Software Artefacts Consistency Management towards Continuous Integration: A Roadmap

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

***Abstract*—Software development in DevOps practices has become popular with the collaborative intersection between development and operations teams. The notion of DevOps practices drives the software artefacts changes towards continuous integration and continuous delivery pipeline. Subsequently, traceability management is essential to handle frequent changes with rapid software evolution. This study explores the process and approaches to manage traceability ensuring the artefact consistency towards CICD in DevOps practice. We address the key notions in traceability management process including artefact change detection, change impact analysis, consistency management, change propagation and visualization. Consequently, we assess the applicability of existing change impact analysis models in DevOps practice. This study identifies the conceptualization of the traceability management process, explores the state-of-art solutions and suggests possible research directions. This study shows that the lack of support in heterogeneous artefact consistency management with well-defined techniques. Most of the related models are limited with the industry-level applicability in DevOps practice. Accordingly, there is inadequate tool support to manage traceability between heterogeneous artefacts. This study identifies the challenges in managing software artefact consistency and suggests possible research directions that can be applied to manage the traceability in the process of software development in DevOps practice.**

Keywords*—*Consistency management; traceability; continuous integration; DevOps; comparative study

I. INTRODUCTION

Software systems perform in a dynamic context, where changes arise due to different factors such as a change in business goals, performance improvements, fault corrections and change of technology. DevOps (Development-Operations) practice is an emerging software development approach that encourages collaborative nature over traditional software development [1]. DevOps practice addresses the frequent artefact changes during the Software Development Life Cycle (SDLC) enabling a Continuous Integrations and Continuous Delivery (CICD) pipeline. Consequently, maintaining software evolution is essential, although it is challenging to manage the artefacts change consistency management [2]. For instance, a software artefact barely exists in isolation as it is associated with other artefacts in the process. Thus, a change in a requirement may have a considerable effect on the entire system. Hence, artefact consistency management plays a major role in achieving software traceability [3].

A software system consists of both homogeneous and heterogeneous artefacts in different formats, making consistency management a complex task. For example, the requirement artefact can be in a natural language, while the source code artefact in Java programming language. Therefore, a change occurred in one artefact does not directly reflect in other artefacts due to the type and format mismatches. The artefact traceability is significant in consistency management [4]. Generally, an artefact change appears as a request for a change without direct action of alteration. Thus, a semi-automated or manual process of maintaining the artefact traceability may subject to errors.

In traceability established system, it is essential to detect artefact changes and identify their impact. The Change Impact Analysis (CIA) process identifies the affected artefacts by following the trace paths using different techniques such as traceability graphs [5], Information Retrieval (IR) and Machine Learning (ML) [6][7]. In graph-based traceability, the artefact changes are mapped to the nodes and the change are propagated via the connected links. However, all the endpoints of the links may not be subjected to changes. Therefore, the impact of the propagated changes at the linked endpoints needs to be measured to identify the actual impacted set of a change. Thus, a CIA approach should be selected based on the artefact types that are used to measure the impact [8][9].

Additionally, the properties of the initial change may highly affect the impact calculation, since the linked endpoints are compared with the initial change. Several studies have used IR techniques to identify the properties of the initial change such as the scope and keywords [10][11]. Moreover, CIA techniques based on probabilistic methods such as association rules, Bayes theorem and Change History have used to measure the impact of a change [12][13][14]. However, it is challenging to address the change ripple effects after the initial impacted endpoint identification, as the changes can propagate continuously.

The main components of the traceability management process include trace-link creation, change detection, CIA, consistency management, change propagation and collaboration. This survey paper addresses the traceability management process in DevOps practice. Section II states an overview of DevOps and the traceability management process. Section III discusses the artefact change detection methods, which is the first step in the CICD process. Section IV elaborates CIA with its terminology, approaches and related studies. Section V describes the existing change propagation and consistency management techniques. Related work on the continuous integration in DevOps practice is presented in Section VI. Section VII states the limitations and challenges of achieving traceability in DevOps practice and the possible future research directions. Section VIII concludes the paper.

