A Reflection on the Predictive Accuracy of Dynamic Impact Analysis

Haipeng Cai
Washington State University, Pullman, USA
haipeng.cai@wsu.edu

***Abstract*—Impact analysis as a critical step in software evolution assists developers with decision making as regards whether and where to apply code changes in evolving software. Dynamic approaches to this analysis particularly focus on the effects of potential code changes to a program with respect to its concrete executions. Given the existence of a number of prior approaches to dynamic impact analysis as opposed to a lack of systematic understanding of their performance, the first comprehensive study of the predictive accuracy of dynamic impact analysis was conducted, comparing the performance of representative techniques in this area against various kinds of realized code changes. This paper reflects on the progress in dynamic impact analysis, concerning the impact of that earlier study on later research. We also situate dynamic impact analysis within the current research and practice on impact analysis in general, and envision relevant future research vectors in this area.**

***Index Terms*—Impact analysis, accuracy, potential, vision**

I. Introduction

Impact analysis is widely recognized as a key step during software development. Extensive research has been invested in impact analysis, mostly focusing on techniques such as novel algorithms (e.g., [1]–[4]) with fewer targeting in-depth studies such as comparative evaluations (e.g., [5]). In this context, a systematic methodology to comprehensively evaluate the predictive accuracy of impact analysis was designed, which predicts the set of code locations (referred to as *impact set*) that are potentially impacted by (any) candidate changes at given change locations (i.e., *query*), with respect to the actual impact of particular, concrete code changes (at the query locations).

The study published in 2015 [6], with a focus on dynamic impact analysis [7] working at method level against the most cost-effective dynamic impact analysis techniques [1], [4] back then, revealed and examined the insufficient performance of these techniques. Accordingly, it proposed several future lines of research that may address the insufficiency (i.e., improving the accuracy of dynamic impact analysis). It also discussed possible avenues of applying the dynamic impact analysis techniques to develop cost-effective techniques and tools serving various software evolution and maintenance tasks. Concerning the work itself, it further identified different ways to expand the comparative study for distilling further insights into the full landscape of dynamic impact analysis regarding its potentials and improvement strategies.

In this reflection paper, we look back at the original contributions of the study (Section II), discuss its connections to later research in the area (Section III) while positioning it in current relevant research and practice (Section IV), and elucidate the potentials of the methodology and results while looking forward to new ideas and future impact (Section V).

II. Original Contributions

A. Problem and Motivation

While prior impact analyses are known to be imprecise due to their conservative nature [5], [7], the imprecision was identified mostly through empirical evaluations using *relative precision* measures only (e.g., [1], [5], [8]). Very few other analyses had their results assessed against ground-truth impact of realized changes, yet they either used only trivial experimental subjects [9], [10] or relied on the knowledge about actual changes *before* the analysis itself can be applied [11]–[13]. Also, impact-analysis evaluations are mostly concerned about precision only, assuming the safety (perfect recall) of their impact sets as a result of the conservative nature of the analyses [1]. An exception is the study of both precision and recall using ground truth derived from repository changes [14], where the technique examined is a class-level static analysis based on call-graph reachability, whose results are thus excessively coarse.

Since developers need to apply impact analysis for understanding the effects of potential changes, it is clearly important to know how accurate the analysis results are expected to be. However, as yet **an empirical assessment that systematically measures the accuracy (precision and recall) of change-impact prediction was missing** [7]. To fill this gap, an experimental approach was designed to systematically study the predictive accuracy of existing dynamic impact analyses that represented the state of the art back in 2015.

B. Technical Contributions

A main design challenge to the study was to involve as many potential change locations as possible so that the accuracy metrics can be representative enough for reaching a conclusive assessment of the techniques studied. There were varied sources of changes for such a study, including real changes made by developers in active open-source software repositories and mechanic faults designed by other researchers. Yet the size and availability of these changes are limited and usually cover only a limited portion of the software. Using sensitivity analysis [15] along with efficient instrumentation, the approach **dynamically generates a large**

number of changed program versions by injecting random modifications to *all* applicable code locations to complement the other types of changes.

Another key element of the study methodology was to **create the ground truth for each impact set M computed by the technique under study**. Given a potential change location C, finding all program entities that are semantically dependent [16] on C can give such ground truth. While computing the full set of semantic dependencies is undecidable, for the experiment purpose, using the semantic dependencies with respect to the same set of executions as used by the evaluated impact analysis for producing M can be justifiably utilized as the ground truth against M. To compute the semantic dependencies, execution differencing [11], [12] was used between the program versions before and after actual changes are applied.

