The Journal of Systems and Software 103 (2015) 248–265

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A comprehensive study of the predictive accuracy of dynamic change-impact analysis

Haipeng Cai*, Raul Santelices

Department of Computer Science and Engineering University of Notre Dame, Notre Dame IN 46556, USA

ARTICLE INFO

Article history:
Received 3 June 2014
Revised 7 February 2015
Accepted 9 February 2015
Available online 16 February 2015

Keywords:
Impact analysis
Impact prediction
Accuracy study

ABSTRACT

The correctness of software is affected by its constant changes. For that reason, developers use change-impact analysis to identify early the potential consequences of changing their software. Dynamic impact analysis is a practical technique that identifies potential impacts of changes for representative executions. However, it is unknown how reliable its results are because their accuracy has not been studied. This paper presents the first comprehensive study of the predictive accuracy of dynamic impact analysis in two complementary ways. First, we use massive numbers of random changes across numerous Java applications to cover all possible change locations. Then, we study more than 100 changes from software repositories, which are representative of developer practices. Our experimental approach uses sensitivity analysis and execution differencing to systematically measure the precision and recall of dynamic impact analysis with respect to the actual impacts observed for these changes. Our results for both types of changes show that the most cost-effective dynamic impact analysis known is surprisingly inaccurate with an average precision of 38–50% and average recall of 50–56% in most cases. This comprehensive study offers insights on the effectiveness of existing dynamic impact analyses and motivates the future development of more accurate impact analyses.

1. Introduction

Modern software is increasingly complex and changes constantly, which threatens its quality, reliability, and maintainability. Failing to identify and fix defects caused by software changes can have serious effects in economic and human terms. Therefore, it is crucial to provide developers with effective support to identify dependencies in code and deal with the impacts of changes that propagate via those dependencies. Specifically, developers must understand the risks of modifying a location in a software system *before* they can budget, design, and apply changes there. This activity, called *(predictive) change-impact analysis* Bohner and Arnold (1996); Li et al. (2013); Rajlich (2011), can be quite challenging and expensive because changes affect not only the modified parts of the software but also the parts where their effects propagate.

An existing important approach to assessing the effects of changes in a program is *dynamic* impact analysis Apiwattanapong et al. (2005); Breech et al. (2005, 2006); Law and Rothermel (2003a, 2003b); Orso et al. (2003, 2004); Ren et al. (2004). This approach uses runtime information such as profiles and traces to identify the entities that might be affected by changes under specific conditions—those created by the test suite for that program. The resulting *impact sets* (affected entities) of dynamic approaches that are safe for the execution sets utilized are smaller Law and Rothermel (2003b), and thus usually more manageable, than those obtained by safe static analyses as they focus on only a particular subset of all possible inputs (and executions accordingly) Apiwattanapong et al. (2005). For scalability, most dynamic impact analyses operate on *methods* as the entities that can be changed and be impacted by changes Apiwattanapong et al. (2005); Breech et al. (2005, 2006); Law and Rothermel (2003a, 2003b); Orso et al. (2003, 2004); Ren et al. (2004). At the statement level, dynamic slicing Agrawal and Horgan (1990); Korel and Laski (1988); Zhang et al. (2003), in its forward version, can be used for impact analysis in greater detail but at a greater computational cost Law and Rothermel (2003b); Masri et al. (2006); Orso et al. (2004).

Despite its attractiveness, however, dynamic impact analysis has not been evaluated for its ability to *correctly predict the actual impacts* that changes have on software. Techniques exist to *describe* the impacts of changes *after* changes have been made (e.g., Apiwattanapong et al. (2007); Ramanathan et al. (2006); Santelices et al. (2010)). However, for predictive purposes—*before* the changes are even known—the usefulness of dynamic impact sets remains a mystery. For instance, CHIANTI (Ren et al., 2004) and its applications (Ren et al., 2006; Stoerzer et al., 2006) evaluate their impact analysis results with respect to affected test cases or changes between pairs of program

* Corresponding author. Tel.: +1 601 8184 273.
E-mail addresses: hcai@nd.edu, chapering@gmail.com (H. Cai), rsanteli@nd.edu (R. Santelices).

http://dx.doi.org/10.1016/j.jss.2015.02.018
0164-1212/© 2015 Elsevier Inc. All rights reserved.

versions, but these approaches are *descriptive* (Bohner and Arnold, 1996) rather than *predictive*. The rest of the literature focuses only on comparing the sizes of dynamic impact sets (i.e., relative precision) and the relative efficiency of the techniques without considering how closely those impact sets approximate the real impacts of changes.

To address this problem, in this paper, we introduce a novel approach for assessing the *accuracy* (precision and recall) of dynamic impact analyses. The approach uses SensA, a sensitivity-analysis technique we recently developed (Cai et al., 2014a; Santelices et al., 2013b). We adapted SensA for making large numbers of random changes efficiently across the software and running dynamic impact analysis on those change locations. While random changes do not necessarily represent all changes, the impacts they find (or not) can help identify *deficiencies in precision and recall* of dynamic impact analyses *across the entire software*. The benefit of this approach is that all methods in a program can be analyzed, in contrast with others based on code repositories which, if available, offer selections of changes that, although supposedly more representative of developer practice, are less comprehensive.