II. Background

A. Concepts of DevOps

DevOps practice broadens the view of software engineering paradigm by improving the collaboration across teams, sharing resources, tools and increasing the project performance. This is based on a maturity model that integrates the development and operations teams [15]. There are stages of a DevOps cycle with respect to SDLC phases that include continuous planning, integration, testing, delivery, deployment and monitoring [1][15][16]. As shown in Fig. 1, in DevOps practice the developers implement the project by enforcing the deployment process and the operations team monitors the progress and faults during the deployment, adhering to CICD process [1]. Mainly the development team is responsible to plan, code, build, test and the operations team is responsible for the project release, deploy, operate and monitor.

The coordination of human resources in a DevOps environment is important to maintain the manageability of collaborative nature. DevOps engineer, that can be an individual or a team, plays a major role by managing the tool support including automation, version control, configuration, maintenance [1]. Consequently, the level of automation in the CICD pipeline is controlled by the DevOps engineer. In this study, we have mainly explored the Continuous Integration (CI) aspects such as change detection, change impact analysis, change propagation and consistency management along with the identification of DevOps tool support and challenges.

B. Traceability Management Process

In a software product, it is essential to maintain artefact consistency whenever a change occurs. Software traceability is required to handle changes during the process of CICD, that integrates the work frequently between the development and operations teams leading to multiple integrations per day [2]. Software traceability follows the life cycle of an artefact both forward and backward and overcomes the inconsistencies during the software development [17]. Thus, each alteration occurs in an artefact is traced among other artefacts and change accordingly based on the impact. The relationship links among artefacts must be updated and maintained consistently. Fig. 2 shows a traceability process model within a collaborative environment. The artefacts with changes in various levels are usually included in each CI task. Thus, the initially established traceability links in the project must be incorporated according to the changes included in each integration. This process consists of change detection, CIA, consistency management and change propagation. Visualization of the traceability links is used to understand the dependencies between the artefacts. It is challenging and costly to manage the consistency of a larger set of artefact relationships whenever a change occurs. Also, the effort of maintaining artefact relations is considerably high though the number of artefacts is minimal [18]. Thus, the accuracy of traceability establishment is important.

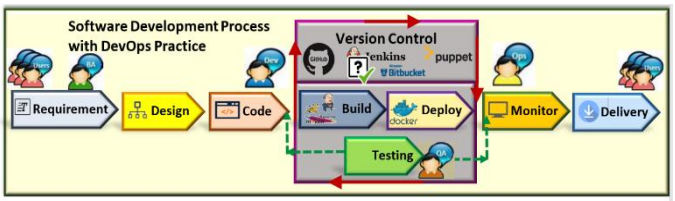

Fig. 1. Software Development Process with DevOps practice.

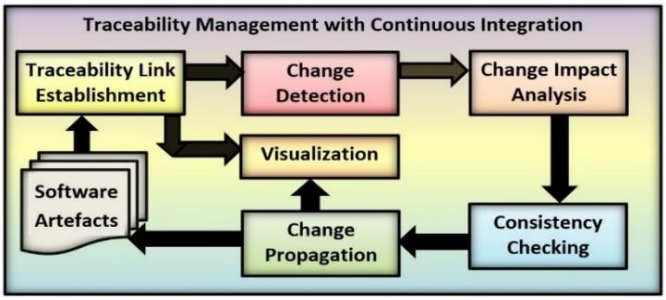

Fig. 2. Traceability Management Process Model.

III. Change Detection

A. Types of Artefact Changes

Change is inevitable in any software development process and should be handled properly to improve product quality while avoiding unnecessary cost [19]. Artefacts in each phase of the SDLC are subjected to change at different frequencies due to change of requirement, technology, managerial decision and fault tolerance. A change in a phase of the SDLC can evolve through the phases based on their dependencies.

Mainly three types of changes can occur during software development. (1) edit changes, that alter the existing elements or sub-elements of the artefacts. For example, changing the name of an attribute in a design class diagram or renaming a source code method name is considered as an edit change; (2) delete changes, that removes one or more existing artefact elements or sub-elements from a traceability model such as removing attributes from a source code class or deleting a class in a design class diagram is considered as a deletion change; (3) add changes, that include new artefact elements or sub-elements to the existing project. The change detection process identifies whether a change has occurred and detect the details of change such as change type and artefact type.

B. Change Detection Techniques

Among many techniques, edit history approach keeps track of the alterations or the edits as a history [20]. Here, each change is considered as an item for the history and records as another edit. This is already in use with most of the software and non-software related tools and methodologies such as text editors, mainly the ‘Undo”, “Redo’ and ‘Restore’ operators [20][21]. However, this technique is mostly used for the change detection of the source code artefact.

