Does Source Code Quality Reflect the Ratings of Apps?

Gemma Catolino
University of Salerno, Department of Computer Science, Italy

ABSTRACT

In the past, bad code quality has been associated with higher bug-proneness. At the same time, the main reason why mobile users negatively rate an app is due to the presence of bugs leading to crashes. In this paper, we preliminarily investigate the extent to which code quality metrics can be exploited to predict the commercial success of mobile apps. Key results suggest the existence of a relation between code quality and commercial success; We found that inheritance and information hiding metrics represent important indicators and therefore should be carefully monitored by developers.

CCS CONCEPTS

•**Software and its engineering** → *Software creation and management;* •**General and reference** → **Mobile Apps;**

KEYWORDS

Mobile Applications, Metrics, Software Quality

1 INTRODUCTION

Nowadays the wide diffusion of smartphones and tablets has led to an ever increasing demand of mobile applications, which has been even more stimulated by the introduction of application distribution markets (e.g., the *Google Play Store*). Such a growth convinced several industrial companies in creating specialized teams in order to enter in the market of apps. Although such apps are generally smaller than traditional systems (e.g., in terms of size or lifecycle duration [1]), their complexity is massively increasing, thus requiring a proper monitoring of source code quality during both development and maintenance phases [2].

An external indicator of quality is the mechanism that allows end users to rate apps using scores and reviews [2], but this is not enough. Indeed, as Palomba et al. [3] showed, the quality of source code has a noticeable impact on the bug-proneness of a system; Moving this concept in the mobile field, the presence of bugs might possibly lead to malfunctions, e.g., crashes. This eventually implies an app to be low-rated by its end users [4], and in the extreme case to be even uninstalled.

Starting from these findings we want to understand whether code quality metrics can be exploited to predict the rate of a mobile app. More specifically, in this paper the goal is to empirically evaluate the predictive power of code quality metrics when determining the commercial success of a mobile apps. To this aim, we built (i) a prediction model that takes into account 13 code metrics and 8 code smell types and (ii) a prediction model where we apply a feature selection algorithm, i.e., *Wrapper* [5], in order to extract only the relevant features starting from the initial set described above. We assessed and compared the performance of the considered models on a dataset of 439 mobile apps. Key results of our study indicate how (i) the model where we applied the *Wrapper* algorithm performed better than the other (the value of *F-Measure* improved of 9%) and (ii) inheritance and information hiding metrics should be carefully monitoring during the development and maintenance activities of mobile apps as a way to control commercial success.

MOBILESoft '18, Gothenburg, Sweden
 978-1-4503-5712-8/18/05…$15.00
DOI: https://doi.org/10.1145/3197231.3198447

2 STUDY DESIGN

The *goal* of the study is to evaluate the usefulness of code quality metrics when predicting the commercial success of a mobile app, with the *purpose* to understand whether code quality can be actually exploited to provide developers with information on the possible commercial success of their apps. The *quality focus* is on the accuracy of a prediction models based on code quality attributes, while the *perspective* is of mobile app developers and tool vendors: (i) The first ones interested in understanding the value of code quality; (ii) The second ones interested in how to better support developers.

As for the *context* of the study, we relied on a publicly available dataset previously defined by Grano et al. [6]. This includes information about (i) rates, (ii) reviews (extracted from the *Google Play Store*), (iii) code metrics, and (iv) code smells of 630 open source Android apps mined from F-Droid and having different size and scope. The dataset contains 395 unique apps, while multiple versions are available for most of them. In our study, we selected all the apps having at least two versions, leading to a selection of 439 total apps. All the pieces of information are reported at app-level, meaning that the dataset contains the average of each metric computed over all the classes composing an app. Further details of the apps are available on their repository [1].

We formulated the following research questions:

RQ1: *To what extent code quality metrics can be exploited to measure the commercial successful of mobile apps?*

RQ2: *Can the application of a feature selection technique have an impact on the performance of commercial success prediction models?*

To answer **RQ1** we used as independent variables the 13 different code quality metrics (e.g., CK Metrics, Android Metrics) as well as the 8 code smell types (e.g., Blob [3]). Due the space limitation we report all the metrics used, their description and how Grano et al. [6] computed them in our online appendix [7]. Starting from the dataset [6], we firstly extracted the metric values for the 439 mobile apps (i.e., the independent variables). Besides the value of metrics, each apps is associated to a rate, which is the average ratings assigned to them by the end users. We used such information as dependent variable. In particular, we discriminate high-

[1] https://github.com/sealuzh/user_quality

and low-rated apps using the heuristic by Khalid et al. [8]: Apps whose average rating was strictly higher than 3.5 were considered as high-rated, otherwise they were marked as low-rated.

Once defined the set of independent and dependent variables, we focused on training a machine learning model. In this step, we also took into account the data unbalance problem; We applied the *Synthetic Minority Over-sampling Technique* [9] (SMOTE) to make the training set uniform between high- and low-rated instances. Afterwards, we built a *Random Forest* [10] model: This choice was driven by previous findings showing that it is among the top machine learning approaches to use, since it is robust to noisy data [11]. Finally, we assessed the performance of the prediction model using a 10-fold cross validation strategy. To evaluate the performance of the model we adopted five widely used metrics, such as *precision* (P), *recall* (R), *F-Measure* (F-M), *Matthews correlation coefficient* (MCC), and *Area Under the ROC Curve* (AUC-ROC) [12].

