Traceability Establishment and Visualization of Software Artefacts in DevOps Practice: A Survey

D. A. Meedeniya[1], I. D. Rubasinghe[2], I. Perera[3]
Dept. of Computer Science and Engineering, University of Moratuwa, Sri Lanka

***Abstract*—DevOps based software process has become popular with the vision of an effective collaboration between the development and operations teams that continuously integrates the frequent changes. Traceability manages the artefact consistency during a software process. This paper explores the trace-link creation and visualization between software artefacts, existing tool support, quality aspects and the applicability in a DevOps environment. As the novelty of this study, we identify the challenges that limit the traceability considerations in DevOps and suggest research directions. Our methodology consists of concept identification, state-of-practice exploration and analytical review. Despite the existing related work, there is a lack of tool support for the traceability management between heterogeneous artefacts in software development with DevOps practices. Although many existing studies have low industrial relevance, a few proprietary traceability tools have shown a high relevance. The lack of evidence of the related applications indicates the need for a generalized traceability approach. Accordingly, we conclude that the software artefact traceability is maturing and applying collaboratively in the software process. This can be extended to explore features such as artefact change impact analysis, change propagation, continuous integration to improve software development in DevOps environments.**

Keywords—Software traceability; visualization; comparative study; DevOps; continuous software development

I. Introduction

Software system is an asset that contributes to enhance products or services. Change of a software system is inevitable as the requirements are evolving over time. Technology improvements, environmental changes, modifications in legal bodies and many factors affect requirement changes. Therefore, a continuous system update is required to cope with the artefact changes by preserving the value of the software. Hence, considering the usage of resources, time, cost and effort, software evolution is preferred over building a new software system to manage the changes [1]. Generally, software evolution is identified as a maintenance task due to new change requests during the Software Development Life Cycle (SDLC). The software evolution mainly depends on the type of software being maintained, the development processes, and directly affects the related software artefacts.

Software artefacts are the intermediate by-products used in each stage of the SDLC that contribute towards the outcome of an intended product. For instance, Software Requirement Specification (SRS), non-functional design reports, design diagrams, source code, test cases, test scripts, bug reports, walkthroughs, inspections, configuration files, build logs, project plans, risk assessments and user manuals are some of the artefacts in the SDLC [2][3]. There are different forms of relationships between the homogeneous and heterogeneous software artefacts. Some artefacts may be highly coupled, and some may depend on other artefacts in different degrees, unidirectionally or bidirectionally. Thus, software artefacts consistency management helps to fine-tune the software process. The incomplete, outdated software artefacts and their inconsistencies mislead both the development and maintenance process. Thus, artefact management is essential such that the changes are accurately propagated to the impacted artefacts without creating inconsistencies. Traceability is the potential to relate artefacts considering their relationships [4][5]; thus, a solution for artefact management. Being an active research topic, many studies have discussed the different aspects of traceability, tool support in different scopes and domains.

At present, DevOps, that unifies the process between the development (Dev) and operation (Ops) teams, has become a popular software development practice. DevOps environment supports to build, test and deliver the product at a high demand and results in faster evolvements of the products [6][7]. The concepts, Continuous Integration (CI) and Continuous Delivery (CD) encourage to accept frequent changes at any phase of the SDLC in DevOps practice [8]. Thus, artefact management is essential to achieve in DevOps environments to avoid artefact inconsistencies. However, it is challenging to ensure traceability with maximum automation due to frequent integrations. Further, the practical use of artefact traceability in DevOps is not widely in use due to the limitations in existing traceability techniques, tools and automation capabilities. Thus, auditability and traceability are challenging in DevOps [9].

We present a survey on artefact traceability management in DevOps practice. The traceability concepts and terminology are described in Section II. Section III and Section IV explore the traceability creation techniques and related visualization methods, respectively. Related traceability management studies in DevOps practice with the conceptual traceability models are discussed in Section V. Moreover, the tool support to manage traceability is explored in Section VI. Section VII explores the traceability evaluation methods using quality aspects and network analysis. The associated challenges and limitations are discussed in Section VIII. Finally, Section IX concludes the survey with the identified suggestions and possible future directions for traceability support in DevOps practice.

II. Background

A. Concept of Traceability

A software system is a combination of several artefacts that evolves through a certain software development process model.

It is important to manage the relationships and dependencies between these software artefacts to maintain the consistency of the product. The outdated artefacts can lead to artefact inconsistency, synchronization issues and lack of stakeholders' trust in artefacts [10]. Thus, it is essential to manage the artefact traceability in software development with DevOps practices that involve frequent Continuous Integration and Continuous Delivery (CICD).

Traceability provides a logical connection between the artefacts of the software development process. It is important to maintain the traceability among both homogeneous and heterogeneous software artefacts throughout the SDLC stages covering the requirements gathering, design, development, testing, maintenance and deployment. For example, the ability to track the relationships between requirements and their sources is essential to revise the initially gathered set of requirements [2]. This concept was initially used as a method of managing requirements artefact during the requirements engineering phase [11]. Generally, traceability is defined as the ability to follow the life cycle of a software requirement both forward and backwards and overcome the inconsistencies during software development [4]. Thus, each alteration occurs in each requirement is traced among other requirements and changed accordingly based on the impact. These traces are used in the requirement validation and verification processes.

Center of Excellence for Software and Systems Traceability (CoEST) has defined traceability as "the ability to interrelate any uniquely identifiable software engineering artefact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process". They have not limited to requirement traceability and have declared traceability in terms of other artefact types such as design documents, source code and test cases with the deployment of an experimental traceability environment for researchers called TraceLab [12]. Traceability is defined as the ability to trace the dependent items within a model and trace the corresponding items in other models [13]. As a result, currently, traceability is used not only in requirements management but also for other artefact types in different software development methodologies like Model-Driven Development (MDD) [14]. This wide range of adaptation of traceability shows its importance in improving software quality, maintenance, evolution and reuse activities.

B. *The Terminology of Software Artefact Traceability*

Traceability refers to the ability or the potential of tracing a change propagation among artefacts in a software system. For a given trace, there can be one or many possible trace paths, while each trace path has a source and target artefacts. An artefact may be a source for a given trace path and a target for another trace path, simultaneously. A trace link or a traceability link is a relationship between a pair of artefacts. All trace links generated between two sets of artefacts are referred to as a trace relation [4]. A trace set is the sum of all generated traces and traceability graph is used to visualize all the relationships. A traceability graph is a traceability network when the edges are directional, or the nodes are embedded with a weight. Further, traceability maintenance manages the consistency of the artefacts and updates the traces for a given change.

