SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. (2012)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/stvr.1475

A survey of code-based change impact analysis techniques

Bixin Li [1,2,*,†], Xiaobing Sun [1,2], Hareton Leung [3] and Sai Zhang [4]

[1] *School of Computer Science and Engineering, Southeast University, Nanjing, China*
[2] *State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[3] *Department of Computing, Hong Kong Polytechnic University, Hong Kong, China*
[4] *Computer Science and Engineering Department, University of Washington, Seattle, WA, USA*

SUMMARY

Software change impact analysis (CIA) is a technique for identifying the effects of a change, or estimating what needs to be modified to accomplish a change. Since the 1980s, there have been many investigations on CIA, especially for code-based CIA techniques. However, there have been very few surveys on this topic. This article tries to fill this gap. And 30 papers that provide empirical evaluation on 23 code-based CIA techniques are identified. Then, data was synthesized against four research questions. The study presents a comparative framework including seven properties, which characterize the CIA techniques, and identifies key applications of CIA techniques in software maintenance. In addition, the need for further research is also presented in the following areas: evaluating existing CIA techniques and proposing new CIA techniques under the proposed framework, developing more mature tools to support CIA, comparing current CIA techniques empirically with unified metrics and common benchmarks, and applying the CIA more extensively and effectively in the software maintenance phase.

Received 24 January 2011; Revised 23 February 2012; Accepted 29 March 2012

KEY WORDS: change impact analysis; survey; source code; application

1. INTRODUCTION

Software maintenance has been recognized as the most difficult, costly and labour-intensive activity in the software development life cycle [1, 2]. Software products are naturally adapted and changed with the changing system requirements to meet user's needs. Software change is a fundamental ingredient of software maintenance. Lehman and Belady proposed and refined five laws that characterize the dynamics of programme evolution, in which the first law is—*change is continual* [3]. Changes can be stemmed from new requirements, fixing faults, change requests and so on. When changes are made to software, they will inevitably have some unexpected effects and may cause inconsistencies with other parts of the original software. Software *change impact analysis* (CIA) involves a collection of techniques for determining the effects of the proposed changes on other parts of the software [4]. It plays an important role in software development, maintenance, and regression testing [4–7]. CIA can be used before or after a change implementation. Before making changes, CIA can be employed for programme understanding, change impact prediction and cost estimation [4, 5]. After changes have been implemented, CIA can be applied to trace ripple effects, select test cases and perform change propagation [6–9].

*Correspondence to: Bixin Li, School of Computer Science and Engineering, Southeast University, Nanjing, China.
†E-mail: bx.li@seu.edu.cn

1.1. Background

In the last 30 years, software engineering researchers have proposed several definitions of CIA and developed many techniques for CIA. In the early 1980s, the work on impact analysis focused on the so-called *ripple effect* [10, 11]. The first definition of impact analysis was proposed by Horowitz *et al.* in 1986 as 'an examination of an impact to determine its parts or elements' [12]. In 1990, Pfleeger *et al.* presented a model of software maintenance, which took impact analysis as one of its key activities [13]. Then in 1991, Pfleeger *et al.* defined impact analysis as 'the evaluation of the many risks associated with the change, including estimates of the effects on resources, effort, and the schedule' [14]. Until now, there is no definition of software change impact analysis in the IEEE Standard Glossary of Software Engineering Terminology [15]. However, impact analysis as a necessary activity of the software maintenance process is explained in the IEEE Standard for Software Maintenance [16]. And in 1996, Bohner *et al.* defined CIA as 'the process of identifying the potential consequences of a change, or estimate what needs to be modified to accomplish a change' in the book *Software Change Impact Analysis* [4]. Since then, this definition has become most frequently used and widely recognized.

1.2. Scope

There have been a wide range of research work on CIA techniques and its applications. Some CIA methods are based on the traceability examination (traceability-based CIA) whereas others concentrate on dependence relationships (dependence-based CIA) to determine the change effects. Researches that rely on traceability analysis try to trace dependence between elements within one level of abstraction with the corresponding items at different levels (i.e. requirements, design documents, source code and test cases) [4, 17, 18]. Its goal is to connect different types of software artefacts (e.g. requirement or design documentation with source code). As De Lucia *et al.* have presented a survey of this type of work in 2008 [18], this survey does not include traceability-based CIA techniques. Another reason for excluding traceability-based CIA techniques is that traceability-based CIA seeks to recover the links between different types of software artefacts whereas dependence-based CIA attempts to estimate the potential change effects of the proposed change.

Dependence-based analysis usually attempts to analyze programme syntax relations, which signify some semantic dependencies among programme entities [19, 20]. Broadly speaking, it performs impact analysis of software artefacts at the same level of abstraction (design to design or code to code). In recent years, most of the dependence-based CIA techniques focus on the source code level whereas a few work on requirement and design level [21–24]. Requirement and design level CIA rely on the abstract models of high level (for example, unified modeling language diagrams [25] or use case maps [26]) to perform CIA. These techniques do not need to access the source code, and they are far from mature when compared with source code-based CIA techniques. So, CIA on these high levels are also not included in the survey. Source code-based CIA techniques have the advantage of identifying the change impact in the final product—program source code. They can improve the precision of the impact results as they directly analyze implementation details. And most of the efforts on CIA focuse on the source code level [27–30], and these techniques are becoming better and better.

Hence, this article focuses on presenting a survey of code-based CIA techniques. It is aimed to define and organize this research area. The survey only covers recent work published between 1997 and 2010, as works before 1997 can refer to Bohner *et al.* [4] and Turver *et al.* [7].

1.3. Motivation

Change impact analysis has become an increasingly popular research area. A rich body of knowledge of different CIA techniques have continuously emerged in recent years. It is exciting to see a multitude of recent work, including CIA methodologies, CIA supporting tools and CIA applications. However, it is difficult for researchers or practitioners to decide which technique is most appropriate

DOI: 10.1002/stvr

for their needs, or how different CIA techniques can be integrated to suit their purposes. Unfortunately, there have been very few works focusing on a comprehensive and systematic framework to comprehend, compare and evaluate existing CIA techniques.

A few works have tried to address this issue by presenting some comparative studies to characterize the effectiveness of different CIA techniques. However, these studies focus on only a limited number of CIA techniques, and none of them is comprehensive and systematic. In 1993, Bohner *et al.* proposed a definition of impact analysis that clearly delineates the basis for comparing impact analysis capabilities [31]. They presented a comparative model based on applications of impact analysis, in which, some are used to perform impact analysis and others are used to measure the effectiveness of these techniques. A few CIA techniques are categorized and compared according to this framework. In 1996, they gave a summarization of CIA techniques and related issues in a book [4]. This book presented the strengths and limitations of various CIA techniques and reported CIA techniques from the perspective of source code dependence analysis and software traceability analysis.

In addition, there are some studies focusing on experimental comparison of a limited number of CIA techniques [32–34]. However, none of the proposed frameworks have been used to characterize a comprehensive view to capture the different dimensions of the CIA techniques. Hence, a unified framework should be developed to support a critical review of all existing CIA techniques. This framework can be used as follows:

1. to identify key properties of CIA technique,
2. to facilitate comparison of CIA techniques,
3. to guide development of new CIA techniques, and
4. to help practitioners select appropriate CIA techniques according to their needs.

The survey may also serve as a roadmap for this important research area by providing a taxonomy, a detailed comparison of existing CIA techniques, as well as review of their key applications in software maintenance.

1.4. Article organization

This article is organized as follows: it starts by presenting some related definitions and terminologies to facilitate comprehension of CIA area. In Section 3, the survey method used for the study is described. Section 4 presents the selected articles on 23 code-based CIA techniques. Section 5 provides a unified comparative framework for code-based CIA techniques and followed by the discussion of its applications in Section 6. In Section 7, some applications of the CIA techniques are introduced in software maintenance. Section 8 discusses the implications for future CIA research. Finally, conclusion is given in Section 9.

2. CHANGE IMPACT ANALYSIS

A change made to a software system may result in an undesirable *side effect* and/or *ripple effect* [4]. The goal of CIA is to identify the ripple effects and prevent side effects of a proposed change. A side effect occurs when changes to the software cause it to reach an erroneous state. The concept of *ripple effect* is often used to measure the likelihood that a change to a particular module may cause problems in the rest of the program [8, 35, 36]. Figure 1 shows the whole CIA process [18, 37]. CIA starts by analyzing the change request and the source code to identify the change set in which the elements could be affected by the change request. Then through the change impact analysis technique, other elements that are probably affected by the elements in the change set are estimated. The resulting set is called estimated impact set (EIS). Once the change is implemented to accomplish the change request, the elements in the actual impact set (AIS) are actually modified. In practice, the AIS is not unique with respect to a change request because a change may be implemented in several ways.

DOI: 10.1002/stvr

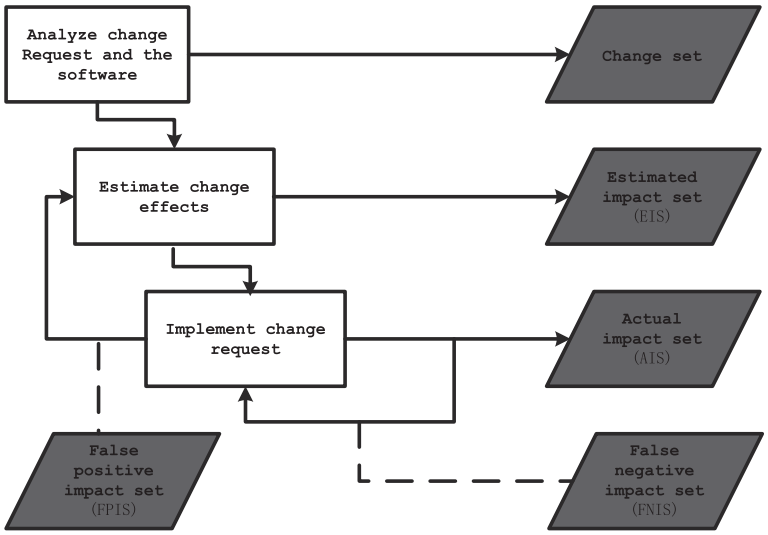

Figure 1. Change impact analysis process.

As shown in Figure 1, the software change impact analysis process is an iterative process. And when a change is implemented, some other impacted elements not in the EIS may be discovered. The set of such elements is called the false negative impact set (FNIS), which represents an under-estimate of impacts [18, 37]. On the other hand, EIS usually includes some elements that do not really need to be modified or reworked, and this set is called the false positive impact set (FPIS), which represents the over-estimate of impacts in the analysis [18, 37]. There is a relationship between these four sets, that is, the union of EIS and FNIS minus the FPIS should result in the AIS.

The goal of the CIA process is to estimate an EIS that is as close to the AIS as possible. Several metrics can be defined to evaluate the accuracy of the impact analysis process [31, 33]. Among the metrics for accuracy of CIA techniques, precision and recall are two widely used metrics [33, 38–40]. They were traditionally used in an information retrieval scenario [41]. In the CIA scenario, precision and recall are defined as follows:

$$Precision = \frac{|EIS \cap AIS|}{|EIS|}$$

$$Recall = \frac{|EIS \cap AIS|}{|AIS|}$$

Precision measures the degree the estimated impacts accord with the real impacts induced by changes whereas recall measures the degree the EIS cover the real changes. With a high precision EIS, maintainers will spend less time on locating and implementing the changes. With a high recall EIS, maintainers are confident that impacts of those proposed changes will be all considered.

2.1. Identifying the change set

The first step of the CIA process is to determine the change set that requires the analysis of the change request specification and both the source code and the software documentation [18]. This step is also called concept location or feature location, which is the activity of identifying an initial location in the source code that implements functionality in a software system [42–44]. Programmers use feature location to find where in the source code the initial change needs to be made. The full extent of the change is then handled by impact analysis, which starts with the source code identified by feature location and finds all the code affected by the change [43]. Different techniques for the identification of concepts or features in the source code have been proposed, and a survey of the concept location techniques can refer to Dit *et al.* [43]. And the articles and

DOI: 10.1002/stvr

research from such related field are not included in this article as they are beyond the scope of this focused survey.

2.2. Estimating the change effect

The second step is the estimation of the change effect induced by the change set. Indeed, impact analysis is necessary during software maintenance for the fact that even a small change in a software system may affect many other parts of the system, thus causing ripple effects [18]. Thus, to accurately estimate the possible effects of the proposed software changes is crucial to the CIA. This step is the focus of most current CIA techniques. The CIA techniques surveyed in this article mainly focus on estimating the change effect from the change set.

3. SURVEY METHOD

This section describes the process to conduct the survey. The process follows the systematic review approach as proposed by Kitchenham [45, 46].

Specifying the right research question(s) is very crucial to a systematic review to obtain detailed and relevant findings of a given domain and to help identify appropriate future research activities [45, 46]. As discussed earlier, this survey aims at defining a framework to summarize and clarify understanding of the current state of the art in change impact analysis research and providing recommendations for future work. Specifically, the following research questions (RQ) are addressed,

(RQ1) What techniques/approaches have been used to perform the change impact analysis?
(RQ2) Which properties can be identified to characterize the research on code-based CIA?
(RQ3) What are the key application areas of CIA?
(RQ4) What are the fruitful areas of future research?

To answer these research questions, the following process is used to conduct the systematic review of the literature:

1. Literature search. A wide range of articles is searched in electronic databases using the title or key words indexing;
2. Study selection. Some inclusion and exclusion criteria are defined on the initial set of articles to select only relevant articles;
3. Data extraction and analysis. Useful information from the final set of selected articles is extracted and analyzed. The research questions listed previously are then answered.

3.1. Literature search

3.1.1. Source. To obtain a broad perspective, the search is first performed widely in electronic databases instead of a limited set of journals, conferences and workshop proceeding as motivated by Dieste *et al.* [47]. The source for the search includes the following four digital libraries:

- ACM Digital Library (http://portal.acm.org/)
- IEEE Computer Society Digital Library (http://ieeexplore.ieee.org/)
- Science@Direct (http://www.sciencedirect.com/)
- Springer Link (http://www.springerlink.com)

These libraries are chosen because they cover the most relevant sources in software engineering [48]. In this survey, grey literature, such as technical reports and work in progress were excluded from the search source for the reason that their quality cannot be guaranteed. In addition, the initial search resulted in overlap among many papers. The duplicates were excluded primarily by manual filtering.

