Classifying code comments in Java Mobile Applications

Pascarella, Luca

DOI
10.1145/3197231.3198444

Publication date
2018

Document Version
Accepted author manuscript

Published in
Conference on Mobile Software Engineering and Systems

Citation (APA)

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.
Classifying Code Comments in Java Mobile Applications
Luca Pascarella
Delft University of Technology, The Netherlands
L.Pascarella@tudelft.nl

ABSTRACT
Developers adopt code comments for different reasons such as document source codes or change program flows. Due to a variety of use scenarios, code comments may impact on readability and maintainability. In this study, we investigate how developers of 5 open-source mobile applications use code comments to document their projects. Additionally, we evaluate the performance of two machine learning models to automatically classify code comments. Initial results show marginal differences between desktop and mobile applications.

CCS CONCEPTS
• Software and its engineering → Maintaining software;

KEYWORDS
Android, Mining Software Repositories, Code Comments

ACM Reference Format:

1 INTRODUCTION
During software development, software engineers make several choices forging computer programs [5] and to document their rationals, developers write code comments [10]. Past researches demonstrate that code comments are crucial to enhance program readability and maintainability [3]. Despite unaligned documentation may exacerbate maintenance [4, 9], researchers globally agree that having a generous commented code is a good practice [2, 6].

Considering all comments the same may bring to incorrect evaluations especially when code comments are used to perceive the quality of inspected codes, for example, by measuring the code/comment ratio [2, 6]. In a recent study, Pascarella et al. confirmed this limitation arguing that code comments contribute to different meanings [8]. They proposed a novel taxonomy to classify Java comments, investigated how developers of OSS systems use comments, and experimented with automatically classifying code comments.

In this study, we aim at corroborating and possibly improving the current knowledge about the use of code comments in mobile apps. Particularly, we (1) measured the distribution of code comments in the given taxonomy and (2) evaluated the performance of two machine learning models to automatically classify code comments. For this purpose, we inspected 325 Java files of 5 open-source Android mobile apps and we manually classified up to 2,100 comment blocks comprising more than 4,500 lines.

Our results confirm the suitability of the proposed taxonomy in the context of mobile apps. We discovered only marginal differences between desktop and mobile apps. Finally, even though the performance of the machine learning classifiers decreases in the context of a cross-project training we detected a promising capability in reusing the provided training set.

2 METHODOLOGY
The intention of this work denotes and extends the goal of the study conducted by Pascarella et al. aimed at understanding the purpose of the code comment written by developers [8]. Particularly, this study focuses on code comments of open-source Android mobile apps by verifying the generalizability of the proposed approach.

Research questions. To this aim, we observed that Pascarella et al. mainly focus on Java desktop apps [8]. Although the desktop and mobile apps share the same programming languages, developers could adopt the same development approach. A study showed the opposite [11]. Consequently, to understand how Android developers use code comments we defined our first research question:

RQ1. How often does each category occur in OSS Android apps using the Pascarella et al. taxonomy?

Then, we investigate to what extent the automated models proposed by Pascarella et al. can be generalized in a cross-project approach. This leads to our second research question:

RQ2. How effective are the proposed machine learning models in classifying code comments in OSS Android apps?

Project selection. To conduct our analysis we selected 5 heterogeneous Android apps written in Java programming language. They are all open-source projects available through GitHub, hosted by GitHub, and with different size and scope. Table 2 summarizes the characteristics of the selected systems reporting for each project the GitHub link, the number of commits, the number of contributors, and the number of Java lines.

Sample validity. To establish a statistical significant sample of Java files used to measure to what extent Java code comments are
We report the top category whose purpose was not covered by the provided taxonomy. We used a classifier tested considering precision to document their code more than desktop apps developers.

Manual classification. To answer our RQ1 we manually inspected and classified all comments of the sampled Java files. To this aim, we used the definitions of the taxonomy proposed by Pascarella et al. [8]. Such a taxonomy is composed of 22 categories organized into 2 levels. The first level discerns 6 top categories while the second level details 16 definitions (inner categories). Thereafter, we used the COMMEAN tool [8] to support the classification process and reduce human errors. Moreover, we annotated possible comments whose purpose was not covered by the provided taxonomy.

Classification technique. To answer our RQ2 we repeated the settings of Pascarella et al. by applying only a subset of machine learning classifiers. Particularly, we used 2 well-known classes of supervised machine learning algorithms based on probabilistic classifiers and decision tree algorithms, Naive Bayes Multinomial and Random Forest, respectively. These algorithms rely on diverse assumptions aimed at reacting to different execution speeds and overfitting ability. Finally, to improve the performance we applied both data balancing and multi-collinearity corrections.

Classification evaluation. To evaluate the performance of the achieved results we relied on precision and recall, which are based on the evaluation of different combinations of True Positive, False Positive, and False Negative. Finally, we evaluated the performance considering 2 different evaluation techniques: (1) a preliminary evaluation adopting the 10-fold cross-validation, then (2) intersecting datasets of different projects to train and test the model and overcome the limitation imposed by data availability [1].

3 RESULTS

We report the top categories’ results (a detailed report is available online with a new dataset of classified mobile apps comments [7]).

RQ1 results. Figure 1 compares the distributions of comments for desktop and mobile apps. An appreciable difference is evident for Under Dev. category that for mobile apps is more than the double. It suggests that these developers frequently deal with unstable features. While this represents a dynamic development it is also a sign of self-admitted technical debts. At same time PURPOSE category suggests that mobile apps developers resort to code comments to document their code more than desktop apps developers.

RQ2 results. Table 2 shows only the performance of the better classifier tested considering precision (P) and recall (R) for 10-fold and cross-project validation. While 10-fold confirms considerable performance, cross-project validation suffers. This drop is due to a well-known limitation of a supervised algorithm that does not tolerate project-specific key terms.

Despite limitations, this classification method may still help the development life cycle providing an overview of the code quality.

4 THREATS TO VALIDITY AND CONCLUSION

Aware of limitations of our dataset (low number of apps and files) and the taxonomy validity (new categories may emerge) we plan to extend this study by considering a higher number of projects besides a revisited taxonomy validation. Additionally, we plan to overcome the limitation of the cross-project train/test model by creating a representative dataset of mobile apps comments. This dataset may be used to pre-filter code comments and improve the performance of self-admitted technical debt methods.

With our preliminary study, we observed that in both desktop and mobile projects code comments contain valuable information for supporting software development. However, mobile apps developers tend to use code comments for a different purpose. For example, the high percentage of Commented Code category may represent a bad practice, such as with negative consequence on readability and maintainability. Finally, we observed a limitation of supervised classifiers when applied to cross-project validation.

REFERENCES