Different traceability classifications exist in the literature as shown in Fig. 1. One such classification is automatic or manual, based on the automation level of the traceability process. Another classification is forward or backwards, that is based on the direction of the traceability path [4]. Forward tracing follows subsequent steps such that from requirements to code; whereas backward tracing follows antecedent steps such that code to design or requirements artefacts. Artefact-level is another criterion that classifies traceability as horizontal or vertical. Horizontal tracing reflects homogeneous artefacts, which are at the same level of abstraction such as tracing between different versions of requirements [15]. Further, this can be sub-classified based on the direction such that horizontal forward tracing or horizontal backward tracing. Tracing heterogeneous artefacts that are in different levels of abstraction, such as the requirement to code, is considered as vertical tracing, which can be either vertical forward tracing or vertical backward tracing. Proactive and reactive tracing is another categorization based on stimuli behaviour. In reactive tracing, the traces are created on demand by responding to a stimulus to initiate the trace capture. Whereas in proactive tracing, traces are created in the background without explicit response to any stimulus [4]. The traceability link generation techniques (see Section IV) that are based on these categories are selected by considering aspects such as the problem domain and the behaviour of the software system.

C. *Traceability in DevOps Practice*

The DevOps concept represents the collaboration of the development and the operational teams [6][9]. DevOps eases the project team management with communication, understandability, integration and relationships by bridging the gap between the development and operational teams. This CICD process increases the rate of change and deploys the features into production faster [16][7]. Thus, DevOps-based software development improves the quality, customer experience and supports simultaneous deployment in different platforms. The associated cross-functionality behaviour enables the early identification of ambiguities, reduction of the error fixing time and reduction of the problem complexities. The importance of DevOps towards the business aspect is also significant to shorten the development life cycle, increase the release velocity and improve the Return on Investment (ROI) by achieving a higher customer satisfaction [6]. Further, rich collaboration and performance-oriented culture encourage the ability to research and innovate within projects. However, the Internet of Things (IoT) and Microservices architecture are identified to be challenging in DevOps [9].

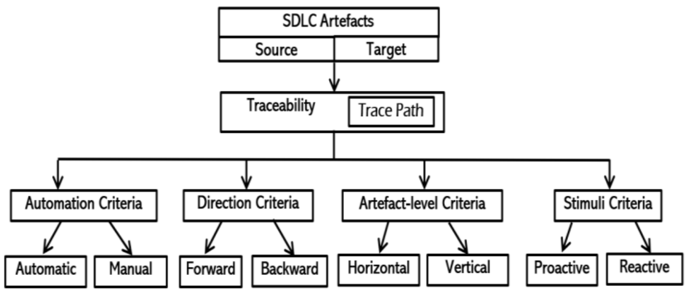

Fig. 1. Summary of Traceability Classification.

DevOps environments associate five main principles: continuous planning, continuous integration, continuous delivery, continuous testing and continuous deployment [6]. The CI process integrates work frequently that leads to multiple integrations per day and deploys effectively [17][18]. Generally, CI verifies the integration using build automation by early detection of integration errors. The ability to trace the artefact changes is essential to notify feedback at an integration failure [19]. Thus, it is important to have software artefact traceability support for the CI process to handle changes.

III. Traceability Establishment

A. *Information Retrieval and Data Pre-Processing*

In software development, different types of artefacts are considered for traceability creation such as requirements to design, requirements to source code and test cases. For instance, requirement traceability shows the mapping of the requirements to other stages of the SDLC; design traceability refers the ability to trace design and requirements to design rationale for verifying and maintaining the architectural design [20]. In a DevOps environment, the design traceability helps to identify the change impacts, trace design evolution, relate design objects and analyse the cross-cutting concerns.

Generally, Information Retrieval (IR) methods and data pre-processing are considered as pre-requisites to the traceability establishment process. Software artefacts consist of different formats such as the requirements in natural language, design artefacts in different Unified Modelling Language (UML) notations and source code artefacts in programming languages. Thus, pre-processing techniques should be applied to extract the required data as an initial task towards the generation of traceability links. Most of the time, the textual content in the artefacts provides descriptive details about its informal semantics. The frequently involved pre-processing steps for textual-based requirements artefacts are Natural Language Processing (NLP) tasks such as tokenization, text normalization, anaphora analysis, morphological analysis and stemming [4][21]. It is assumed that the artefacts are conceptually related if their textual contents are similar Thus, trace links can be created among them. Hence, the other types of artefacts can be pre-processed with different file readers, UML parsers and programming language specific parsers.

IR methods enable the extraction and analysis of the embodied textual contents in artefacts with a less pre-processing effort [4]. It minimizes the cost of traceability link recovery as it does not consider predefined vocabulary or grammar. The key steps in a generalize IR process that follows a pipelined architecture are: (1) document parsing, extraction and pre-processing, (2) corpus indexing with an IR method, (3) ranked list generation and (4) analysis of candidate links. Moreover, most IR related techniques are Vector Space Model (VSM), Latent Semantic Indexing (LSI) and Term Frequency-Inverse Document Frequency metric (tf-idf) and they have provided better performance outcomes in the literature [22].

B. *Traceability Establishment Approaches*

Different types of approaches have been used in the literature to generate traceability links between software artefacts. This section discusses the widely used software artefact traceability establishment approaches and Table I states a comparison of these approaches.

TABLE I. Traceability Establishment Approaches

Method	**Description**	**Advantages**	**Limitations**
Rule-based	Defines a rule set based on artefact attributes. Manages traceability with rule re-evaluation [23].	Works well with artefacts such as requirements, use cases, object models [23].	Structural changes are hard to identify [4].
Hypertext	Manage traceability using XML markup specifications [23].	Works with requirements and code [4].	Weekly supports the other types of artefacts.
Event-based	Manage traceability using publish-subscribe links and event-based subscriptions [24].	Maintains dynamic links.	Scalability issues in maintaining the dynamicity of the traceability [24].
Constraint-based	Provides a set of constraints which must not be violated by traceability links [25].	Views artefact types as constraints among them.	Difficult to refer all the traceability links with the constraints [25].
Transformations	Uses incremental [26] and graph-transformation based methodologies.	Suited for model-based software systems [26].	Difficult to apply to artefacts that are not generated using MDD [26].
Goal-centric (GCT)	Manages the change impact of non-functional requirements. Use soft goal interdependency graph and traceability matrix [27].	Ensure quality by assessing the change impact of functional vs. non-functional aspects [27].	Lack of scalability and tool support [30].
Model based	Manages traceability using template-based models [28].	Supports different artefact types [28].	Lack of support towards non-MDDD [28].

