J Inf Process Syst, Vol.17, No.5, pp.972~988, October 2021
https://doi.org/10.3745/JIPS.04.0225

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

SAT-Analyser Traceability Management Tool Support for DevOps

Iresha Rubasinghe*, Dulani Meedeniya*, and Indika Perera*

Abstract
At present, DevOps environments are getting popular in software organizations due to better collaboration and software productivity over traditional software process models. Software artefacts in DevOps environments are vulnerable to frequent changes at any phase of the software development life cycle that create a continuous integration continuous delivery pipeline. Therefore, software artefact traceability management is challenging in DevOps environments due to the continual artefact changes; often it makes the artefacts to be inconsistent. The existing software traceability related research shows limitations such as being limited to few types of artefacts, lack of automation and inability to cope with continuous integrations. This paper attempts to overcome those challenges by providing traceability support for heterogeneous artefacts in DevOps environments using a prototype named SAT-Analyser. The novel contribution of this work is the proposed traceability process model consists of artefact change detection, change impact analysis, and change propagation. Moreover, this tool provides multi-user accessibility and is integrated with a prominent DevOps tool stack to enable collaborations. The case study analysis has shown high accuracy in SAT-Analyser generated results and have obtained positive feedback from industry DevOps practitioners for its efficacy.

Keywords
Computer-Aided Software Engineering, Continuous Integration, Software Artefact Consistency Management

1. Introduction

Software system development is a collective process of creating software programs that embody different stages in the system development life cycle (SDLC) [1]. The intermediate by-products corresponding to each phase are known as software artefacts. Software requirements, design diagrams, source code, test files and configuration files are some of the artefacts found in SDLC. Different software process models can be found based on specific perspectives such as convenience in development, increase profits and schedule optimization. Most of the traditional software development models appreciate well-defined requirements, minimum artefact changes during the process and high visibility of project progression [1]. However, the prototype costs and increasing quality concerns pushed the software development practices toward Agile software processes [2].

Agile principles motivate customer collaboration, embrace software artefact changes at any stage in contrast to traditional processes and ensure the incremental delivery of software at regular intervals. Thus,

Manuscript received March 27, 2019; first revision July 10, 2019; second revision September 26, 2019; accepted October 14, 2019.
Corresponding Author: D. A. Meedeniya (dulanim@cse.mrt.ac.lk)
* Dept. of Computer Science and Engineering, University of Moratuwa, Sri Lanka (ireshar@cse.mrt.ac.lk, dulanim@cse.mrt.ac.lk, indika@cse.mrt.ac.lk)

Agile reduces the communication gap between customer and developer preventing software requirement mismatches. DevOps practice has become a popular software development approach that eliminates the gap between developer and operations teams to ensure incremental delivery in Agile's release cycle [2,3]. The collaboration between the customer, the development team, and the operations team is increased. DevOps supports continuous integration (CI), continuous build, continuous testing and continuous delivery (CD) formulating the continuous integration continuous delivery (CICD) pipeline [2,4,5]. This ensures rapid software delivery with continuously changing software artefacts; a comprehensive tool stack exists such as Jenkins, Git, Docker, Trello to help with each DevOps practice [3,5]. However, artefact consistency management throughout the project is still a challenge in DevOps. Software artefact traceability support is important to avoid the heterogeneous artefact management overheads in DevOps during frequent changes [1,6]. Thus, there is a need for traceability solutions that are compatible with DevOps practices and the existing DevOps tool stack, without any overhead.

This paper presents a traceability solution, Software Artefact Traceability Analyser (SAT-Analyser) that manages artefact traceability in software development in DevOps practice. The main modules considered are traceability establishment, visualization, artefact change detection, change impact analysis (CIA), change propagation, consistency management during the CICD pipeline. Further, this tool integrates with DevOps tool stacks such as Jenkins and Trello [7] toward high collaboration.

The paper is organized as follows. Section 2 reviews related work on traceability management. The design of the SAT-Analyser tool, CICD support and evaluation are discussed from Sections 3 to Section 7, respectively. Finally, Section 8 concludes the remark with future research directions.

2. Related Work

2.1 Traceability Aspects

Traceability enables the ability to track different types of artefacts involved throughout a software project. This helps to maintain consistency among artefacts as they go through the changes during software development. For instance, artefact changes can cause inconsistency among other connected artefacts and traceability helps to identify such occurrences through the continuous tracking among artefacts. Many techniques such as rule-based, hypertext-based, event-based, constraint-based, transformation-based, goal-centric and model-driven have been used to support traceability in [6]. Information retrieval techniques such as vector space model (VSM), term frequency-inverse document frequency (TF-IDF), and latent semantic indexing (LSI) are also used in traceability studies for artefact data extraction and relationship establishment [8]. However, each of these techniques has certain pros and cons. For instance, rule-based traceability approaches are not widely used to recognize structural changes, event-based traceability techniques have scalability issues, and the transformational and model-driven approaches are domain specific; thus, do not offer generic software development environments with process-domain independence.

The artefact change detection techniques include edit history, tree differencing with abstract syntax tree (AST) and custom differencing algorithms [9,10]. However, most of the related studies have incorporated these techniques only for the change detection in source code artefact in code repositories. CIA is important to identify the impact of the changes, consequences and to trace ripple effects.

Among different CIA categories, traceability-based CIA that aims to recover the trace links, dependence-based CIA targets to estimate the impact of the change, static impact analysis considers all possible behaviors [11,12]. Table 1 summarizes CIA related studies with their features and limitations [9,12-18]. Accordingly, the lack of support for the heterogeneous artefact types, poor tool support with automation limits the applicability of these studies in a collaborative and change-intensive DevOps environment.

Table 1. Change impact analysis related work analysis

Related work	Method	Advantages	Limitations
Code change detection to identify impacts [9].	Edit history, heuristic rules and distance-based methods.	High performance. Finds the optimal change propagation.	Limited for source code artefact.
A static impact method to compute impact sets [12].	Rules based on change type.	Improve precision of impact sets.	Only applicable with object-orientation.
A goal-driven approach to manage requirements changes [13].	Design structure matrix (DSM) is used, and it is divided into four sub matrices.	Visually organize project tasks. Enables team communication with CIA.	Limited for requirements artefact.
Improving CIA in requirements [14].	Formal semantics. Implemented as an extension for a tool named TRIC.	Eliminates false positive impacts. Enables change consistency. Semi-automatic.	Limited for requirements artefact.
A rule-based CIA method for software lifecycle objects [15].	Change propagation rules and a CIA algorithm. Entity Dependency Graph (EDG).	The experiments have shown the effectiveness of the CIA algorithm.	No GUI support.
A recommendation system named “ImpRec” for CIA at process automation [16].	ImpRec uses a knowledge base to recommend impacted artefacts. Compute impact using network analysis.	The knowledge in the historical CIA reports can be reused to provide decision support.	Lacks source code impact. Requires the manual creation of the knowledge base.
Stochastic dependencies over structured techniques for source code CIA [17].	Stochastic dependencies. Logical coupling.	Better accuracy. Effective in volatile systems having frequent refactoring.	Limited for source code artefact.
Analyzes change propagation among different artefacts [18].	Impact propagation rules. Rule-based dependency analysis.	Enables developers to easily understand and retrace the propagation of changes.	Needs to refine and add dependency detection rules.