A framework realizing this methodology was implemented, and used for assessing two representative contemporary techniques (PI/EAS [1], [4]) on eight Java projects, of which most are real-world systems, and a large scale and multiple types of concrete code changes. **The framework can be utilized for assessing the predictive accuracy of other types of dynamic impact analysis as well** (e.g., non-dependence-based analysis). Moreover, the framework's component used for dynamically generating large sets of program versions can have even broader range of applications (e.g., mutation analysis).

C. Main Empirical Findings

The empirical results confirmed the imprecision of studied techniques while also disclosing the low recall with respect to the real impact based on semantic dependencies. Specifically, the results for both artificial (automatically generated) and real (repository) changes showed that **the most cost-effective dynamic impact analysis known by then was surprisingly inaccurate** with an average precision of 38-50% and average recall of 50-56% in most cases. In comparison, **the accuracy of the studied techniques was generally lower against the real changes than against artificial changes**; when applied to fewer subjects and only artificial changes, these techniques had 47-52% and 56-87% recall according to a preliminary version of this study [17]. The in-depth investigation further suggested that methods in the resulting impact sets were more likely to be *actually* impacted when they executed sooner after the execution of the changed locations. In all, this comprehensive study offered insights on the effectiveness of the most cost-effective dynamic impact analyses back then. Thus, we expected the results to **motivate future development of more accurate dynamic impact analyses**.

III. Overview of Citing Research

After the publication of the original study, later research that cited the paper can be categorized into two classes: (1) *relevant research*—those that referred to it as a representative dynamic impact analysis technique when proposing techniques immediately on or closely related to impact analysis but did not attempt to improve the accuracy of impact analysis, and (2) *improving research*—those that were motivated to actually develop a more precise impact analysis technique.

A. Relevant Research

In [18], the authors developed a novel change impact analysis solution that combines multiple information retrieval (IR) techniques. Like other impact analysis techniques that are not based on code dependencies, this IR-based approach complements the dependence-based techniques targeted in our comparative study. The paper primarily focused on exploring a different approach to impact analysis by treating code text as bags of words in order to apply IR techniques. The authors **compared our work as one of the state-of-the dependence-based impact analyses**, but only in terms of the technical approach; empirically, the paper used a coupling measure based impact analysis as its baseline to demonstrate its merits. In addition, it highlighted the benefits of combining multiple IR techniques over using individual techniques for impact prediction through the empirically shown (precision and recall) advantages of the former.

While this IR-based technique also works at method level (as did the techniques we studied), it does not fall in the class of dependence-based impact prediction techniques. Thus, it did not attempt to improve the accuracy of dependence-based techniques as we called for in our paper. On the other hand, **our recommendation for incorporating more diverse program information in impact analysis shares the same spirit as this citing research** combining the information produced by multiple (two) IR techniques (i.e., one based on bags of words and another based on neural network). Although not explicitly acknowledged so by the authors, it was possible that our recommendations and study results have influenced in certain ways the technical design of this later research.

The technique proposed by Kabeer [19] predicts impact of software change requests on files as well as effort and duration required through an analogy based reasoning method, representing a distinct approach to impact analysis. Partush and Yahav [20] cited our work in discussing topics on differential program analysis—in our study, the experimentation pipeline includes an approach for producing the ground truth through execution differential analysis. Yet like a few other papers on presenting new impact analysis techniques [21], [22], [22], Kabeer and Partush et al. mainly considered our work as an impact analysis approach based on code dependencies that works by propagating change effects along data/control flows.

B. Improving Research

Gyori et al. [23], [24] aimed to develop a more precise change impact analysis technique that is aware of the semantics of code changes. The key idea is to infer equivalence relations between variables of two program versions (the original version and the changed one) so as to make the dataflow-based impact analysis aware of change semantics, and then the change-semantics-awareness leads to greater precision than prior techniques that do not consider the semantics. The technique differs from our work in multiple ways: it is a *static* impact analysis that *computes the actual impact* of realized changes at *statement level*—the concrete changes are known and given to the analysis, while the techniques we evaluated in our work were both *dynamic* impact analysis that *predicts potential impact* of any possible changes at the given query locations at *method* level—the concrete changes were not known nor available to the analysis.