Rule-based traceability generates different types of trace links between artefacts based on the semantics and the grammatical features of their words [29]. First, the traceability link generation rules are defined based on the attributes of the artefacts and then the traceability maintenance phase re-evaluates the rules. Moreover, rule-based approaches can be combined with event-driven approaches. Thus, the traceability maintenance can be conducted in two phases: (1) recognizing changes based on events and (2) re-evaluating the rules that governing link updates [26]. However, these rule-based traceability techniques are not applicable to all types of artefacts rather than requirements and source code [23].

Hypertext-based traceability uses an underlying XML representation along with the conformance analysis. This is suitable for complex and versioning of traceability links [4]. However, hypertext-based traceability support technique is also limited to software artefacts such as requirements and source code [23].

The event-based approaches consider the events occur during the software development activities to maintain the traceability links. For example, the deletion of an artefact can be made as a trigger to delete all the connected traceability links. Maro et al., have addressed this using a similar conceptual technique; publish-subscribe mechanism, that connects traceability maintenance tasks to events [26]. However, this technique has scalability issues [24][25].

Various other approaches that can be used to establish traceability are available in the literature. The model-based traceability establishment approach manages the inter-relationship of models using XML representations, without narrowed only into UML, Business Process Model and Notation (BPMN), feature models and systematic review for architecture to code traceability [27][28]. Constraint-programming is another approach that declares the valid rules for traceability links [25]. Here, the traceability links that are not referenced by constraints are considered as consistent by default. The transformation-based approaches [26], mainly graph-transformations are used to generate traceability links based on the artefact transformations. However, these approaches are not widely used in practice. The Design Decision Tree (DDT) provides the ability to connect requirements to architecture decision and design elements under the traceability establishment. The traceability model presented in [20], has addressed the traceability in a design rationale model using the conceptual UML notations. However, it captures relationships between only two entities, architecture rationale and architecture elements. Further, a scoped-based approach was explored by Patricia et. al., [30], that emphasizes the current status of considering traceability in a given situation rather than presenting all the traces. This has been an attempt to minimize the traceability cost by engaging a manageable minimal set of links.

IV. Traceability Visualization

Software artefact traceability visualization helps the decision-making process to analyse the relationships among artefacts. However, it is challenging to visualize many traceability links and paths among software artefacts in real-time with the evolving inter-relationships. The challenges include scalability and visual clutter related issues. Although there are data visualization techniques and tools to analyse large temporal data, the selection of an optimal representation depends on different properties in the traceability links. This section discusses traceability visualization techniques and Table II summarizes a comparison of these techniques.

Traceability matrix is used to record the trace relations. Initially, requirements traceability matrix has used to associate requirement artefacts during the requirements engineering process [31]. It shows the associated or dependent pairs of artefacts using the trace links [4]. The work done by Cleland-Huang has shown the possibility of increasing the cost of traceability creation and maintenance by using this technique, although the row-column structure is simple. Since this representation is easily readable by the stakeholders, a single repository is sufficient to document both forward and backward traceability results, which is an advantage in terms of storage. However, representing many artefacts and trace links using a traceability matrix is less practical due to the complexity in access, search and update operations.

The hierarchical tree is a node-link based representation that uses lines to connect parent and child nodes. This representation is easily understandable and communicates a hierarchical structure. There are two sub-approaches in this hierarchical tree visualization. The first approach has edges between relevant children nodes and group edges using the hierarchical edge bundling technique. However, this method has the drawback of visual clutter with a larger number of traceability links [35]. The second approach directly adds traceability links as children of leaf nodes. Further, this technique is used to represent detailed dependency information of an item. In related work [31], hierarchical tree visualization is used as a supplement for Tree-map visualization to illustrate the detailed information on each trace.

TABLE II. Traceability Visualization Techniques

Method	Features	Advantages	Limitations
Lists [32]	Show data in 1-dimension, sequentially.	Efficiency due to simplicity.	Limited for a small-scale data due to a single dimension.
Traceability matrix [31][33][34]	Stores data in 2-dimensional grid structure.	Well-represents a small set of artefacts.	Impractical to represent a larger number of trace links.
Cross-reference [31]	Represents data in a table structure.	Provides a list of related links for an artefact.	Hard to show the full trace structure. Scalability issues.
Tree-map [33][35]	Uses a tree data structure to represent data in 2D.	Represents a large tree by optimum display space.	Hard to communicate with the layers. Complex for a larger set of traceability links.
Hierarchical tree [35]	Node-link representation to show data hierarchically.	Gives trace dependency data. Simple to understand.	Visual clutter when a larger number of traceability links are involved.
Traceability graph [33][36][37]	Show data as nodes, links as edges.	Visualizes structured data with relations.	Limit the graph view for excessive nodes. Performance issues.
Sunburst and Netmap [32][33]	Use a radial layout.	Browse, navigate with user orientation.	Not filter the visualization links.

Traceability graph visualizes a trace set in a node-link format, where trace artefacts as nodes and trace links as edges [4]. The associated visual clutter can be reduced and can enhance the readability of the traceability graph by using colour codes for nodes and edges based on their type, category and direction [38]. The traceability outcome can be analysed by graph traversal and graph analysis methods. Moreover, network analysis methods discussed in Section VII can be applied to analyse these network graphs. However, this technique has performance issues when there is an excessive number of nodes and links.

Among many other visualization techniques, lists are a single dimension primary approach that can be applied for a small set of data. This technique shows efficient performance due to its simple structure [32]. Cross-reference is a tabular structure that represents a list of related links for a given software artefact. However, this technique does not support to obtain an overall traceability representation [31]. Tree-map approach represents trace data in a 2D tree structure and supports a larger set of data. However, the communication among each trace artefacts by traversing the tree tends to be complex [33]. Further, Sunburst and Netmap visualization [32] is a radial layout representation approach.

Most of the traceability visualization techniques have slightly considered model driven features; thus, there is a limit of supporting to a range of software types [36][37]. Many related works have addressed issues such as visual clutter and scalability [32][33] and several tools are integrated with a specific Integrated Development Environment (IDE). However, most of the studies have not addressed different types of software artefacts, as they have considered a certain type of artefacts such as either requirements or source code. Thus, there is a potential need for a generic software artefact visualization methodology that can accommodate artefact representation independent of its type and scale.

V. Traceability in Practice

A. Models for Traceability in DevOps Practice

Although there are some attempts in the recent literature to adapt traceability into Agile environments, traceability management in DevOps practices has not been addressed well. A generic Agile Traceability Model (ATM) for managing Non-Functional Requirements (NFR) has presented by Firdaus et al. [39]. They have traced the effects of the frequent Functional Requirement (FR) changes on NFRs such as security and performance. This model is based on an example ATM [4] that trace requirements, source code and test cases and an NFR traceability metamodel that links the stakeholders and the requirement grouping component of the project. Each artefact and its elements of the FRs are used to trace NFR with a mediate association component. Fig. 2 shows the integration of ATM (blue in colour) and NFR traceability metamodel (grey in colour). Here, the code is backtracked to the requirements through testing. It is required to have test cases for NFRs without adding code to the overall model. Thus, the impact of development changes is traced using test cases. This conceptual model has been implemented as a prototype. Although the average value of precision and recall of the process model is 0.46, they have stated that the integration is mapped correctly.