2.2 DevOps Aspects

DevOps culture is about sharing responsibilities between the developers and operations by increasing transparency, collaboration, and communication across the software development [11]. In practice, different tools are used to share the tasks at every level of the CICD pipeline. Git is a widely used source code version management system and Jenkins is a build automation tool for CI [5]. Puppet, Docker, and Travis are also some of the leading DevOps tool stacks used within the CICD pipeline [2,19]. In this practice, project management (PM) tools support collaboration among different teams of a project. Tools such as Trello [7], Jira, Slack, Zoho sprints and Bitrix24 [19] support to manage the projects providing features such as tracking software changes and task allocation. Accordingly, a new tool in the DevOps tool stack should be compatible with these prominent tools to work collaboratively. One of the major characteristics of DevOps tools is their multi-user accessibility. For example, the Git version management allows multiple authorized users to merge continuous implementations, Jenkins build automation tool enables multiple people to schedule builds and trigger events, Docker deployment tool allows many containers created on a single code base and Trello PM tool supports multiple users to allocate tasks, contribute to activities and notifies to a larger audience simultaneously. Considering the inter-relationships of the DevOps tool stack, each tool can function independently and work in an integrative mode with other tools. These tools have their in-built plug-ins that support other tools; for instance, Jenkins offers various plug-ins for Git, Docker, Travis, Puppet, Jira, Slack [2,5,19].

3. System Design

We have designed the SAT-Analyser tool by addressing the existing traceability management challenges in any software development practice, especially for an Agile-based DevOps environment. Fig. 1 shows the research model of the proposed traceability tool, which considers diverse artefacts; requirements, design diagram, source code, test script and configuration files representing the major phases in SDLC. These heterogeneous artefact types are the inputs to the tool throughout a software project timeline. The artefacts selection is conducted based on the industry demand obtained via a survey among DevOps engineers, developers and operations teams who experience CICD pipelines in software companies in Sri Lanka. The CICD supportability of the tool is ensured by the artefact consistency management during artefact version management with CI, artefact change detection, impact analysis, change propagation and status notification in the DevOps practice.

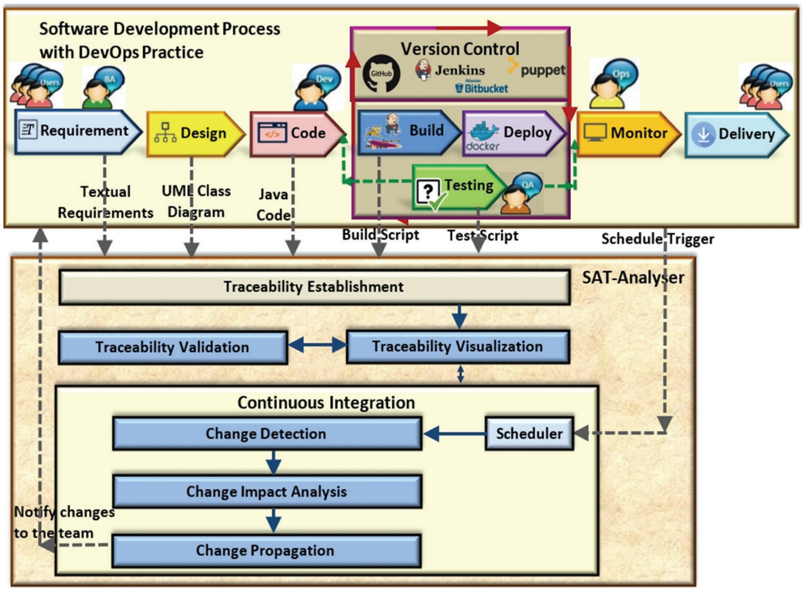


Fig. 1. Integration of the tool within DevOps practice.

Generally, Agile-based DevOps does not promote extensive documentation compared to traditional software processes. Instead, Agile-based DevOps targets working software over comprehensive documentation [1-3]. Thus, the artefact inconsistencies that have a high tendency to occur at any time are resolved by the SAT-Analyser tool. The 3-tier architecture of the tool with modular levels is shown in Fig. 2. The presentation layer contains the inputs and outputs such as visualization, while all the functionalities are taken place within the business logic layer with the data management support of the data access layer. Artefact manager considers the supportive heterogeneous artefact types at the initiation of a software project. During the CICD pipeline, it synchronizes the latest source code, test scripts and build-script artefacts that are merged through CI activities. The visualization manager generates multiple graph variations; analytical, interactive and informative traceability graphs for each update. The DevOps tool stack integration with SAT-Analyser is shown in the business logic layer that holds the application programming interface (API) modules for Git, Jenkins, and Trello. The artefacts received via artefact manager are pre-processed using data extractors specific for each category and converted to a common XML format. Traceability generator is comprised of string comparison, threshold handlers, validator with accuracy and network centrality metrics. The traceability and consistency management processes are linked with the CI process, using a scheduler in the change detector module that can invoke in fixed intervals or trigger dynamically whenever CI occurs. Moreover, the impact analyzer includes a mathematical model with a graph traversal submodule.

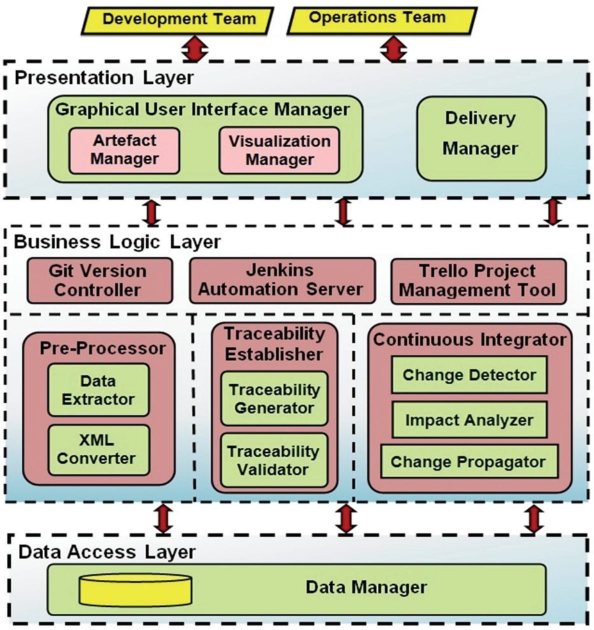


Fig. 2. SAT-Analyser modular level architecture.