Tree differencing is another main change detection technique that represents elements as Abstract Syntax Tree (AST) and calculates the differences to extract detailed change information [22]. AST is a tree representation method of the syntactic structure in the source code, where every node denotes a construct occurrence in the code. This is a well-defined method, for instance, the study done in [19], has transformed a tree to another in hierarchically structured data based on the idea of matching and minimum cost edit scripts. Here, the authors have separated the change detection problem as 'good matching' and 'minimum conforming edit script'. However, it is required the data to be in a tree format. The approach of calculating differences between ASTs is widely used for source code artefacts [22][23].

The customized differencing algorithm is another technique for software artefact change detection. It is used in software maintenance aspects such as program-profile estimation (stale profile propagation). A software artefact type can be taken as the input for a differencing algorithm [22]. However, the implementation of a general algorithm for all types of software artefacts is identified to be impractical rather than having a set of differencing algorithms for each type of software artefacts.

IV. Change Impact Analysis

A. Terminology of Change Impact Analysis

Each change in a single artefact may affect one or more other related artefacts in different degrees. The goal of the CIA is to detect the consequences of an artefact alteration to other artefacts in the software system [12]. Traceability is a key notion to identify the affected artefacts and decide whether evolution is sustainable [24]. Generally, the impact is analyzed before or after a change implementation. The prior analysis of the impact results in better program understandability, change impact prediction and cost estimations. Correspondingly, conducting impact analysis after implementation of a change can be beneficial in tracing ripple effects, selecting test cases and in performing change propagation [1]. Fig. 3 shows the iterative CIA process in software development [25].

Initially, the process analyses a change request to determine the set of changes in which the artefacts can be affected. It is referred to as feature location, with the aim of finding a place that requires the initial change. Then, the CIA is performed to estimate the effects in changes resulting in an Estimated Impact Set (EIS). Afterwards, the change is implemented and the elements in the Actual Impact Set (AIS) are altered. The AIS is not considered to be unique for a given change request as a change can be implemented in different ways.

Fig. 4 shows the types of change impact sets. The Starting Impact Set (SIS) denotes the set of entities initially affected by the change. Then, a subset of it called Candidate or Estimated Impact Set (CIS or EIS) is a subset of SIS, includes the identified potential impact entities that are traced from SIS. The AIS is required to be identified from the EIS. Discovered Impact Set (DIS) are the artefacts that are impacted by the change but have not identified by CIS due to the challenging effect of artefact type mismatches, development mistakes and inconsistencies in artefact naming. These hidden dependencies of the artefacts can be identified manually or using a knowledge-based technique [26][27]. The False Positive Impact Set (FPIS) denotes the artefacts that are overestimated as belong to the CIS, but which are not actually impacted yet.

B. Change Impact Analysis Techniques

One category of change impact analysis approach is a traceability-based and dependence-based technique to identify the effect of a change [25]. Traceability-based CIA is narrowed in recovering the traceability links among software artefacts. Dependence-based CIA evaluates the impact of a proposed change. This technique is biased towards in analyzing program syntax relations and performing CIA of artefacts in the same level of abstraction such as in the level of software design or within the level of source code. The higher-level Unified Modeling Language (UML) models and use case maps are mainly involved in requirement and design level impact analysis. In addition, the source code-based CIA techniques are more capable to determine change impacts of the final software product with improved precision as they analyze the implementation details directly. Table I explores different CIA techniques with their advantages and limitations.

Another categorization of CIA technique is static and dynamic impact analysis [9][28][29]. Static CIA techniques consider all the possible behaviours and inputs. It analyzes the syntax and semantic dependencies of source code and constructs intermediate representations using call graphs and program dependence graphs [9]. Then, the CIA is conducted based on the representations resulting in large impact sets that are difficult to use in practice. Thus, lower precision is a major drawback in static CIA techniques. Besides, dynamic CIA techniques overcome this drawback by considering only a part of the inputs. Hence, their impact sets are identified to be highly precise though lower in safety. Furthermore, dynamic CIA depends on the analysis of the data obtained during the execution such as trace information, relation and coverage information to assess the impact sets.