A traceability approach named Trace++, that transforms a traditional software development to Agile environments has been proposed in [40]. They have addressed four transition issues such that (i) amount of rework per sprint, (ii) high-level understanding of a project scope before starting a sprint, (iii) lack of NFR documentation, and (iv) losing management control. They have considered extended information as shown in Fig. 3 when transforming to an Agile environment.

Another semi-automated traceability prototype for Agile development is proposed in [41], with the intention of achieving reusability of requirements artefacts. This work is based on an extension of an existing metamodel named TmM model [42]. The authors have identified reusable traceability links with the automated traceability link generation and Agile integration approaches as shown in Fig. 4. The user story change process is started when a change to a user story in the Agile team backlog occurs. If the responsible link is not found in the link repository, then the link generation process obtains rules from the metamodel and sends the result to the link maintenance process that modifies or removes the impacted links. The automated link generation process requires a manual confirmation from an authorized Agile team member via the link generation reviewer in IDE.

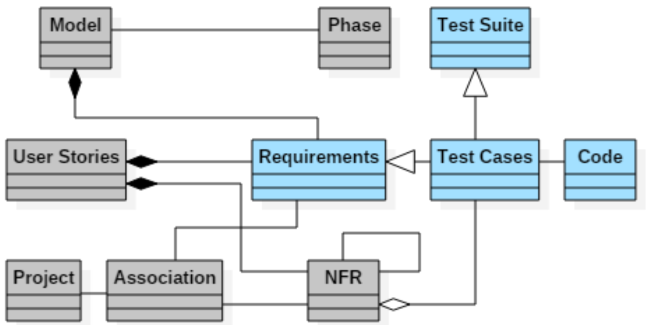

Fig. 2. Proposed NFR Agile Traceability Model [39].

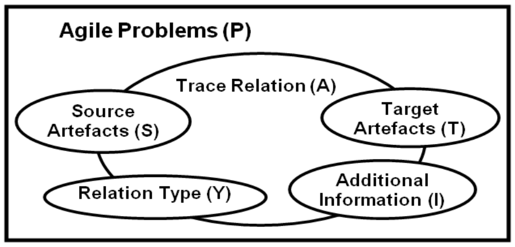

Fig. 3. Trace++ Traceability Solution Structure [40].

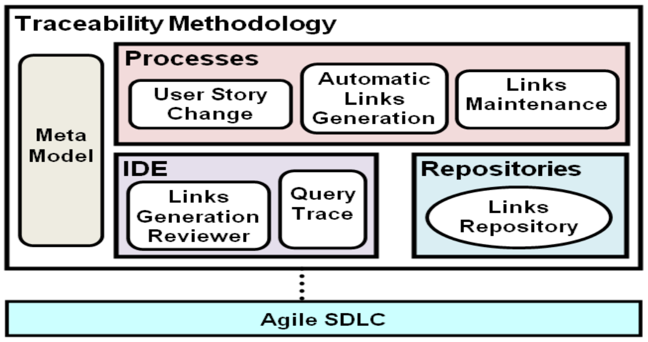

Fig. 4. Traceability Methodology for Requirements Reuse in Agile [41].

Generally, in a DevOps environment, distributed cloud platforms integrated software projects are monitored using logs and runtime information. These data are scattered in large-scale projects, due to the use of multiple tools and dashboards, hence hard to identify accurately. A traceability approach to monitoring these data in a DevOps environment has been proposed using explicit links between runtime information and source code in [43]. This work has used an online context graph to show the connections such that edges denote the implicit connections among information fragments. The conceptual model of this solution is shown in Fig. 5. A prototype of this context analytics model has been evaluated using cloud applications. The cost associated with the artefact pre-processing is low in this approach, as it considers runtime information with short textual artefacts. The results have shown a 48% reduction of efforts required in analysis steps and an average reduction of 40% for the required inspected traces.

A heuristic named SPEQTRA is proposed to locate the traces of automated tests in the CI process [44]. This approach enables the efficient continuation of a project when a test case is failed during the DevOps practice. Fig. 6 shows the trace execution of successful and unsuccessful test cases. This approach executes each test case records the traces and identifies the closet item set to transform traces to sequences via SPEQTRA. Then, the classes with sequences are ranked using the Jaccard similarity coefficient based on the fault likelihood to localize the classes with faults.

Similarly, another test case artefact-based tracing approach that supports CI for the automotive industry domain is proposed in [45]. They have used black-box testing following the input and output signals through process controllers as shown in Fig. 7. Every keyword in the test suites are used to trace the most suitable tests for a given CI task and a mapping table is maintained for the selection.

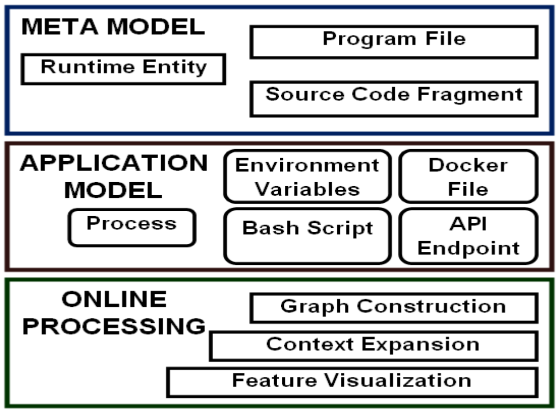

Fig. 5. DevOps Context-Based Analytics Conceptual Model [43].

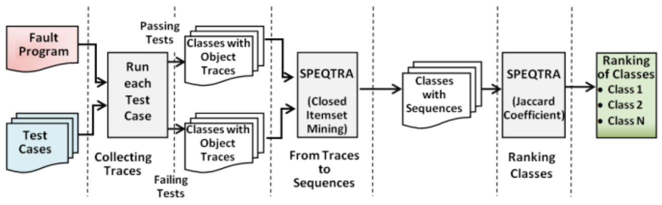

Fig. 6. SPEQTRA Heuristic Workflow [44].