3.1.2. Search criteria. The initial search keywords used were broad in order to cover as many papers with different uses of terminology as possible. The initial set of key words included

DOI: 10.1002/stvr

<impact analysis> and (<software> or <program>). During this process, the 'title', 'keyword', or 'abstract' field is used for indexing. That is, the title, keyword, or abstract of the paper including these key words are included in the search results. After a preliminary search, some of the other key words derived from the concepts of impact analysis were added, such as 'ripple effect' and 'change propagation'. During the search process, the database fields of title, keyword and abstract were searched, and as explained earlier, the start year was 1997 whereas the last date was 2010. And only papers published in English were included. The initial search found 2357 potentially relevant papers.

3.2. Study selection

To ensure high quality results, the paper selection process was divided into two stages, as depicted in Figure 2. The first stage (initial study selection) was based on only the titles and abstracts, and the second stage (final study selection) is based on the full text of the research papers. During the process, three researchers were involved. All the papers were firstly assessed by two of the researchers independently. In case of disagreement, the third researcher acted as a checker, and a discussion among them was conducted.

In the initial selection stage, the exclusion strategy was performed, that is, the duplicates and irrelevant papers from the initial set of 2357 papers were removed. The exclusion criteria used include the following:

1. The research focused on some other problems rather than the CIA technique; and
2. The research focused on the traceability-based analysis techniques or high level model-based (design level and requirement level CIA) rather than the code-based CIA technique.

Of the 2357 papers, some papers may mention the CIA technique or CIA process, but their main focus was on CIA application (e.g. fault localization, regression testing, etc. [9, 49–52]) or the presented techniques to support CIA (program slicing, information flow analysis, etc. [53, 54]). These papers were removed according to the first criterion. In addition, some of the papers presented the CIA technique at a high abstract level, such as requirement or specification level, or design level [21, 21, 23, 24]. These papers were also filtered. Finally, 78 papers were left.

In the final study selection phase, a full text analysis was conducted on the remaining 78 papers. During this process, a detailed analysis on the full text of these papers was performed. This process was also conducted by the previous three researchers in the same way. For this step, the inclusion strategy was used. The inclusion criteria aimed at providing answers to the research questions and included one of the following conditions:

1. The research focused on a specific CIA technique, stated the change impact analysis to be one of its goals and provided empirical validation of the CIA technique from the change impact analysis perspective;
2. The research provided empirical comparison of some CIA techniques.

On the one hand, the first inclusion criterion shows that a paper exhibits a profound relation to CIA. That is, the paper must state CIA to be a goal; furthermore, some experimental evaluation must be performed and the evaluation must demonstrate the purpose of the approach from the CIA perspective. The paper fitted to this criterion was selected. On the other hand, some paper focused on empirical comparison of several CIA techniques, which can provide the advantages and

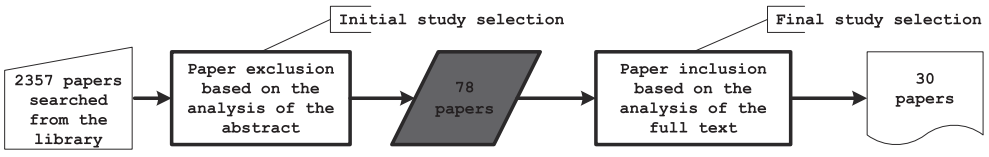

Figure 2. Overview of the study selection process.

DOI: 10.1002/stvr

disadvantages of a CIA technique over another. This type of paper was also selected in the final study. After the final study selection phase, a total of 30 papers were selected. Figure 3 shows the distribution of the selected papers by years and across different publication venues. And Table I shows the publication venues of selected papers.

3.3. Data extraction

From the selected papers, some basic information is extracted:

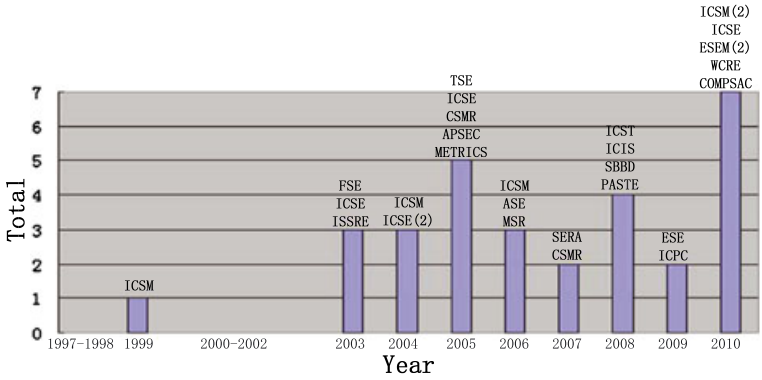

Figure 3. Distribution of the selected papers in different years and across different venues.

Table I. Selected papers across different venues.

Type	Acronym	Description	Total
Conference	ICSM	International Conference on Software Maintenance	5
	ICSE	International Conference on Software Engineering	5
	CSMR	European Conference on Software Maintenance and Reengineering	2
	FSE	ACM SIGSOFT International Symposium on Foundations of Software Engineering	1
	ASE	International Conference on Automated Software Engineering	1
	WCRE	Working Conference on Reverse Engineering	1
	PASTE	ACM Workshop on Program Analysis for Software Tools and Engineering	1
	APSEC	Asia-Pacific Software Engineering Conference	1
	ESEM	International Symposium on Empirical Software Engineering and Measurement	2
	ISSRE	International Symposium on Software Reliability Engineering	1
	MSR	International Workshop on Mining Software Repositories	1
	ICST	International Conference on Software Testing, Verification, and Validation	1
	ICPC	International Conference on Program Comprehension	1
	COMPSAC	International Conference on Computer Software and Applications	1
	ICIS	International Conference on Computer and Information Science	1
	SBBD	Brazilian Symposium on Databases	1
	SERA	International Conference on Software Engineering Research, Management and Applications	1
	METRICS	International Software Metrics Symposium	1
Journal	ESE	Empirical Software Engineering Journal	1
	TSE	IEEE Transactions on Software Engineering	1

DOI: 10.1002/stvr

- The co-occurrences of various properties of the CIA technique;
- Prototype tool of the CIA technique (if available);
- Empirical validation approach;
- A list of statement relevant to the research questions.

According to the information distilled from these 30 papers,

1. 23 papers present 23 different CIA techniques with empirical studies;
2. two papers cover the same CIA technique;
3. four papers extend previous CIA techniques; and
4. two papers provide comparison of different CIA techniques.

The CIA techniques (T1–T23) are mapped to different publications (P1–P30) as shown in Table II. Table III summarizes the approach of various CIA techniques. During this process, three previous researchers were also involved in the recording of the properties of these 23 different CIA techniques, independently. Then, according to the recorded properties, decision on the finally selected properties to characterize the CIA techniques is discussed and determined.

4. CODE-BASED CHANGE IMPACT ANALYSIS TECHNIQUES

In this section, various approaches of CIA proposed between 1997 and 2010 are reviewed. The CIA techniques that have been empirically evaluated are focused.

From Table III, it shows that many CIA techniques (T6, T8, T10, T15, T17, T21) are based on traditional static program analysis techniques involving the dependency graph. For example, the impact set is computed through reachability analysis on the dependency graph.

Table II. Thirty papers reporting 23 change impact analysis techniques.

Publication	Technique	Reference
P1	T1	Briand *et al.* [27]
P2	T2	Orso *et al.* [28]
P3	T3	Law *et al.* [20]
P4	T4	Zimmermann *et al.* [55]
P5	T5	Apiwattanapong *et al.* [30]
P6	T6	Badri *et al.* [56]
P7	T7	Ramanathan *et al.* [57]
P8	T8	Breech *et al.* [58]
P9	T9	Canfora *et al.* [59]
P10	T10	Huang *et al.* [60]
P11	T11	Beszedes *et al.* [61]
P12	T12	Jashki *et al.* [62]
P13	T13	Hattori *et al.* [63]
P14	T14	Sherriff *et al.* [64]
P15	T15	Hattori *et al.* [38]
P16	T16	Poshyvanyk *et al.* [39]
P17	T17	Petrenko *et al.* [65]
P18	T18	Kagdi *et al.* [66]
P19	T19	Torchiano *et al.* [67]
P20	T20	Ceccarelli *et al.* [68]
P21	T21	Sun *et al.* [69]
P22	T22	Gethers *et al.* [70]
P23	T23	Ahsan *et al.* [71]
P24	T3	Law *et al.* [19]
P25	T3	Breech *et al.* [72]
P26	T4	Zimmermann *et al.* [73]
P27	T2, T3	Orso *et al.* [34]
P28	T2, T3, T5	Breech *et al.* [32]
P29	T9	Canfora *et al.* [74]
P30	T20	Canfora *et al.* [75]

DOI: 10.1002/stvr

Table III. Change impact analysis techniques.

Technique	Reference	Description
T1	Briand *et al.* [27]	Use object oriented coupling measurement to identify the impact set.
T2	Orso *et al.* [28]	Use the coverage information of the field data collected from users to support dynamic CIA.
T3	Law *et al.* [20]	Provide a technique for dynamic CIA based on whole path profiling.
T4	Zimmermann *et al.* [55]	Apply data mining to version histories in order to extract the co-change coupling between the files for CIA.
T5	Apiwattanapong *et al.* [30]	Use the execute-after relation between entities to support dynamic CIA.
T6	Badri *et al.* [56]	Use the control call graph to perform static CIA.
T7	Ramanathan *et al.* [57]	Use dynamic programming on instrumented traces of different programme binaries to compute the impact set.
T8	Breech *et al.* [58]	Analyze influence mechanisms of scoping, function signatures and global variable accesses to support CIA.
T9	Canfora *et al.* [59]	Use textual similarity to retrieve past change request in the software repositories for CIA.
T10	Huang *et al.* [60]	Perform dependency analysis in object oriented programs for CIA.
T11	Beszedes *et al.* [61]	Use the measure of dynamic function coupling between two functions for CIA.
T12	Jashki *et al.* [62]	Create clusters of closely associated software program files in the software repository for CIA.
T13	Hattori *et al.* [63]	Apply two different data mining algorithms $Apriori$ and DAR in the software repository for CIA.
T14	Sherriff *et al.* [64]	Analyze change records through singular value decomposition to produce cluster of co-change files for CIA.
T15	Hattori *et al.* [38]	Use call graph to compute the impact set.
T16	Poshyvanyk *et al.* [39]	Use conceptual coupling measurement for CIA.
T17	Petrenko *et al.* [65]	Use a hierarchical model to interactively compute the impact set.
T18	Kagdi *et al.* [66]	Blend conceptual and evolutionary couplings to support CIA.
T19	Torchiano *et al.* [67]	Use source code comments and changelogs in software repository to support CIA.
T20	Ceccarelli *et al.* [68]	Use multivariate time series analysis and association rules to perform CIA.
T21	Sun *et al.* [69]	Analyze impact mechanisms of different change types for CIA.
T22	Gethers *et al.* [70]	Use relational topic based coupling to capture topics in classes and relationships among them for CIA.
T23	Ahsan *et al.* [71]	Use single and multi-label machine learning classification for CIA.

To improve the precision of the CIA techniques, some researchers propose CIA techniques (T2, T3, T5, T7, T10) based on analysis of the information collected during the program execution, such as execution traces information, coverage information and execution relation information, to compute the impact set.

Recently, there are many CIA techniques (T4, T9, T12–T14, T18–T20, T23) based on mining information from the software repositories. Some evolutionary dependencies between program entities that can not be distilled by the traditional program analysis technique can be mined from these repositories. These CIA techniques are based on identifying the co-change coupling between the files (or program entities) that are changed together in the software repository.

There are also some CIA techniques (T1, T11, T16, T18, T22) performed based on some coupling measurement, such as structural coupling, conceptual coupling, dynamic function coupling and relational topical based coupling. Their impact sets are predicted by analyzing the program elements that are coupled with the changes.

DOI: 10.1002/stvr

There are some attempts in combining several CIA techniques. For example, T18 is based on a combination of coupling measurement and mining software repositories. T10 uses both traditional dependency analysis and the execution information.

In addition, four publications (P24, P25, P29, P30) extend previous CIA techniques, with detail shown in Table IV. P24 and P25, both of which extend the dynamic CIA (T3) as online, show that the time and space cost of the online CIA is smaller than that of the dynamic CIA, but the precision of these CIA techniques is similar. P29 extends T9 at the finer line of code granularity level, and shows that fine grained CIA at the line of code level improves precision at the cost of higher computational time, compared with the coarser file level CIA. And P30 validates the advantages of the hybrid approach that combines ranking of both association rules and Granger together to perform CIA, which is only assumed in P20.

To summarize, these 30 publications on 23 different CIA techniques can be classified into four perspectives: traditional dependency analysis, software repositories mining, coupling measurement and execution information collection. The distribution of these publications from different perspectives in different years is shown in Figure 4. It shows that the most widely studied CIA techniques are based on mining software repositories, and they are receiving increasing attention in recent years. The technique of mining software repositories can uncover useful and important historical dependencies between project artefacts, such as functions, documentation files, or configuration files. Maintainers can use these dependencies to propagate changes to related software artefacts, instead of relying on solely traditional static or dynamic code dependencies, which may fail to capture important dependencies. For example, a change to the code that writes data into a file may require corresponding changes to the code that reads data from this file, although there exists no data and control flow between both pieces of code. Hence, mining software repositories may become a good complement to change impact analysis with traditional static or dynamic CIA techniques.

Table IV. Papers on extending existing change impact analysis techniques.

