (Storey et al. 2008; Sridhara 2016).

In this position paper, we argue about the importance of content in todo comments and envision several techniques to automatically maintain and resolve those comments.

This position paper is to a large extent inspired by our extensive analysis of a large corpus of open-source projects. Specifically, we analyzed over 30k open-source projects, which are available on GitHub, totaling 585 million lines of code (not counting comments). We found that these projects include over 297 million lines of comments (~30% of the total lines). Our analysis also uncovered more than 700k todo comments in the used corpus. We manually inspected (and discussed) hundreds of comments, code and comment changes, and commit messages. In the following subsections, we will frequently refer to this dataset and our findings related to this dataset. All examples of code and comments that we provide in this paper are taken from one of the analyzed open-source projects.

This paper mostly focuses on todo comments that contain valuable information on increasing software quality, performance, maintenance, and reliability. We consider the following three categories of todo comments. First, task comments explain what features are currently not supported or what optimizations need to be implemented (e.g., from the Google Guava project: “For optimal performance, use a binary search when `targets.size() < size()/log(size())`’). Second, trigger-action comments talk about changes to the code repository that would be necessary if something else is modified by developers (e.g., from Guava: “check more preconditions (as `bufferSize >= chunkSize` if this is ever public”). Finally, question comments are concerned with alternative implementations, potential optimizations, and testing, which may be explored by developers only if time permits (e.g., from Guava: “Is this faster than `System.arraycopy()` for small arrays?”).

Regardless of the category of todo comments, as software evolves and development teams re-organize, these comments may be dangling, i.e., resolved but forgotten (Storey et al. 2008; Sridhara 2016). For example, a trigger may hold (e.g., “if this is ever public”) but the action may not be executed by developers (for very long time or ever), and developers may never have enough time to consider alternative algorithms and fine tune their existing implementations.

With the goal to help developers increase the reliability of their software, we propose several techniques to (1) synthesize code described in task comments, (2) make trigger-action comments executable, (3) answer question comments, (4) improve the quality of all todo comments, and (5) automatically detect dangling comments.

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.
protected AbstractStreamingHasher(int chunkSize, int bufferSize) {
    // TODO: check more preconditions (bufferSize, chunkSize)
    if (T.RIGI.T odio:check more preconditions (bufferSize, chunkSize)
    "if this is ever public"
    checkArgument(bufferSize >= chunkSize);  
    checkArgument(bufferSize % chunkSize == 0); ... }

(a) Example from Google Guava (AbstractStreamingHasher)
    public void testDynamicAttributesSupport() throws Exception
    dispatcher.serviceAction(request, response, mapping);
    // TODO: remove expected JDK 1.5 and 1.6 after switching to Java 1.6
    if (T.RIGI.TRod, T.RIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.TRIGI.T
**Table 1: Example todo comments in open-source projects**

<table>
<thead>
<tr>
<th>Project (on GitHub)</th>
<th>File (.java)</th>
<th>Todo Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache/Incubator-wave</td>
<td>Pretty</td>
<td>Remove this if itmlViewImpl implements getAttributes</td>
</tr>
<tr>
<td>Apache/Struts</td>
<td>FreemarkerResultMockedTest</td>
<td>Remove expected/Dir16 and if() after switching to Java 1.6</td>
</tr>
<tr>
<td>Apache/Poi</td>
<td>TextXSBugs</td>
<td>Delete this test case when sBValue and vARK are implemented</td>
</tr>
<tr>
<td>Google/Guiava</td>
<td>Types</td>
<td>Once we are on Java 8, delete this abstraction</td>
</tr>
<tr>
<td>Google/Guiava</td>
<td>AbstractStreamingHasher</td>
<td>Check preconditions (as bufferSize &gt;= chunkSize) if this is ever public</td>
</tr>
<tr>
<td>Google/Guiava</td>
<td>MapTest</td>
<td>Replace with Ascii_caseInsensitiveEquivalence() if it exists</td>
</tr>
<tr>
<td>KangProject/Frameworks_base</td>
<td>SslCertificate</td>
<td>If deprecated constructors are removed, this should always be available</td>
</tr>
<tr>
<td>Morristech/Gwt</td>
<td>DefaultFilters</td>
<td>This class needs to be revisited, when Gwt’s Ant is upgraded</td>
</tr>
<tr>
<td>Morristech/Gwt</td>
<td>Simplifier</td>
<td>If the AST were normalized, we wouldn’t need this</td>
</tr>
</tbody>
</table>

**Improving Todo Comments**

We will develop techniques to help software engineers to write meaningful todo comments. While manually analyzing hundreds of todo comments, we found a number of comments that were hard to understand even if we read the code near those comments. We were also in disagreement about their meaning in several cases, and although we could understand a comment (e.g., from the Square Retrofit project: “TODO non-suck message”), it was clear that any technique would have a hard time to extract any useful data.

