Information and Software Technology 67 (2015) 220–235

Contents lists available at ScienceDirect

Information and Software Technology

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

Learning dependency-based change impact predictors using independent change histories

Hani Abdeen*, Khaled Bali, Houari Sahraoui*, Bruno Dufour

DIRO, Université de Montréal, QC, Canada

ARTICLE INFO

Article history:
Received 5 November 2014
Revised 5 June 2015
Accepted 23 July 2015
Available online 11 August 2015

Keywords:
Change impact analysis
Impact set prediction
Change history analysis
Change impact graph
Software maintenance

ABSTRACT

Context: Recent studies showed that combining present data, which are derived from the current software version, with past data, which are derived from previous software versions, can improve the accuracy of change impact predictions. However, for a specific program, existing combined techniques can rely only on version history of that program, if available, and the prediction results depend on the variety of available change impact examples.

Objective: We propose a hybrid probabilistic approach that predicts the change impact for a software entity using, as training data, existing version histories of whatever software systems.

Method: Change-impact predictors are learned from past change impact graphs (CIGs), extracted from the version history, along with their associations with different influencing factors of change propagation. The learning examples in CIGs are not specific to the software entities that are covered in those examples, and the change propagation influencing factors are structural and conceptual dependencies between software entities. Once our predictors are trained, they can predict change impacts regardless of the version history of the software under-analysis. We evaluate our approach using four systems in two scenarios. First, we use as training data the CIGs extracted from previous versions of the system under-analysis. Second, for each analyzed system, we use only the training data extracted from the other systems.

Results: Our approach produces accurate predictions in terms of both precision and recall. Moreover, when training our classifiers with a large variety of CIGs extracted from the change histories of different projects, the recall scores of predicted impact sets were significantly improved.

Conclusion: Our approach produces accurate predictions for new classes without recorded change histories, as well as for old classes. For the systems considered in our evaluation, once our approach is trained with a variety of CIGs it can predict change impacts with good recall scores.

1. Introduction

Change impact analysis (CIA) continues to gain a considerable attention in software engineering research [1–3]. In the field of impact set prediction, the ultimate goal is to identify program elements that should be modified in order to accommodate a change request, and hence assist maintainers in estimating the consequences and cost of a given change [4].

In literature, many CIA techniques have been proposed to predict the impact of change requests at code level [3]. Those techniques use a variety of data on the software under analysis, extracted by static (*e.g.,* [5–8]) or dynamic (*e.g.,* [9,10]) analysis, and/or by mining software repositories (MSR) *e.g.,* [11–14]. In another category, a considerable body of CIA techniques in the literature aims at predicting change coupling (co-change) instead of identifying the impact set of a specific change request. Such MSR-based techniques exploit co-change histories with the heuristic that software entities that changed together in the past are likely to change together in the future [15–20]. In these techniques, a mandatory prerequisite is that the software version history must be available and well established. Moreover, even if the software version history is available, in these techniques, association rules (called change patterns) are specific to the existing software entities which have changed in the past.

Recent studies showed that combining present data (*e.g.,* dependencies between software entities in the present software snapshot of the system under-analysis) with past data, which are derived from multiple prior software versions (*i.e.,* MSR-based techniques), can

* Corresponding authors.
E-mail addresses: hani.abdeen@gmail.com, abdeenha@iro.umontreal.ca (H. Abdeen), balikhal@iro.umontreal.ca (K. Bali), sahraouh@iro.umontreal.ca (H. Sahraoui), dufour@iro.umontreal.ca (B. Dufour).

http://dx.doi.org/10.1016/j.infsof.2015.07.007
0950-5849/© 2015 Elsevier B.V. All rights reserved.

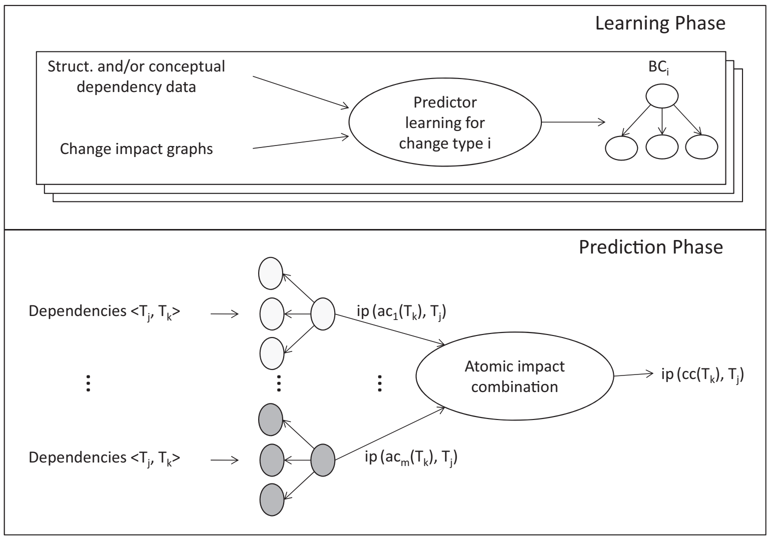
Learning Phase
Struct. and/or conceptual dependency data
Change impact graphs
Predictor learning for change type i
BC$_i$
Prediction Phase
Dependencies <T$_j$, T$_k$>
ip (ac$_1$(T$_k$), T$_j$)
Atomic impact combination
ip (cc(T$_k$), T$_j$)
Dependencies <T$_j$, T$_k$>
ip (ac$_m$(T$_k$), T$_j$)