Technique	Publication	Difference
T3	P3, P24	P24 provides an improved technique to be applied incrementally as a system evolves and avoids the overhead of completely recomputing the information needed for CIA as shown in P3.
T3	P3, P25	P25 presents a completely online (i.e. during program execution) CIA technique, and it avoids storage and postmortem analysis of program traces, even compressed, as shown in P3.
T9	P9, P29	P29 extends the CIA technique at a finer level of granularity (i.e. lines of code) based on that in P9, which is at file granularity level.
T20	P20, P30	P30 defines and validates a hybrid approach that combines ranking of both association rules and Granger over which only shows the probability of this approach in P20.

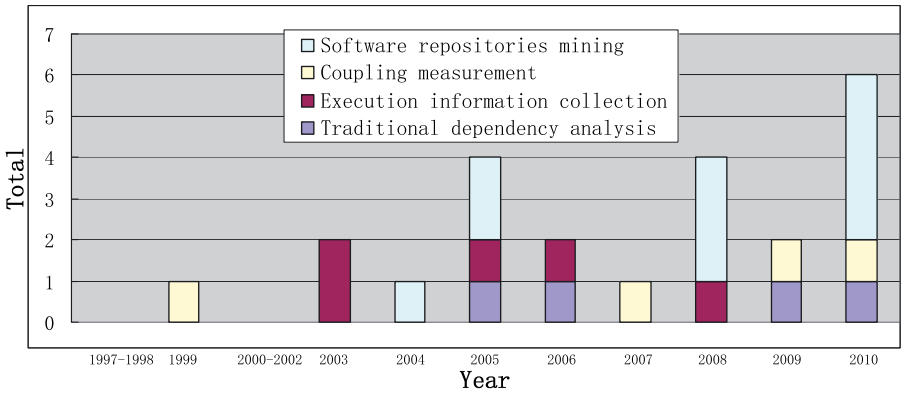

Figure 4. Distribution of the change impact analysis techniques from four perspectives.

DOI: 10.1002/stvr

In addition, from Figure 4, it is shown that a large proportion of CIA techniques are conducted based on dynamic information. It is because dynamic CIA techniques can improve the precision of the impact results compared with traditional static CIA techniques.

5. A COMPARATIVE FRAMEWORK FOR CODE-BASED CHANGE IMPACT ANALYSIS TECHNIQUES

One of the objectives of this study is to construct a framework that can be used to characterize the CIA techniques. This framework should support the identification and comparison of existing CIA techniques based on the specific needs of the user. In addition, it should provide guidelines to support development of new CIA techniques. That is, researchers can develop a new CIA technique based on this framework. In this section, the process used to arrive at such a framework is described, as well as the key components of the resulting framework.

Firstly, all the 30 selected papers were studied, and words of interest that could be relevant to the CIA techniques were marked down. The result after reviewing all 23 CIA techniques was a large set of initial properties.

Then, the initial properties set were analyzed, and some of them are generalized to improve their reusability. For example, the property Java, C and C++ can intuitively be generalized to language support. After this data synthesis, the resulting property set consisted of seven main properties, as follows:

- *Object*. The object of CIA is the change set and the source for analysis. The object is an important factor when selecting a CIA technique. Practitioners may want to know the following: What kinds of source and change set does the CIA need to perform the analysis? Does the source come from the current program version or previous versions (from a software repository)?
- *Impact set*. The output of CIA is the impact set, which composes of impacted elements of the system. The output is also one of the driving factors when selecting a CIA technique. It provides direct results for its application to support various maintenance tasks. For example, from the size of the impact set, the scale of the ripple effects of the proposed changes is known, and an estimation of the cost and effort can be predicted to perform such changes. In addition, with the impact set, the change ripples can be traced when conducting change propagation. Applications of CIA mainly rely on the impact set. Practitioners may consider what can be learnt from the impact set? How to apply the generated impact set? Can the impact set be effectively used?
- *Type of analysis.* There are two main types of analysis: static analysis and dynamic analysis. Each of them can be further divided into subtypes. Subtypes of the static analysis include historical analysis, textual analysis and structural static analysis. And subtypes of dynamic CIA include offline and online CIA. Different types of analysis need different resource, cost and user involvement.
- *Intermediate representation.* CIA is often performed based on the assumption that: if there are some kinds of dependence between two program entities, one of them may be affected when the other entity is changed. Hence, dependence-based CIA techniques are the most commonly used techniques for analyzing the effects of changes on semantic dependence between entities. The intermediate representation may be derived from the current version of the program, or previous versions from a software repository. It affects the cost, accuracy, and extensibility of a CIA technique. The success or failure of a CIA technique depends highly on its intermediate representation. When adopting a particular CIA, the practitioners may consider: What intermediate representation is required? Can the intermediate representation be easily generated? How does the CIA technique utilize the intermediate representation?
- *Language support.* When comparing CIA, researchers and practitioners may consider whether the CIA can support various programming paradigms, such as procedure-oriented programs and object-oriented programs. In addition, they may like to know whether the CIA technique can be easily extended to new programming features.

DOI: 10.1002/stvr

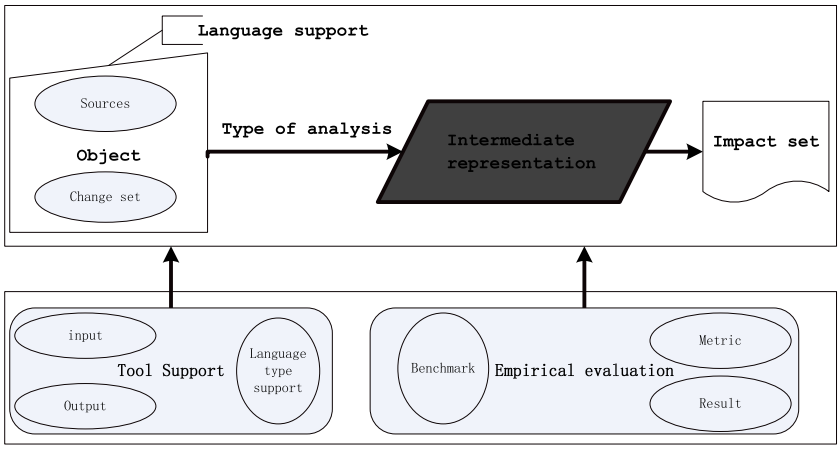

Figure 5. Overview of the framework of the change impact analysis techniques.

- *Tool support.* Since CIA plays a crucial role in software maintenance and evolution, its automation is necessary to encourage wider adoption. The decision to select a particular CIA technique may be affected by the availability of tool support.
- *Empirical evaluation.* All proposed CIA techniques should be empirically validated so that practitioners can select those proven CIA techniques. In addition, development of new CIA techniques may follow similar empirical approach. Researchers are interested in the following questions: What benchmarks can be used to evaluate the CIA technique? What metric can be used to measure the effectiveness of the CIA technique? What can be learned about CIA techniques from the empirical studies? What are the relative performance of these CIA techniques? Which CIA technique is the best based on empirical evidence?

These properties are important and necessary when evaluating a CIA technique. As pointed out earlier, some properties can be divided into finer sub-properties. For example, the sub-properties of the object property include source and change set. Details of properties and sub-properties are presented in the next section. From the identified properties, a framework of CIA techniques is developed as shown in Figure 5. This framework captures the whole process and key components of the CIA, including empirical evaluation and tool support. In Figure 5, the upper rectangle presents the necessary and internal properties of the CIA technique, and the under rectangle shows the external properties of the CIA technique. On the one hand, the internal properties often have a great impact on the effectiveness of the CIA technique, and they must be considered when a new CIA technique is proposed. On the other hand, the external properties constitute the auxiliary properties of the CIA technique, and they are important when the CIA technique is evaluated or selected for application.

6. APPLICATION OF THE COMPARATIVE FRAMEWORK

In this section, the framework proposed earlier is applied to the selected CIA techniques. A comparison of these techniques based on multiple properties in the framework is also available online[‡].

6.1. Object

As shown in Figure 5, the object is composed of two sub-properties: source and change set. Table V shows the object required by various CIA techniques. Source represents the context in which the CIA technique is applied. It can be a single program version or multiple program versions

[‡]http://ise.seu.edu.cn/people/XiaobingSun/survey.xls

DOI: 10.1002/stvr

Table V. Object of the change impact analysis techniques.

	Object								
	Source			Change set					
Technique	Single version	Multiple version	Execution data	Textual change request	File changes	Class changes	Method changes	Field changes	Statement changes
Briand *et al.* [27]	●	·	·	·	·	●	·	·	·
Orso *et al.* [28]	●	·	●	·	·	·	●	·	·
Law *et al.* [20]	●	·	●	·	·	·	●	·	·
Zimmermann *et al.* [55]	·	●	·	·	●	·	·	·	·
Apiwattanapong *et al.* [30]	●	·	●	·	·	·	●	·	·
Badri *et al.* [56]	●	·	·	·	·	·	●	·	·
Ramanathan *et al.* [57]	●	·	●	·	·	·	●	·	·
Breech *et al.* [58]	●	·	·	·	·	·	●	·	·
Canfora *et al.* [59]	·	●	·	●	·	·	·	·	·
Huang *et al.* [60]	●	·	●	·	·	·	●	●	·
Beszedes *et al.* [61]	●	·	●	·	·	·	●	·	·
Jashki *et al.* [62]	·	●	·	·	●	·	·	·	·
Hattori *et al.* [63]	·	●	·	·	·	●	●	·	·
Sherriff *et al.* [64]	·	●	·	·	●	·	·	·	·
Hattori *et al.* [38]	●	·	·	·	·	●	●	●	·
Poshyvanyk *et al.* [39]	●	·	·	·	·	●	·	·	·
Petrenko *et al.* [65]	●	·	·	·	·	●	●	●	●
Kagdi *et al.* [66]	●	●	·	·	●	·	●	·	·
Torchiano *et al.* [67]	·	●	·	●	·	·	·	·	·
Ceccarelli *et al.* [68]	·	●	·	·	●	·	·	·	·
Sun *et al.* [69]	●	·	·	·	·	●	●	●	·
Gethers *et al.* [70]	●	·	·	·	·	●	·	·	·
Ahsan *et al.* [71]	·	●	·	●	·	·	·	·	·
Law *et al.* [19]	●	·	●	·	·	·	●	·	·
Breech *et al.* [72]	●	·	●	·	·	·	●	·	·
Zimmermann *et al.* [73]	·	●	·	·	●	·	·	·	·
Orso *et al.* [34]	●	·	●	·	·	·	●	·	·
Breech *et al.* [32]	●	·	●	·	·	·	●	·	·
Canfora *et al.* [74]	·	●	·	●	·	·	·	·	·
Canfora *et al.* [75]	·	●	·	·	●	·	·	·	·

DOI: 10.1002/stvr

(software repositories). For some dynamic CIA techniques, their source may include a set of tests for the program. The sub-property change set includes all the elements that will be changed. For example, the change set may be composed of a set of files, classes and methods or even a change request in natural language. When the context of the CIA technique is the source code of the current version (e.g. T1, T2, T3, T15–T17), traditional program analysis technique is often used to analyze the syntactic and/or semantic information of the source code. Then, CIA is performed based on these information. Also, some CIA techniques need to execute the program to compute the ripple effect. They use information collected from a specific set of program execution, operational profiles or specific test suites (T2, T3, T5). This 'execution information' helps to improve the precision of the impact results.

On the other hand, useful information about software modifications can be found in software repositories. *Mining software repositories* (MSR), as a broad spectrum of investigations into the examination of software repositories, has become popular. The software repositories offer a wealth of information and provide a unique view of the actual evolutionary path taken to realize a software system. Software repositories usually include historical repositories (i.e. source control repositories, bug repositories), run-time repositories (i.e. deployment logs), and code repositories (i.e. Sourceforge.net). The MSR techniques analyze the rich data in software repositories to uncover interesting and useful information about software systems and projects [76, 77]. For example, some dependence between program entities that can not be distilled by the traditional program analysis technique can often be mined from these repositories. CIA is then performed based on these 'additional' dependence (e.g. T4, T9, T12–T14). Recently, Kagdi *et al.* utilized both source code of the current program version and previous versions from software repositories to obtain better impact results (T18), when compared with using them independently.

The change set of a CIA technique may include changes at different granularity levels, that is, textual change request (T9, T19, T23), files (T12, T14, T20), classes (T1, T16, T22), class members (T15, T17, T21, etc.), or even statements (T17). The granularity of the change set may affect the accuracy of the impact results. Petrenko *et al.* validates that the finer the elements in the change set, the more precise the impact results [65]. Some techniques perform impact analysis at just one granularity-level (T1–T9, etc.) whereas others can tackle change set at multiple granularity levels from the coarser file or class level to the finer class member or statement level (T15, T17, T21). Currently, most of the CIA techniques accept change sets at the source code level. But there are also some works on the ripple effect extracted from non-source code files changes, which may ripple to other source code files (T12, T14). In addition, some CIA techniques were proposed based on the information in the change request form (T9, T19, T29). This approach makes use of the changes suggested by the user, who may have little knowledge of the source code.

Figure 6 presents the distribution of different types of source and change sets for the CIA techniques. From Figure 6, there are 15 techniques needing to use the current program for analysis, nine techniques use multiple versions, and seven need execution of the software to perform impact

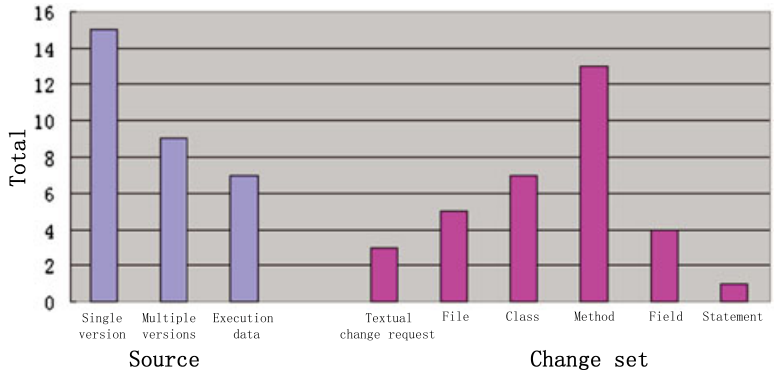

Figure 6. Distribution of various source and change sets of the change impact analysis techniques.