4. Traceability Process Model

The software artefacts are pre-processed to extract the main and sub-elements with the relationships. The pre-processing uses natural language processing (NLP) techniques such as tokenization, anaphora analysis, morphological analysis, stemming and redundancy elimination. The implementation is based on Stanford CoreNLP. Initially, this process identifies the pronouns in the requirements. Then, coreferences, the terms that refer to the same pronoun, is identified for each line using anaphora analysis. A rule set based on NLP is used to identify the elements and intra-relationships. For instance, if a noun is followed by a verb, that noun is extracted as the main element. Sub elements are extracted from nouns and adjectives that are not following a verb. Relationship discovery is used to extract the associations and generalizations. Finally, morphological analysis is used to eliminate redundancy [20].

The UML parser with JSON readers is used to pre-process design diagram files. The source code artefacts in Java language and test script artefacts in JUnit are pre-processed using Java Grammar and ANother Tool for Language Recognition (ANTLR) to extract object-oriented classes, methods, attributes with their intra dependencies as the conceptual entities and interrelations [21]. The XML parsers are applied to pre-process the selected configuration artefact in Maven build script format. Fig. 3 shows the traceability link generation procedure followed in the SAT-Analyser tool.

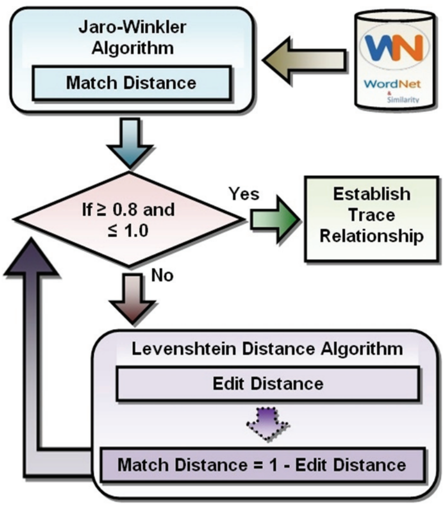

Jaro-Winkler Algorithm
Match Distance
WordNet Similarity
If ≥ 0.8 and ≤ 1.0
Yes
Establish Trace Relationship
No
Levenshtein Distance Algorithm
Edit Distance
Match Distance = 1 - Edit Distance

Fig. 3. SAT-Analyser traceability establishment process.

Accordingly, all the extracted heterogeneous artefacts are converted into an intermediate XML structure prior to traceability establishment. The specified tag structures in the XML includes artefact type, name, ID, visibility, artefact sub-elements, intra-relationships and inter-relationships in a source-target style. The XML structure helps to build complex graphs and improves developer readability.

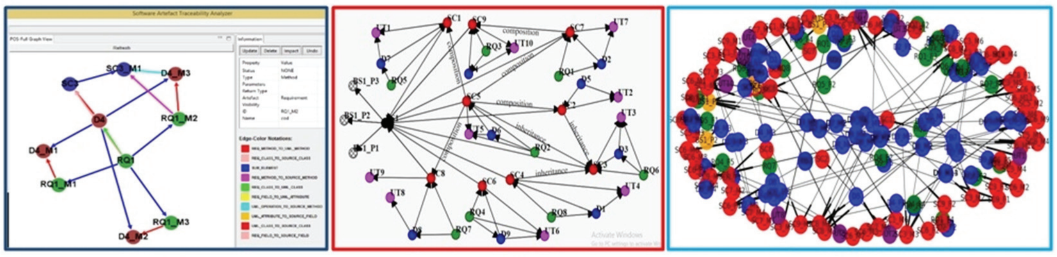
We have used two prominent string comparison algorithms Jaro-Winkler algorithm and the Levenshtein distance algorithm along with fixed threshold values for decision-making [22,23]. The artefacts are checked for traceability pair-wise. A similarity score is obtained for each pair through the Jaro-Winkler algorithm with the association of WordNet ontology for the support of synonyms. Hear, a trace relation is created if the score is in the range of 0.8 and 1.0. Else, a pair of artefact strings are sent through the Levenshtein distance algorithm in the second layer for a deeper comparison to finalize with the trace relation creation. The threshold values such as 0.8 are changeable and can be improved to be auto selected with learning models. These integrated views of artefacts and the fine-grained links for the data extraction are further discussed in the case study in Section 6. Jaro-Winkler algorithm shown in Eq.

(1) is applied first due to its efficiency over the Levenshtein algorithm given in Eq. (2) and later is used for deeper similarity computation at the second level for higher accuracy. The notations d_j, S_1, S_2, m and t in Eq. (1) represent Jaro distance, string one, string two, number of matching characters in both strings and half the number of transpositions, respectively. The term $ld_{s1,s2}(i,j)$ in Eq. (2) denotes the Levenshtein distance between the first i characters of string one ($s1$) and the first j characters of string two ($s2$). Further, $k_{(s1_i \neq s2_j)}$ is the indicator function equal to 0 when $s1_i = s2_j$ and equal to 1 otherwise.

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases} \tag{1}$$

$$ld_{s1,s2}(i,j) = \begin{cases} \max(i,j) & \mathit{if}\ \min(i,j) = 0 \\ \min \begin{cases} ld_{s1,s2}\,(i-1,j) + 1 \\ ld_{s1,s2}\,(i,j-1) + 1 \\ ld_{s1,s2}\,(i-1,j-1) + k_{(s1_i \neq s2_j)} \end{cases} & \mathit{Otherwise} \end{cases} \tag{2}$$

Fig. 4 shows the three views: informative view to illustrate the artefact, relationship types with details; interactive view to easily move the traceability graph dynamically with drag and drop, zoom, hover functions; and analytical view for the network analysis, of the SAT-Analyser tool. The graph database Neo4j [24] and Gephi [25] open graph platform are used in the informative traceability graph that contains a separate node-link data section by side. Moreover, Python NetworkX and Matplotlib libraries [26] for the analytical graph and JavaScript D3.js library [27] with JSON for the interactive graph are used. Scalability issues and visual clutter are mitigated in this multi-analysis view. The informative graph shows an overall view with node data with visibility levels of private, public, protected, or filtered views to identify intra-relationships of a given artefact.



(a) (b) (c)

Fig. 4. Traceability visualization graph variations: (a) informative view, (b) interactive view, and (c) analytical view.

