INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING & COMPUTER SYSTEMS (IJSECS)
ISSN: 2289-8522 e-ISSN: 2180-0650
VOL. 7, ISSUE 1, 57 – 66
DOI: https://doi.org/10.15282/ijsecs.7.1.2021.6.0082

ORIGINAL ARTICLE

Improving the Accuracy of Static Source Code Based Software Change Impact Analysis Through Hybrid Techniques: A Review

S.R. Yusuff[1], A.O. Bajeh[2], T.O. Aro[3], K.S. Adewole[4]

[1]Department of Computer Sciece, Kwara State University, Malete, Kwara State, Nigeria.
[2,4]Department of Computer Science,University of Ilorin, Ilorin, Nigeria.
[3]Department of Mathematical and Computing Sciences,KolaDaisi University, Ibadan, Oyo State, Nigeria

ABSTRACT – Change is an inevitable phenomenon of life. This inevitability of change in the real world has made software change an indispensable characteristic of software systems and a fundamental task of software maintenance and evolution. Changes to software may arise as a result of feature enhancement requests, bug fixes, technological advancements amongst others. The continuous evolution process of software systems can greatly affect the quality and reliability of such systems, if proper mechanisms to manage them are not adequately provided. Consequently, Software change Impact Analysis (CIA) has been identified as an approach to help address the problem. CIA is an essential activity for comprehending and identifying the impacts of potential software as a way of preventing the system from entering into an erroneous state. A good CIA technique, is one which helps to reduce maintenance costs. Hybrid CIA technique is a blend of multiple CIA techniques. A number of hybrid CIA techniques have been proposed by researchers in the literature. However, there has been no study that reviewed Hybrid CIA techniques holistically. The paper tries to fill this gap by presenting a summary of the methods and techniques so far adopted in code based Hybrid CIA techniques with a view to suggesting possible future directions. A number of literature including journal articles, conference proceedings, and workshop papers published between 2009 and 2019 related to the topic were reviewed. The following themes were employed in the analysis of the review based on their mention in most of the reviewed literature: size and type of subject software systems; level of granularity; CIA techniques and methods; and evaluation metrics. The results from the review, reveals that a combination of a minimum of two CIA techniques is sufficient to gain improved performance. Likewise, hybrid CIA techniques have always shown significant improvement in performance, over baseline technique. However, comparison of existing hybrid CIA techniques, in terms of performance, is yet to be carried out. In addition, findings from the paper, isolated Latent Semantic Indexing (LSI) as the main method utilized for analyzing textual source code data despite advancement in the field of Information Retrieval (IR). The paper further highlights areas for future research to include a performance evaluation of existing hybrid CIA techniques. To achieve this, it is proposed to have a universal benchmark source code dataset of different programming languages, size and scope. Furthermore, it is necessary to try out other categories of IR models such as Latent Dirichlet Allocation (LDA) topic model and those based on deep learning techniques like doc2vec. It would also be a good way forward, if other possible CIA combinations can be implemented, particularly in the aspect of utilizing the syntactic and semantic information inherent in source code to achieve a holistic source code CIA.

ARTICLE HISTORY
Revised: 02 Jan 2021
Accepted: 07 Apr 2021

KEYWORDS
Software change, Software maintenance, Software evolution, Hybrid Change Impact Analysis, IR models, Topic model

INTRODUCTION

Change is inevitable in software engineering and maintenance [1][2]. A majority of modern day software systems are models of the real world which is continually changing. As a result, software systems must continuously evolve with the changing reality being modeled or risk becoming less useful and obsolete [3]. Typically, software systems evolve to adapt to new user requirements, enhance functionality and performance, changing technology and business models, fix existing faults amongst others [4]. Hence, software change has been identified as an indispensable characteristic and a fundamental element of software maintenance and evolution [5] The task of implementing changes in evolving software systems is non-trivial due to the size and complexity of software systems [6]. Therefore, it becomes necessary to effectively make an assessment of proposed changes before carrying out such changes, so as to prevent undesirable and erratic behavior of the system that can be caused by unidentified ripple effects of changes. Software Change Impact analysis (CIA) is the approach used to address this problem of change implementation [7]. A number of definitions for CIA has been given in

*CORRESPONDING AUTHOR | | ✉ taiwo774@gmail.com

the literature. Noteworthy of mention, is a widely used definition of CIA defined as " the process of identifying the potential consequences of a change, or estimates what needs to be modified to accomplish a change" [8].