DOI: 10.1002/stvr

Table VI. Impact set of the change impact analysis techniques.

	Impact set						
	Granularity					Ranked	
Technique	File	Class	Method	Field	Statement	Ranked	Not ranked
Briand *et al.* [27]	·	●	·	·	·	●	·
Orso *et al.* [28]	·	·	●	·	·	·	●
Law *et al.* [20]	·	·	●	·	·	·	●
Zimmermann *et al.* [55]	●	●	●	·	·	●	·
Apiwattanapong *et al.* [30]	·	·	●	·	·	·	●
Badri *et al.* [56]	·	·	●	·	·	·	●
Ramanathan *et al.* [57]	·	·	●	·	·	·	●
Breech *et al.* [58]	·	·	●	·	·	·	●
Canfora *et al.* [59]	●	·	·	·	·	●	·
Huang *et al.* [60]	·	·	●	●	·	·	●
Beszedes *et al.* [61]	·	·	●	·	·	·	●
Jashki *et al.* [62]	●	·	·	·	·	·	●
Hattori *et al.* [63]	·	●	●	·	·	·	●
Sherriff *et al.* [64]	●	·	·	·	·	·	●
Hattori *et al.* [38]	·	●	●	●	·	·	●
Poshyvanyk *et al.* [39]	·	●	·	·	·	●	·
Petrenko *et al.* [65]	·	●	●	●	●	●	·
Kagdi *et al.* [66]	●	·	·	·	·	·	●
Torchiano *et al.* [67]	●	·	●	·	·	●	·
Ceccarelli *et al.* [68]	●	·	·	·	·	·	●
Sun *et al.* [69]	·	●	●	●	·	·	●
Gethers *et al.* [70]	·	●	·	·	·	·	●
Ahsan *et al.* [71]	●	·	·	·	·	·	●
Law *et al.* [19]	·	·	●	·	·	·	●
Breech *et al.* [72]	·	·	●	·	·	·	●
Zimmermann *et al.* [73]	●	●	●	·	·	●	·
Orso *et al.* [34]	·	·	●	·	·	·	●
Breech *et al.* [32]	·	·	●	·	·	·	●
Canfora *et al.* [74]	●	·	·	·	●	●	·
Canfora *et al.* [75]	●	·	·	·	·	·	●

analysis. From the perspective of the granularity of change set, the most used change set includes method granularity-level elements. The file and class level change sets are also widely used for impact analysis.

6.2. Impact set

The impact sets of the CIA techniques are shown in Table VI. Similar to the change sets, the impact sets are also at different granularity levels. The granularity level of the impact set of a CIA technique is often corresponding to that of the change set, that is, when the change set is at a certain granularity-level (files, classes, class members), the impact set is also at the same granularity-level. The impact set composes of a set of potentially impacted entities (T2, T3, T5, T6, T7, T8). This set is often large, thus difficult for practical use, particularly for static CIA techniques [4, 29, 78]. Given such an impact set, users often do not know where to start the inspection of the impacted results. Hence, some researchers compute a ranked list of entities marked with the priority for checking (T1, T4, T9, T16, T17, T19). This prioritized list can better support various maintenance tasks. When users select only those higher priority entities, the impact set will have fewer false-positives; when the users select impacted results of all priority, the impact set may include more false-negatives. Hence, such impacted results can provide an eclectic solution that matches the needs of users.

Impact set of the CIA techniques can be generated at file, class, method, field and statement level. Figure 7 shows the distribution of the impact set at different granularity levels. This figure shows that most of the impact sets include entities at method granularity level. Comparing Figure 6 with

DOI: 10.1002/stvr

Figure 7, on the one hand, they are similar as most of the change sets and impact sets are at method granularity level; on the other hand, the classification of the change set and impact set is different, that is, the changes of some CIA techniques are textual change request, but the corresponding impact set can be the source code (classes, methods, and statements). This shows that some CIA techniques use a change set at a certain level but generate an impact set at a different level.

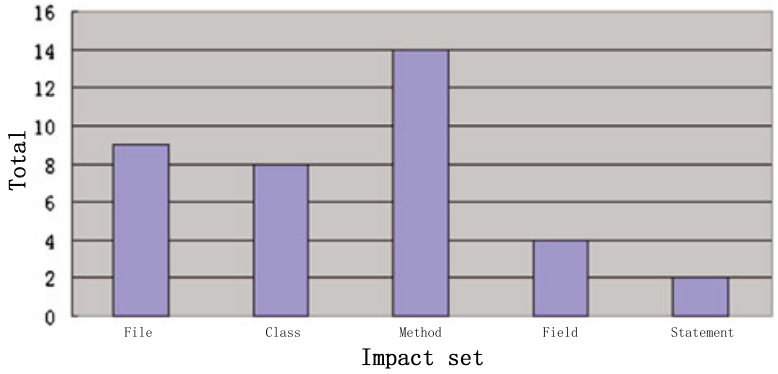

Figure 7. Distribution of impact sets of the change impact analysis techniques.

Table VII. Type of analysis of the change impact analysis techniques.

	Type of analysis				
	Static			Dynamic	
Technique	Structural static analysis	Historical analysis	Textual analysis	Offline analysis	Online analysis
Briand *et al.* [27]	●	·	·	·	·
Orso *et al.* [28]	·	·	·	●	·
Law *et al.* [20]	·	·	·	●	·
Zimmermann *et al.* [55]	·	●	·	·	·
Apiwattanapong *et al.* [30]	·	·	·	●	·
Badri *et al.* [56]	●	·	·	·	·
Ramanathan *et al.* [57]	·	·	·	●	·
Breech *et al.* [58]	●	·	·	●	·
Canfora *et al.* [59]	·	●	●	·	·
Huang *et al.* [60]	·	·	·	●	·
Beszedes *et al.* [61]	·	·	·	●	·
Jashki *et al.* [62]	·	●	·	·	·
Hattori *et al.* [63]	●	●	·	·	·
Sherriff *et al.* [64]	·	●	·	·	·
Hattori *et al.* [38]	●	·	·	·	·
Poshyvanyk *et al.* [39]	·	·	●	·	·
Petrenko *et al.* [65]	●	·	·	·	·
Kagdi *et al.* [66]	·	●	●	·	·
Torchiano *et al.* [67]	·	●	●	·	·
Ceccarelli *et al.* [68]	·	●	·	·	·
Sun *et al.* [69]	●	·	·	·	·
Gethers *et al.* [70]	·	·	●	·	·
Ahsan *et al.* [71]	·	●	·	·	·
Law *et al.* [19]	·	·	·	·	●
Breech *et al.* [72]	·	·	·	·	●
Zimmermann *et al.* [73]	·	●	·	·	·
Orso *et al.* [34]	·	·	·	●	·
Breech *et al.* [32]	·	·	·	●	●
Canfora *et al.* [74]	·	●	●	·	·
Canfora *et al.* [75]	·	●	·	·	·

DOI: 10.1002/stvr

6.3. Type of analysis

Current researches in CIA have varied from relying on static information (T1, T4, T6, T8, T9) to dynamic information (T2, T3, T5, T7) in working out the impact set. Table VII shows various types of analysis used by the selected CIA techniques.

Static CIA techniques take all possible behaviour and inputs into account, and thus are conservative at the cost of precision. As Figure 8 shows, static CIA techniques are often performed by analyzing the syntax and semantic, or evolutionary dependence of the program (or its change history repositories). The resultant impact set often has many false-positives, with many of its elements not really impacted [20, 28, 79]. Static analysis can be further divided into structural static analysis, textual analysis, and historical analysis [70]. Structural static analysis focuses on static analysis of the structural dependence of the program and construction of the dependence graph (T1, T6, T8, T13, T15, T17, T21). Textual analysis extracts some conceptual dependence (coupling) based on the analysis of the comments and/or identifiers in the source code. These coupling measures provide a new perspective to traditional structural coupling measures (T9, T16, T22). Historical analysis is performed by mining the information from multiple evolutionary versions of the software in software repositories (T4, T12, T13, T18, T20).

Dynamic analysis considers some specific inputs and relies on the analysis of the information collected during program execution (e.g. execution traces information, coverage information and execution relation information) to calculate the impact set [20, 28, 30]. The process of dynamic CIA techniques is shown in Figure 9. It starts with a set of test data as input and collects the execution information for analysis. The impact set tends to be more precise than that of the static analysis. However, the cost of dynamic CIA techniques is higher than that of static CIA because of the overhead of expensive dependency analysis during program execution [72]. Moreover, their impact set often includes some false-negatives. Dynamic CIA can be performed online or offline. Offline CIA is performed after the program finishes its execution, whereas online CIA performs all analysis using information collected as the program is executing. Breech *et al.* proposed an approach of the whole program path-based online impact analysis [72]. Online CIA techniques are proposed to alleviate the need to obtain the whole runtime execution information after instrumented execution. The impact set can be calculated for any number of multiple runs of the same program depending on the set of the inputs used for inferring the dynamic behaviour of the system [32]. The precision of online impact analysis and offline impact analysis has been empirically validated [32, 72]. The results show that online CIA techniques compute impact sets as precise as offline CIA techniques, but scale better [32].

Figure 10 shows the distribution of the types of analysis of the CIA techniques. This figure shows that 70% of the CIA techniques belong to static analysis, and the remaining 30% belong to dynamic CIA. In addition, historical analysis is the most used techniques in CIA. Historical analysis focuses on the co-change coupling to predict future change, which naturally contributes to its popularity. In addition, structural static analysis is also widely used. As most of the structural analysis is performed based on the structural dependence (control or data dependence) in the programs, it can be used to predict which elements are impacted based on these dependence. However, these dependence lack

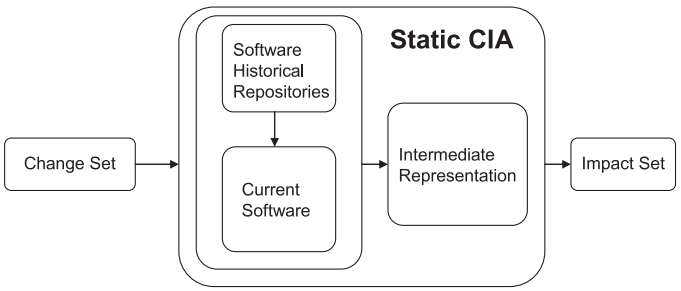

Figure 8. Static change impact analysis process.

DOI: 10.1002/stvr

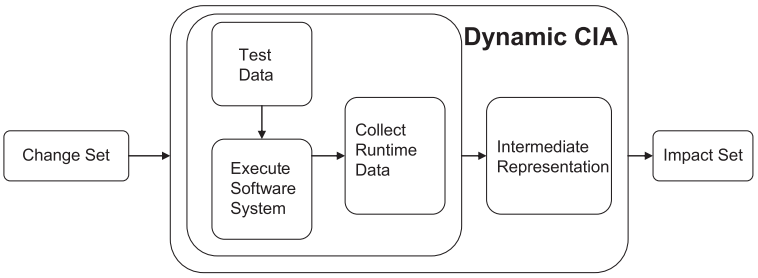

Figure 9. Dynamic change impact analysis process.

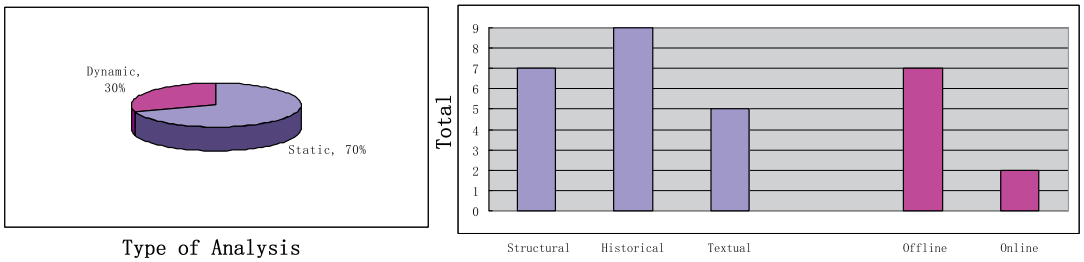

Figure 10. Distribution of the analysis types of the change impact analysis techniques.

the ability to identify conceptual dependence. For example, implicit relationships encoded by developers in identifiers and comments of source code cannot be distilled from structural analysis. But these entities are also important because they express the intent of the software. Two parts of the software with similar intent will most likely refer to similar concepts in the problem or solution domains of the system. Hence, textual analysis of these comments and identifiers have attracted an increasing concern.

6.4. Intermediate representation

Intermediate representation is an important factor to develop a new CIA technique. It influences the cost, accuracy and extensibility of a CIA technique, and impacts the effectiveness of the CIA technique. It is the main property to distinguish between different types of CIA techniques. Intermediate representations of the selected CIA techniques are shown in Table VIII.

Various dependence graphs, such as call graph (T6, T15), influence graph (T8) and class and member dependency graph (T17) are common representations for program analysis. Different CIA techniques are often conducted based on different representations according to specific situations, that is, programming paradigm [52, 65, 80], analyzed subject [27, 59, 63, 64], types of analysis [9, 20, 28, 30] and so on. For example, traditional CIA techniques use static slicing [81–85], dynamic slicing [86–88], or hybrid slicing [89, 90] to trace the change effects for different programming paradigms.

Static CIA technique often resorts to the structural analysis of dependence among entities in the system to compute the impact set. As these dependence are constructed based on all possible runtime behaviours of the system, the impact results will not be very precise. To overcome this problem, some researchers propose dynamic CIA techniques, which focus on a set of specific program executions. Dynamic CIA techniques are based on runtime data and dynamic interactive behaviours of the system. They use dependence information extracted during the program execution, such as program execution trace and execution coverage information. Thus, their intermediate representations include the whole program path directed acyclic graph (T3), execute-after relation (T5), and dynamic dependency graph (T10).