The color codes for nodes and edges are used with a separate data pane to elaborate details of each node. The results can be filtered based on artefact types or edge types from the menu. In contrast, the interactive view supports dynamic movements, hover, search and identify immediate traceability relations of a node by clicking on a single view. This is also facilitated to represent the results of the CIA and change propagation. For traceability outcome validation, the analytical graph is used with network analysis considering the analytical centrality measures in NetworkX features [26]. In addition, other features such as zoom, record, move, save functionalities are supported by this view.

Further, the tool is featured with a traceability validation module using two approaches. The statistical validation uses the accuracy measures namely precision, recall and F-measure, whereas network analysis-based validation uses centrality measures including degree centrality, closeness centrality, betweenness centrality and eigenvector centrality (EVC) [28,29]. This module is only used to validate the tool functionality and not to be used in real-world applications. The tool validation process is described in Section 6 in the case study.

5. Continuous Integration Support

5.1 Consistency Management

In this CICD compatible traceability process model, artefact consistency management during CI is handled in three major steps, i.e., artefact change detection, CIA and change propagation. The existing traceability management solutions lack the support of quantitative measures to identify the impact of a detected artefact change. Our CIA model, as shown in Fig. 5, is implemented based on EVC by measuring the impact quantitatively in traceability network analysis. It traces the change impact of artefacts due to additions, modifications or deletions. The identification of those changes is crucial in a change-intensive DevOps environment to keep accurate traceability data. XML based change detection process is implemented with a version management subsystem as part of SAT-Analyser architecture. The XML files of the latest and the previous artefact versions are compared using a generalized tool “diffmk” from Oracle Sun developers.

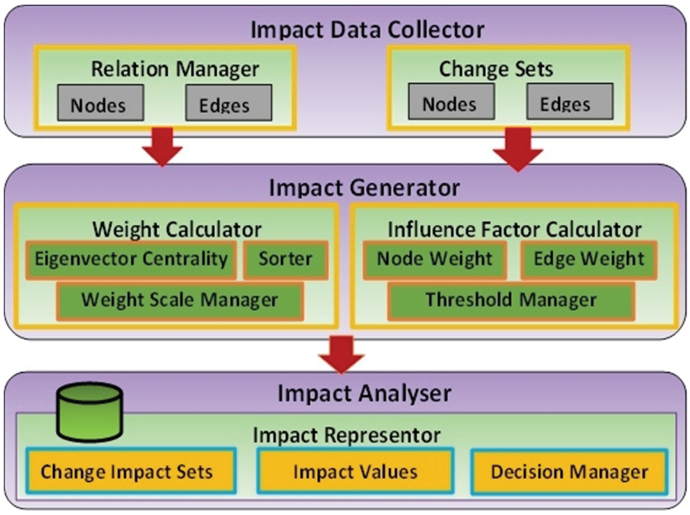


Fig. 5. SAT-Analyser change impact analysis (CIA) model workflow.

EVC measures the influence of a node within the traceability network [29]. Thus, when a given artefact is detected as changed, we look for its influential factor to determine the change impact of that node. Accordingly, the impact generator partitions each node into a two-state influential factor value as either high or low based on its eigenvector value. The CIA follows a rule-based approach to decide the impact paths. This helps to optimize the computations by identifying only the possible trace paths that impact. Thus, the less significant trace paths are removed using the dynamic impact ruleset. Accordingly, if a changed artefact node has a highly influential factor value; it is designed to be considered as its outgoing edges' endpoint nodes are also getting a high impact and low otherwise. Thus, the impact value relationship in the CIA model can be depicted as in Eq. (3), where $w()$ function represents the two-state influential factor weight, i for ith artefact node or its any outgoing trace relationship or edge, $Node^{source}$ for source artefact node and $Node^{target}$ denotes the endpoint node of a source node's outgoing edge. Further, SAT-Analyser validates the CIA using precision, recall and F-measure to analyse the accuracy of the obtained impact sets and allows modifiability. The artefact change propagation on the traceability graph and traceability links are updated after merging the artefact changes.

$$\forall i, w\left(Node_i^{target}\right) = w(Edge_i) = w\left(Node_i^{source}\right) \tag{3}$$

Further, SAT-Analyser contains a change propagation model using the graph traversal and a rule-based approach as shown in Fig. 6 and Algorithm 1. The single source Dijkstra algorithm is applied for the graph traversal since the weights involved in the CIA model are non-negative. The obtained complete impact path set for any changed artefact item in the change set would be filtered immediately based on the defined change propagation rules. The rules are defined based on the artefact types such that for instance, a change on a build script artefact should not impact any other artefacts; however, the impact on a design artefact item can impact its related source code, test script, and build script artefacts. SAT-Analyser updates the traceability graph and simultaneously notifies the CIA results that propagated on traceability graph to DevOps teams via the PM tool Trello [7].

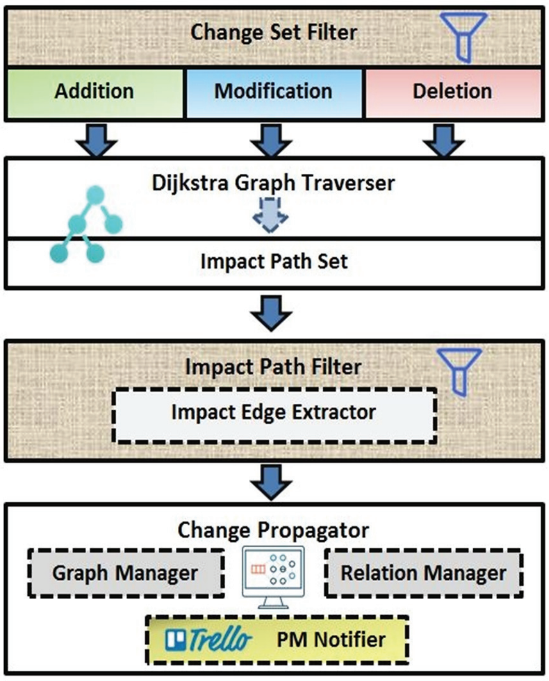


Fig. 6. SAT-Analyser change propagation model workflow.