The CIA process is an iterative one that begins by identifying the initial location in the software components that implements some functionality in the software based on a change request [9]. This initial step can be done through some feature location techniques. A comprehensive survey of feature location techniques on source code data can be found in [10]. The result of the initial step is the identification of direct impacts due to software change, known as the change set or the Starting Impact Set (SIS) [9][11]. Consequently, indirect impacts that are probably affected by the components in the SIS are further estimated by the CIA technique. The resulting impact set is called the Estimated Impact Set (EIS) or Candidate Impact Set (CIS). However, not all components in the EIS are actually modified as a result of a proposed change; thus, the false set of components in the EIS is called the False Positive Impact Set (FPIS). The FPIS represents an over-estimate of impacts. Also, not all truly affected software components (called the Actual Impact Set (AIS)) are included in the EIS. This underestimate of impacts not reflected in the EIS is called the False Negative Impact Set (FNIS). The relationships between all the sets identified above are given in equations 1, 2 and 3 while a summary of the CIA process is depicted in Figure 1.

$$AIS = EIS \cup FNIS - FPIS \quad (1)$$

$$FPIS = EIS - (AIS \cap EIS) \quad (2)$$

$$FNIS = AIS - (AIS \cap EIS) \quad (3)$$

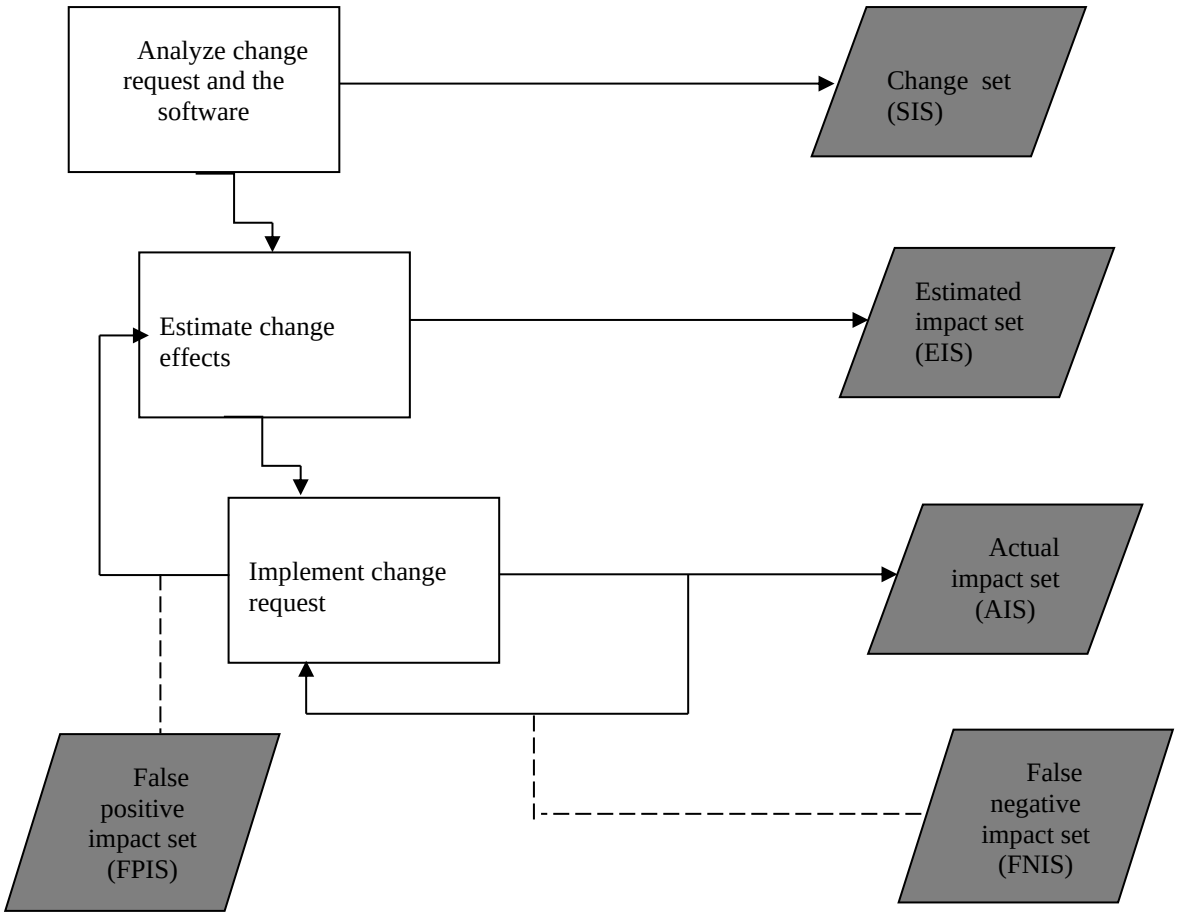

Figure 1. The Change Impact Analysis Process [9][11]

CIA techniques have been classified based on underlying analysis [8][9][11] and scope [12] which refers to the type of software artifact utilized. CIA classification based on underlying analysis is identified as traceability; dependency and experiential analysis while CIA techniques based on scope is given as source code; formal models; miscellaneous artifacts and combined scope [11] [12]

Traceability analysis, seeks to identify relationships between software artifacts at different scopes such as tracing the implementation of a source code component to its underlying design or requirement model. Dependency analysis on the other hand, identifies the relationship between various components of a single software artifact. For example, dependency analysis can be used to identify the relationship between a method A and other methods in a source code [9]. Experiential

analysis relies on informal methods such as the expertise of project team members, discussions and meetings [11]. For a comprehensive review of existing CIA techniques in each classification, the reader is referred to [11][12].