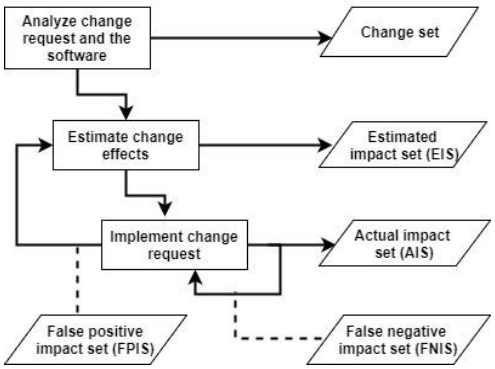

Fig. 3. Change Impact Analysis Process [25].

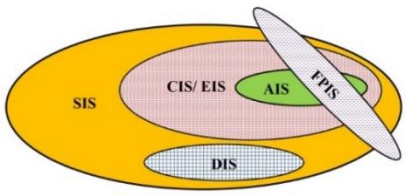

Fig. 4. Change Impact Analysis Categorization.

TABLE I. COMPARISON OF CHANGE IMPACT ANALYSIS TECHNIQUES

Category	Technique	Description
Statistical analysis	Data flow analysis, relational language, program slicing, static call graphs [30]	Has identified that the string analysis is not precise for schema CIA. There is a precision versus computational cost trade-off in this analysis.
	Comparative analysis: Study on impact analysis algorithms, techniques using Precision, Recall and Harmonic mean [11][26][31][32][33][34].	Results certify that existing algorithms require enhancements and effective mechanisms to facilitate automated tools for the CIA. Have identified required characteristics in impact analysis. Discovered the possibility of transferring impact analysis tools in academia to industry to help developers during maintenance and evolution activities.
Probabilistic-based	Change history and Bayes' theorem [26]	Maintenance of object-oriented critical systems is addressed. Limited for object-oriented software.
	Call graphs, Entity Dependency Graph [35][30][36][37][38][39][38]	Explain the concept of two dependency states; namely, persistent relationship state and immediate relationship state in change propagation. Better program understanding and debugging.
	Formal Semantics [40] Logical dependencies and classification [39][41]	Removal of false positive impacts and consistency checking. Adds valuable information. Restricted for particular change and relation types.
	Rule-based [36][41][42]	Allows developers to smoothly retrace the changes.
	Data mining, Apriori algorithm [6]	Useful in change predictions.
History-based	Historical co-change analysis, change history [6][43][44]	Use version histories to identify logical/ evolutionary couplings between entities. Predict impact files after a change.
	Machine learning [7][10][26]	Classification models to predict the validity of the candidate links. Use unsupervised learning to identify hidden dependencies. Less human involvement
	Logical coupling [27]	Use logical coupling with a Markov model. Better accuracy.

Acharya and Robinson [28] have presented a static CIA framework that is developed as a tool named Imp, to analyse the impact of source code artefacts during the frequent builds.

This mathematical model is used forward slicing consists of three criteria such as range, dependences and summary edges to assess the impact sets. Additionally, this work has used Andersen's algorithm with pointer analysis. This methodology consists of two variations one for high setting impact analysis which is expensive and another for low setting impact analysis which can be performed frequently and faster at a low cost.

Another static impact analysis technique is addressed in [29], that have created clusters of associated source codes based on their co-modification history. Here, dimensionality reduction approaches have used to reduce complexity and perform the impact analysis efficiently. Initially, they have mined the changed repository to find co-occurring source files and developed a matrix containing a degree of closeness in each pair of files. Then, an intrinsic dimensionality method based on eigenvalues has conducted to estimate the lower dimensional representation of the matrix and Principle Component Analysis (PCA) was used to reduce the matrix. Finally, the matrix rows were taken as coordinates of the files; the distance between each pair of files was measured and passed to five clustering methods. However, quantitative measures have not used to evaluate the model impact.

In dynamic CIA approach, the main techniques are path impact and coverage impact [30][35]. Path impact performs at the method level with the use of compressed execution traces to determine impact sets. It processes forward and backward traces to identify the impact of the changes. The forward traces determine all the methods called after the altered methods, while the backward traces find methods into which the execution can return. The coverage impact technique uses the coverage information to identify the executions that traverse at least one method in the change set and it marks the covered methods in each execution. Next, it assesses a static forward trace from each change from the marked methods. Thus, the methods in computed traces become the impact set. Moreover, it is analytically identified that the path impact technique is more precise compared to the coverage impact technique as it makes use of traces instead of the coverage [35]. However, the time and space overhead of the path impact technique is high. The required time in path impact tends to depend on the size of the analyzed trace, though the coverage impact needs a constant time in updating bit vectors at each of the method entries. Besides, the space complexity of coverage impact technique is linear over the size of the program, while it is also proportional to the size of the traces in the path impact technique. The use of dependency network measures such as centrality measures havw also used for the CIA. The work by Nguyen et al. [45] has shown the applicability of dependency network analysis measures in practical applications.