Algorithm 1: Change propagation

```
Require: Impact value assigned change set
Ensure: Change propagation of change set
1.    Input: An impact assigned change set
2.    For each item in the impacted change set
3.        Dijkstra single source graph traverser (G, item, influential factor value)
4.            Impact Path Set = $Impact\_Path_{Complete}$
5.    For each $Impact\_Path_{Complete}$ in Impact Path Set
6.        $Impact\_Path_{Relevant}$ Set = Impact path filter rules ($Impact\_Path_{Complete}$)
7.    Propagate changes ($Impact\_Path_{Relevant}$ Set)
8.        Extract edges ($Impact\_Path_{Relevant}$ Set)
9.        Update graph manager & relation manager
10.       Project management notifier
11.   Output: Relevant Impact path sets
```

5.2 DevOps Tool Stack Compatibility of SAT-Analyser

DevOps environments involve a tool stack that is rapidly evolving with newer features to ease the tasks and maximize the gain [19]. Generally, software development companies adopt their own set of DevOps tools depending on the organizational structure, financial status, employee experience and the software projects they are working on. DevOps tools should work together due to the highly collaborative nature of the CICD pipeline. For example, Jenkins has a plugin for the separate tool Docker, which can be integrated easily in a development environment during the CICD process [5]. Accordingly, the proposed SAT-Analyser is equipped with the DevOps compatibility features and integrated with Git, Jenkins and Trello [5,7] to work smoothly in a DevOps environment. Moreover, SAT-Analyser is featured with a multi-user, browser-based web version to make it usable in a DevOps environment. Further, the tool has a performance analysis feature to assess the resource consumption during traceability establishment and CIA process of a software project in terms of elapsed time, memory and central processing unit (CPU) consumption. Thus, the DevOps environments can properly manage their resource allocation for traceability support as currently, traceability is lacking in practice due to the maintenance cost and resource consumption.

6. Experimental Case Study

SAT-Analyser is evaluated using software projects from different domains. Each project is different in size measures such as requirements, line-of-code (LOC) and function calls, making them unique to each other. SAT-Analyser is already evaluated using a point-of-sales (POS) system [28] and a tour management system [29]. In this evaluation, we use a software project for an E-school management system as a prototype. It is an information management system for the activities of students and teachers. This has 10 major requirements, 6 design elements as shown in Fig. 7, 5460 LOC and 690 function calls [30]. The identified main artefacts followed by the tool generated identifier of each artefact are given in Fig. 8. There exist artefact sub-elements for methods, attributes (fields), plug-ins as partially shown in Fig. 8 with _M, _F, and _P, respectively. The established traces among the identified artefacts are written to the predefined conceptual data model using the source to the target format in the Relations.xml file as shown in Fig. 9. For instance, it shows two relations from SC8:Enroll and SC10:MyProfile source classes to the BS1: project's Maven build script artefact.

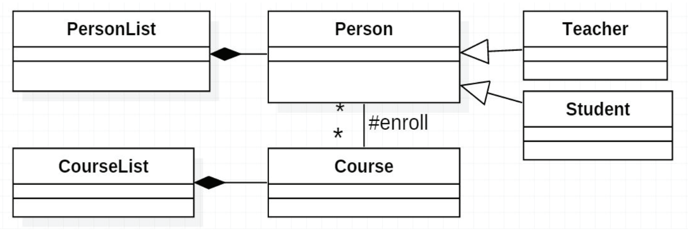
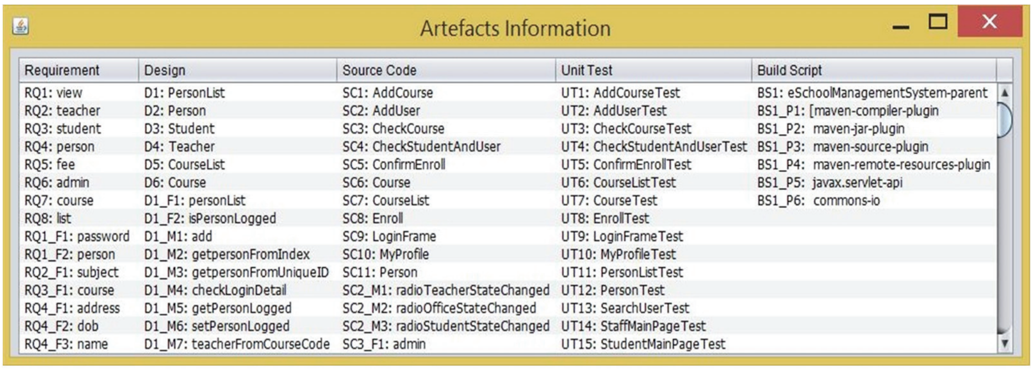
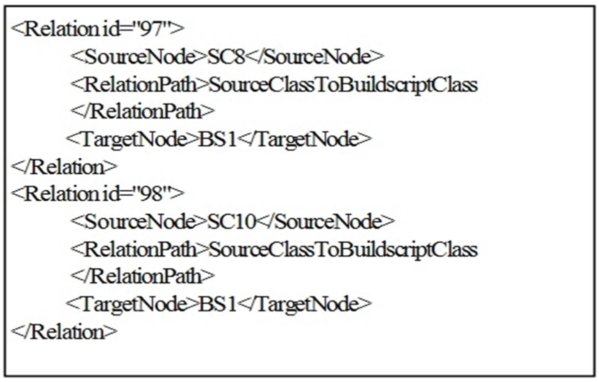


Fig. 7. E-school management system design diagram.



Requirement	Design	Source Code	Unit Test	Build Script
RQ1: view	D1: PersonList	SC1: AddCourse	UT1: AddCourseTest	BS1: eSchoolManagementSystem-parent
RQ2: teacher	D2: Person	SC2: AddUser	UT2: AddUserTest	BS1_P1: [maven-compiler-plugin
RQ3: student	D3: Student	SC3: CheckCourse	UT3: CheckCourseTest	BS1_P2: maven-jar-plugin
RQ4: person	D4: Teacher	SC4: CheckStudentAndUser	UT4: CheckStudentAndUserTest	BS1_P3: maven-source-plugin
RQ5: fee	D5: CourseList	SC5: ConfirmEnroll	UT5: ConfirmEnrollTest	BS1_P4: maven-remote-resources-plugin
RQ6: admin	D6: Course	SC6: Course	UT6: CourseListTest	BS1_P5: javax.servlet-api
RQ7: course	D1_F1: personList	SC7: CourseList	UT7: CourseTest	BS1_P6: commons-io
RQ8: list	D1_F2: isPersonLogged	SC8: Enroll	UT8: EnrollTest	
RQ1_F1: password	D1_M1: add	SC9: LoginFrame	UT9: LoginFrameTest	
RQ1_F2: person	D1_M2: getpersonFromIndex	SC10: MyProfile	UT10: MyProfileTest	
RQ2_F1: subject	D1_M3: getpersonFromUniqueID	SC11: Person	UT11: PersonListTest	
RQ3_F1: course	D1_M4: checkLoginDetail	SC2_M1: radioTeacherStateChanged	UT12: PersonTest	
RQ4_F1: address	D1_M5: getPersonLogged	SC2_M2: radioOfficeStateChanged	UT13: SearchUserTest	
RQ4_F2: dob	D1_M6: setPersonLogged	SC2_M3: radioStudentStateChanged	UT14: StaffMainPageTest	
RQ4_F3: name	D1_M7: teacherFromCourseCode	SC3_F1: admin	UT15: StudentMainPageTest	

Fig. 8. SAT-Analyser main artefact summary for E-school management system.