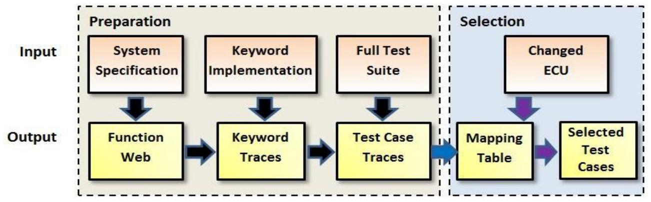

Fig. 7. Process Model for Trace-Based Test Selection [45].

Although several models are available that can manage software artefact traceability, most of them are restricted to a few artefact types. The model ATM [39] has considered requirement level artefacts, SPEQTRA [44] is based on test case artefacts, Trace++ [40] has addressed issues in the Agile-based development and other approaches have emphasized only on development level artefacts. A single model is not sufficiently addressed the traceability between heterogeneous artefacts in a CICD pipeline due to the inbuilt complexity; Thus, traceability management in a DevOps practice is still an active research area.

B. *Related Studies on Traceability Management*

Consistency management of a large set of artefact relationships, when a change occurs during the software development process is a costly task that consumes more time and effort. Although there are few studies on managing traceability in many artefacts, there are several studies that have focused on traceability maintenance of a small set of artefacts. It is important to ensure the correctness of traceability over time [26][23]. The proper identification of a feasible traceability management approach could minimize the associated cost and effort.

Traceability recovery management system incorporating IR techniques such as VSM and LSI were used in [46]. They have performed an incremental semi-automated traceability recovery by integrating into an artefact management tool called ADMS [47]. Moreover, they have identified quality issues in the textual descriptions of the traced artefacts using IR tools. However, their results have shown that IR techniques are not sufficient to identify all the traceability links and required to remove a large number of false-positives manually. An approach named SCOTCH+ (Source code and COncept based Test to Code traceability Hunter) was presented in [48] that has addressed the JUnit test to code traceability. The techniques; dynamic slicing and textual information analysis were used in this approach and have shown results with high accuracy.

The tool Software Artefact Traceability Analyser (SAT-Analyser) [21][48] has addressed the traceability among requirements artefact, UML class diagrams regarding the design artefact and the source code artefact in Java programming language. It has used NLP and traceability has been established based on a string similarity computation using the Jaro-Winkler algorithm and Levenshtein Distance algorithm along with WordNet synonyms and pre-defined dictionary ontology. Next, the artefacts and their established trace links were parsed through the Document Object Model (DOM) parser and converted into a predefined XML structure for traceability graph representation. However, this approach lacks the artefact support for the entire SDLC. Subsequently, this work has extended to support DevOps based software development [49][50]. This approach manages the traceability among heterogeneous artefact types that are involved in a DevOps environment covering each stage in SDLC. Additionally, this work has addressed continuous changes using change detection, change impact analysis and change propagation. Further, this study has supported the collaborative nature in DevOps practice by integrating the SAT-Analyser tool with the existing project management tools.

Accordingly, these related studies have mainly considered the automation of traceability management. Requirement, design level artefact traceability and enhancing traceability with visualizations have been slight considerations. The interest in obtaining complete automation in literature is advantageous for DevOps in practice. But having traceability support for all artefact types in overall SDLC phases is also required along with automation to be applicable to DevOps which is not significantly addressed together in literature.

VI. Traceability Management Tools

Variety of traceability management tools is available in the literature. These tools can be classified as proprietary tools: commercial and open source and research-based prototypes. Additionally, these tools can be grouped as non-Graphical User Interface (GUI) supportive and tools with fine-tuned IDEs. Among them, only a few sets of tools support traceability management in a DevOps environment, that considers all the artefacts from the requirement phase to delivery and maintenance phases. The overall analysis of commercial tools is presented in Table III. Considering the proprietary tools, TraceMaintainer [15] is an independent tool that supports any Computer-Aided Software Engineering (CASE) tool by allowing to work in any heterogeneous environment. However, this tool is limited to the requirements and design artefacts.

The tool TraceME [48] is a freely available integrative tool within the Eclipse IDE as a plugin. Tools such as TIRT [34], are limited to specific application domains such as software product line-based information retrieval. The artefacts such as test scripts, configuration files, deployment and delivery related artefacts in a DevOps environment have not significantly addressed in these existing tools. Thus, there is a need for tools with CI features that have addressed all the artefact types and support the CICD process.

Accordingly, lack of tool support for all types of software artefacts in the DevOps practice with a minimum of dependencies, such as depending on a given IDE or a platform like Windows or Ubuntu is observable. Further, most of the tools lack proper visualization of traceability information. Only some tools are adapting the traceability graph visualization and still, those representations are not using colour codes or interactive usability attributes.

TABLE III. Traceability Management and Visualization Tools

Tool name	**Artefacts**	**Traceability technique**			**Visualization technique**				
		Rule-based	***Hypertext-based***	***Integrative***	***Traceability matrix***	***Cross-reference***	***Tree-map***	***Hierarchical tree***	***Traceability graph***
TraceMaintainer [15]	Requirements, structural UML	X		X				X	
TraceME [48]	All			X				X	X
Rational DOORS [51]	Requirements		X	X		X		X	
Rational RequisitePro [52]	Requirements			X	X		X	X	X
Cradle [53]	Requirements			X		X		X	
ReqView [54]	Requirements			X		X			X

VII. Quality Measures

One major hindrance in the practical application of traceability is the cost and resource consumption. In generic software development organizations are reluctant to spare the resources on traceability. Therefore, validating the traceability aspects and ensuring the quality measures are important to encourage real-world usage. Usability measures consider the user experience and interactivity based on evolving user expectations. The usefulness, ease of use, learnability and likeability are treated as the general concepts of the usability [55][56]. Generally, a larger user base is considered to measure the usability aspects of a traceability tool. Further, the degree of automatization by reducing human effort in trace link generation, time, user interface improvements with the aid of colour codes and help-tips are used as usability metrics [57].

The traceability coverage refers to the set of identified correct links after a traceability link recovery process. The link coverage analysis helps to identify the poorly traced artefacts. Traceability coverage can be defined as:

$$\text{Traceability Coverage} = \frac{|\text{links\_a (targets)}|}{|\text{targets}|} \tag{1}$$

where, *targets* is the set of target artefacts and *links_$_a$ (targets)* represents the set of links traced between the artefact *a* and the artefacts in the set *target* [4].