Fig. 1. Overview.

significantly improve change impact predictions [11–13]. Here again, for a specific software/entity, existing techniques rely heavily on the version history of that software/entity, when this history is available and well established. Indeed, the accuracy of change impact predictions depends on the number and variety of impact observations recorded in the software version history for that specific program/entity. For instance, consider the prediction of the impact set of a change *ch*, which consists of removing a method and modifying another method in a class *X*. If *X* is a new class, or if it did not experience frequent changes similar to *ch* in the past, the predictions will not be reliable.

This paper proposes a novel contribution which uses dependencies between prior changes (*i.e.,* prior *Change Impact Graphs*: CIGs shortly), instead of prior co-changes or evolutionary coupling, as *past data* to train predictors of the impact set of future changes.

More precisely, we propose a probabilistic approach that initially learns a set of Bayesian classifiers to predict the impact of atomic change types. Then, it combines the results of these classifiers with different integration strategies to predict the impact of a complex change. Our approach can predict impact sets using, as input, only dependencies that can propagate the change between software entities, extracted from the current version of the software system under-analysis (*i.e.,* structural dependencies, conceptual dependencies, or both). However, to allow the prediction, Bayesian classifiers have to be trained with prior CIGs and the dependencies associated to those prior CIGs. Hence, for a specific change request *ch* within a class *X*, the prediction of *ch*'s impact set is done by using the classifiers previously trained with change-impact examples in prior CIGs (with their corresponding dependencies) that involve changes similar to the ones involved in *ch*. A major advantage over existing impact prediction techniques is that, these prior CIGs can be extracted by comparing successive versions, available, of whatever software systems. Indeed, our approach does not draw any assumptions on the relations between the software system/entities under-analysis and the software systems/entities that are used to extract training data –with the exception that all the used software systems should belong to the same programming domain (*e.g.,* Object-Oriented) and, hence, they share the same influencing factors of change propagation.

We evaluate our approach with two different scenarios:

- *Scenario 1 – Use change history of the system under-analysis:* the system under-analysis has an available version history, and thus we extract prior CIGs from its change history to train the classifiers.
- *Scenario 2 – Borrow change histories of different systems:* regardless of change history of the system under-analysis, in this scenario, we borrow the prior CIGs from other systems and use them as training data. Although used systems for extracting prior CIGs may differ from the system under-analysis, in terms of coding practices, structure and relationships between software entities, we believe that this scenario can improve the accuracy of prediction results due to the increased number and variety of used CIGs for training the classifiers.

We applied both scenarios on four Object-Oriented Java systems having from 5 to 22 released versions, and the size of used last releases, goes from 69 to 331 classes. The results show that our approach produces good predictions (in terms of precision and recall) in both above-mentioned scenarios. Moreover, we show that, for our study sample, there is no difference in terms of prediction precision when the training data is taken from the history of the system under analysis (Scenario 1) or borrowed from the histories of other systems (Scenario 2). However, the recall is improved in the case of Scenario 2 (statistically significant).

The reminder of this paper is organized as follows. Section 2 describes our approach for building and using the change prediction models. The setting and the results of our empirical study are detailed respectively in Sections 3 and 4, together with the threats to validity (Section 5). Section 6 gives an overview of the different approaches used for change impact analysis. Finally, a conclusion is provided in Section 7.

2. Prediction approach

This section presents our probabilistic approach to predict change impact. Our approach targets the context of object-oriented software systems at the class granularity level.

2.1. Overview

As shown in Fig. 1, our approach has two phases: learning and prediction. In the learning phase, we train Bayesian classifiers to predict the impact of individual atomic changes (e.g., *change method, add attribute, remove class*). Actually, we define a Bayesian classifier for each atomic change type. For a specific atomic change $ac(T_1)$ in a class T_1, the trained Bayesian classifier, which is associated to the *ac*'s type, determines the probability that another class T_2 will be impacted by $ac(T_1)$, knowing the dependencies between T_1 and T_2.