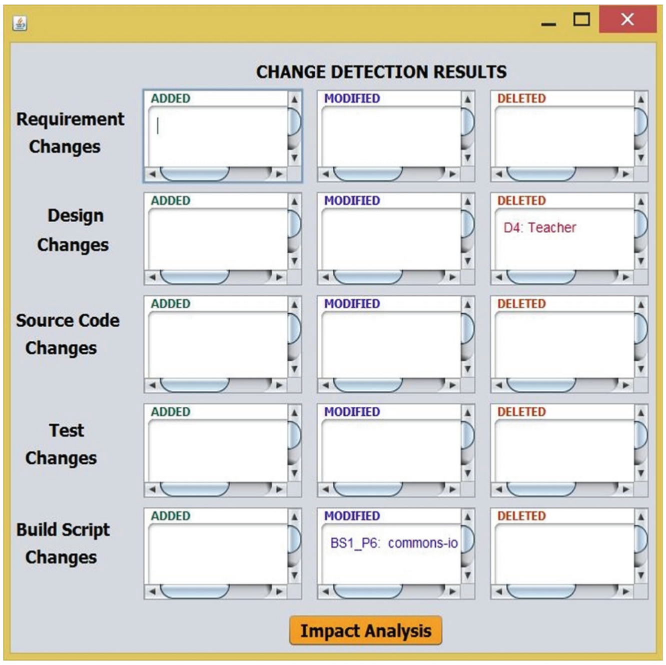


```
<Relation id="97">
        <SourceNode>SC8</SourceNode>
        <RelationPath>SourceClassToBuildscriptClass
        </RelationPath>
       <TargetNode>BS1</TargetNode>
</Relation>
<Relation id="98">
        <SourceNode>SC10</SourceNode>
        <RelationPath>SourceClassToBuildscriptClass
        </RelationPath>
       <TargetNode>BS1</TargetNode>
</Relation>
```

Fig. 9. Section of the traceability Relations.xml.

Moreover, we have validated the trace link creation between artefacts by performing a network analysis over the traceability graph. For instance, the build script artefact, BS1 has the maximum values for betweenness, closeness and in-degree centrality measures, among all other artefacts. Since the case study has one Maven build script and it is linked with each source class artefact, the obtained centrality result are considered as accurate. The isolated nodes show the minimum betweenness centrality, as they are not related to other artefacts.

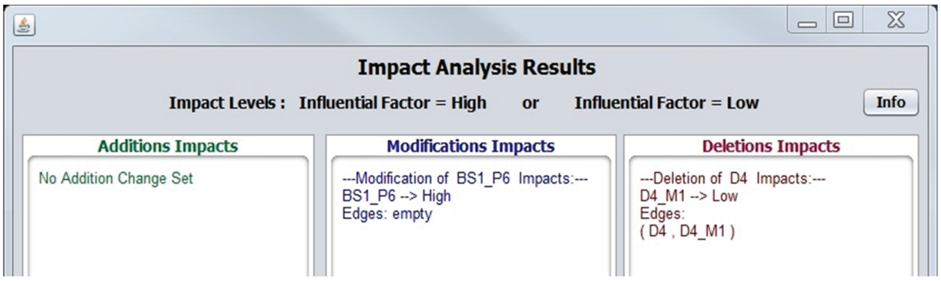
SAT-Analyser supports 17 change types for the impact analysis process [30], in which two change types are applied for this case study, i.e., (1) delete a design component and (2) modify a configuration artefact. Figs. 10 and 11 show the change detection results for these changes and the corresponding CIA results, respectively. The deletion of D4:Teacher design artefact has impacted its sole method existed (D4_M1). However, the modification performed on the BS1_P6 plugin name has not impacted any other as it is a leaf node with no outgoing edges to propagate, though its influential factor is high.



CHANGE DETECTION RESULTS

	ADDED	MODIFIED	DELETED
Requirement Changes			
Design Changes			D4: Teacher
Source Code Changes			
Test Changes			
Build Script Changes		BS1_P6: commons-io	

Fig. 10. Change detection results window for E-school management system.



Impact Analysis Results

Impact Levels : Influential Factor = High or Influential Factor = Low

Additions Impacts	Modifications Impacts	Deletions Impacts
No Addition Change Set	---Modification of BS1_P6 Impacts:--- BS1_P6 --> High Edges: empty	---Deletion of D4 Impacts:--- D4_M1 --> Low Edges: (D4 , D4_M1)

Fig. 11. A part of SAT-Analyser CIA window for the case study.

Further, a section of the propagated changes is shown in Fig. 12 as an interactive traceability graph, where nodes denote the artefact and edges show the links. For instance, the node BS1 (black) represents the Maven pom.xml build script and each source code class (SC) is shown in red nodes. The notations UT, D, RQ stand for the unit test class, design diagram class, and requirement item, respectively. The node names are shown when hovered over them in real-time. The nodes D4 and its impacted D4_M1 have been completely removed, while BS1_P6 is marked as modified in a larger size.

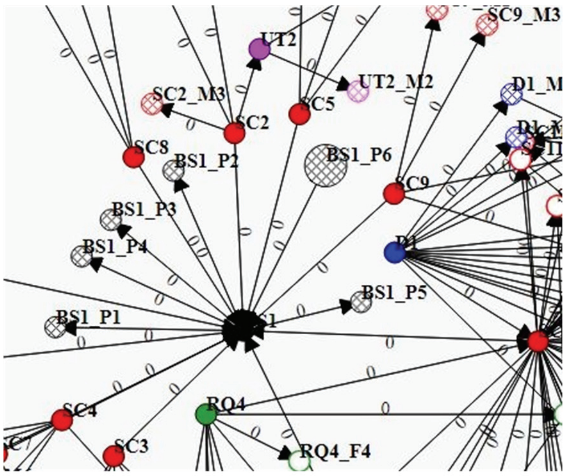


Fig. 12. Section of the change propagated interactive traceability graph.

7. Discussion

The traceability solution of SAT-Analyser addresses the major limitations in related studies such as being limited for a certain type of artefacts, lack of automation and incapability to handle continuous integration activities. Table 2 presents a comparative analysis between existing software traceability management tools and the proposed SAT-Analyser tool [31-39]. Mainly, this tool considers heterogeneous artefacts throughout the SDLC, presents an artefact consistency management approach to cope with frequent changes in a DevOps environment, and provides independent tool support that is compatible with DevOps. SAT-Analyser is designed with a traceability process model that addresses requirements, design, source code, test script and configuration artefacts in their raw formats. The trace-link visualization consists of interactive, analytical and informative graphs to handle the scalability issues and visual clutter in an excessive number of artefacts.