The statistical methods precision, recall and F-measure are widely used accuracy measures [39]. Precision refers to the number of correct instances among all the retrieved instances (2). Recall or the sensitivity denotes the number of correct ones among a total number of relevant instances (3) [58]. The higher precision saves time in locating and implementing the changes, while higher recall is useful in confirming that all proposed changes will be taken into consideration. Moreover, F-measure (F1 Score) is a measure of accuracy and defined as the weighted harmonic mean of the precision and recall of a test. This conveys the balance between the Precision and the Recall. F-measure assumes values to be in the interval [0,1].

$$\text{Precision} = \frac{|\text{relevant trace links} \cap \text{retrieved trace links}|}{|\text{retrieved trace links}|} \tag{2}$$

$$\text{Recall} = \frac{|\text{relevant trace links} \cap \text{retrieved trace links}|}{|\text{relevant trace links}|} \tag{3}$$

$$\text{F1 Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{4}$$

Ensuring reliability is important in traceability establishment and management. It ensures that a process will perform its intended tasks, without any failures for a given time. The Hidden Markov Chain is one of the algorithms that can be applied to measure software reliability. For instance, Lee et al., have proposed an approach using Markov Chain for measuring reliability in UML by supporting traceability to overcome the limitations in analysis and modelling [59]. Further, traceability matrix-based techniques help to preserve the reliability with respect to requirement artefacts.

Network analysis is originated from graph theory and applied on problems that are represented in a node-link structure as a graph. This technique is actively used for graph analysis in various domains such as food chains in ecosystems to internet traffic in computer systems. Network analysis comprises of several centrality measures that can be used to validate and assess the accuracy of the traceability links in networks [60]. Fig. 8 shows an example of four centrality measure values, degree centrality (D), closeness centrality (C), betweenness centrality (B), eigenvector centrality (E), obtained using the Python NetworkX libraries. The maximum and minimum centrality measures of nodes in a traceability graph is useful to identify the most and least connected artefact, centralized artefact, the artefacts with control over the network and influential artefacts. Further, these four centrality measures have used in validating traceability results [61].

Degree centrality (5) describes the status of a node in accordance with its adjacent links by counting the neighbouring nodes [60][62]. It has two versions as in-degree that counts incoming relationships and out-degree that counts a number of outgoing connections. Thus, a node that has a higher degree centrality value denotes that it is more central and has more power to be visible due to a maximum number of relationships in that network with respect to other nodes. For instance, nodes R and T have the maximum degree centrality in Fig. 8 with a value of 4 since they both have four connections.

$$C_{\text{Degree}}(v) = \deg(v) \qquad \text{where, } v \text{ denotes a node.} \tag{5}$$

Closeness centrality (6) defines the nearest node to most nodes [60][62]. It considers the sum of a node to all the other nodes available in a network. Hence, a maximum closeness centrality value depicts that the distance from that node to the majority of other nodes in a lower value having the ability to send information fast. In Fig. 8, maximum closeness centrality value belongs to two nodes; R and T.

$$C(x) = \frac{N}{\sum_y d(x,y)} \tag{6}$$

where, *d(x,y)* denotes the distance between vertex *x* and vertex *y*. *N* denotes the number of nodes in the graph.

Betweenness centrality (7) denotes the occurrences of a node being a bridge along the shortest path to other nodes [60][62]. Here, it is assumed that information flow is performed over the shortest paths between nodes. Accordingly, a node with a higher betweenness centrality value may have control within the network in terms of data passing as in Fig. 8 the node T has the maximum betweenness centrality value.

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{7}$$

where, *v*, *s*, *t* denote nodes and $\sigma_{st}(v)$ is all the shortest paths between nodes *s* to *t* that pass through node *v*.

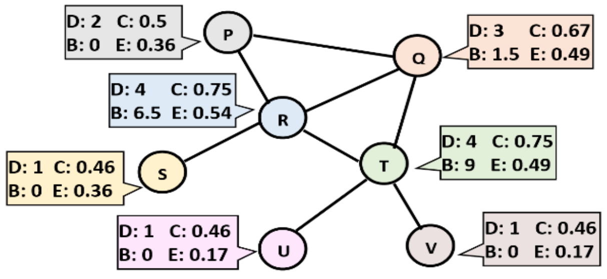

Fig. 8. Example of Network Centrality Measures.

Moreover, EigenVector Centrality (EVC) also named as eigen centrality defines the most influential nodes in terms of the connectivity of a node with the other well-connected nodes in a network [60][62]. EVC is the principal eigenvector of an adjacency matrix defining the network. If a node's EVC is a higher value, it shows that it is a more influential element in that network with respect to remaining nodes [61]. A node is influential, if it affects other highly influential nodes than the lesser nodes [63]. Thus, in EVC, nodes influence the linked nodes without being restricted to the shortest path or the adjacency of node connectivity [62]. For example, node R has the maximum EVC value in Fig. 8. Due to these factors, EVC measure is applied in Google web page ranking and to analyze the traceability establishment accuracy [62][63].

VIII. Discussion

One major challenge in tracing software artefacts is the heterogeneity due to different abstraction levels and the lack of defined data formats of the artefacts. Thus, it is essential to identify the key elements from a given artefact for proper relationship establishment. Although there are a variety of related studies, most of the literature has certain limitations such as being addressing only a few artefacts types, not addressing aspects related to software development in DevOps practice, lack of support towards CICD and lack automation [40][45]. Considering the tool support, many tools have traceability visualization and scalability issues leading to the inability to manage traceability with many artefacts.

Moreover, the need for techniques and tools to recover traceability links in legacy systems is particularly important for a variety of software evolution tasks such as transitioning from traditional software process models to DevOps. The tasks include general maintenance tasks, impact analysis, program comprehension and more encompassing tasks such as systematic reuse traceability types and reverse engineering for redevelopment [4]. Some existing studies have addressed these aspects separately such as change impact analysis rather than along with traceability [64][65]. Hence, overall there is a lack of traceability management support to cope with the continuous integrations in DevOps practice.

Traceability support in a DevOps based software development environment can be achieved by addressing the identified limitations. Several future possible research directions can be suggested based on this survey. The efficiency and effectiveness of the software artefact extraction process can be improved by exploring data pre-processing and information retrieval techniques that support heterogeneous artefact types, which can result in a more accurate traceability establishment process. Moreover, a generalized framework can be modelled to manage traceability, so that the automation can be achieved irrespective of the project domain and scale. Consequently, it is important to have a traceability representation with a scalable visualization technique that can lead to better decision making with minimum visual clutter. Additionally, traceability result validation is important to avoid inconsistencies among artefacts and reduce the associated cost due to re-work. Therefore, it is essential to determine a traceability methodology to synchronize software artefacts, such that the changes made to an artefact in any phase of the SDLC can preserve the consistency across all the artefacts. Furthermore, the traceability management process can be refined with automation and cutting-edge CI features including impact analysis and change propagation addressing the frequent artefact changes with a minimum cost.