[12] reveals that about 65% of CIA techniques are based on analysing source code for identifying change impacts. Source code artifact, when compared to other artifacts like formal models, documentation and configuration files, is available, reliable and complete, since it analyzes implementation details [9][11]. As such, a number of dependency analysis techniques based on analyzing source code have continued to emerge in the field. Dependency analysis approaches are further classified as dynamic or static analysis based on the method utilized for obtaining the information needed to identify change impacts [13][14][15]. Dynamic analysis requires that program information be collected during runtime for estimating change impacts and as such, incurs a high overhead cost. Also, their impact set often misses a lot of truly affected entities, since not all program inputs are considered during analysis [9][11][12][16]. Furthermore, dynamic analysis can be carried out offline or online depending on whether the analysis is performed after or during the program execution respectively [11] [12]. On the contrary, static analysis is carried out on the program when it is not in execution and thus, considers all possible program behaviors. Consequently, the resulting change impacts often include too many false positives [9][11][12][16]. Static CIA techniques are further classified as structural, textual or evolutionary analysis depending on whether they analyze the structural, textual or historical information residing in software repositories respectively [14][17][18].

Structural analysis has been identified as the origin of a vast number of static CIA techniques [19]. They analyze the interactions and relationships among source code components such as classes and methods in software systems. Existing structural analysis techniques have utilized concepts like call graphs, program slicing and structural coupling metrics to capture relationships between software components [20][21][22][23][24].

Textual analysis serves as a good complement to structural analysis by providing a way to identify conceptual dependencies in software. They utilize the textual source code information such as comments and identifier names which reflect the problem solution of the software [6]. CIA techniques utilizing textual source code information have relied on the use of Natural Language Processing (NLP) and Information Retrieval (IR) techniques such as Latent Semantic Indexing (LSI) [16][25][26] and Relational Topic Models (RTM) [27] for identifying the conceptual dependencies or couplings in software.

Evolutionary analysis identifies co-change couplings in software repositories such as version control systems, bug repositories, mailing lists to mention a few [28]. The premise for this is that software entities that have frequently changed together in the past are likely candidates for sharing a relationship. Evolutionary analysis is carried out by mining information from multiple versions of the software in software repositories hence, the name Mining software Repositories (MSR) [29]. Data mining techniques such as Association rule and item set mining have been employed for identifying co-changing components in software [28][29][30]. A taxonomy of existing CIA techniques derived from the CIA classification in [8][9][11] is depicted in Figure 2.

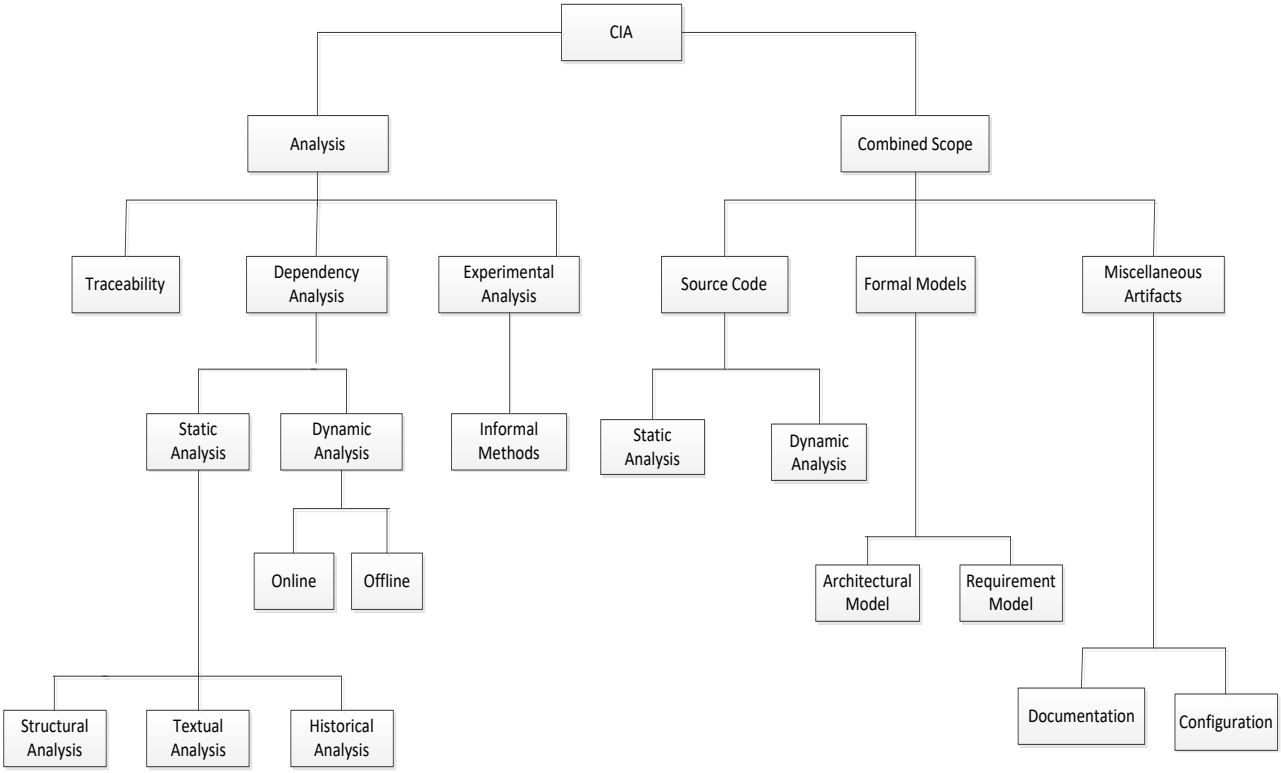

Figure 2. A Taxonomy of CIA Techniques

Regardless of the underlying analysis and scope of the CIA, the overarching goal of the CIA is to achieve an EIS that is as close as possible to the AIS. The precision and recall metrics from the Information retrieval domain have been used extensively for evaluating the performance of CIA techniques [9][11][21][26]. In the context of software CIA, the precision metric measures how much of the EIS correlates with the AIS while the recall metric is a measure of how complete the EIS is. The precision and recall metrics are given in equations 4 and 5.