DOI: 10.1002/stvr

Table VIII. Intermediate representation of the change impact analysis techniques.

Technique	Intermediate representation
Briand *et al.* [27]	Structural coupling measures
Orso *et al.* [28]	Static forward slice and coverage bit vector
Law *et al.* [20]	Whole program path directed acyclic graph
Zimmermann *et al.* [55]	Association rules
Apiwattanapong *et al.* [30]	Execute-after relation
Badri *et al.* [56]	Control call graph
Ramanathan *et al.* [57]	Memory traces and dynamic programming
Breech *et al.* [58]	Influence graph
Canfora *et al.* [59]	CR query description, XML file descriptor representation, and textual similarity
Huang *et al.* [60]	Dynamic dependency graph
Beszedes *et al.* [61]	Dynamic function coupling
Jashki *et al.* [62]	Co-occurrence matrix, and vector-space representation of program files
Hattori *et al.* [63]	*Apriori* and *DAR* algorithms
Sherriff *et al.* [64]	Singular value decomposition
Hattori *et al.* [38]	Call graph
Poshyvanyk *et al.* [39]	Conceptual coupling measures
Petrenko *et al.* [65]	Class and member dependency graph
Kagdi *et al.* [66]	Conceptual couplings, and evolutionary couplings
Torchiano *et al.* [67]	Keywords combination
Ceccarelli *et al.* [68]	Multivariate time series, and association rules
Sun *et al.* [69]	Object-oriented class and member dependency graph
Gethers *et al.* [70]	Relational toping based coupling measure
Ahsan *et al.* [71]	Single and multi-label machine learning classification
Canfora *et al.* [74]	Line history table

In addition, some CIA techniques rely on the coupling measures between program entities to compute a ranked list of impacted entities. These coupling measures include traditional structural couplings (T1), conceptual couplings (T16, T18), dynamic function coupling (T11) and relational topic based coupling (T22). There is a rich body of work on structural couplings [91]. These couplings have been comprehensively described within a unified framework for coupling measurement [91]. There are coupling between objects (CBO), response for class (RFC), message passing coupling (MPC), information-flow-based coupling (ICP), and others. Conceptual coupling is based on measuring the degree to which the identifiers and comments from different classes relate to each other [39]. Poshyvanyk *et al.* reported the accuracy comparison of the impact results of structural couplings against conceptual couplings and showed that one of the conceptual couplings, $CCBC_m$ (maximum conceptual coupling between two classes), appears to be superior to existing structural coupling measures [39]. Relational topic based coupling (RTC) of classes is also a kind of conceptual coupling, which uses relational topic models (RTM) to capture latent topics in source code classes and their relationships. Gethers *et al.* showed that this coupling is a good complement to the conceptual coupling proposed by Poshyvanyk *et al.* [70]. In addition, dynamic function coupling (DFC) between two methods is proposed based on the assumption that the closer the execution of a method is to the execution of another method in some execution of the program, the more likely they are dependent on each other. Then, impact sets may be computed based on this kind of coupling [61].

These discussions focus on the intermediate representation obtained from the current version of the program. There are also some intermediate representations derived from multiple versions in software repositories. Some CIA techniques mine dependence information from software repositories. They try to identify some co-change dependence (e.g. for entities changed together during the same cycle of program evolution, these entities are said to have such co-change dependence) between program entities from various software repositories, which can not be extracted from traditional program analysis techniques. Intermediate representations for these CIA techniques include singular value decomposition (T14), co-occurrence matrix (T12), association rules (T4), line history table (T9), and multivariate time series (T20). These representations attempt to show

DOI: 10.1002/stvr

the evolutionary dependence between program entities instead of dependence within a specific program version.

Table VIII illustrates that different CIA techniques use different intermediate representations, which are based on specific object, type of analysis and so on. It is difficult to identify the best representation, as each serves a different purpose. However, the finding can provide some references to develop a new CIA technique. Moreover, the effectiveness of these representations may be obtained through the empirical evaluation, which will be introduced later.

6.5. Language support

Existing CIA techniques have varied from supporting traditional procedure-oriented programs [20, 28–30, 92] to object-oriented programmes [9, 39, 63, 65, 69, 79, 93]. For the emerging language and design paradigms, researchers usually either construct a novel representation to express the new language features or extend traditional representations to include these features to support CIA. Table IX presents the language paradigms supported by the CIA techniques.

Most of traditional CIA techniques support procedure-oriented programs to compute the method-level impact set (T2–T9, T12–T15). These CIA techniques rely on capturing the relationship between methods to generate the impact results. Of course, they can also be extended and applied to other programs such as object-oriented programmes.

Object-oriented paradigm has introduced some new design and programming concepts, such as encapsulation, inheritance and polymorphism. In recent years, a lot of work has been studied to support impact analysis for object-oriented programs (T1, T10, T15–T18, T21–T22). These CIA

Table IX. Language support of the change impact analysis techniques.

	Language support	
Technique	Procedure-oriented	Object-oriented
Briand *et al.* [27]	·	●
Orso *et al.* [28]	●	●
Law *et al.* [20]	●	●
Zimmermann *et al.* [55]	●	●
Apiwattanapong *et al.* [30]	●	●
Badri *et al.* [56]	●	●
Ramanathan *et al.* [57]	●	●
Breech *et al.* [58]	●	●
Canfora *et al.* [59]	●	●
Huang *et al.* [60]	·	●
Beszedes *et al.* [61]	●	●
Jashki *et al.* [62]	●	●
Hattori *et al.* [63]	●	●
Sherriff *et al.* [64]	●	●
Hattori *et al.* [38]	·	●
Poshyvanyk *et al.* [39]	·	●
Petrenko *et al.* [65]	·	●
Kagdi *et al.* [66]	·	●
Torchiano *et al.* [67]	●	●
Ceccarelli *et al.* [68]	●	●
Sun *et al.* [69]	·	●
Gethers *et al.* [70]	·	●
Ahsan *et al.* [71]	●	●
Law *et al.* [19]	●	●
Breech *et al.* [72]	●	●
Zimmermann *et al.* [73]	●	●
Orso *et al.* [34]	●	●
Breech *et al.* [32]	●	●
Canfora *et al.* [74]	●	●
Canfora *et al.* [75]	●	●

DOI: 10.1002/stvr

techniques take concrete features of the object-oriented programmes into account and generate the potentially impacted classes, class methods and/or class fields.

Overall, most of current CIA techniques can be used in object-oriented programmes, and there is a need to develop new CIA techniques for newer programming paradigms such as aspect-oriented programmes.

6.6. Tool support

In spite of so many surveyed CIA techniques, there are only 10 available tools to support these techniques. These tools are listed in Table X, together with their required input data, output and the programming languages supported.

Some tools such as *Impala* and *JRipples* assist in static analysis of the current system. *Impala* supports CIA before the change implementation [63]. Its input includes classes, methods and fields. Impact analysis is started by searching in the dependency graph from the elements in the change set according to the change type and the dependency type. Those elements, which are reachable from the elements in the change set, are collected as the output of this tool. *JRipples* supports program comprehension during incremental changes [92]. It analyzes the program and automatically marks the impacted classes. Later when a modification is designated, other impacted classes will be automatically shown.

There are also some tools, such as *EAT* and *JDIA*, which use the execution information to support dynamic CIA. *EAT* uses execute-after sequences [30]. Its input is the changed methods and execution information, whereas its output is the dynamic impact set at the method level. *JDIA* works on object-oriented programs. It takes the changed entities, the program, and some executions as the input, and outputs the impacted methods and fields, which can be saved as text file or XML file.

Couplings between entities can be used to measure various dependence between entities. For techniques based on the coupling measures to support impact analysis, coupling calculation is very important. *Columbus* is a very effective tool to compute almost all the structural coupling measures [94]. The *IRC*^{2}*M* tool can compute the conceptual coupling measures [96]. And *LDA* can be used to compute the relational topic based coupling [70]. These three tools take the program as the input and output of the coupling measures between the classes in the program. According to the coupling results, those elements which, have higher coupling with the elements in the change set, are indicated to be more probably impacted.

In addition, some tools such as *ROSE* [73] and *Jimpa* [95], have been developed to analyze the software repositories for impact analysis. *ROSE* is a plug-in for the eclipse environment. It analyzes the version history and the given changes, and outputs the likelihood that further changes should be applied to the given location. *Jimpa* is also a plug-in, which starts from a change request description and the set of historical source file revisions. Then based on an information retrieval approach, some impacted files are identified by referencing similar past change requests.

As introduced in Section 4, CIA techniques can be classified from four perspectives: traditional dependency analysis, software repositories mining, coupling measurement and execution information collection. The distribution of the tool support for these types of CIA techniques is shown in Figure 11. From this figure, it shows that there are two to three tools to support each type of CIA techniques. Practitioners can select corresponding tools to support their CIA activity according to Table X.

6.7. Empirical evaluation

The selected CIA techniques have been validated by empirical studies. Approaches for empirical evaluation of the CIA techniques are shown in Table XI, which lists the size of the benchmark and the measure used to show the effectiveness of the CIA technique.

There are many different types of benchmarks. The empirical study can be classified based on the size of the benchmark. An empirical study is said to be small when the artefact in the benchmark has less than 5000 lines of code (KLOC); large when its artefact has more than 100 KLOC; and medium when its artefact between 5 and 100 KLOC. These classifications are somewhat arbitrarily defined. Figure 12 shows that up to 47% of the empirical studies belong to medium empirical

DOI: 10.1002/stvr

Table X. Tool support of the change impact analysis techniques.

	Tool support				
Technique	Name	Input	Output	Language	Reference
Briand *et al.* [27]	*Columbus*	Object-oriented system	Structural coupling measures	C++	[94]
Zimmermann *et al.* [55]	*ROSE*	Software historical repositories; current program; changes	Impacted parts	Java	[73]
Apiwattanapong *et al.* [30]	*EAT*	Execution information; proposed changed methods	Impacted methods	Java	[30]
Ramanathan *et al.* [57]	*Sieve*	Programme binaries of original and modified program	Impacted methods and code regions in modified program	*C*	[57]
Canfora *et al.* [59]	*Jimpa*	A change request description; historical source files repositories	Impacted files	Any*	[95]
Huang *et al.* [60]	*JDIA*	Changes; the program; some executions	Impacted methods and fields	Java	[60]
Hattori *et al.* [38]	*Impala*	The system; changes	Impacted elements	Java	[38]
Poshyvanyk *et al.* [39]	*IRC*^{2}*M*	A project	Conceptual coupling measures	Any*	[96]
Petrenko *et al.* [65]	*JRipples*	The system; changed classes	Impacted classes	Java	[92]
Gethers *et al.* [70]	*LDA*	A software project	Relational topic based coupling	Any*	[70]

*Any, represents that this tool supports the CIA technique, which is immune to its supported language type.

DOI: 10.1002/stvr

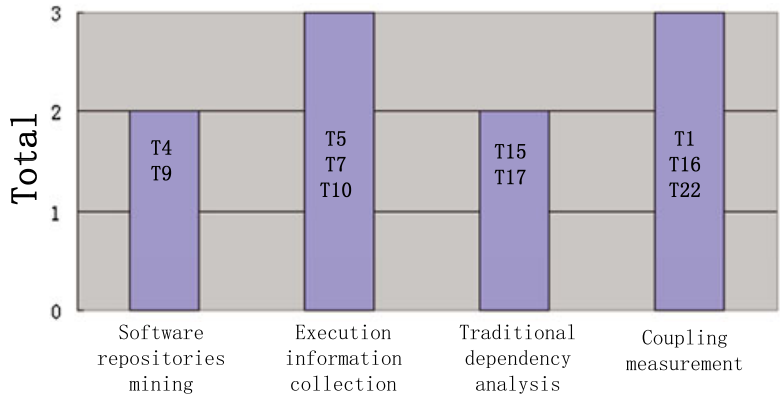

Figure 11. Distribution of tool support of the change impact analysis techniques.

Table XI. Empirical evaluation of the change impact analysis techniques.