IX. Conclusion

Software artefact traceability in DevOps practice is an evolving research area with the need for software evolution and maintenance. Software development in DevOps practice considers frequent software change requests and facilitates for continuous integration and continuous delivery process. This study addressed the applications of artefact traceability in a DevOps environment, which is challenging than managing traceability in a traditional software development process. We have identified the traceability concepts, terminologies, traceability models in DevOps practice, traceability establishment techniques, visualization approaches and traceability evaluation methods. Moreover, this study explored the related work on artefact traceability management and the existing traceability tool support. We have identified that there is a lack of traceability management features, which can be practically applicable in DevOps software development environments. This study has identified the requirement of having a generalized and automated traceability management solution that can handle any type of frequent artefact change in a scalable manner with a proper visualization approach.

Acknowledgment

The authors acknowledge the support received from the University of Moratuwa, Sri Lanka in publishing this paper.

References

[1] V. Rajlich, "Software evolution and maintenance," in Future of Software Engineering (FOSE), 2014, pp. 133–144.

[2] I. Sommerville, Software Engineering, 10th ed. New York: Addison-Wesley Professional, 2010.

[3] R. Arora and N. Arora, "Analysis of SDLC Models," Int. J. Curr. Eng. Technol., vol. 6, no. 1, pp. 268–272, 2016.

[4] J. Cleland-Huang, A. Zisman, and O. Gotel, Software and Systems Traceability, 1st ed. London: Springer-Verlag London, 2012.

[5] D. A. Meedeniya I. D. Rubasinghe I. Perera, "Software Artefacts Consistency Management Towards Continuous Integration: A Roadmap", International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 10, No. 4, April 2019.

[6] L. J. Bass, I. M. Weber, and L. Zhu, DevOps : A Software Architect's Perspective, 1st ed. Addison-Wesley Professional, 2015.

[7] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," J. Softw. Evol. Process, vol. 29, no. 6, p. e1885, 2017.

[8] M. Meyer, "Continuous integration and its tools," IEEE Software, vol. 31, no. 3, pp. 14–16, May 2014.

[9] G. B. Ghantous and A. Gill, "DevOps: Concepts, Practices, Tools, Benefits and Challenges," in 21st Pacific Asia Conference on Information Systems, 2017, pp. 1–13.

[10] I. Pete, D. Balasubramaniam, "Handling the Differential Evolution of Software Artefacts A Framework for Consistency Management", in 22nd IEEE International Conference on Software Analysis Evolution and Reengineering (SANER 2015), pp. 599-600, 2015.

[11] M. Osborne and C. K. MacNish, Requirements Engineering, 2nd International Conference on (ICRE '96). IEEE Computer Society, 1996.

[12] E. Keenan et al., "TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate

traceability solutions," in 34th International Conference on Software Engineering (ICSE), 2012, pp. 1375–1378.

[13] K. Mohan, P. Xu, L. Cao, and B. Ramesh, "Improving change management in software development: Integrating traceability and software configuration management," Decis. Support Syst., vol. 45, no. 4, pp. 922–936, Nov. 2008.

[14] J. S. Her, H. Yuan, and S. D. Kim, "Traceability-centric model-driven object-oriented engineering," Inf. Softw. Technol., vol. 52, no. 8, pp. 845–870, Aug. 2010.

[15] P. Mäder, O. Gotel, T. Kuschke, and I. Philippow, "traceMaintainer - Automated Traceability Maintenance," in 16th IEEE International Requirements Engineering Conference, 2008, pp. 329–330.

[16] G. Kim, P. Debois, J. Willis, J. Humble, and J. Allspaw, The DevOps Handbook, 1st ed. IT Revolution Press, 2016.

[17] P. M. Duvall, S. Matyas, and A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, 1st ed. Addison-Wesley Professional, 2007.

[18] G. Pinto, F. Castor, R. Bonifacio, and M. Rebouças, "Work practices and challenges in continuous integration: A survey with Travis CI users," Softw. Pract. Exp., vol. 48, no. 12, pp. 2223–2236, Dec. 2018.

[19] D. Ståhl, K. Hallén, and J. Bosch, "Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework" Empir Softw Eng,vol.22,no. 3, pp. 967–995, 2017.

[20] A. Tang, Y. Jin, and J. Han, "A rationale-based architecture model for design traceability and reasoning," J. Syst. Softw., vol. 80, no. 6, pp. 918–934, Jun. 2007.

[21] A. Arunthavanathan et al., "Support for Traceability Management of Software Artefacts using Natural Language Processing," in Moratuwa Engineering Research Conference (MERCon), 2016, pp. 18–23.

[22] J. H. Hayes, A. Dekhtyar, S. K. Sundaram, E. A. Holbrook, S. Vadlamudi, and A. April, "REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery," Innov. Syst. Softw. Eng., vol. 3, no. 3, pp. 193–202, Sep. 2007.

[23] P. Mäder and O. Gotel, "Towards automated traceability maintenance," J. Syst. Softw., vol. 85, no. 10, pp. 2205–2227, Oct. 2012.

[24] J. Cleland-Huang, C. K. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," IEEE Trans. Softw. Eng., vol. 29, no. 9, pp. 796–810, 2003.

[25] H. Schwarz, J. Ebert, and A. Winter, "Graph-based traceability: A comprehensive approach," Softw. Syst. Model., vol. 9, no. 4, pp. 473–492, Sep. 2010.

[26] S. Maro, A. Anjorin, R. Wohlrab, and J.-P. Steghöfer, "Traceability maintenance: factors and guidelines," in 31st IEEE/ACM International Conference on Automated Software Engineering, 2016, pp. 414–425.

[27] M. A. Javed and U. Zdun, "A systematic literature review of traceability approaches between software architecture and source code," in 18th International Conference on Evaluation and Assessment in Software Engineering (EASE), 2014, pp. 1–10.

[28] I. Omoronyia, G. Sindre, and T. Stålhane, "Exploring a Bayesian and linear approach to requirements traceability," Inf. Softw. Technol., vol. 53, no. 8, pp. 851–871, Aug. 2011.

[29] G. Spanoudakis, A. Zisman, E. Pérez-Miñana, and P. Krause, "Rule-based generation of requirements traceability relations," Journal of Systems and Software, vol. 72, no. 2, pp. 105–127, Jul. 2004.

[30] P. Lago, H. Muccini, and H. van Vliet, "A scoped approach to traceability management," Journal of Systems and Software, vol. 82, no. 1, pp. 168–182, 2009.

[31] X. Chen, J. Hosking, and J. Grundy, "Visualizing traceability links between source code and documentation," in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2012, pp. 119–126.

[32] W. B. Santos, E. S. De Almeida, and S. R. Silvio, "TIRT: A traceability information retrieval tool for software product lines projects," in 38th EUROMICRO Conf Softw. Eng. Adv. Appl. (SEAA), pp. 93–100, 2012.