$$Precision = \frac{|\text{AIS} \cap \text{EIS}|}{\text{EIS}} \tag{4}$$

$$Recall = \frac{|\text{AIS} \cap \text{EIS}|}{\text{AIS}} \tag{5}$$

While some CIA techniques overestimate the level of change impact thereby producing false positives, others underestimate them thereby producing false negatives. Thus, achieving a good balance between precision and recall is highly desired of any valuable CIA technique in order to reduce the efforts of software maintainers as well as to ensure that all impacts are considered. As such, the F-measure metric is also employed to avoid optimizing for either precision or recall [18]. This is defined as the harmonic mean of precision and recall and is given as:

$$F - measure = \frac{2 \times \text{Precision} \times \text{Recall}|}{\text{Precion} + \text{Recall}} \tag{6}$$

Accordingly, studies have proposed hybrid CIA techniques which are a blend of two or more CIA techniques utilizing the same or different analysis approach. Results from these studies have shown that a combination of multiple CIA techniques can improve performance by complementing one another [16][19] [28][29][31]. These promising results are the motivation for the current paper on hybrid CIA techniques with a view to not only discussing the state of art in code based hybrid CIA techniques but also identifying areas for future studies that could bring about further improvement. This will go a long way in ensuring that efforts spent on maintaining software is significantly reduced while ensuring that all impacts of a proposed change are considered.

The rest of the paper is organized as follows: Section 2 presents the methodology adopted for the review. Section 3 explores existing hybrid CIA techniques with emphasis on static source code techniques while section 4 presents discussion on the findings of the paper. Finally, the conclusion and future direction is given in section 5.

RESEARCH METHOD

This section describes the research methodology and the review process of the paper. The following research questions were formulated for the review with a view to summarizing the state of art in static code based hybrid CIA and also give recommendations for future research.

RQ1 What number of techniques have been combined for code based hybrid CIA*?*
RQ2 What types of techniques and combination strategies have been utilized thus far for code based Hybrid CIA?
RQ3 What metrics have been used to evaluate the performance of code based hybrid techniques?
RQ4 What is the future direction for code based hybrid CIA?

To address the research questions listed above, the study adopted a three-stage process proposed in [9]. The steps are given as follows:

1. Literature Search: The literature search was carried out on the following electronic databases and search engine with an extensive reportage on the topic:

- IEEE Computer Society Digital Library (http://ieeexplore.ieee.org/)
- Elsevier (http://www.sciencedirect.com/)
- ACM Digital Library (http://portal.acm.org/)
- Academia.com
- GoogleScholar

The following keywords were employed in the initial search: ‘impact analysis’ OR ‘static based impact analysis’ OR ‘source code impact analysis’ OR ‘hybrid impact analysis’. A total of 654 papers were found from the initial search which was indexed using the title, abstract and keywords.

2. Selection of relevant study: Next, the search result from the first step was refined based on the specified exclusion and inclusion criteria. The following exclusion criteria were utilized on the search result:

1. The research utilized only a single technique
2. The research combined other CIA techniques that excluded source code
3. The research did not include any static analysis.

At the end of this step, only about 48 papers were left in the selection process. A further refinement was carried out using the following inclusion criteria:

1. The research utilized more than one technique.
2. The research included at least one static source code analysis technique
3. The research carried out empirical evaluation of the CIA technique.

A total of 6 studies were found to meet the criteria and were selected for the review.

3. Extraction of relevant information: Detailed study of the selected papers was carried out and relevant information was extracted. Table 1 gives a summary of the selected papers classified based on the literature type.

Table 1. Summary of selected papers.

Paper type	Description	Author/Year	Type of Analysis	Techniques used	Empirical evaluation
Conference proceedings	Working Conference on Reverse Engineering (WCRE)	[28]	Textual and Historical	Latent Semantic Analysis (LSI) and item set mining were used.	Precision and Recall metric
	European Conference on software maintenance and Reengineering (CSMR)	[19]	Structural and dynamic	Program Graph and Execute After Sequence (EAS)	Precision and Recall
	International Conference on software Engineering (ICSE)	[29]	Textual, dynamic and historical	LSI, Execution traces and item set mining	Precision and Recall
Journal	Advances in Engineering Software	[20]	Structural	Concept lattice and call graph	Precision and Recall
	Journal of computers and application	[25]	Structural and textual	LSI and method dependencies	Precision and Recall
	Scientific programming	[16]	Textual	LSI and doc2vec	Precision, Recall and F-measure

RELATED WORK

This section presents a review of code based hybrid CIA techniques. However, it is imperative to mention previous reviews on CIA to have an understanding of what has been done on the topic.