Mainly the CIA techniques can be categorized into a graph, formal, historical and scope-based. The comparative and classification-based approaches are used in most related work while some have developed a specific tool. Among the used CIA techniques, call graphs and dependence graphs are widely used to handle the changes that enable the backtracking ability to debug easily. Most of the artefact types including design, code and test cases are influenced by these call graphs related techniques and IR based: Latent Semantic Indexing (LSI), Frequency-Inverse Document Frequency (TF-IDF) and Vector Space Model (VSM) techniques. In contrast, the work in [46] has shown the drawbacks of dependence graphs and proposed a

modular-based approach for large-scale product lines. The formal semantics, First Order Logic (FOL) and review-based analysis have addressed the requirement artefact type solely. While most of these techniques are semi-automatic, the use of data mining and ML in impact analysis has become a newer trend that tries to completely avoid the human effort with full automation. However, in terms of the scope-based CIA related work, there is a lack of support for later phases of SDLC in particular testing and maintenance phases as the majority are restricted for requirements artefact or source code [17][47][48]. Thus, that limits the applicability of these CIA approaches and related works into a DevOps environment.

C. Change Impact Analysis Models

A change type classification method for code revision has presented in [21]. The authors have proposed a taxonomy for change types and defined code changes in the form of tree edit operations on the AST of a program. The changes are classified based on the significance level such as low, where local changes to have a minor significance, medium, high or crucial, where interface changes are critical. This work has extended with a tool called ChangeDistiller in [49]. This model assigns a label to every code entity in the AST that denotes the type of each entity and adds a textual named value, covering the actual code. Additionally, they have extracted change couplings from their release history database which have an identical total significance level. These changes have weighted based on the number of transactions they occurred in. Accordingly, the remaining change couplings were the most likely candidates for being affected by changes to coupled entities. This model is shown in Fig. 5, and implemented as the ChangeDistiller [49] plug-in for Eclipse. However it is limited for small projects.

Another model for the CIA for Java programs has presented in [38]. Fig. 6 shows the architecture of a Java source code impact analysis tool called 'Chianti', which is implemented as a plugin in Eclipse IDE.

There are three main submodules in this tool. The initial one is to derive atomic changes from two Java source code versions which are done via pairwise AST comparisons. One other module is responsible for reading test call graphs to detect the original and edited codes. Also, it assesses impacted tests and impacting changes. The other module visualizes the change impact details to the user. Accordingly, this plugin model is mainly based on the call graphs and does not involve calculations for each impact in a quantitative value.

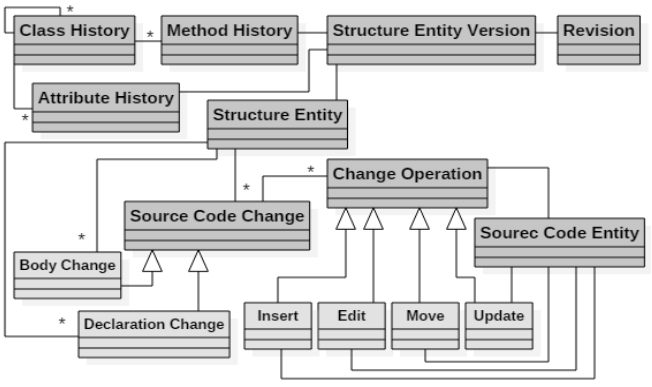

Fig. 5. ChangeDistiller Model [49].

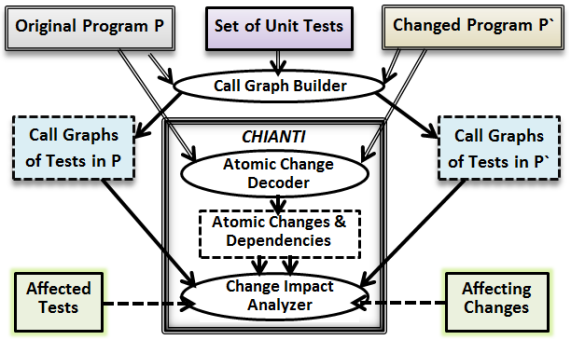

Fig. 6. Chianti Tool Architecture [38].