Despite these differences, **this later research fits technically with our vision** about future work that we discussed in the original journal paper. **First**, this research focuses on precision improvement of static impact analysis. Our study strongly revealed surprising imprecision of dynamic impact analysis, which is generally expected to be precise with respect to the underlying executions analyzed, while static impact analysis is expected to be generally *less* precise than dynamic approaches. **Second**, this research presents a dataflow-based impact analysis, where fine-grained (statement-level) data flow analysis forms a basis of the greater analysis precision than prior approaches that either only produce very-coarse (e.g., class-level) impact sets or do not consider data flow facts (e.g., dependencies). In our study, we also motivated the evaluation to target method-level techniques by the excessive level of imprecision of techniques working at overly coarse levels (e.g., class or package)—meanwhile, we did not address statement-level impact analysis because of efficiency concerns of those techniques and that most prior (representative) techniques worked at method level. **Third**, we identified incorporating fine-grained program dependencies as a major direction for future precision improvement in dynamic impact analysis. On the other hand, this later research did not explicitly cite our work as a motivating study for improving impact analysis precision; instead, they referred to our work as one of the representative previous impact analysis techniques based on detailed data and control flow analysis [24], [25].

In [26], the authors aimed to reduce the rate of false positives in (hence improve the precision of) change-impact prediction, by leveraging data-sharing and calling dependencies among code entities. In addition, the research attempts to understand how varied kinds of dependencies affect the effectiveness of change impact prediction. This citing paper falls exactly in the same category of impact analysis approaches as the techniques we studied in our earlier work—predicting potential, rather than computing actual, impact sets of code changes. Although like the work by Gyori et al., this paper targets a static, instead of dynamic, approach to impact analysis, the proposed technique reasons about actual impact based on code (data/control) dependencies. The main rationale behind its precision improvement is to incorporate these program dependencies, **consistent with what we discussed in our paper about the strategies for improving dynamic impact prediction accuracy**. In particular, we pointed out that the low precision of the studied techniques (PI/EAS) was very likely to be the result of insufficient use of program dependence information—PI/EAS only considers (method-level) control flows and computes impact sets based on execution order between methods. This citing research found that data dependencies are complementary to calling (control) dependencies in improving the precision of impact prediction.

Our earlier suggestion on incorporating more program dependence information is probably echoed most strongly in the later research conducted by Malhotra et al. [27]. The authors proposed to combine multiple types of dependencies—symbol dependencies, temporal dependencies, and include dependencies—in addition to the conventional types of (i.e., data and control) dependencies in order to improve the accuracy of their static impact prediction. They further demonstrated the significant contribution of the three newly introduced classes of dependencies to the precision improvement. Like the other two accuracy-improving research, this paper also cited our work as a different type of prior approach to impact analysis, instead of explicitly using our study results to motivate their effort for improving the accuracy. A plausible reason is that all these three improving techniques are static approaches, while our study reported the precision insufficiency with dynamic impact analysis.

IV. Positioning in Current Research and Practice

More recent research on impact analysis has explored diverse approaches beyond dependence-based solutions in terms of the analysis scope, (non-code) information incorporated, and (non-dependence-based) techniques utilized. For instance, the analysis in [28] computes impact sets that apply to system configurations in software product lines—the impact considered is not that on code. For another example, the technique in [29] focuses on change impact at design time, concerning the effects of proposed modifications to ERP entity dependencies on system design and operations of ERP software. In [30], the authors presented an analysis that addresses the impact of changes to software requirements, rather than the impact of changes in code, using natural language processing (NLP) methods. The study on impact analysis presented in [31] targets the consequences of changes in the body of safety evidence (i.e., artifacts concerning the safe operations of software in a given environment) instead of code changes. Using a recommendation system based on IR and repository mining techniques, the approach in [28] also targets non-code artifacts when computing impact sets.

Thus, **in the holistic spectrum of impact analysis, the techniques studied in our work represent one particular class** that (1) infers/predicts potential impact of candidate changes based on code dependencies and (2) applies for understanding the consequences of changes *in code* while concerning only the consequences *on code* as well. In fact, even within the scope of *code-based* impact analysis [7], a number of techniques other than dynamic impact *prediction* have been proposed. For example, descriptive impact analysis (e.g., [11], [13]) addresses the impact of realized changes between two program versions, as opposed to the techniques we studied being predictive. Traceability-based approaches, as opposed to dependence-based solutions, utilize various forms of software artifacts, ranging from requirements and specifications to design documents and source code, to compute change impact by defining and tracing the relationships among them. Most of the recently advanced change impact analyses, including the ones discussed above, belong to this traceability-based category.

