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:
Luca Pascarella. 2018. Classifying Code Comments in Java Mobile Applications. In *MOBILESoft '18: MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3197231.3198444

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) we 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.

MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5712-8/18/05...$15.00
https://doi.org/10.1145/3197231.3198444

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?*

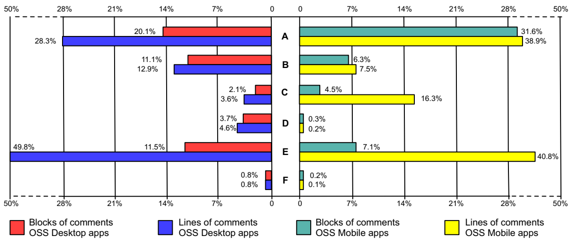
Project selection. To conduct our analysis we selected 5 heterogeneous Android apps written in Java programming language. They are all open-source projects available through Google Play, hosted by GitHub, and with different size and scope. Table 2 summarizes the characteristics of the selected systems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

Table 1: Overview of the projects used in this study

Project	GitHub	Google Play	Commits	Cont.	Lines
AFWall+	https://goo.gl/xzExNE	https://goo.gl/rjhZ2h	1,346	21	28k
Amaze File Manager	https://goo.gl/8k67AJ	https://goo.gl/yV5jJE	2,913	89	41k
AntennaPod	https://goo.gl/XkS14Y	https://goo.gl/t76sCB	2,913	103	61k
ownCloud	https://goo.gl/5b9BVG	https://goo.gl/NhnVAq	6,397	76	63k
WordPress	https://goo.gl/qfkTQA	https://goo.gl/d5dnJe	6,397	131	140k

Sample validity. To establish a statistical significant sample of Java files used to measure to what extent Java code comments are

Figure 1: Comparison of code comments frequency.



used by mobile apps developers we considered all 2108 Java files, we defined a confidence level of 95%, and we set a confidence interval of 5%. We obtained a representative sample of 325 Java files. This sample creates a reasonable large dataset composed of about 2, 100 block of comments and more than 4, 500 lines of comments.

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 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 Multinominal* 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

Table 2: Performance of the Random Forest classifier

Categories	P/R	10-fold	P1	P2	P3	P4	P5
Purpose	P	0.98	0.63	0.70	0.94	0.98	0.98
	R	0.97	1.00	0.98	0.99	1.00	1.00
Notice	P	0.94	1.00	0.00	1.00	1.00	1.00
	R	0.96	0.54	0.00	0.60	1.00	1.00
Under	P	0.99	1.00	0.75		0.00	
development	R	0.99	0.33	0.30		0.00	
Style	P	0.84	1.00				0.00
& IDE	R	0.91	1.00				0.00
Metadata	P	1.00	1.00	1.00	1.00	1.00	1.00
	R	1.00	0.95	0.73	0.50	1.00	1.00
Discarded	P	1.00			0.00		
	R	1.00			0.00		

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

[1] Alberto Bacchelli, Tommaso dal Sasso, Marco D'Ambros, and Michele Lanza. 2012. Content Classification of Development Emails. In *ICSE*.
[2] Manuel J Barranco Garcia and Juan Carlos Granja-Alvarez. 1996. Maintainability as a key factor in maintenance productivity: a case study. In *ICSM*.
[3] Carl S. Hartzman and Charles F. Austin. 1993. Maintenance Productivity: Observations Based on an Experience in a Large System Environment. (1993).
[4] Zhen Ming Jiang and Ahmed E. Hassan. 2006. Examining the Evolution of Code Comments in PostgreSQL. In *Workshop on Mining Software Repositories*.
[5] Yan Liang, Ying Liu, Chun Kit Kwong, and Wing Bun Lee. 2012. Learning the "Whys": Discovering design rationale using text mining-An algorithm perspective. *Computer-Aided Design* (2012).
[6] Paul Oman and Jack Hagemeister. 1992. Metrics for assessing a software system's maintainability. In *Software Maintenance*.
[7] Luca Pascarella. 2018. Appendix. (2018). https://ndownloader.figshare.com/files/10838253.
[8] Luca Pascarella and Alberto Bacchelli. 2017. Classifying code comments in Java open-source software systems. In *Mining Software Repositories (MSR)*.
[9] L. Pascarella, A. Ram, A. Nadeem, D. Bisesser, N. Knyazev, and A. Bacchelli. 2018. Investigating Type Declaration Mismatches in Python. *MaLTeSQuE* (2018).
[10] Jef Raskin. 2005. Comments are more important than code. *Queue* (2005).
[11] D. Zhang and B. Adipat. 2005. Challenges, methodologies, and issues in the usability testing of mobile apps. *Journal of human-computer interaction* (2005).