Wong and Cai [50], have proposed a model to extract the logical models from UML class diagram. This is based on a logical framework named Augmented Constraint Network (ACN). Fig. 7 depicts the structure of an ACN translated class. The classes are weighted based on their number of augmented constraint networks to determine impacted elements. A higher rank is assigned, if there are more sub augmented constraint networks. The distance between the two classes also affects the weight and closely related classes get a higher ranking. The weights are then multiplied with the co-change frequency of entities that are mined from version histories. Accordingly, ten of the highest ranked elements are output to the user.

One of the earliest CIA models [8], has calculated the transitive closure to identify the impacted elements using conceptual models. The concept of intra-method and inter-method data dependency graphs are used to calculate impacts on entities in method bodies and change dependencies between methods, respectively. Object-oriented dependency graphs are used to compute the change impact at the system level. Additionally, they have declared four types of impacts between every two related entities including contaminated (both are impacted), clean (both are not impacted), semi-contaminated (the source does not impact the target though the source can be impacted) and semi-clean (the source is not impacted, but it propagates changes to the target). These types are used to assign weights to relationships among entities, based on their impact relation type. Then the total change impact weight is assessed as the sum of all weights, which are assigned to the relations between two entities. Finally, the total change impact weight is assigned to all graphs to enable impact calculation.

A CIA approach for architectural models using OCL4 to express explicit rules has proposed in [51]. They have searched the impacted elements and a distance measure is used to control the propagation of changes to the indirectly related software entities. It either cancels the change propagation or weights the impact paths based on their nesting depth. Also, they have presented a taxonomy of change types that provide three elemental change types such as add, remove and change. The impact analysis model presented in [52], has extracted changes from a repository and compiled into a matrix. The authors have computed association clusters using singular value decomposition. Every file in a cluster was assigned a weight based on its degree of participation in that cluster. High singular values have denoted that a file was subjected to be impacted by changes to other files of the same cluster.

```
Constraint Network:
    A_interface : {orig, other}
    A_impl : {orig, other}
    A_impl = orig => A_interface = orig
Dominance Relation:
    (A_impl, A_interface)
```

Fig. 7. ACN Translation of a Class [50].

Hattori et al. [53], have proposed another approach to estimate the possible impact elements using a dependency graph that represents the source code and a reachability analysis over the graph. They have extracted the change sets and analysed the impacted entities using a probabilistic algorithm based on Bayes' theorem that removes the false-positives. Apriori and Disjunctive association rule algorithms were used to weight and sort the impacts based on their likelihood. Another change impact analysis model is presented by Arnold and Bohner in the area of traceability [54]. They have considered the changes occurred in documentation and source codes to identify the SIS. Then, dependency graphs were used to obtain CIS based on direct impacts. Moreover, the reachability graph visualizations were used to identify the indirect impacts. They have applied this process incrementally to identify the CIS to minimize the false positive rate. A source code artefact CIA approach based on the change requests provided in natural language document such as bug reports is presented in [55]. They have IR technique to estimate the impact set; LSI to analyze textual change requests, dynamic analysis to evaluate the execution information and data mining to examine the evolutionary information. This work has shown that the accuracy improves regarding the metrics; precision, recall and f-measure by combining these multiple approaches.

Although there are several CIA studies, the majority has been limited only up to design level or source code artefact in considering the artefact types while operational level artefacts like build scripts are not addressed.

V. Consistency Management

During SDLC, various artefacts process through different stages and it is essential to maintain the consistency between all the artefacts, whenever an artefact change occurs. After predicting the artefact change effects during the CIA, consistency management and change propagation are performed to trace the ripple effects [25]. Thus, maintaining consistency plays a major role in managing artefact consistency during the CICD process in a DevOps environment.

A. Change Propagation Techniques

Heuristic rules are one of the techniques that can be used to aggregate the detected changes to propagate the changes. This enables to obtain the optimal solution of the change propagation path with higher performance irrespective of the completeness of the information [56]. Source code artefact change propagation is addressed in [20] using an aggregation algorithm with heuristic rules.