[12] presented a comprehensive investigation of the CIA based on a literature review and taxonomy for CIA. A summary and classification of about 150 CIA approaches was presented based on the taxonomy. The CIA techniques were analyzed based on the scope of the artifact. Findings from the study revealed that about 65% of the reviewed papers were based on analyzing source code changes, 11% analyzed change impacts on software architectures, 7% analyzed requirement documents, 4% analyzed miscellaneous artifacts such as documentation and configuration files while 13% combined two or more software artifacts in their analysis. The study also showed that about 33% of all approaches were not empirically validated. The study however included only those techniques that existed before the year 2012. With advancement in the field, there is a need for more studies to be carried out. In another study, [9] presented a review of 23 code based change impact analysis techniques. A comparative framework based on seven properties was proposed for characterizing CIA techniques and areas of further research was suggested. Although the study is closely related to the current paper, it did not extend beyond the year 2010 with no emphasis on hybrid techniques.

More recently, [11] presented a systematic review of the recent development, techniques and tools for the CIA. A total of 33 papers were considered which cut across traceability analysis, dependency analysis and dynamic analysis. Results from the study revealed that traceability and dependency analysis were the most worked on approaches. However, the focus of the paper was not on source code or hybrid CIAtechniques.

The current paper differs from previous reviews in that it focuses on CIA techniques that have combined two or more techniques, particularly those having at least one static source code analysis in them. Moreso, recent CIA techniques not covered in some of the previous preview are considered.

An early attempt at combining multiple CIA techniques is found in the study by [28] which combined textual and evolutionary approaches for CIA. Their approach utilized an Information retrieval technique known as Latent Semantic Indexing on textual source code data to derive conceptual coupling from a single version of the source code while evolutionary couplings were derived from source code commits in multiple versions of the software using item set mining. Results of their experiment on five open source software systems showed better performance over the baseline techniques particularly when disjunctive combination was used. An improvement of up to 20% and 45% in recall was reported over baseline IR and evolutionary techniques respectively.

[19] also proposed a hybrid CIA technique based on a combination of static and dynamic analysis. The technique utilized program graphs extracted from the source code in identifying structural dependencies and Execute After sequence (EAS) from software traces to identify dynamic dependencies between source code entities. The final change impacts were ranked based on dynamic dependencies. Evaluation of the technique was carried out on five software systems at the method and class level of granularity. Results of their experiment were compared with baseline static and dynamic analysis techniques using the precision and recall metrics. An improvement in recall values of between 90 and 115% over baseline static structural technique and between 21.2% and 39% over baseline dynamic technique was observed. However, precision scores did not match up to that of baseline structural analysis technique although not worse than that of dynamic analysis. It was also observed that the use of dynamic analysis incurred additional time overhead resulting from collecting run- time information.

Another study in this direction is found in [29] which proposed a scenario-driven CIA approach based on a combination of IR, dynamic analysis and evolutionary analysis. Their technique utilized LSI on a single version of textual source code data based on some change request. The technique configured the best-fit combination should additional sources of information such as execution traces and source code commits be available. Results of their experiments on four open source systems showed that a combination of all three CIA approaches yielded the best result when compared to a combination of any two techniques. Nonetheless, all forms of combination performed better than the individual techniques with improvements in precision and recall values of about 17% and 41% respectively.

The study [20] combined concept lattice and call graph at the class and method level of granularity to obtain a ranked list of change impacts. Concept lattice was used to predict class level changes and call graph was then applied on method level changes obtained thereafter. A disjunctive combination method was used to obtain the final change impacts. Results of the experiments carried out on four real-world programs showed an improvement of between 18 and 20% in precision when compared with results from either concept lattice or call graph without severely decreasing recall. However, more time overhead was incurred.

[25] proposed a new conceptual coupling metric based on the analysis of structural and semantic relationships between methods and classes of object-oriented systems. The technique augmented the conceptual coupling metric based on LSI presented in [26] with method dependencies and their direct dependencies along with the cosine similarity of their LSI representation. Results from experiments on six subject systems showed that the metric was able to capture aspects of coupling among classes not captured by existing structural and conceptual coupling metrics. Also, the new metric also showed a better performance on average in ranking classes required to effect a change as well as in tasks of fault proneness and software maintainability.

In [16], an approach to support the CIA that integrates multiple IR techniques was proposed. A combination of LSI and doc2vec was used to derive conceptual couplings from the textual source code data. They also proposed an algorithm for directly computing Cosine and Euclidean similarity functions and performed experiments on three open source software systems. Experimental results based on precision, recall and F-score measures indicated a significant improvement of between 7.3% and 17.3% in F-score values over the baseline techniques.