Nevertheless, it is important to also incorporate in a study of impact analysis the changes that developers typically make to complement the comprehensiveness of the new approach with the representativity of real changes. Thus, we designed our approach to support repository changes in addition to the random changes inserted by SensA. Specifically, our approach takes changes committed by developers into SVN repositories and also changes (bug fixes) from the SIR repository Do et al. (2005) made by other researchers for their own studies.

To find the *ground truth*—the code actually impacted by changes—our approach uses *execution differencing* (Ramanathan et al., 2006; Santelices et al., 2010; Sumner and Zhang, 2013) on the program before and after each change is applied to determine which code is really affected (i.e., code that changes states or occurrences (Podgurski and Clarke, 1990)). By design, we use the *same* test suite as the dynamic impact analysis to assess the accuracy of that analysis under the same runtime conditions. The similarities and differences between this ground truth and the impact sets indicate how accurate the evaluated impact analysis can be for predicting actual impacts.

Using this approach with both random and repository-based changes, we performed a comprehensive empirical study of the accuracy of dynamic impact analysis on multiple Java subjects. For dynamic impact analysis, we chose the best known and most cost-effective technique from the literature: PathImpact (Law and Rothermel, 2003a) with *execute-after-sequences* (EAS) (Apiwattanapong et al., 2005), which we call PI/EAS. (Another technique, InfluenceDynamic (Breech et al., 2006), is only marginally more precise yet much more expensive, and also more complicated, than PI/EAS.) For different sets of changed methods in each subject, we obtained the impact set predicted by PI/EAS and computed its precision and recall with respect to the ground truth.

The results of our study are surprising. On average for all subjects, the *precision* of the impact sets ranged between 38% and 50% depending on the change type. In other words, at most one in two methods reported by PI/EAS was actually impacted by the studied changes. Moreover, the average *recall* of PI/EAS was about 50–56% except for SIR changes, for which the average recall was 87%. These results reveal that dynamic impact analysis can also miss many real impacts. Interestingly, the accuracy of PI/EAS was lower for SVN changes, made by developers in practice, than for artificial changes (random and SIR). These results suggest that developers should not expect a great accuracy from existing dynamic impact analyses and that there is plenty of room for improving such techniques.

Our study also showed that, often, the precision was high and the recall was low or vice versa. We hypothesized and confirmed that, when the program execution is shorter *before* a change (when predictive impact analysis is performed) than after a change, runtime effects are missed (e.g., many methods execute only in the changed program). Interestingly, the precision in such cases is greater than usual, suggesting that methods in dynamic impact sets are more likely to be truly impacted if they execute relatively soon after the change.

In all, the main contributions of this paper are:

- An approach for evaluating the accuracy of dynamic change-impact analysis techniques with respect to the actual impacts of source-code changes
- An implementation of the approach that applies massive numbers of changes to support accuracy studies with both artificial and repository changes
- A comprehensive study—the first of its kind—on multiple Java subjects that estimates the accuracy of the most representative and cost-effective dynamic impact analysis known and shows the inadequacy of existing techniques for predicting the effects of changes

The rest of this paper is organized as follows. Section 2 details the problem addressed by, and the motivation of, this work. Section 3 provides the necessary background and a working example. Section 4 discusses the qualities of PI/EAS that affect its accuracy. Then, Section 5 presents our approach for assessing that accuracy with artificial and repository changes. Sections 6 and 7 present our studies using this approach for both types of changes. Finally, Section 8 discusses related work and Section 9 concludes.

2. Problem and motivation

The new paradigm of software engineering focuses on software evolution, which is characteristic of incremental changes (Rajlich, 2006; Rajlich and Gosavi, 2004). One of the two steps of designing incremental changes is impact analysis, a key activity during software development that assesses the full extent of the changes (Rajlich, 2006, 2014). In fact, several industrial user studies have also shown that developers widely recognize the crucial role of impact analysis in their daily tasks (LaToza et al., 2006; de Souza and Redmiles, 2008; Tao et al., 2012), with views on impact analysis issues varying with different perspectives and organization levels (Rovegard et al., 2008).

However, developers face many challenges to impact analysis (Acharya and Robinson, 2011; LaToza and Myers, 2010; LaToza et al., 2006; Tao et al., 2012), and one of the most critical issues is the *uncertain* results produced by existing analyses (Rovegard et al., 2008; de Souza and Redmiles, 2008). In addition, an even more critical issue reported by developers is that available analyses are *incomplete* (Rovegard et al., 2008). Taken together, these studies show that developers have already realized and encountered the *inaccuracy* of today's impact analysis in practice. And furthermore, such inaccuracy has been suggested as an issue with existing analysis techniques and tool supports that block their adoption in practice (Acharya and Robinson, 2011; Rovegard et al., 2008).

On the other hand, despite of a large and growing body of research on impact analysis (Lehnert, 2011; Li et al., 2013), the empirically suggested inaccuracy has not yet been formally studied or systematically quantified (Li et al., 2013). Although a great number of automatic impact-analysis tools have been developed as well (e.g., Breech et al. (2006); Law and Rothermel (2003a); Orso et al. (2003)), the accuracy of most existing impact analyses was evaluated using relative measures only (e.g., the ratios of impact-set sizes of one technique over the other) with respect to the execution sets utilized by the analysis (Li et al., 2013). Particularly, when it comes to *predictive* impact analysis, empirical accuracy measurement with respect to actual impact sets (as ground truth) is still missing.

While predictive impact analysis plays a vital role in driving software evolution as it enables developers to assess potential risks and consequences of candidate changes during the planning phase for