Although *Random Forest* should automatically select the most appropriate features to use [10], in **RQ2** we tested whether the application of a second step of feature selection might boost the overall performance of the approach. To this aim, we made sure to avoid model over-fitting by considering the best combination of predictors given as output by the *Wrapper* algorithm [5].

As a result, only 4 metrics were considered meaningful i.e., *Number of Children* (NOCH), *Depth of Inheritance Tree* (DIT), *Percent Public Instance Variables* (PPIV) and *Access to Public Data* (APD). There metrics were used to train the Random Forest classifier. At the same time, we evaluated them using the same procedure i.e., 10-fold cross, and metrics, e.g., F-Measure, AUC-ROC, MCC.

3 RESULTS

Table **1** reports the overall performance achieved by the experimented models. The first row of the table showed the performance of the model without the second step of feature selection (NFS), while the second row reports the results for the model where we applied the Wrapper algorithm (FS). The dataset used in this study ia available in the online appendix [7].

Table 1: Overall Performance (in %) of the prediction model.

MODEL	P	R	F-M	MCC	AR
NFS	56	53	54	8	53
FS	65	61	63	26	65
IMPROVEMENT	**+9%**	**+8%**	**+9%**	**+18%**	**+12%**

The first thing that leaps to the eyes is the large different between the performance achieved by NFS and the FS model. Indeed, using a feature selection technique (FS model) the value *F-Measure* improves of 9%; This is mainly due to the improvement of the overall *Recall* i.e., 8%. From a practical point of view, this means that the model is able to discover a greater number of true positive instances, thus being able to better discriminate between low- and high-rated apps. The improvement is observable for all the other evaluation metrics (e.g., *MCC*). Thus, we claim that a second step of feature selection can improve the prediction capabilities, even in cases where the classifier has a way to reduce the risk of multi-collinearity.

Looking more in depth into the results, we focused our attention on the metrics that the *Wrapper* technique selected as more relevant. Interestingly, NOCH and DIT belong to the category of inheritance metrics, i.e. indicators of the extent to which a class makes use of inheritance mechanisms. Yu et al. [13] showed that such inheritance metrics can strongly affect the bug-proneness of software systems. Similarly, PPIV and APD are metrics that quantify the degree a class exposes its data to other classes: As shown by Aggarwal et al. [14], low information hiding leads to higher bug-proneness. From a practical point of view this means that a "negative" value of the four more relevant metrics might lead to an increasing of the probability that a system will contain bugs. As a likely consequence, an app might be rated lower than apps that keep under control the value of such metrics [4].

As a consequence of our findings, *we argue that code quality and, in particular inheritance and information hiding metrics should be carefully monitored during the development and maintenance activities of mobile apps as a way to control commercial success.*

4 FUTURE DIRECTIONS

Our study showed how code quality metrics might be exploited in order to predict the commercial success of mobile apps. For this reason, we plan a further investigation taking into account other factors related to usability and feedback of the end users.

REFERENCES

[1] A. I. Wasserman, "Software engineering issues for mobile application development," in *Proc. of workshop on Future of software engineering research*, 2010.

[2] G. Catolino, "Just-in-time bug prediction in mobile applications: the domain matters!" in *Proc. of the 4th Int. Conf. on Mobile Soft.Engineering and Systems*, 2017.

[3] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation," *Empirical Soft. Engineering*, 2017.

[4] F. Palomba, M. Linares-Vásquez, G. Bavota, R. Oliveto, M. Di Penta, D. Poshyvanyk, and A. De Lucia, "Crowdsourcing user reviews to support the evolution of mobile apps," *Journal of Systems and Soft.*, 2018.

[5] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, 1997.

[6] G. Grano, A. Di Sorbo, F. Mercaldo, C. A. Visaggio, G. Canfora, and S. Panichella, "Android apps and user feedback: a dataset for software evolution and quality improvement," in *Proc. of the 2nd ACM SIGSOFT Int. Workshop on App Market Analytics*.

[7] G. Catolino. (2018) Do high-rate app developers can more about source code quality? - https://figshare.com/s/de9e4f62ed689d3843c3.

[8] H. Khalid, E. Shihab, M. Nagappan, and A. E. Hassan, "What do mobile app users complain about?" *IEEE Soft.*, 2015.

[9] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 2002.

[10] T. Dietterich, "Overfitting and undercomputing in machine learning," *ACM Comput. Surv.*, 1995.

[11] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *ACM/IEEE 30th Int. Conf. on Soft.Engineering*. IEEE, 2008.

[12] Baeza-Yates *et al.*, *Modern information retrieval*.

[13] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo metrics. an industrial case study," in *6th Eur. Conf. on Soft. Maintenance and Reengineering*. IEEE, 2002.

[14] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: a replicated case study," *Soft. process: Improvement and practice*, 2009.