DISCUSSION

This section presents insights gained from the study in a bid to summarize the state of art in code based hybrid CIA techniques and also provide recommendations for further research. The discussion is centered around the research questions formulated in section 2 and provided the findings of the review. For clarity purposes the discussion has been classified viz-a-viz the following points based on the formulated research questions: subject systems; combination strategy and method; level of granularity of software; and evaluation metrics. This is presented as follows:

SUBJECT SOFTWARE SYSTEMS

A good number of the software systems utilized for evaluating the CIA techniques presented are open source systems written in Java with only a few of the systems written in C/C++. The reason for this may not be unrelated to the advantages offered by java such as ease, platform independent, ability to handle large high level software systems amongst others which has made it a first choice in many enterprise software solutions in the market. Also, it was observed that the sizes of the software systems utilized have varied between 678 and 367KLOC. It was also identified that three subjects systems have been used for evaluating the techniques on an average.

Nonetheless, there is a need to evaluate CIA techniques across a variety of software written in different programming languages and having different features, functionalities and sizes. A comparison of the hybrid CIA techniques surveyed based on the subject systems utilized is summarized in Table 2.

Table 2. Comparison of Hybrid Techniques Based on Subject Software Systems

Author, year	Subject systems				
	Programming Languages			Size of system	number of systems
	Java	*C++*	*C*		
[28]	√	√	√	70-367KLOC	4
[19]	√	√	√	678-5969LOC	5
[29]	√	-	-	74-148KLOC	4
[20]	√	-	-	10-149KLOC	4
[25]	√	-	-	-	6
[16]	√	-	-	74-103KLOC	3

COMBINATION METHODS AND STRATEGIES

All hybrid CIA techniques surveyed except one combined two CIA techniques. This shows that a combination of two techniques is sufficient to bring about an improvement in the accuracy of estimated impacts without incurring too much time overhead. Also, all hybrid techniques analyzing textual source code data have utilized Latent Semantic Indexing (LSI) for computing conceptual dependencies. However, LSI has been found not to have some shortcomings such as the inability to effectively resolve polysemy problems and also difficulty in interpretation of results thereby affecting its performance [32][33]. Since the choice of Information Retrieval (IR) model employed for textual analysis is crucial to getting best results for analyzing textual information and overall best results for estimating source code change impacts [34][35] , there is a need to utilize other categories of IR models, particularly topic models like Latent Dirichlet Allocation (LDA) and its extensions as well as those based on deep learning techniques. Another point worthy of mention is the fact that hybrid techniques which rely on run time or evolutionary information may not always be applicable since such information may not be available. It is therefore necessary to come up with hybrid techniques that depend on sources of information like the structured and textual information found in source code data which will always be available. A comparison of the hybrid CIA techniques surveyed based on the types of CIA techniques utilized is given in Table 3.

Table 3. Comparison of Hybrid Techniques Based on Type of CIA Techniques Utilized

Author, year	Combination Techniques				
	CIA approaches				Number of techniques
	Structural	***Textual***	***Evolutionary***	***Dynamic***	
[28]	-	LSI	Itemset mining	-	2
[19]	Program graph	-	-	Execution traces	2
[29]	-	LSI	Itemset mining	Execution analysis	3
[20]	Formal Concept Lattice & Call graph	-	-	-	2
[25]	**S**tructural Coupling	LSI	-	-	2
[16]	-	LSI & doc2vec	-	-	2

LEVEL OF GRANULARITY

Granularity levels adopted by the surveyed techniques have varied between methods, classes and file levels of granularity with a majority of the techniques utilizing the method level granularity. This may be as a result of the fine

grained details provided by method level granularity as compared to the coarse details provided by file granularity level. Still, a few studies have combined two granularity levels. Nevertheless, there is a need to incorporate a variable granularity level in hybrid CIA techniques, such that the software maintainers can decide on the level of granularity to work on. [22] proposed a CIA technique which allowed software maintainers to select a variable granularity level when performing CIA. Table 4 denotes the granularity levels adopted in the hybrid CIA techniques.

Table 4. Comparison of Hybrid CIA Techniques Based on Level of Granularity

Author, year	**Level of Granularity**				
	Method	*Class*	*File*	*Change Request*	*Variable granularity*
[28]	√	-	√	-	-
[19]	√	√	-	-	-
[29]	√	-	-	√	-
[20]	√	√	-	-	-
[25]	√	√	-	-	-
[16]	√	-	-	-	-

EVALUATION METRICS

Precision and recall are two widely used metrics for evaluating CIA techniques. However, these metrics are reciprocal and so, may not be able to give adequate insight into the performance of the CIA technique. Thus, the need to utilize the F-score measure to reflect the true performance of the technique. Unfortunately, only two of the surveyed techniques made use of the F-score measure for evaluating their technique, others relied solely on the precision and recall metrics. Additionally, it was observed that evaluation in all surveyed techniques was based on comparison with the baseline techniques. While this has revealed the strength of combining multiple CIA techniques, it however fails to evaluate the performance of the techniques against other hybrid techniques. To effectively carry out such evaluations, there is a need for a benchmark dataset which will include all information sources such as source code, run time and evolutionary information. Table 5 presents the evaluation parameters used in the studies.