Yet **dependence-based impact analysis still represents a major type of impact analysis approaches**, especially when the scope of analysis concerns the *code* artifacts. In contrast to descriptive and non-dependence-based techniques, predictive dependence-based impact analysis such as PI/EAS essentially models the influences among code entities, despite concrete changes at query locations. Thus, improving the accuracy of predictive dependence-based impact analysis generally resorts to the improvement in the accuracy of the underlying dependence analysis. Accordingly, studying the predictive accuracy of dynamic impact prediction as we did essentially examines the accuracy of dynamic dependence analysis with respect to how the dependencies (impact sets) guide developers' change decisions rather than to the ground-truth syntactic dependencies among code entities at runtime.

Concerning practical adoption, however, industrial case studies in the literature seem to have suggested preferences for static approaches within the category of code-based impact analysis techniques [32], [33], and more generally for non-code-based approaches [28], [29], [31], [34], over dynamic dependence-based impact analysis. While answering why this has been the case would need a dedicated study comparing code-based versus non-code-based approaches as well as dependence-based versus non-dependence-based approaches, there are three plausible reasons to consider.

First, non-code-based approaches usually directly address change requests (typically in natural language) with respect to software specifications by considering diverse kinds of (non-code) artifacts (e.g., requirements, design documents, repository commits, code comments, etc.). In contrast, code-based approaches might be too narrow in terms of the information sources considered (i.e., only the code as the single type of artifact). **Second**, dependence-based approaches further rely on the accuracy of underlying dependence analysis to be practically accurate, yet for real-world, large code bases, there tends to be a huge amount of dependencies to sort out—even latest dependence-based impact prediction techniques can still produce too many potentially impacted code entities to be affordably inspected by developers. **Finally**, dynamic approaches to dependence-based impact analysis appeared to be even less adopted in practice—there have not been published studies reporting the use of such approaches in industry, probably because these techniques compute potentially impacted code entities only with respect to specific program executions. Indeed, **dynamic impact analysis tends to be more applicable in the context of its application/client analyses** than directly for making change decisions in general. For example, it could have been more often (albeit implicitly) used as an underlying technique for testing (e.g., for regression test selection and prioritization) and debugging (e.g., for iteratively applying and validating bug-fixing changes with respect to the bug-revealing execution).

V. Vision and Impact

In retrospect, during the three years after it was published, our study was moderately referred to by later research on impact analysis. Importantly, **the aim of the following work was closely connected to (although not explicitly acknowledged as motivated by) the future work we discussed in our paper**. Meanwhile, the citing research generally all considered our work as a technical approach to impact analysis, **rather than directly referring to our empirical findings**, although our paper mainly addressed the comparative evaluation of prior analysis techniques instead of proposing a new technique for dynamic impact analysis.

We also noted that **there has not been later work exactly developing more precise *dynamic* impact analysis or conducting comparative studies of impact analysis techniques**, except for our follow-up work on improving the cost-effectiveness of static [35], [36] and dynamic [37]–[40] impact prediction by utilizing diverse program information. One plausible explanation is that dependence-based dynamic impact analysis has been studied quite extensively with various approaches already explored. Another reason, as mentioned earlier, is that significant advancement in this area immediately requires that in the underlying dependence analysis. However, precise dynamic dependence analysis is well-known as hard to scale to large, complex software systems, while achieving user-perceived (i.e., from users' perspective in particular use scenarios) desirable precision remains a grand challenge.

Nevertheless, we believe that **our work has unexploited potentials**. **First**, given the nature of the studied techniques, our evaluation methodology essentially assesses the user-perceived accuracy of forward dynamic dependencies (at method level). Thus, it is more broadly applicable to techniques based on forward dynamic dependence analysis, than just to dynamic impact prediction techniques in particular. In fact, we applied the same methodology for evaluating the accuracy of forward dynamic slicing at statement level [41]. **Second**, our experiment framework features an automated pipeline of large-scale runtime program variant generation which simulates realistic code changes by developers, as well as an automatic mechanism that computes dynamic change impact using the impact analysis technique under assessment. Flexible interfaces are provided to allow for plugging in any impact prediction tool to be evaluated. The framework automates results presentation and statistics computation as well. Thus, this framework provides immediate utilities for conducting similar other studies. **Finally**, our research explored for the first time the long-standing problem about the trustworthiness of program dependence analysis, a technique that has been widely utilized for many software engineering tasks. Importantly, our study revealed the gap between the accuracy of syntactical dependencies and that perceived by users (developers) in specific task scenarios, and provided the first empirical evidence about the significance of this gap.