	Empirical evaluation						
	Benchmark size			Measure			
Technique	Small	Medium	Large	Accuracy	Cost	Comparison	Others
Briand *et al.* [27]	·	●	·	●	·	·	·
Orso *et al.* [28]	·	●	·	·	·	●	●
Law *et al.* [20]	·	●	·	·	·	●	●
Zimmermann *et al.* [55]	·	●	●	●	·	·	·
Apiwattanapong *et al.* [30]	●	●	·	·	●	●	●
Badri *et al.* [56]	·	●	·	·	·	●	·
Ramanathan *et al.* [57]	·	●	·	·	●	●	●
Breech *et al.* [58]	●	●	·	·	●	●	●
Canfora *et al.* [59]	·	●	●	●	·	·	·
Huang *et al.* [60]	●	●	·	·	●	●	●
Beszedes *et al.* [61]	●	·	·	●	·	●	·
Jashki *et al.* [62]	·	●	●	●	●	·	·
Hattori *et al.* [63]	●	·	·	·	·	·	●
Sherriff *et al.* [64]	·	●	●	·	·	●	●
Hattori *et al.* [38]	●	●	·	●	·	·	·
Poshyvanyk *et al.* [39]	·	·	●	●	·	·	·
Petrenko *et al.* [65]	·	·	●	·	·	·	●
Kagdi *et al.* [66]	·	●	●	●	·	·	·
Torchiano *et al.* [67]	·	●	●	●	·	·	·
Ceccarelli *et al.* [68]	·	●	·	●	·	·	·
Sun *et al.* [69]	●	·	·	●	·	·	·
Gethers *et al.* [70]	·	●	●	●	·	·	·
Ahsan *et al.* [71]	·	·	●	●	·	·	·
Law *et al.* [19]	·	●	·	·	●	●	·
Breech *et al.* [72]	●	●	·	·	●	●	●
Zimmermann *et al.* [73]	·	●	●	●	·	·	·
Orso *et al.* [34]	●	●	·	·	●	●	●
Breech *et al.* [32]	●	●	·	·	●	●	●
Canfora *et al.* [74]	·	●	●	●	●	·	·
Canfora *et al.* [75]	·	●	·	●	·	·	·

studies; 30% belongs to small; and only 23% employ large-size artefacts. In addition, different CIA techniques use different artefacts as their benchmarks for empirical studies. The comparison of CIA techniques should be facilitated by the existence of common benchmarks that could be used to consistently evaluate the approaches. Currently, there are a number of systems that have been used in the evaluation of many CIA techniques. It is hoped that some popular artefacts will emerge

DOI: 10.1002/stvr

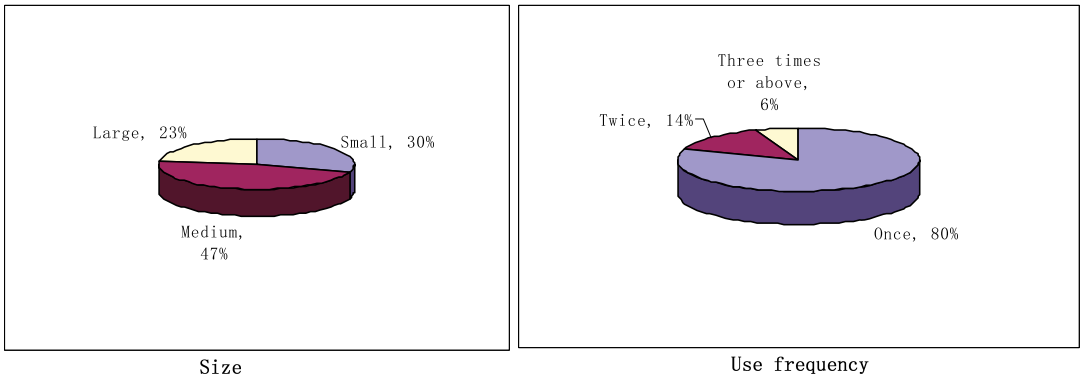

Figure 12. Distribution of the size and use frequency of the benchmarks in empirical evaluation.

as common benchmarks for empirical validation of all CIA techniques. Figure 12 also shows the use of frequency of the benchmarks in empirical studies. From this figure, it shows that most of the artefacts (up to 80%) are only used once in empirical studies. Some are used twice, such as *JABA* and *ArgoUML*. The most popular artefacts are *space*, *Mozzilla*, *Eclipse* and *JEdit*, which are used three times or more.

For the measure used in the empirical evaluation, some approached measure the effectiveness from the cost perspective (T5, T8, T10); some from the accuracy perspective (T1, T4, T9, T11, T12); some also from other perspectives, such as the size of the impact set. Cost is measured in terms of time and space to perform the CIA. With the increasingly improved performance brought by the hardware advancement, time and space are no longer the main concern of researchers and practitioners. The other measures consider the quality of the elements in the impact set. Traditional CIA techniques often assess the CIA by measuring the size of the impact set or the ratio between the impact set and the whole system (T2, T3, T5, T8, T9). This measure method takes the scale (size) perspective. However, they can not identify those entities in the impact set that are really affected (true-positives), those entities are in fact not affected (false-positives), and those entities are in fact affected, but not in the estimated impact set (false-negatives). To do this, an analysis of the content of the impact set is needed. As introduced in Section 2, precision and recall are two widely used metrics to validate the accuracy of the CIA techniques. Precision measures the degree the EIS accord with the real impacts induced by changes, whereas recall measures the degree the EIS cover the real changes during change implementation. With these two metrics, users can know which CIA techniques provide more precise results and which CIA techniques have good recall, and then select the appropriate CIA technique that fits their needs. As seen in Figure 13, at present, accuracy (precision and recall) is the most widely used metrics to evaluate the CIA techniques.

In addition, many of the CIA techniques are measured by comparing with other CIA techniques, such as T2, T3, T5–T8. The empirical studies in these papers focus on comparison of CIA techniques

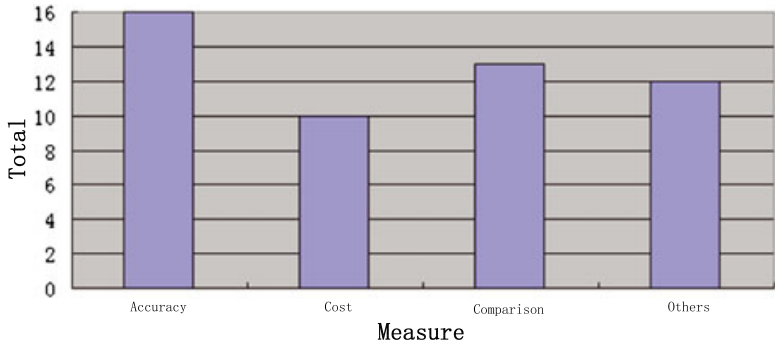

Figure 13. Distribution of effectiveness measures of the change impact analysis techniques.

DOI: 10.1002/stvr

and identification of those effective ones. From these empirical studies, the relative advantages of CIA techniques can be obtained. Figure 14 shows the comparison results of these CIA techniques along dimensions of accuracy (left) and cost (right). If two CIA techniques (CIA_1 and CIA_2) are connected by a line, and CIA_1 is placed on top of CIA_2, then CIA_1 is better than CIA_2. For any two CIA techniques that are not connected, it indicates that no empirical studies have been carried out for them. From Figure 14, T7, T8, T10, T11 and T18 are good CIA techniques to use from the accuracy perspective. Of these five CIA techniques, T7, T8, T10 and T11 are dynamic CIA techniques, and T18 is a hybrid CIA technique, which used both conceptual coupling and historical information. This analysis illustrates that dynamic CIA, textual static CIA and historical CIA may be better in accuracy compared with traditional structural static CIA. From the cost perspective in Figure 14, it shows that online CIA, T2, and call graph based CIA cost less than other CIA techniques. However, from the accuracy perspective, T2 and call graph based CIA perform relatively poor. Overall, online CIA technique is more suitable for practical use from both accuracy and cost perspectives.

6.8. Evaluation of the proposed framework

In this section, the comparative framework is evaluated from two perspectives: expressiveness of the framework and effectiveness for comparison.

The expressiveness of a comparative framework is related to its ability to cover a wide spectrum of CIA techniques. This can be seen from Tables V–XI. These selected CIA techniques (T1–T23) can be characterized by the properties in the framework. And from the properties of the CIA techniques in the framework, the key components of a CIA technique and what can be learned or obtained from a CIA technique can be identified. Using the proposed framework, the CIA technique that fits practical demands for a specific situation can be easily selected.

Effectiveness for comparison measures the ease and comprehensiveness of comparison of the CIA techniques. To compare different CIA techniques, on the one hand, the properties and/or sub-properties can be compared to determine which CIA technique is suitable for practical use. For example, from the perspective of object of the CIA, some CIA techniques need only the current software system, and some need multiple historical versions. In addition, from the sub-property change set, some CIA techniques analyze change at the class level, some analyze at the finer method level, and some even analyze changes from textual change request. Figures 6, 7 and 10–13 present the statistics of the distribution of the key properties for the CIA techniques. On the other hand, the better CIA techniques can be obtained, that is, those with fewer false-positives and/or false-negatives

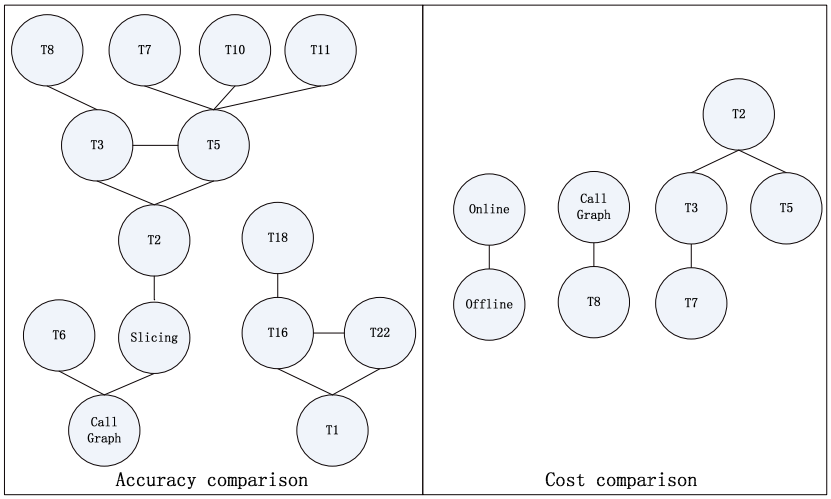

Figure 14. Comparison of the change impact analysis techniques in empirical studies.

DOI: 10.1002/stvr

through the empirical evaluation, as shown in Tables XI. And, Figure 14 shows the comparison results of the CIA techniques from the accuracy and cost perspectives.

From this analysis, the framework is shown to be expressive and effective for surveying the CIA techniques.

7. APPLICATIONS OF CHANGE IMPACT ANALYSIS TECHNIQUES

Change impact analysis plays an important role in software development, maintenance and regression testing [4–6]. During the survey, many applications of CIA in software maintenance were found, that is, software comprehension, change propagation, selection of regression test cases, debugging and measuring various software attributes such as changeability and maintenance cost. Thus, CIA can indirectly support maintainers to make decision among various change solutions, prepare change schedule, estimate resources and costs, and trace the change effect. Table XII lists the main application areas of CIA and associated tools support (if available). 'N/A' in the table denotes that no tool can be found to support the application in the literature.

7.1. Software comprehension

Before a change can be implemented, a detailed knowledge of the system must be acquired to determine where and how to make the required change. CIA can assist in understanding the original programs. If the potential effects of changes can be calculated before the changes are implemented, the programmer can accurately and efficiently perform the required changes.

As introduced in Section 4, many CIA techniques have the capability of mining the dependence between program entities and thus improving program comprehension [29, 65, 73]. In particular, the intermediate representation constructed by some CIA techniques can aid understanding of the program. Several tools are developed for this purpose. For example, Buckner *et al.* provided a tool, named *JRipples*, to support program comprehension during incremental changes [92]. In addition, *ROSE* can present the transaction rules between different entities [73].

7.2. Change propagation

Change propagation is a major application of CIA. As changes are made, maintainers must ensure that other entities in the software system are consistent with these new changes. Change propagation

Table XII. Change impact analysis applications.

Application area	Tools support	Reference
Software comprehension	*JRipples*	Buckner *et al.* [92]
	ROSE	Zimmermann *et al.* [73]
	N/A	Kagdi [97], Tonella [29], Kagdi [66]
Change propagation	*JTracker*	Gwizdala *et al.* [98]
	JRipples	Buckner *et al.* [92]
		Hassan *et al.* [99], Rajlich *et al.* [100]
	N/A	Hassan *et al.* [101], Malik *et al.* [102]
Regression testing	*Chianti*	Ren *et al.* [9, 103]
	Celadon	Zhang *et al.* [104]
	N/A	Martin *et al.* [105], Wang *et al.* [106], Pourgalehdari *et al.* [107], Orso [28]
Debugging	*Crisp*	Chesley *et al.* [108]
	AutoFlow	Zhang *et al.* [109]
	Chianti	Ren *et al.* [103]
Software measurement	*ROSE*	Zimmermann *et al.* [73]
	N/A	Fluri *et al.* [110], Chaumun *et al.* [111]

DOI: 10.1002/stvr

is defined as the changes required of other entities of the software system to ensure the consistency of assumptions within the system after a particular entity is changed [99]. It is often performed during incremental changes [100]. Impact analysis is first used to predict the change effects, which can then be checked to see whether they need modification.

JTracker is a tool to assist in change propagation. Whenever the programmer changes a class, this tool can mark the potentially impacted neighbouring classes [98]. If changes of these neighbouring classes are not necessary, the propagation can be stopped; otherwise, the programmer implements the changes. After a modification, *JTracker* automatically marks additional classes potentially affected by the changes. *JRipples* is also a useful tool to support change propagation during incremental changes.

7.3. Regression testing

Change impact analysis can also be used for regression testing by selecting the test cases that need to be rerun and adding new test cases to cover the changes and affected parts not tested by the original test suite. This will help build confidence in the integrity of the system after modification.

Chianti [9] and *Celadon* [104] are two tools, which can be directly used to support regression testing. The outputs of these two tools contain the affected test cases. The main difference between them is that one is utilized for Java programs and the other for AspectJ programs.

Orso *et al.* presented a novel approach that leverages field data to support regression testing [28, 34]. They utilized the impact sets generated from CIA to help select, augment and prioritize the test cases from the original test suite. They first identified an initial set of test cases that traversed at least one change in the change set based on coverage information. Then, they employed the impact set to assess whether the initial test suite set was adequate according to the field data. If it was not adequate, the initial test suite set needs to be augmented by adding those test cases, which have not covered the entities not traversed by the original test cases in the impact set. In addition, they prioritized the test cases in the new test suite by giving a higher priority to those test cases, which cover more affecting entities of the impact set.

7.4. Debugging

Programmers often spend a significant amount of time debugging programs to locate the faults and make correction. Typically, CIA techniques can be employed when regression tests fail unexpectedly by isolating a subset of responsible changes for that failing test. Then, failure-inducing changes can be found out more effectively. Chesley *et al.* proposed a tool *Crisp* for debugging Java programs. When a test fails, *Crisp* firstly uses the input from *Chianti*, which supports CIA, to select parts of the edit that affect the failing test and to add them to the original program [108]. Thus, an intermediate version of the program guaranteed to compile is created for the programmer to re-execute the test. Based on this mechanism, individual (or sets of) affecting changes are iteratively selected or undone to effectively find a small set of changes contributing to the failing test.

A similar tool, *AutoFlow*, for AspectJ software debugging has been developed by Zhang *et al.* [109]. It can automatically reduce a large portion of irrelevant changes in the early phase, and effectively locate faulty changes. *AutoFlow* integrates the delta debugging algorithm with CIA to narrow down the search for faulty changes. CIA is firstly employed to identify a subset of changes responsible for the failing test, then these changes are ranked according to the proposed heuristic. Finally, the tool utilizes an improved delta debugging algorithm to determine a minimal set of faulty changes.