Table 5. Evaluation of Hybrid CIA Techniques

Author, year	**Evaluation Techniques**				
	Accuracy Metrics			**Mode of Comparison**	
	Precision	*Recall*	*F-Measure*	*Baseline*	*Hybrid*
[28]	√	√	-	√	-
[19]	√	√	-	√	-
[29]	√	√	-	√	-
[20]	√	√	-	√	-
[25]	√	√	-	√	-
[16]	√	√	√	√	-

CONCLUSION

This paper reviewed studies that have combined at least one static code CIA approach for predicting impacts of proposed software changes. Results from the review revealed that code based hybrid CIA techniques have consistently shown the capability of improving the accuracy of estimated impacts when compared to the baseline techniques. The paper juxtaposed the different hybrid CIA techniques based on the software systems used in the studies, the combination techniques used, the level of granularity of the analysis and evaluation metrics. Predominantly, Java-based software systems have been used followed by C/C++ software. All the studies, except [29] combined only two techniques which indicated that a minimum of two techniques is enough to achieve improvement in code based hybrid techniques.

Furthermore, method-level granularity has been the most considered level of granularity in the studies while the precision and recall metrics have mostly been employed for evaluating Code based CIA techniques. Consequently, the following are recommendations from the review: More studies need to be carried out to not only validate the results achieved so far but also to explore other possible combinations that have not been studied. These studies should include a combination of structural, textual, evolutionary and dynamic CIA approaches that need to be considered. In doing so, several techniques under each approach can be explored. Also, there is a need to consider variable granularity levels in a single technique in order to give room for options for the maintainers. Most of the techniques involving textual analysis have utilized the LSI model as such, there is a need to consider other models such as Latent Dirichlet Allocation (LDA) and those based on deep learning in order to compare performance. In addition, there is a need to carry out a comparative analysis on the performance of existing code based hybrid CIA techniques in order to shed more light on further research. This however may require the need for a benchmark source code dataset to allow for a fair comparison as well as the need to include additional evaluation criteria such as time taken for analysis.

REFERENCES

[1] A. O. Bajeh, B. Shuib, and T. . Low, "Empirical Validation of Object-Oriented Inheritance Hierarchy Modifiability Metrics," in *Proceedings of the 6th International Conference on Information Technology and Multimedia (ICIMU)*, 2014, pp. 189–194.

[2] A. O. Bajeh, M. A. Olatunji, and R. O. Oladele, "Investigating Self-Regulation Property of Evolving Open Source Systems: An Empirical Study," *J. Sustain. Technol.*, vol. 10, no. 1, pp. 68–76, 2019.

[3] M. W. Godfrey and D. M. German, "The Past , Present and Future of Software Evolution," in *2008 Frontiers of Software Maintenance (FoSM)*, 2008, pp. 129–138.

[4] G. Canfora *et al.*, "In Memory of Manny Lehman, 'Father of Software Evolution,'" *J. Softw. Maint. Evol. Res. Pract.*, vol. 23, no. 3, pp. 137–144, 2011, doi: 10.1002/smr.537.

[5] V. Rajlich, "Software Evolution and Maintenance," in *Proceedings of the Future of Software Engineering - FOSE 2014*, 2014, pp. 133–144, doi: 10.1145/2593882.2593893.

[6] M. Alenezi, "Extracting High-Level Concepts from Open-Source Systems," *Int. J. Softw. Eng. its Appl.*, vol. 9, no. 1, pp. 183–190, 2015, doi: 10.14257/ijseia.2015.9.1.16.

[7] W. Chen, "A Hybrid Software Change Impact Analysis for Large-scale Enterprise Systems," Mcmaster University, School of Graduate Studies, 2015.

[8] S. Bohner and S. A. Arnold, "An Introduction to Software Change Impact Analysis," in *Software Change Impact Analysis*, Los Alamitos, CA, USA: IEEE Computer Society Press, 1996, pp. 1–26.

[9] B. Li, X. Sun, H. Leung, and S. Zhang, "A Survey of Code-Based Change Impact Analysis Techniques," *J. Softw. Testing, Verif. Reliab.*, vol. 23, no. 8, pp. 613–646, 2012, doi: 10.1002/stvr.

[10] A. Dhamija and S. Sikka, "A Systematic Review of Feature Location Techniques Under Software Change Impact Analysis," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 3, pp. 184–192, 2019.

[11] A. Dhamija and S. Sikka, "A Systematic Study of Advancements in Change Impact Analysis Techniques," *Int. J. Innov. Technol. Explor. Eng.*, vol. 8, no. 8, pp. 435–443, 2019, [Online]. Available: http://ieeexplore.ieee.org.

[12] S. Lehnert, "A Review of Software Change Impact Analysis," Ilmenau, Germany, 2011. [Online]. Available: http://www.db-thueringen.de/servlets/DocumentServlet?id=19544.

[13] N. Ajienka, A. Capiluppi, and S. Counsell, "Managing Hidden Dependencies in OO Software : A Study based on Open Source Projects," in *In Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2017, pp. 141–150, doi: 10.1109/ESEM.2017.21.

[14] X. Sun, B. Li, H. Leung, B. Li, and J. Zhu, "Static Change Impact Analysis Techniques: A Comparative Study," *J. Syst. Softw.*, vol. 109, pp. 137–149, 2015, doi: 10.1016/j.jss.2015.07.047.