Table 2. Comparative analysis between existing tools and proposed SAT-Analyser

Tool	Artefacts	Traceability creation	Visualization	Change detection	CIA	Change propagation
JTracker [31]	Source code		√	√	√	√
JRipples [32]	Source code			√	√	√
TraceME [33]	Requirement, design, source code, test script	√	√		√	
TraceViz [34]	Multimedia	√	√			
DCTracVis [35]	Requirement, source code	√	√	√	√	√
Imp [36]	Source code				√	
FaultTracer [37]	Source code, test scripts			√	√	√
ImpactMiner [38]	Source code				√	√
ChangeDistiller [39]	Source code				√	
SAT-Analyser	Requirement, design, code, test script, build script	√	√	√	√	√

The artefact consistency management module consists of artefact change detection, CIA and change propagation that manages frequent changes in the CICD pipeline [40]. Artefact changes are detected following an XML comparison approach with artefact version management during CI activities. CIA is computed on the detected artefact changes using eigenvector centrality measure in traceability network analysis. Then, SAT-Analyser propagates the changes on impacted artefacts by updating the traceability graphs accordingly using a single-source Dijkstra graph traversal algorithm with a rule-based approach. Simultaneously, it notifies the involved DevOps environment teams about the traceability status in every change propagation instance. Thus, SAT-Analyser handles heterogeneous artefact consistency management in a change-intensive development environment in parallel to traceability support.

SAT-Analyser is designed and implemented as an independent tool similar to existing solutions. It is available in both stand-alone desktop and multi-user browser-based web version. Additionally, this tool can function collaboratively with prominent DevOps tools Git, Jenkins and Trello. The tool installation does not require additional resources as it uses only applications such as PyThon packages, wamp server, JDK Java, WordNet. Further, the performance analysis feature helps for resource management in DevOps practice without letting traceability support a burden in practice. Therefore, we have addressed all the existing traceability management related research questions in an applicable way into the CICD pipeline in a DevOps environment [41,42].

Further analysis of the SAT-Analyser tool is done in an experimental space using different case studies such as POS system [28], tour management system [29] and resulted in high accuracy. Finally, we performed a user study for the tool SAT-Analyser among the DevOps practitioners. All the participants in the system usability scale (SUS) study are experienced software practitioners in leading software companies in Sri Lanka, who are working in both DevOps and Agile environments. They found SAT-Analyser is innovative and useful since they could minimize manual tracing activities using it. The SUS is applied in assessing the user study results and above 70% of the users declared the tool as “well integrated” and as “likely to use frequently.” Further, we collected the user feedback in expressive terms and generated a tag cloud to visualize the overall user ideas about the tool. Correspondingly, the terms “usable,” “supportive,” “innovative,” “improvable,” and “novelty” are highlighted in the tag cloud depending on the frequency of feedback terms.

8. Conclusion

This paper presented a traceability tool, SAT-Analyser, and addressed the research hindrance of software traceability management within Agile-based DevOps environments. We have proposed an approach for software artefact consistency management that can cope with frequent artefact changes in a DevOps environment by supporting the CICD process. The main modules of this process are artefact change detection, CIA, change propagation, visualization, and validation. The compatibility of SAT-Analyser into CICD in practice is ensured by integrating it with leading DevOps tools, facilitating web-based multi-user accessibility for DevOps teams and performance analysis that helps for optimum resource utilization in DevOps environments. Case studies have shown the applicability of the tool in different software projects domains with continuous artefact changes. Thus, SAT-Analyser fulfils the traceability management support into DevOps by addressing the research gap and practical limitations. This research can be extended in several ways. NLP models can be improved for efficient artefact pre-processing. Machine learning techniques can be used to establish trace links to mitigate the artefact naming convention issues. The tool can be extended by incorporating more artefact types such as multiple design diagram categories, support for many programming languages and deployment scripts. Further, SAT-Analyser can be extended as a recommender system to measure the artefact quality in a software development environment.

Acknowledgement

We acknowledge the support received from the Conference & Publishing Support (Senate Research Committee), University of Moratuwa, Sri Lanka in publishing this paper.

References

[1] I. Sommerville, *Software Engineering*, 9th ed. Boston, MA: Pearson, 2011.

[2] G. Kim, J. Humble, P. Debois, and J. Willis, *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. Portland, OR: IT Revolution Press, 2016.

[3] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Boston, MA: Pearson, 2015.

[4] P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Boston, MA: Pearson, 2007.

[5] A. Berg, *Jenkins Continuous Integration Cookbook*. Birmingham, UK: Packt Publishing Ltd., 2012.

[6] J. Cleland-Huang, O. Gotel, and A. Zisman, *Software and Systems Traceability*. London, UK: Springer, 2012.

[7] Trello [Online]. Available: https://trello.com/.

[8] R. Oliveto, "Traceability management meets information retrieval methods-strengths and limitations," in *Proceedings of 2008 12th European Conference on Software Maintenance and Reengineering*, Athens, Greece, 2008, pp. 302-305.

[9] E. Kitsu, T. Omori, and K. Maruyama, "Detecting program changes from edit history of source code," in *Proceedings of 2013 20th Asia-Pacific Software Engineering Conference (APSEC)*, Bangkok, Thailand, 2013, pp. 299-306.

[10] K. Kamalabalan, T. Uruththirakodeeswaran, G. Thiyagalingam, D. B. Wijesinghe, I. Perera, D. Meedeniya, and D. Balasubramaniam, "Tool support for traceability of software artefacts," in *Proceedings of 2015 Moratuwa Engineering Research Conference (MERCon)*, Moratuwa, Sri Lanka, 2015, pp. 318-323.

[11] D. Meedeniya, I. Rubasinghe, and I. Perera, "Artefact consistency management in DevOps practice: A survey," in *Tools and Techniques for Software Development in Large Organizations: Emerging Research and Opportunities*. Hershey, PA: IGI Global, 2020, pp. 98-129.

[12] X. Sun, B. Li, C. Tao, W. Wen, and S. Zhang, "Change impact analysis based on a taxonomy of change types," in *Proceedings of 2010 IEEE 34th Annual Computer Software and Applications Conference*, Seoul, Korea, 2010, pp. 373-38.

[13] W. T. Lee, W. Y. Deng, J. Lee, and S. J. Lee, "Change impact analysis with a goal-driven traceability-based approach," *International Journal of Intelligent Systems*, vol. 25, no. 8, pp. 878-908, 2010.

[14] A. Goknil, I. Kurtev, K. Van Den Berg, and W. Spijkerman, "Change impact analysis for requirements: A metamodeling approach," *Information and Software Technology*, vol. 56, no. 8, pp. 950-972, 2014.