Further, according to our results, dynamic impact analysis is not a problem that has really been solved, and there are several **emerging ideas that may help advance this area**. **First**, developers' knowledge can be exploited to improve user-perceived precision. In particular, the underlying dependence analysis would be more precise to users if the resulting dependencies are labeled with inspection priorities computed with user feedback incorporated through machine learning (e.g., reinforcement learning to use the order that dependencies were actually explored by users for improving future ranking of dependencies). This direction may lead to improvements in user-perceived precision of impact prediction based on such prioritized dependencies. **Second**, to make dynamic impact analysis more attractive to practitioners, we may offer a more friendly environment for using the technique and demonstrate the value of the improvement in the analysis itself (e.g., the precision) in the context of its applications. For instance, the greater user-perceived precision would lead to greater savings in time and effort for regression testing and iterative debugging. **Third**, the resulting impact sets can be utilized to guide the incremental approach to various analysis tasks—given the analysis results of one program version, just reanalyzing the changed and impacted entities when doing the analysis of the changed version of the program.

VI. Acknowledgments

We thank the reviewers for constructive comments. This work is supported in part by NSF grant CCF-1936522.

References

[1] A. Orso, T. Apiwattanapong, and M. J. Harrold, "Leveraging field data for impact analysis and regression testing," in *Proceedings of ESEC/FSE*, 2003, pp. 128–137.

[2] J. Law and G. Rothermel, "Whole program path-based dynamic impact analysis," in *Int'l Conf. on Softw. Engg.*, 2003, pp. 308–318.

[3] P. Tonella, "Using a concept lattice of decomposition slices for program understanding and impact analysis," *Software Engineering, IEEE Transactions on*, vol. 29, no. 6, pp. 495–509, 2003.

[4] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in *Int'l Conf. on Softw. Engg.*, 2005, pp. 432–441.

[5] A. Orso, T. Apiwattanapong, J. B. Law, G. Rothermel, and M. J. Harrold, "An empirical comparison of dynamic impact analysis algorithms," in *Int'l Conf. on Softw. Eng.*, 2004, pp. 491–500.

[6] H. Cai and R. Santelices, "A comprehensive study of the predictive accuracy of dynamic change-impact analysis," *Journal of Systems and Software*, vol. 103, pp. 248–265, 2015.

[7] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

[8] L. Huang and Y.-T. Song, "A dynamic impact analysis approach for object-oriented programs," *Advanced Software Engineering and Its Applications*, vol. 0, pp. 217–220, 2008.

[9] ——, "Precise dynamic impact analysis with dependency analysis for object-oriented programs," in *Int'l Conf. on Softw. Engg. Res., Manag. & Apps*, 2007, pp. 374–384.

[10] M. C. O. Maia, R. A. Bittencourt, J. C. A. de Figueiredo, and D. D. S. Guerrero, "The hybrid technique for object-oriented software change impact analysis," in *Euro. Conf. on Software Maintenance and Reengineering*, 2010, pp. 252–255.

[11] M. K. Ramanathan, A. Grama, and S. Jagannathan, "Sieve: A tool for automatically detecting variations across program versions," in *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, 2006, pp. 241–252.

[12] R. Santelices, M. J. Harrold, and A. Orso, "Precisely detecting runtime change interactions for evolving software," in *Int'l Conf. on Softw. Testing, Verification and Validation*, Apr. 2010, pp. 429–438.

[13] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *ACM Conf. on Obj. Oriented Prog. Syst., Lang., and Appl.*, 2004, pp. 432–448.

[14] L. Hattori, D. Guerrero, J. Figueiredo, J. Brunet, and J. Damasio, "On the precision and accuracy of impact analysis techniques," in *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science*, 2008, pp. 513–518.

[15] H. Cai, S. Jiang, Y.-j. Zhang, Y. Zhang, and R. Santelices, "Sensa: Sensitivity analysis for quantitative change-impact prediction," in *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. IEEE, 2014, pp. 165–174.

[16] A. Podgurski and L. Clarke, "A formal model of program dependences and its implications for software testing, debugging, and maintenance," *TSE*, vol. 16, no. 9, pp. 965–979, 1990.