[15] S. Basri, N. Kama, R. Ibrahim, and S. A. Ismail, "A Change Impact Analysis Tool for Software Development Phase," *Int. J. Softw. Eng. its Appl.*, vol. 9, no. 9, pp. 245–256, 2015, doi: 10.14257/ijseia.2015.9.9.21.

[16] W. Wang, Y. He, T. Li, J. Zhu, and J. Liu, "An Integrated Model for Information Retrieval Based Change Impact Analysis," *Sci. Program.*, vol. 2018, pp. 1–21, 2018, doi: 10.1155/2018/5913634.

[17] X. Sun, X. Liu, J. Hu, and J. Zhu, "Empirical studies on the NLP techniques for source code data preprocessing," in *ACM International Conference Proceeding Series*, 2014, no. May, pp. 32–39, doi: 10.1145/2627508.2627514.

[18] X. Sun, B. Li, B. Li, and W. Wen, "A comparative study of static CIA techniques," in *4th Asia-Pacific Symposium on Internetware, Internetware 2012*, 2012, pp. 1–8, doi: 10.1145/2430475.2430498.

[19] M. C. O. Maia, R. A. Bittencourt, J. C. A. De Figueiredo, and D. D. S. Guerrero, "The Hybrid Technique for Object-Oriented Software Change Impact Analysis," in *2010 14th European Conference on Software Maintenance and Reengineering*, 2010, pp. 252–255, doi: 10.1109/CSMR.2010.48.

[20] B. Li, X. Sun, and H. Leung, "Combining Concept Lattice with Call Graph for Impact Analysis," *Adv. Eng. Softw.*, vol. 53, pp. 1–13, 2012, doi: 10.1016/j.advengsoft.2012.07.001.

[21] X. Sun, B. Li, C. Tao, W. Wen, and S. Zhang, "Change Impact Analysis based on a Taxonomy of Change Types," in *Proceedings - International Computer Software and Applications Conference*, 2010, pp. 373–382, doi: 10.1109/COMPSAC.2010.45.

[22] M. Petrenko and V. Rajlich, "Variable Granularity for Improving Precision of Impact Analysis," in *17th IEEE*

International Conference on Program Comprehension (ICPC'09), 2009, pp. 10–19.

[23] X. Li and Y. Yin, "A Unified Framework for Software Coupling Measurement," in *In Proceedings of 2nd International Conference on Software Engineering, Knowledge Engineering and Information Engineering (SEKEIE 2014)*, 2014, no. January 2014, pp. 156–161, doi: 10.2991/sekeie-14.2014.37.

[24] L. C. Briand, J. Wuest, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *Conference on Software Maintenance*, 1999, pp. 475–482, doi: 10.1109/icsm.1999.792645.

[25] M. Alenezi and K. Magel, "Empirical Evaluation of a New Coupling Metric: Combining Structural and Semantic Coupling," *Int. J. Comput. Appl.*, vol. 36, no. 1, pp. 34–44, 2014, doi: 10.2316/Journal.202.2014.1.202-3902.

[26] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using Information Retrieval Based Coupling Measures for Impact Analysis," *Empir. Softw. Eng.*, vol. 14, no. 1, pp. 5–32, 2009, doi: 10.1007/s10664-008-9088-2.

[27] M. Gethers and D. Poshyvanyk, "Using Relational Topic Models to Capture Coupling among Classes in Object-Oriented Software Systems," in *In 2010 IEEE International Conference on Software Maintenance*, 2010, pp. 1–10, doi: 10.1109/ICSM.2010.5609687.

[28] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard, "Blending conceptual and evolutionary couplings to support change impact analysis in source code," in *Proceedings - Working Conference on Reverse Engineering, WCRE*, 2010, pp. 119–128, doi: 10.1109/WCRE.2010.21.

[29] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," in *Proceedings - International Conference on Software Engineering*, 2012, pp. 430–440, doi: 10.1109/ICSE.2012.6227172.

[30] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empir. Softw. Eng.*, vol. 18, no. 5, pp. 933–969, 2013, doi: 10.1007/s10664-012-9233-9.

[31] H. Cai, R. Santelices, and S. Jiang, "Prioritizing Change-Impact Analysis via Semantic Program-Dependence Quantification," in *IEEE Transactions on Reliability*, 2016, vol. 65, no. 3, pp. 1114–1132, doi: 10.1109/TR.2015.2481000.

[32] L. H. Anaya, "Comparing Latent Dirichlet Allocation and Latent Semantic Analysis as Classifiers," University of North Texas, 2011.

[33] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug Localization using Latent Dirichlet Allocation," *Inf. Softw. Technol.*, vol. 52, no. 9, pp. 972–990, 2010, doi: 10.1016/j.infsof.2010.04.002.

[34] M. Belford, B. Mac Namee, and D. Greene, "Stability of topic modeling via matrix factorization," *Expert Syst. Appl.*, vol. 91, pp. 159–169, 2018, doi: 10.1016/j.eswa.2017.08.047.

[35] A. Agrawal, W. Fu, and T. Menzies, "What is Wrong with Topic Modeling?(and How to Fix it Using Search-based Software Engineering)," in *Information and Software Technology*, 2018, vol. 98, pp. 74–88, doi: 10.1016/j.infsof.2018.02.005.