[15] Y. Wang, J. Zhang, and Y. Fu, "Rule-based change impact analysis method in software development," in *Proceedings of the 2nd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2017)*, Wuhan, China, 2016, pp. 384-391.

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

[17] S. Wong, Y. Cai, and M. Dalton, "Change impact analysis with stochastic dependencies," 2011 [Online]. Available: https://www.cs.drexel.edu/files/jmf323/wong-icse11b_0.pdf.

[18] S. Lehnert and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," in *Proceedings of 2013 17th European Conference on Software Maintenance and Reengineering*, Genova, Italy, 2013, pp. 209-218.

[19] V. Farcic, *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices*. Victoria, Canada: Leanpub, 2016.

[20] A. Arunthavanathan, S. Shanmugathasan, S. Ratnavel, V. Thiyagarajah, I. Perera, D. Meedeniya, and D. Balasubramaniam, "Support for traceability management of software artefacts using natural language processing," in *Proceedings of 2016 Moratuwa Engineering Research Conference (MERCon)*, Moratuwa, Sri Lanka, 2016, pp. 18-23.

[21] I. Rubasinghe, D. Meedeniya, and I. Perera, "Tool support for software artefact traceability in DevOps practice: SAT-Analyser," in *Tools and Techniques for Software Development in Large Organizations: Emerging Research and Opportunities*. Hershey, PA: IGI Global, 2020, pp. 130-167.

[22] Jaro-Winkler Distance [Online]. Available: https://scholar.harvard.edu/jfeigenbaum/software/jaro-winkler-distance.

[23] The Levenshtein-Algorithm [Online], Available: http://www.levenshtein.net.

[24] Neo4j Inc., "Graph Visualization Tools," 2021 [Online]. Available: https://neo4j.com/developer/tools-graph-visualization/.

[25] Gephi, "The Open Graph Viz Platform," 2017 [Online]. Available: https://gephi.org/.

[26] NetworkX [Online]. Available: https://networkx.github.io.

[27] D3.js: data-driven documents [Online]. Available: https://d3js.org/.

[28] I. D. Rubasinghe, D. A. Meedeniya, and I. Perera, "Software artefact traceability analyser: a case-study on POS system," in *Proceedings of the 6th International Conference on Communications and Broadband Networking*, Singapore, 2018, pp. 1-5.

[29] I. Rubasinghe, D. Meedeniya, and I. Perera, "Automated inter-artefact traceability establishment for DevOps practice," in *Proceedings of 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS)*, Singapore, 2018, pp. 211-216.

[30] SAT-Analyser [Online]. Available: https://sites.google.com/cse.mrt.ac.lk/sat-analyser/case-studies.

[31] S. Gwizdala, Y. Jiang, and V. Rajlich, "Tracker: a tool for change propagation in Java," in *Proceedings of the 7th European Conference on Software Maintenance and Reengineering*, Benevento, Italy, 2003, pp. 223-229.

[32] J. Buckner, J. Buchta, M. Petrenko, and V. Rajlich, "JRipples: a tool for program comprehension during incremental change," in *Proceedings of the 13th International Workshop on Program Comprehension (IWPC)*, St. Louis, MO, 2005, pp. 149-152.

[33] G. Bavota, L. Colangelo, A. De Lucia, S. Fusco, R. Oliveto, and A. Panichella, "TraceME: traceability management in eclipse," in *Proceedings of 2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Trento, Italy, 2012, pp. 642-645.

[34] R. Dautriche, R. Blanch, A. Termier, and M. Santana, "TraceViz: a visualization framework for interactive analysis of execution traces," in *Actes de la 28ième conference francophone sur l'Interaction Homme-Machine*, Fribourg, Switzerland, 2016, pp. 115-125.

[35] X. Chen, J. Hosking, J. Grundy, and R. Amor, "DCTracVis: a system retrieving and visualizing traceability links between source code and documentation," *Automated Software Engineering*, vol. 25, no. 4, pp. 703-741, 2018.

[36] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in *Proceedings of 2011 33rd International Conference on Software Engineering (ICSE)*, Honolulu, HI, 2011, pp. 746-755.
[37] L. Zhang, M. Kim, and S. Khurshid, "Faulttracer: a change impact and regression fault analysis tool for evolving java programs," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, Cary, NC, 2012, pp. 1-4.
[38] B. Dit, M. Wagner, S. Wen, W. Wang, M. Linares-Vasquez, D. Poshyvanyk, and H. Kagdi, "Impactminer: a tool for change impact analysis," in *Companion Proceedings of the 36th International Conference on Software Engineering*, Hyderabad, India, 2014, pp. 540-543.
[39] H. C. Gall, B. Fluri, and M. Pinzger, "Change analysis with evolizer and changedistiller," *IEEE Software*, vol. 26, no. 1, pp. 26-33, 2009.
[40] I. D. Rubasinghe, D. A. Meedeniya, and I. Perera, "Towards traceability management in continuous integration with SAT-Analyzer," in *Proceedings of the 3rd International Conference on Communication and Information Processing*, Tokyo, Japan, 2017, pp. 77-81.
[41] D. A. Meedeniya, I. D. Rubasinghe, and I. Perera, "Software artefacts consistency management towards continuous integration: a roadmap," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 4, pp. 100-110, 2019.
[42] D. A. Meedeniya, I. D. Rubasinghe, and I. Perera, "Traceability establishment and visualization of software artefacts in DevOps practice: a survey," *International Journal of Advanced Computer Science and Applications*, vol. 10, no. 7, pp. 66-76, 2019.

Iresha Rubasinghe **https://orcid.org/0000-0001-9232-3648**

She received B.Sc. (Hons) in Software Engineering from the University of Colombo, School of Computing. Her M.Phil. is from the Department of Computer Science and Engineering from the University of Moratuwa, Sri Lanka.

Dulani Meedeniya **https://orcid.org/0000-0002-4520-3819**

She is a senior lecturer in the Department of Computer Science and Engineering, at the University of Moratuwa, Sri Lanka. She holds a Ph.D. from the University of St Andrews, UK. Her main research interests are Software Engineering and Data Analytics. She is a fellow of HEA (UK), MIET and C.Eng. (UK).

Indika Perera **https://orcid.org/0000-0001-5660-248X**

He is a professor at the University of Moratuwa, Sri Lanka. He holds a Ph.D. (St Andrews, UK) M.BS. (Colombo), M.Sc. (Moratuwa), and B.Sc. Eng. (Hons) (Moratuwa). His research interests include Software Engineering and HCI. He is a Fellow of HEA (UK), MIET, SMIEEE and a C.Eng. at EC (UK) and IE (SL).