The Journal of Systems & Software 143 (2018) 14–28

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Enhancing change prediction models using developer-related factors

Gemma Catolino*,[a], Fabio Palomba[b], Andrea De Lucia[a], Filomena Ferrucci[a], Andy Zaidman[c]

[a] University of Salerno, Italy
[b] University of Zurich, Switzerland
[c] Delft University of Technology, The Netherlands

ARTICLE INFO

Keywords:
Change prediction
Mining software repositories
Empirical study

ABSTRACT

Continuous changes applied during software maintenance risk to deteriorate the structure of a system and are a threat to its maintainability. In this context, predicting the portions of source code where specific maintenance operations should be focused on may be crucial for developers to prevent maintainability issues. Previous work proposed change prediction models relying on product and process metrics as predictors of change-prone source code classes. However, we believe that existing approaches still miss an important piece of information, i.e., developer-related factors that are able to capture the complexity of the development process under different perspectives. In this paper, we firstly investigate three change prediction models that exploit developer-related factors (e.g., number of developers working on a class) as predictors of change-proneness of classes and then we compare them with existing models. Our findings reveal that these factors improve the capabilities of change prediction models. Moreover, we observed interesting complementarities among the prediction models. For this reason, we devised a novel change prediction model exploiting the combination of developer-related factors and product and evolution metrics. The results show that such a combined model is up to 22% more effective than the single models in the identification of change-prone classes.

1. Introduction

Software systems are subject to continuous evolution, driven by changes in the requirements imposed by the stakeholders on the one hand and by the resolution of bugs threatening their reliability on the other hand (Lehman and Belady, 1985). Unfortunately, the more changes developers apply to the software system the more complex the system is likely to become, thereby eroding the original design and possibly reducing the overall maintainability (Parnas, 1994). While change is unavoidable, it needs to be *controlled* by developers. In this context, the up front identification of code elements potentially exhibiting a higher change-proneness may be important for developers for two main reasons: on the one hand, change-proneness can be considered a quality indicator that can be used to warn developers when touching code that should be refactored (Zhou et al., 2009); on the other hand, developers can plan preventive maintenance operations, such as refactoring (Fowler et al., 1999), peer-code review (Bacchelli and Bird, 2013; Beller et al., 2014), and testing (Soetens et al., 2016; Moonen et al., 2008), aimed at increasing the quality of the code and reducing future maintenance effort and costs (Fowler et al., 1999).

Change prediction is the branch of software engineering aimed at identifying the entities more prone to be modified in the future, helping developers in both planning preventive maintenance actions and keeping the complexity of source code under control (Koru and Liu, 2007). Previous research focused on (i) the analysis of the factors influencing the change-proneness of classes (Bieman et al., 2003; Di Penta et al., 2008; Khomh et al., 2012; Miryung Kim, 2014; Soetens et al., 2016) and (ii) the definition of prediction models able to support developers by recommending the classes on which preventive maintenance actions should be performed (Sharafat and Tahvildari, 2007; Han et al., 2008; Sharafat and Tahvildari, 2008; Han et al., 2010).

An important body of previous work has explored the possibility to use product metrics (e.g., the Chidamber and Kemerer Object Oriented metric suite Chidamber and Kemerer, 1994) as indicators of the change-proneness of classes. In this case, the underlying assumption is that classes having low code quality are more prone to be modified in the future. As an example, Zhou et al. (2009) proposed a change prediction model relying on cohesion, coupling, and inheritance metrics, finding that code metrics can be exploited for predicting change-prone classes, while the number of lines of code often represents a confounding effect

* Corresponding author.
E-mail addresses: gcatolino@unisa.it (G. Catolino), palomba@ifi.uzh.ch (F. Palomba), adelucia@unisa.it (A. De Lucia), fferrucci@unisa.it (F. Ferrucci), a.e.zaidman@tudelft.nl (A. Zaidman).
URL: http://docenti.unisa.it/001775/home (F. Ferrucci).

https://doi.org/10.1016/j.jss.2018.05.003
Received 27 July 2017; Received in revised form 28 April 2018; Accepted 2 May 2018
Available online 03 May 2018
0164-1212/

worsening the performance of prediction models.

In the recent past, Elish and Al-Rahman Al-Khiaty (2013) investigated the role of process metrics in the context of change prediction models. More specifically, they devised the so-called *evolution metrics*, i.e., metrics characterizing the history of a class under different perspectives (e.g., the number of past modifications of a class in a certain time window). Afterwards, they found that a change prediction model based on such evolution metrics performs better than the one built using code metrics.