[17] H. Cai, R. Santelices, and T. Xu, "Estimating the accuracy of dynamic change-impact analysis using sensitivity analysis," in *International Conference on Software Security and Reliability*, 2014, pp. 48–57.

[18] W. Wang, Y. He, T. Li, J. Zhu, and J. Liu, "An integrated model for information retrieval based change impact analysis," *Scientific Programming*, vol. 2018, 2018.

[19] S. J. Kabeer, "An analogy based technique for predicting the vector impact of software change requests," Master's thesis, 2018.

[20] N. Partush and E. Yahav, "Differential program analysis," Ph.D. dissertation, Computer Science Department, Technion, 2017.

[21] J. A. Aguilar, C. Tripp, A. Zaldivar, O. Garcia, and C. E. Zurita, "Evaluating a requirements change request in a goal-oriented requirements engineering model," *IEEE Latin America Transactions*, vol. 14, no. 5, pp. 2411–2417, 2016.

[22] M. Basri, "An algorithmic-based software change effort prediction model using change impact analysis for software development," Ph.D. dissertation, Universiti Teknologi Malaysia, 2016.

[23] A. Gyori, S. K. Lahiri, and N. Partush, "Interprocedural semantic change-impact analysis using equivalence relations," *arXiv preprint arXiv:1609.08734*, 2016.

[24] ——, "Refining interprocedural change-impact analysis using equivalence relations," in *Proceedings of ISSTA*, 2017, pp. 318–328.

[25] A. Gyori, "Proactively detecting unreliable tests," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2017.

[26] X. Liu, L. Huang, A. Egyed, and J. Ge, "Do code data sharing dependencies support an early prediction of software actual change impact set?" *Journal of Software: Evolution and Process*, vol. 30, no. 11, p. e1960, 2018.

[27] M. Malhotra and J. K. Chhabra, "Improved computation of change impact analysis in software using all applicable dependencies," in *International Conference on Futuristic Trends in Network and Communication Technologies*. Springer, 2018, pp. 367–381.

[28] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Change impact analysis for evolving configuration decisions in product line use case models," *Journal of Systems and Software*, vol. 139, pp. 211–237, 2018.

[29] M. Parhizkar and M. Comuzzi, "Impact analysis of erp post-implementation modifications: Design, tool support and evaluation," *Computers in Industry*, vol. 84, pp. 25–38, 2017.

[30] C. Arora, M. Sabetzadeh, A. Goknil, L. C. Briand, and F. Zimmer, "Change impact analysis for natural language requirements: An nlp approach," in *2015 IEEE 23rd International Requirements Engineering Conference (RE)*. IEEE, 2015, pp. 6–15.

[31] J. L. de la Vara, M. Borg, K. Wnuk, and L. Moonen, "An industrial survey of safety evidence change impact analysis practice," *TSE*, vol. 42, no. 12, pp. 1095–1117, 2016.

[32] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *ICSE*, 2011, pp. 746–765.

[33] Y. Tao, Y. Dang, T. Xie, D. Zhang, and S. Kim, "How do software engineers understand code changes?: an exploratory study in industry," in *FSE*, 2012, pp. 51:1–51:11.

[34] M. Borg, K. Wnuk, B. Regnell, and P. Runeson, "Supporting change impact analysis using a recommendation system: An industrial case study in a safety-critical context," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 675–700, 2016.

[35] H. Cai and R. Santelices, "Abstracting program dependencies using the method dependence graph," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 49–58.

[36] ——, "Method-level program dependence abstraction and its application to impact analysis," *Journal of Systems and Software*, vol. 122, pp. 311–326, 2016.

[37] ——, "A framework for cost-effective dependence-based dynamic impact analysis," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 2015, pp. 231–240.

[38] H. Cai, R. Santelices, and D. Thain, "Diapro: Unifying dynamic impact analyses for improved and variable cost-effectiveness," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 25, no. 2, p. 18, 2016.

[39] H. Cai, R. Santelices, and S. Jiang, "Prioritizing change-impact analysis via semantic program-dependence quantification," *IEEE Transactions on Reliability*, vol. 65, no. 3, pp. 1114–1132, 2016.

[40] H. Cai, "Hybrid program dependence approximation for effective dynamic impact prediction," *IEEE Transactions on Software Engineering (TSE)*, vol. 44, no. 4, pp. 334–364, 2018.

[41] S. Jiang, R. Santelices, M. Grechanik, and H. Cai, "On the accuracy of forward dynamic slicing and its effects on software maintenance," in *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, 2014, pp. 145–154.