Our initial task will be to detect todo comments that are not specific enough, as well as those comments that do not follow the conventions already used in the same project. The techniques that we will develop will build on our work on text specificity (Li and Nenkova 2015) and program analysis. When we detect an unspecific comment, we will either notify a developer to provide additional clarification, highlight a part of the comment that does not follow the style (in a similar way that spellcheckers highlight typos in comments inside IDEs), or automatically reformat the comment to be consistent with other comments in the same repository. We will also provide automated comment style checkers, where the rules can be expressed by developers; this is similar to code style checkers, which are used in practice. Having specific comments that follow the same style will enable techniques from prior sections.

**Detecting Dangling Todo Comments**

Prior work has shown that developers may resolve todo comments but forget to remove these comments from source code (Storey et al. 2008; Sridhara 2016); these dangling comments can waste developers’ time during program comprehension and maintenance.

We are working on a technique, based on machine learning, to automatically detect dangling todo comments. Our detection technique learns from existing software repositories. As mentioned earlier, we have already collected more than 700k todo comments. This large dataset provides examples for todo comments that were removed by developers (over 20k). We are using these examples as distant supervision signals, where we are exploring automatic labeling of examples (e.g., todo comments that are in the same file with removed todo comments). Our models are exploiting commit messages and static code analysis of changes. In the future, we plan to also utilize software histories to extract necessary context when todo comments were introduced. We will also reason about co-evolution of code and comments from when a todo comment was introduced until...
it was resolved by a developer. Specifically, for each code change, we will compute its distance from todo comments, word similarity with each comment, and code structure that may be described in a comment. These sources of information provide complementary views to feature development and complementary models, so we plan to build on our prior work in co-training and ensemble models.

Related Work

Li et al. (2006) used text classification to validate the representativeness of their study of bug characteristics. Fluri, Wursch, and Gall (2007) empirically showed that code and comments frequently co-evolve. Padioleau, Tan, and Zhou (2009) manually studied over one thousand comments, and found that 50% of comments can be leveraged by various techniques. Haouari, Sahraoui, and Langlais (2011) introduced a taxonomy of comments and found that todo comments are the second most common type of comments. Movshovitz-Attias and Cohen (2013) used topic modeling and language models to generate comments from Java source files. Several work tackled automated generation of commit messages and mining relation from commit messages (Linares-Vásquez et al. 2015; Jiang and McMillan 2017; Andersson, Ericsson, and Wingkvist 2014; Loyola, Marrrese-Taylor, and Matsuo 2017).

Tan et al. (2007) detected inconsistencies between code and comments and proposed a technique to test Javadoc comments. Zhong et al. (2011) developed a technique to infer specification from natural language API documentation and used it to detect issues in client code.

Conclusion

We argued that comments used to communicate among developers (todo comments) contain valuable content that is currently neglected. We described several techniques – synthesizing code from comments, making comments executable, answering questions in comments, improving comment quality, and detecting dangling comments. These techniques, based on natural language processing and program analysis, have potential to substantially simplify software maintenance and increase software reliability.

Acknowledgments

We thank Rishabh Rai for the initial discussion on this work. This work was partially supported by the US National Science Foundation under Grant No. CCF-1704790.

References


Andersson, R.; Ericsson, M.; and Wingkvist, A. 2014. Mining relations from Git commit messages: An experience report. In SLTC.


Chaurasia, S., and Mooney, R. 2017. Dialog for language to code. In IJCNLP.


Ernst, M. D. 2017. Natural language is a programming language: Applying natural language processing to software development. In SNAPL, volume 71.


Haouari, D.; Sahraoui, H.; and Langlais, P. 2011. How good is your comment? A study of comments in Java programs. In ESEM.


Li, J. J., and Nenkova, A. 2015. Fast and accurate prediction of sentence specificity. In AAAI.


Loyola, P.; Marrrese-Taylor, E.; and Matsuo, Y. 2017. A neural architecture for generating natural language descriptions from source code changes. In ACL.


Padioleau, Y.; Tan, L.; and Zhou, Y. 2009. Listening to programmers’ taxonomies and characteristics of comments in operating system code. In ICSE.


Polikarpova, N.; Kuraj, I.; and Solar-Lezama, A. 2016. Program synthesis from polymorphic refinement types. In PLDI.

Quirk, C.; Mooney, R.; and Galley, M. 2015. Language to code: Learning semantic parsers for if-this-then-that recipes. In ACL.

Raychev, V.; Vechev, M.; and Krause, A. 2015. Predicting program properties from “Big Code”. In POPL.

Sridhara, G. 2016. Automatically detecting the up-to-date status of ToDo comments in Java programs. In ISEC.

Storey, M.-A.; Ryall, J.; Bull, R. I.; Myers, D.; and Singer, J. 2008. TODO or to bug. In ICSE.

Tan, L.; Yuan, D.; Krishna, G.; and Zhou, Y. 2007. /*Comment: Bugs or bad comments*/. In SOSP.

Vasilescu, B.; Casalnuovo, C.; and Devanbu, P. T. 2017. Recovering clear, natural identifiers from obfuscated JS names. In FSE.