Despite the effort devoted by the research community over the years, we believe that current approaches missed an important piece of information, i.e., they do not consider developer-related factors, which can provide information on how developers perform modifications and how complex the development process is. In our previous paper (Catolino et al., 2017a) we conjectured that such aspects can be a useful source of information to predict classes more likely to be changed in the future. To verify the conjecture, we empirically evaluated the performance of three prediction models based on developer-related factors previously defined in literature. Specifically, we experimented with (i) Basic Code Change Model (BCCM) proposed by Hassan (2009) which relies on the entropy of changes applied by developers, (ii) the Developer Changes Based Model (DCBM) devised by Di Nucci et al. (2017) that considers to what extent developers apply scattered changes in the system, and (iii) the Developer Model (DM) proposed by Bell et al. (2013) which analyzes how many developers touched a code element over time. Although such models were originally proposed in the context of bug prediction, we selected them since they are based on metrics possibly influencing the change-proneness of classes as well. For instance, the lack of coordination between multiple developers working on the same code element may lead to the introduction of design pitfalls that negatively influence the maintainability of source code (Kraut and Streeter, 1995), possibly making it more change-prone. Furthermore, to have a comprehensive view of the usefulness of developer-related factors in change prediction, we also compared the performance of the developer-based models with the ones proposed by Elish and Al-Rahman Al-Khiaty (2013) and Zhou et al. (2009).

The results demonstrated that the developer-based prediction models reached an overall F-Measure ranging between 57% and 68% and an *Area Under the ROC Curve* (AUC-ROC) ranging between 56% and 70%. Among them, the DCBM model was the one obtaining the highest accuracy. When compared to the model exploiting the evolution metrics devised by Elish and Al-Rahman Al-Khiaty (2013), we found that the developer-based prediction models improved the F-Measure by up to 9% and the AUC-ROC by up to 9%. More importantly, all the investigated prediction models showed interesting complementarities in the set of change-prone classes correctly predicted. Indeed, we discovered that different models capture different change-prone instances, e.g., change-prone classes modified by several developers can only be captured by developer-based models and not by the approaches relying on product or evolution metrics, while poorly cohesive classes changed by few developers are only detectable using a model based on product metrics. Such a complementarity paves the way for new prediction models exploiting a combination of the predictors used by the investigated models.

In this paper, we extend our previous work (Catolino et al., 2017a) with the aim of (i) designing a combined change prediction model that exploits the complementarities among the investigated product, process, and developer-based models and (ii) increasing the generalizability of our findings by considering a larger dataset. More specifically, we:

1. Devise and evaluate the performance of a new change prediction model based on a combination of the metrics used by the previously investigated models. On the basis of the complementarities discovered among the investigated change prediction models, we performed a detailed study—exploiting the *Information Gain* algorithm (Quinlan, 1986)—with the aim of finding the subset of predictors more relevant for the identification of change-prone classes. Then, we exploited them to build and evaluate a combined change prediction model.
2. Extend the empirical evaluation of developer-based change prediction models and their comparison with the state of the art. While we previously analyzed 197 releases of 10 software systems having a total of 105,693 commits and 358 developers, this study considers 192,274 commits made by 657 developers over 408 releases of 20 software systems having different size and scope.

On the one hand, the results of the study confirm our previous findings showing the usefulness of developer-related factors in change prediction. On the other hand, we found that the novel combined change prediction model clearly outperforms the baseline models, being more accurate in the predictions by up to 22% in terms of F-Measure.

Structure of the paper: In Section 2 we discuss background and related literature on change prediction. In Section 3 the design of the empirical study is described, while Section 4 reports the results achieved when evaluating the performance of the experimented change prediction models. Section 5 discusses the threats that could affect the validity of our study. Finally, Section 6 concludes the paper.

2. Background and related work

In this section we firstly present a background on the problem of change prediction and how it can be used to improve the quality of source code; then, we overview the related literature.

2.1. The problem of predicting change-prone classes

Change-prone classes represent pieces of code that, for different reasons, tend to change more often: this may be due to the importance of a class for the business logic of the system or because it is not properly designed by developers (e.g., in the presence of code smells Khomh et al., 2012; Palomba et al., 2017). Keeping track of these classes can be relevant to create awareness among developers about the fact that these classes tend to change frequently, possibly hiding design issues that should be solved.

It is important to note that this type of classes must not be confused with bug-prone code elements. The two sets of classes might have some relationships but they still remain conceptually disjoint. In the first place, bug-proneness indicates source code that is more likely to have bugs in the near future, thus, the fact that a class has bugs does not imply that it changes more often. Secondly, bug-prone classes might also be change-prone (changes are made to correct faults), but corrections are not the only reason for changes, as classes might change due to software evolution. Thus, change- and bug-proneness of classes might have some relation, but are not the same.

Change prediction models represent an established way to identify change-prone classes (Zhou et al., 2009). In this context, a supervised technique is exploited, where a set of independent variables (i.e., metrics characterizing a class) are used by a machine learning classifier (e.g., Logistic Regression Le Cessie and Van Houwelingen, 1992) to predict a dependent variable (i.e., the change-proneness of classes). In a real-case scenario, change prediction models might be directly integrated in developers software analytics dashboards (e.g., BITERGIA[1]), thus continuously providing feedback on the source code classes that are more likely to change in the future. Such feedback can be used by developers as input for performing preventive maintenance activities before putting the code into production: for instance, in a continuous

[1] https://www.bitergia.com.