7.5. Software measurement

To properly plan for maintenance tasks, some predictive metrics are needed, such as changeability, maintainability, maintenance cost and effort [112–117]. CIA can often help maintainers to make decision among various change solutions, prepare change schedule, estimate the resources, maintenance efforts and costs and trace the change effects. Some metrics, such as changeability,

DOI: 10.1002/stvr

change cost and change complexity, are often employed to help managers make decisions. For example, with several candidate changes satisfying the same changing requirement, the impact analysis information can be used to do trade-offs between alternative changes.

Chaumun *et al.* proposed an approach to measure changeability for object-oriented systems by computing the impacts of the changes made to classes of the system [111]. They firstly classify the changes according to the affected elements. Then, a change impact model is defined at the conceptual level and mapped to the C++ language. Based on the change impacts, they measure the changeability of the system. They can then determine whether the proposed modification is acceptable. The size of the modification, which is a key factor for maintenance effort and cost estimation, has been estimated for object oriented programs by Antoniol *et al.* [118]. The impacted classes of a change request are utilized to predict the size of the proposed modification.

There are also some work that utilizes ripple effect as software metric to measure software complexity, stability and coupling. The ripple effect metric shows the degree of impact of changes on the rest of the system. Black *et al.* conducted extensive studies on computation of ripple effect measures [8, 35]. The results of the ripple effect computation help with estimating the cost of the change, which can then help managers to evaluate the proposed changes or choose between alternative change proposals.

8. IMPLICATIONS

This article surveyed 23 different code-based CIA techniques distributed in 30 publications from 1997 to 2010. Despite so many publications on the CIA techniques, many open issues remain. There are still many fruitful areas of research, as follows:

- *Extensibility.* This article tries to offer a wide overview of the CIA techniques, which have been empirically evaluated. Many papers have not been included in the survey because they do not fit the selection criteria. For example, some study has proposed new CIA techniques that have not been validated empirically [29]. These CIA techniques may provide promising results, and may provide ideas to develop new CIA techniques. Therefore, to expand the scope of the survey is necessary in a future review.
- *Utility.* A comparative framework is proposed to characterize the properties of the code-based CIA techniques. This framework is found to be expressive and effective. On the one hand, existing CIA techniques can be compared under this framework. On the other hand, new CIA techniques can be developed using this framework, that is, novel CIA techniques can be developed by considering the properties in the framework.
- *Scalability.* Most of existing CIA techniques have been experimentally validated based on some open systems, which give some indication of whether the technique can scale. But general claims can not be obtained from the limited data provided by the researchers. Hence, it is difficult to assess the scalability of the CIA techniques. To properly assess the scalability of a CIA technique, one not only has to take into account the input size but also the required user involvement, and the required application environment.
- *Tool support.* Although some tools have been developed to support CIA, most of these tools are just prototypes. No CIA tools have been commercialized until now. Since CIA is increasingly pivotal, good tool support is in great need. A few tools such as *JRipples* [92] have approached maturity and stability. These tools should be fully evaluated, promoted and refined based on industrial use. Additional tools should be developed to support all aspects of CIA.
- *Common benchmark.* So far, only a few work provide a detailed analysis of the effectiveness of their CIA techniques and attempt some empirical comparisons with other techniques. Most CIA techniques cannot be compared with others because of the following: (1) they were performed on different cases, (2) they were presented in a non-compatible format (different input and output, static or dynamic), or (3) they were validated using different metrics. Hence, there is a strong need to compare the effectiveness of different CIA techniques with a common benchmark and under similar usage scenarios. To obtain better insights into the strengths and

DOI: 10.1002/stvr

weaknesses of the CIA techniques, it is advisable to validate them based on a common set of case studies and a common set of metrics.
- *Applications.* CIA plays a crucial role in software change analysis. Current CIA researches mainly focus on predicting the effects of the given changes. Only a small number of works study the applications of CIA. More studies should be devoted to this area. As discussed in Section 7, CIA can support various software maintenance tasks. In addition, some CIA techniques compute a ranked list of potentially impacted entities marked with the priority to be inspected. This result can be applied to software maintenance activities, such as comparing various change schemes, selecting regression testing cases and so on. Some work is needed to integrate CIA into the whole software maintenance process.

9. CONCLUSIONS

In this article, a survey of code-based CIA techniques is presented based on 30 selected publications from 1997 to 2010. Useful information has been extracted from these studies and synthesized against four defined research questions. The key contributions are listed as follows.

1. There are 23 empirically evaluated code-based CIA techniques published, which meet the defined criteria. Figure 4 shows the distribution of these 23 CIA techniques from four perspectives: software repository mining based, execution information based, traditional dependence analysis based and coupling measurement based.
2. A comparative framework for code-based CIA techniques is proposed. The objective of this framework is to support the identification and comparison of existing CIA techniques based on the specific needs of the user. This framework consists of seven key properties of CIA techniques: object, impact set, intermediate representation, type of analysis, language support, tool support and empirical evaluation. Any CIA technique can be distinguished based on this framework. On the one hand, researchers who aim to develop new CIA techniques may consider these properties; on the other hand, practitioners can select an appropriate CIA technique according to its properties.
3. CIA is found to be applied to many software maintenance activities as shown in Table XII. This table shows that CIA supports many critical activities such as software comprehension, change propagation, selection of regression test cases, debugging and measuring various software attributes. As software systems are naturally adapted and changed, a large part of effort and energy will be spent on software maintenance, which involves many cycles of changes to the system. Software CIA will become increasingly crucial in software evolution.
4. Finally, some fruitful areas of future research are presented: evaluating existing CIA techniques and proposing new CIA techniques under the proposed framework, developing more mature tools to support CIA, comparing current CIA techniques empirically with unified metrics and common benchmarks, and applying the CIA more extensively and effectively in the software maintenance phase.

ACKNOWLEDGEMENTS

This work is supported partially by the National Natural Science Foundation of China under Grant No. 60973149, partially by the open funds of State Key Laboratory of Computer Science of the Chinese Academy of Sciences under Grant No. SYSKF1110, partially by a doctoral fund of the Ministry of Education of China under Grant No. 20100092110022, and partially by the Scientific Research Foundation of Graduate School of Southeast University under Grant No. YBJJ1102.

The authors would also like to thank, in particular, Taweesup Apiwattanapong, Vaclav Rajlich, and Paolo Tonella, for their comments and discussions, which made the paper more understandable and stronger. Special thanks to the anonymous reviewers who provided useful suggestions on earlier versions of this paper.

DOI: 10.1002/stvr

REFERENCES

1. Li W, Henry S. Maintenance support for object-oriented programs. *Journal of Software Maintenance: Research and Practice* 1995; **7**(2):131–147.
2. Schneidewind NF. The state of software maintenance. *IEEE Transactions on Software Engineering* 1987; **13**(3):303–310.
3. Lehman MM, Belady LA. *Program Evolution: Processes of Software Change*. Academic Press: London, 1985.
4. Bohner S, Arnold R. *Software Change Impact Analysis*. IEEE Computer Society Press: Los Alamitos, CA, USA, 1996.
5. Bohner SA. Impact analysis in the software change process: a year 2000 perspective. *Proceedings of the International Conference on Software Maintenance*, Monterey, CA, USA, 1996; 42–51.
6. Rovegard P, Angelis L, Wohlin C. An empirical study on views of importance of change impact analysis issues. *IEEE Transactions on Software Engineering* 2008; **34**(4):516–530.
7. Turver RJ, Munro M. Early impact analysis technique for software maintenance. *Journal of Software Maintenance: Research and Practice* 1994; **6**(1):35–52.
8. Black S. Deriving an approximation algorithm for automatic computation of ripple effect measures. *Information and Software Technology* 2008; **50**(7-8):723–736.
9. Ren X, Shah F, Tip F, Ryder BG, Chesley O. Chianti: a tool for change impact analysis of Java programs. *Proceedings of the International Conference on Object Oriented Programming, Systems, Languages and Applications*, Vancouver, BC, Canada, 2004; 432–448.
10. Stevens W, Myers G, Constantine L. Structured design. *IBM Systems Journal* 1974; **13**(2):115–139.
11. Yau SS, Collofello JS, MacGregor T. Ripple effect analysis of software maintenance. *Proceedings of the International Conference on Computer Software and Applications*, Atlanta, Georgia, USA, 1978; 60–65.
12. Horowitz E, Williamson RC. SODOS: a software documentation support environment—its definition. *IEEE Transactions on Software Engineering* 1986; **12**(8):849–859.
13. Pfleeger SL, Bohner SA. A framework for software maintenance metrics. *Proceedings of the International Conference on Software Maintenance*, Washington, DC, 1990; 320–327.
14. Pfleeger SL, Lawrence S. *Software Engineering: The Production of Quality Software*. Macmillan Publishing Co.: New York, USA, 1991.
15. IEEE. The Institute of Electrical and Electronics Engineers, Inc., 1990. IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology.
16. IEEE. The Institute of Electrical and Electronics Engineers, Inc., 1998. IEEE Std 1219-1998: IEEE Standard for Software Maintenance.
17. Knethen A, Grund M. QuaTrace: a tool environment for (semi-) automatic impact analysis based on traces. *Proceedings of the International Conference on Software Maintenance*, Amsterdam, Netherlands, 2003; 246–255.
18. De-Lucia A, Fasano F, Oliveto R. Traceability management for impact analysis. *Proceedings of the International Conference on Software Maintainence*, Beijing, China, 2008; 21–30.
19. Law J, Rothermel G. Incremental dynamic impact analysis for evolving software systems. *Proceedings of the International Symposium on Software Reliability Engineering*, Denver, CO, USA, 2003; 430–441.
20. Law J, Rothernel G. Whole program path-based dynamic impact analysis. *Proceedings of the International Conference on Software Engineering*, Portland, Oregon, USA, 2003; 308–318.
21. Hewitt J, Rilling J. A light-weight proactive software change impact analysis using use case maps. *Proceedings of the International Workshop on Software Evolvability*, Budapest, Hungary, 2005; 41–46.
22. Shiri M, Hassine J, Rilling J. A requirement level modification analysis support framework. *Proceedings of the International Workshop on Software Evolvability*, Maison Internationale, Paris, France, 2007; 67–74.
23. Briand LC, Labiche Y, Sullivan LO. Impact analysis and change management of UML models. *Proceedings of the International Conference on Software Maintenance*, Amsterdam, Netherlands, 2003; 256–265.
24. Briand LC, Labiche Y, O'Sullivan L, Swka MM. Automated impact analysis of UML models. *Journal of Systems and Software* 2006; **79**(3):339–352.
25. OCL—Object Constraint Language, 2010. Available from: http://www.omg.org/spec/OCL/2.2.
26. Use Case Maps, 2004. Available from: http://www.usecasemaps.org.
27. Briand LC, Wust J, Lounis H. Using coupling measurement for impact analysis in object-oriented systems. *Proceedings of the International Conference on Software Maintenance*, Oxford, England, UK, 1999; 475–482.
28. Orso A, Harrold MJ. Leveraging field data for impact analysis and regression testing. *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, Helsinki, Finland, 2003; 128–137.
29. Tonella P. Using a concept lattice of decomposition slices for program understanding and impact analysis. *IEEE Transactions on Software Engineering* 2003; **29**(6):495–509.
30. Apiwattanapong T, Orso A, Harrold MJ. Efficient and precise dynamic impact analysis using execute-after sequences. *Proceedings of the International Conference on Software Engineering*, St. Louis, Missouri, USA, 2005; 432–441.
31. Bohner S, Arnold R. Impact analysis—towards a framework for comparison. *Proceedings of the International Conference on Software Maintenance*, Montréal, Quebec, Canada, 1993; 292–301.