[33] D. Holten, "Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data," IEEE Trans. Vis. Comput. Graph., vol. 12, no. 5, pp. 741–748, Sep. 2006.

[34] S. Kugele and D. Antkowiak, "Visualization of Trace Links and Change Impact Analysis," in IEEE 24th International Requirements Engineering Conference Workshops (REW), 2016, pp. 165–169.

[35] T. Merten, D. Jüppner, and A. Delater, "Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualizations," in 4th International Workshop on Managing Requirements Knowledge (MaRK), 2011, pp. 17–21.

[36] I. Santiago, J. M. Vara, V. De Castro, and E. Marcos, "Visualizing Traceability Information with iTrace," in 9th International Conference on Evaluation of Novel Approaches to Software Engineering, 2014, pp. 5–15.

[37] I. D. Rubasinghe, D. A. Meedeniya, and I. Perera, "Software Artefact Traceability Analyser : A Case-Study on POS System," in 6th International Conference on Communications and Broadband Networking (ICCBN 2018), 2018, pp. 1–5.

[38] A. Rodrigues, M. Lencastre, and G. A. de A. C. Filho, "Multi-VisioTrace: Traceability Visualization Tool," in 10th International Conference on the Quality of Information and Communications Technology (QUATIC), 2016, pp. 61–66.

[39] A. Firdaus, I. Ghani, D. N. Abg Jawawi, and W. M. N. Wan Kadir, "Non functional requirements (NFRs) traceability metamodel for agile development," J. Teknol., vol. 77, no. 9, pp. 115–125, 2015.

[40] F. Furtado and A. Zisman, "Trace++: A Traceability Approach to Support Transitioning to Agile Software Engineering," in IEEE 24th International Requirements Engineering Conference (RE), 2016, pp. 66–75.

[41] R. Elamin and R. Osman, "Towards Requirements Reuse by Implementing Traceability in Agile Development," in IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), 2017, vol. 2, pp. 431–436.

[42] A. E. Limón and J. G. Sopeña, "Tackling Traceability Challenges through Modeling Principles in Methodologies Underpinned by Metamodels.," Proc. 3rd IFIP TC2 Cent. East Eur. Conf. Softw. Eng. Tech. CEE-SET 2008, no. May 2014, 2008.

[43] J. Cito, F. Oliveira, P. Leitner, P. Nagpurkar, and H. C. Gall, "Context-based analytics - establishing explicit links between runtime traces and source code," in 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP), 2017, pp. 193–202.

[44] G. Laghari, A. Murgia, and S. Demeyer, "Localising Faults in Test Execution Traces," in 14th International Workshop on Principles of Software Evolution (IWPSE), 2015, pp. 1–8.

[45] S. Vöst and S. Wagner, "Trace-based test selection to support continuous integration in the automotive industry," Proc. Int. Work. Contin. Softw. Evol. Deliv. (CSED), pp. 34–40, 2016.

[46] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Trans. Softw. Eng. Methodol., vol. 16, no. 4, pp. 13:1-13:50, 2007.

[47] A. De Lucia, F. Fasano, R. Oliveto, and G. Tortora, "Fine-grained management of software artefacts: the ADAMS system," Softw. Pract. Exp., vol. 40, no. 11, pp. 1007–1034, Oct. 2010.

[48] K. Kamalabalan et al., "Tool support for traceability of software artefacts," in Moratuwa Engineering Research Conference (MERCon), 2015, pp. 318–323.

[49] I. D. Rubasinghe, D. A. Meedeniya, G. I. U. S. Perera, "Automated Inter-artefact Traceability Establishment for DevOps Practice", in 17th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2018), 2018, pp. 211-216.

[50] I. Rubasinghe, D. Meedeniya, I. Perera, "Traceability Management with Impact Analysis in DevOps based Software Development", in 7th international conference on advances in computing, communications and informatics (ICACCI), 2018, pp. 1956-1962.

[51] IBM, "Ditch the documents and spreadsheets - manage requirements efficiently and more accurately with IBM Rational DOORS Next Generation - IBM Software White Paper," 2014.

[52] "P. Zielczynski, Requirements Management Using Ibm® Rational® Requisitepro®, First ed., IBM Press, 2007.

[53] Y. Klochkov, A. Gazizulina, M. Ostapenko, E. Eskina and N. Vlasova, "Classifiers of nonconformities in norms and requirements," in 5th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO), 2016, pp. 96-99.

[54] “ReqView,” 2017. [Online]. Available: https://www.reqview.com/. [Accessed: 07-May-2018].

[55] A. I. Martins, A. Queirós, A. G. Silva, and N. P. Rocha, “Usability Evaluation Methods,” in Human-Computer Interaction: Concepts, Methodologies, Tools, and Applications, S. Daeed et al., Eds. IGI Global, 2014, pp. 613–636.

[56] S. Winkler, “On Usability in Requirements Trace Visualizations,” in Requirements Engineering Visualization, 2008, pp. 56–60.

[57] A. Sünnetcioglu, E. Brandenburg, U. Rothenburg, and R. Stark, “ModelTracer: User-friendly Traceability for the Development of Mechatronic Products,” Procedia Technol., vol. 26, pp. 365–373, 2016.

[58] T. Zeugmann et al., “Precision and Recall,” in Encyclopedia of Machine Learning, Boston, MA: Springer US, 2011, pp. 781–781.

[59] J. Lee, B. Cho, H. Youn, and E. Lee, “Reliability analysis method for supporting traceability using UML,” in Communications in Computer and Information Science, Springer Berlin Heidelberg, 2009, pp. 94–101.

[60] J. Scott, Social Network Analysis, Third. SAGE Publications, London, 2013.

[61] G. Bavota, L. Colangelo, A. De Lucia, S. Fusco, R. Oliveto, and A. Panichella, “TraceME: Traceability Management in Eclipse,” in 28th IEEE International Conference on Software Maintenance (ICSM), 2012, pp. 642–645.

[62] S. P. Borgatti, “Centrality and network flow,” Soc. Networks, vol. 27, no. 1, pp. 55–71, Jan. 2005.

[63] I. Perera, A. Miller, and C. Allison, “A Case Study in User Support for Managing OpenSim Based Multi User Learning Environments,” IEEE Trans. Learn. Technol., vol. 10, no. 3, pp. 342–354, Jul. 2017.

[64] S. Park and D. H. Bae, “An approach to analyzing the software process change impact using process slicing and simulation,” J. Syst. Softw., vol. 84, no. 4, pp. 528–543, 2011.

[65] J. Dick, E. Hull, and K. Jackson, “Advanced Traceability,” in Requirements Engineering, Springer International Publishing, 2017, pp. 239–254.