DOI: 10.1002/stvr

32. Breech B, Tegtmeyer M, Pollock L. A comparison of online and dynamic impact analysis algorithms. *Proceedings of the European Conference on Software Maintenance and Reengineering*, Manchester, UK, 2005; 143–152.
33. Hattori L, Guerrero D, Figueiredo J, Brunet J, Damsio J. On the precision and accuracy of impact analysis techniques. *Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science*, Portland, Oregon, 2008; 513–518.
34. Orso A, Apiwattanapong T, Law J, Rothermel G, Harrold MJ. An empirical comparison of dynamic impact analysis algorithms. *Proceedings of the International Conference on Software Engineering*, Edinburgh, Scotland, UK, 2004; 491–500.
35. Black S. Computing ripple effect for software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice* 2001; **13**(4):263–279.
36. Mansour N, Salem H. Ripple effect in object oriented programs. *Journal of Computational Methods in Sciences and Engineering* 2006; **6**(5-6(sup 1)):23–32.
37. Bohner SA. Software change impacts—an evolving perspective. *Proceedings of the International Conference on Software Maintenance*, Montréal, Canada, 2002; 263–272.
38. Hattori L, Guerrero D, Figueiredo J, Brunet J, Damasio J. On the precision and accuracy of impact analysis techniques. *Proceedings of the Seventh International Conference on Computer and Information Science*, Portland, Oregon, USA, 2008; 513–518.
39. Poshyvanyk D, Marcus A, Ferenc R, Gyimothy T. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering* 2009; **14**(1):5–32.
40. Sun XB, Li BX, Tao CQ, Wen WZ, Zhang S. Change impact analysis based on a taxonomy of change types. *International Conference on Computer Software and Applications*, Seoul, Korea, 2010; 373–382.
41. Baeza-Yates RA, Ribeiro-Neto B. *Modern Information Retrieval*. Addison–Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1999.
42. Biggerstaff TJ, Mitbander BG, Webster D. The concept assignment problem in program understanding. *Proceedings of the 15th International Conference on Software Engineering*, Baltimore, Maryland, USA, 1993; 482–498.
43. Dit B, Revelle M, Gethers M, Poshyvanyk D. Feature location in source code: a taxonomy and survey. *Journal of Software Maintenance and Evolution: Research and Practice* 2011. DOI: 10.1002/smr.567.
44. Rajlich V, Wilde N. The role of concepts in program comprehension. *Proceedings of the 10th International Workshop on Program Comprehension*, Paris, France, 2002; 271–278.
45. Kitchenham BA. Guidelines for performing systematic literature review in software engineering. *Technical Report*, Keele University, 2007.
46. Kitchenham BA, Brereton OP, Budgen D, Turner M, Bailey J, Linkman S. Systematic literature reviews in software engineering—a systematic literature review. *Information and Software Technology* 2009; **51**(1):7–15.
47. Dieste O, Grimán A, Juristo N. Developing search strategies for detecting relevant experiments. *Empirical Software Engineering* 2009; **14**:513–539.
48. Brereton P, Kitchenham BA, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of System and Software* 2007; **80**:571–583.
49. Ren X, Chesley O, Ryder BG. Identifying failure causes in Java programs: an application of change impact analysis. *IEEE Transactions on Software Engineering* 2006; **32**(9):718–732.
50. Zheng J, Williams L, Robinson B. Pallino: automation to support regression test selection for COTS-based applications. *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, Atlanta, Georgia, USA, 2007; 224–233.
51. Ryder BG, Tip F. Change impact analysis for object oriented programs. *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, Snowbird, Utah, USA, 2001; 46–53.
52. Zhang S, Gu Z, Lin Y, Zhao JJ. Change impact analysis for AspectJ programs. *Proceedings of the International Conference on Software Maintenance*, Beijing, China, 2008; 87–96.
53. Agrawal H, Horgan J. Dynamic program slicing. *Proceedings of the International Conference on Programming Language Design and Implementation*, White Plains, New York, 1990; 246–256.
54. Xu G, Rountev A. AJANA: a general framework for source-code-level interprocedural dataflow analysis of AspectJ software. *Proceedings of the 7th International Conference on Aspect-Oriented Software Development*, Brussels, Belgium, 2008; 36–47.
55. Zimmermann T, Weisgerber P, Diehl S, Zeller A. Mining version histories to guide software changes. *Proceedings of the 26th International Conference on Software Engineering*, Edinburgh, Scotland, United Kingdom, 2004; 563–572.
56. Badri L, Badri M, Yves SD. Supporting predictive change impact analysis: a control call graph based technique. *Proceedings of the Asia-Pacific Software Engineering Conference*, Taipei, Taiwan, China, 2005; 167–175.
57. Ramanathan MK, Grama A, Jagannathan S. Sieve: a tool for automatically detecting variations across program versions. *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, Tokyo, Japan, 2006; 241–252.
58. Breech B, Tegtmeyer M, Pollock L. Integrating influence mechanisms into impact analysis for increased precision. *Proceedings of the International Conference on Software Maintenance*, Philadelphia, Pennsylvania, USA, 2006; 55–65.

DOI: 10.1002/stvr

59. Canfora P, Cerulo L. Impact analysis by mining software and change request repositories. *Proceedings of the International Software Metrics Symposium*, Como, Italy, 2005; 29–38.
60. Huang LL, Song YT. Precise dynamic impact analysis with dependency analysis for object-oriented programs. *Proceedings of the International Conference on Advanced Software Engineering and Its Applications*, Hainan Island, China, 2008; 217–220.
61. Beszedes A, Gergely T, Farago S, Gyimothy T, Fischer F. The dynamic function coupling metric and its use in software evolution. *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, Amsterdam, Netherlands, 2007; 103–112.
62. Jashki MA, Zafarani R, Bagheri E. Towards a more efficient static software change impact analysis method. *Proceedings of the International Workshop on Principles of Software Evolution*, Lisbon, Portugal, 2005; 81–90.
63. Hattori L, Santos G D Jr, Cardoso F, Sampaio M. Mining software repositories for software change impact analysis: a case study. *Proceedings of the Brazilian Symposium on Databases*, Campinas, Brazil, 2008; 210–223.
64. Sherriff M, Williams L. Empirical software change impact analysis using singular value decomposition. *Proceedings of the International Conference on Software Testing, Verification, and Validation*, Lillehammer, Norway, 2008; 268–277.
65. Petrenko M, Rajlich V. Variable granularity for improving precision of impact analysis. *Proceedings of the International Conference on Program Comprehension*, Vancouver, BC, Canada, 2009; 10–19.
66. Kagdi H, Gethers M, Poshyvanyk D, Collard ML. Blending conceptual and evolutionary couplings to support change impact analysis in source code. *Proceedings of the IEEE Working Conference on Reverse Engineering*, Beverly, MA, USA, 2010; 119–128.
67. Torchiano M, Ricca F. Impact analysis by means of unstructured knowledge in the context of bug repositories. *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Bolzano/Bozen, Italy, 2010; 47:1–47:4.
68. Ceccarelli M, Cerulo L, Canfora G, Penta MD. An eclectic approach for change impact analysis. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, Cape Town, South Africa, 2010; 163–166.
69. Sun XB, Li BX, Tao CQ, Wen WZ, Zhang S. Change impact analysis based on a taxonomy of change types. *International Conference on Computer Software and Applications*, Seoul, Korea, 2010; 373–382.
70. Gethers M, Poshyvanyk D. Using relational topic models to capture coupling among classes in object-oriented software systems. *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, Timisoara, Romania, 2010; 1–10.
71. Ahsan SN, Wotawa F. Impact analysis of SCRs using single and multi-label machine learning classification. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, Bolzano/Bozen, Italy, 2010; 1–4.
72. Breech B, Danalis A, Shindo S, Pollock L. Online impact analysis via dynamic compilation technology. *Proceedings of the International Conference on Software Maintenance*, Chicago Illinois, USA, 2004; 453–457.
73. Zimmermann T, Zeller A, Weissgerber P, Diehl S. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 2005; **31**(6):429–445.
74. Canfora G, Cerulo L. Fine grained indexing of software repositories to support impact analysis. *Proceedings of the International Workshop on Mining Software Repositories*, Shanghai, China, 2006; 105–111.
75. Canfora G, Ceccarelli M, Cerulo L, Penta MD. Using multivariate time series and association rules to detect logical change coupling: an empirical study. *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, Timisoara, Romania, 2010; 1–10.
76. Diehl S, Gall RC, Hassan AE. Guest editors introduction: special issue on mining software repositories. *Empirical Software Engineering* 2009; **14**(3):257–261.
77. Hassan AE. The road ahead for mining software repositories. *Proceedings of the Frontiers of Software Maintenance*, Beijing, China, 2008; 48–57.
78. Zhao JJ. Change impact analysis for aspect-oriented software evolution. *Proceedings of the International Workshop on Principles of Software Evolution*, Orlando, Florida, USA, 2002; 108–112.
79. Huang L, Song YT. Dynamic impact analysis using execution profile tracing. *Proceedings of the International Conference on Software Engineering Research, Management and Applications (SERA 2006)*, Seattle, Washington, USA, 2006; 237–244.
80. Cavallaro L, Monga M. Unweaving the impact of aspect changes in AspectJ. *Proceedings of the Workshop on Foundations of Aspect-Oriented Languages*, Charlottesville, Virginia, USA, 2009; 13–18.
81. Hammer P, Snelting G. An improved slicer for Java. *Proceedings of the ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, Washington, DC, USA, 2004; 17–22.
82. Horwitz S, Reps T, Binkley D. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems* 1990; **12**(1):27–60.
83. Li BX, Fan XC, Pang J, Zhao JJ. A model for slicing Java programs hierarchically. *Journal of Computer Science & Technology* 2004; **19**(6):848–858.
84. Tip F. A survey of program slicing techniques. *Technical Report*, Centre for Mathematics and Computer Science (CWI), 1994.
85. Weiser M. Program slicing. *Proceedings of the International Conference on Software Engineering*, 1981; 439–449.

DOI: 10.1002/stvr

86. Agrawal H, Horgan JR. Dynamic program slicing. *Proceedings of Conference on Programming Language Design and Implementation*, White Plains, New York, 1990; 246–256.
87. Mund GB, Mall R. An efficient interprocedural dynamic slicing method. *Journal of Systems and Software* 2006; **79**(6):791–806.
88. Zhang XY, Gupta R, Zhang YT. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams. *Proceedings of the International Conference on Software Engineering*, Edinburgh, Scotland, UK, 2004; 502–511.
89. Gupta R, Soffa ML. Hybrid slicing: an approach for refining static slices using dynamic information. *Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering*, Washington, DC, USA, 1995; 29–40.
90. Gupta R, Soffa ML, Howard J. Hybrid slicing: integrating dynamic information with static analysis. *ACM Transactions on Software Engineering and Methodology* 1997; **6**(4):370–397.
91. Briand LC, Daly J, Wst J. A unified framework for coupling measurement in object oriented systems. *IEEE Transactions on Software Engineering* 1999; **25**(1):91–121.
92. Buckner J, Buchta J, Petrenko M, Rajlich V. JRipples: a tool for program comprehension during incremental change. *Proceedings of the International Workshop on Program Comprehension*, St. Louis, MO, USA, 2005; 149–152.
93. Lee M, Offutt AJ, Alexander RT. Algorithmic analysis of the impacts of changes to object-oriented software. *Proceedings of Technology of Object-Oriented Languages and Systems*, St. Malo, France, 2000; 61–70.
94. Ferenc R, Beszdes A, Tarkiainen M, Gyimthy T. Columbus—reverse engineering tool and schema for C++. *Proceedings of the International Conference on Software Maintenance*, Montréal, Canada, 2002; 172–181.
95. Canfora P, Cerulo L. Jimpa: an eclipse plug-in for impact analysis. *Proceedings of the Conference on Software Maintenance and Reengineering*, Bari, Italy, 2006; 341–342.
96. Poshyvanyk D, Marcus A. The conceptual coupling metrics for object-oriented systems. *Proceedings of the International Conference on Software Engineering Research, Management and Applications*, Busan, Korea, 2006; 469–478.
97. Kagdi H. Improving change prediction with fine-grained source code mining. *Proceedings of the International Conference on Automated Software Engineering*, Minneapolis, Minnesota, USA, 2007; 559–562.
98. Gwizdala S, Jiang Y, Rajlich V. JTracker—a tool for change propagation in Java. *Proceedings of the European Conference on Software Maintenance and Reengineering*, Benevento, Italy, 2003; 223–229.
99. Hassan AE, Gall RC. Replaying development history to assess the effectiveness of change propagation tools. *Empirical Software Engineering* 2006; **11**(3):335–367.
100. Rajlich V, Gosavi P. Incremental change in object-oriented programming. *Empirical Software Engineering* 2004; **21**(4):62–69.
101. Hassan AE, Holt RC. Predicting change propagation in software systems. *Proceedings of the International Conference on Software Maintenance*, Chicago Illinois, USA, 2004; 284–293.
102. Malik H, Hassan AE. Supporting software evolution using adaptive change propagation heuristics. *Proceedings of the International Conference on Software Maintenance*, Beijing, China, 2008; 177–186.
103. Ren XX. Change impact analysis for Java programs and applications. *PhD Thesis*, Rutgers University, 2007.
104. Zhang S, Gu Z, Lin Y, Zhao JJ. Celadon: a change impact analysis tool for aspect-oriented programs. *Proceedings of the International Conference on Software Engineering*, Leipzig, Germany, 2008; 913–914.
105. Martin P, Xie P. Automated test generation for access control policies via change-impact analysis. *Proceedings of the International Workshop on Software Engineering for Secure Systems*, Washington, DC, USA, 2007.
106. Wang D, Li BX, Cai J. Regression testing of composite service: an XBFG-based approach. *Proceedings of the Congress on Services*, Honolulu, Hawaii, USA, 2008; 112–119.
107. Pourgalehdari O, Zamli KZ, MatIsa NA. A meta level dynamic approach to visualize impact analysis for regression testing. *Proceedings of the International Conference on Computer and Communication Engineering*, Kuala Lumpur, Malaysia, 2008; 928–931.
108. Chesley OC, Ren XX, Ryder BG. Crisp: a debugging tool for Java programs. *Proceedings of the International Conference on Software Maintenance*, Budapest, Hungary, 2005; 401–410.
109. Zhang S, Gu Z, Lin Y, Zhao JJ. AutoFlow: an automatic debugging tool for AspectJ software. *Proceedings of the International Conference on Software Maintenance*, Beijing, China, 2008; 470–471.
110. Fluri B, Gall HC. Classifying change types for qualifying change couplings. *Proceedings of the International Conference on Software Engineering*, Minneapolis, Minnesota, USA, 2007; 97–98.
111. Chaumun MA, Kabaili H, Keller RK, Lustman F. A change impact model for changeability assessment in object-oriented software systems. *Proceedings of the European Conference on Software Maintenance and Reengineering*, Chapel of St. Agnes, Amsterdam, Netherlands, 1999; 130–138.
112. Fioravanti F, Nesi P. Estimation and prediction metrics for adaptive maintenance effort of object-oriented systems. *IEEE Transactions on Software Engineering* 2001; **27**(12):1062–1084.
113. Grover PS, Kumar R, Kumar A. Measuring changeability for generic aspect-oriented systems. *ACM SIGSOFT Software Engineering Notes* 2008; **33**(6):1–5.
114. Kafura D, Reddy GR. The use of software complexity metrics in software maintenance. *IEEE Transactions on Software Engineering* 1987; **13**(3):335–343.

DOI: 10.1002/stvr

115. De-Lucia A, Pompella E, Stefanucci S. Effort estimation for corrective software maintenance. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, 2002; 409–416.
116. Nesi P, Querci T. Using metrics to evaluate software system maintainability. *Journal of Systems and Software* 1998; **42**(1):89–102.
117. Schneidewind NF. The state of software maintenance. *IEEE Transactions on Software Engineering* 1987; **13**(3):303–310.
118. Antoniol G, Canfora G, De-Lucia A, Lucarelli PB, Roma P. Estimating the size of changes for evolving object-oriented systems: a case study. *Proccedings of the Sixth International Symposium on Software Metrics*, Boca Raton, FL, USA, 1999; 250–259.

DOI: 10.1002/stvr