The distance-based technique is mainly based on temporal and spatial distance. For the change propagation, this method considers the time taken between changes and the location distance among two modifications. These distances are measured using AST and the graph-based representations associated with graph traversal algorithms [20][51].

B. Consistency Management Approaches

Artefact changes or refinements happen at any time. Their consequences may not result in a uniform pattern, where some refinements may reflect and impact on other artefacts. Thus, the stability among artefacts can be inconsistent and can fail in representing the expected software solution with stakeholder dissatisfaction. Therefore, consistency management, the ability to preserve the synchronization among software artefacts along with the occurring changes, is essential to minimize efforts in software system maintenance [57]. Hence, an artefact alteration or the presence of outdated artefacts should consistently reflect on the other affected artefacts.

Several studies have addressed consistency management and change propagation with CIA [36][40][42]. A predictive model is presented in [14] to predict the change propagation. The approach does not require access to the source code. It derives information from expert knowledge, user manuals and maps them in a weighted dependency graph. Reachability analysis is done on that weighted graph to reason the change propagation. The tool 'JTracker' is popular for assisting change propagation and CIA in such that when a programmer changes a class, this tool creates the potentially impacted neighbouring classes. The propagation is terminated if the changes in neighbouring classes are not necessary. 'JRipples' is another significant tool to support change propagation during the incremental changes [12][25].

In an earlier related study, Lee [58] has addressed impact analysis algorithms and evaluation metrics, which have been developed in a tool called 'ChaT' tool leading path to research on CIA for object-oriented programs. That algorithm relies on the computation of the transitive closure of object-oriented data dependency graphs. Those graphs based on control and data flow information extracted from the code program. The algorithms analyze the relationships between components and weight them according to the type of relation. It is expressed as a set of impact propagation rules.

VI. Continuous Integration and DevOps Tools

Continuous Integration is the repetitive integration process of developing and testing during a software development process [2][16]. It elaborates the frequent merging of the sole components of a software system to a shared branch by preserving the healthiness of the code. The automation of the CI process is significant to reduce the risks associated with software development such as lack of deployable software, late discovery of defects and lower project visibility [2].

In CI, the code commits to the version control repositories are frequently pushed to the CI servers and generate build scripts to integrate the new changes to the software. For instance, the concept of a single source point is encouraged using version control repositories such as CVS, Subversion, Perforce and Visual SourceSafe that allows accessing all the source codes from a single primary location. After each build script execution by the CI servers, the feedback mechanism notifies the status of the build. It is recommended to fix the

discovered pipeline failures at the earliest possible way to preserve CI. Moreover, CI and testing are intricately linked together [16][59]. In order to trace the software artefacts, this method uses scripts based on version controlling to control the code rather than the individual commands. The 'Echo' approach is an evolving tool-based solution that addresses traceability in requirements, as it is impractical to use static documents to track the requirement artefacts in an Agile environment [60].

DevOps enables efficient deliverables by speeding-up the customer query processing using tool support [15]. The tool support in a DevOps environment, such as Jenkins, Travis, Ansible, Docker, Sonar, Maven and OpenStack helps to maintain CI and traceability. For instance, the existing high-level plugins such as Hudson post-build scripts enable automated analysis of CI operations in Jenkins. However, many supportive tools have mainly considered the source code artefact integrations regardless of other artefact combinations such as a modification in requirements, design and test cases. Further, the consequences of changes in integrations have not analyzed in terms of CIA and change propagation aspects. This section explores the main features of DevOps supportive tools.

Jenkins is a prominent opensource DevOps support tool that monitors frequently executed jobs. It is a rapid CI server with error detection. Jenkins server performs a set of tasks supported by a trigger, that can be a change in a version management system [61]. The main tasks of theworkflow are acquiring Git source code, trigger the job, build the source project and notify the results. The list of tasks includes performing a software build with Apache Maven, executing a shell script, archiving the build results and starting the integration tests. Jenkins builds and tests software systems continuously and supervises the job executions even though it is running on a remote machine. Further, Jenkins configuration is simple, deployable in large scale environments and call slaves from the cloud by adhering to a slave topology [62].

Puppet is another configuration tool in DevOps, based on deploying microservices efficiently [59]. The configurations are described using a set of scripts defined in a Domain Specific Language (DSL). Puppet provides a unified interface for activities such as starting system services that require different tasks in the various Operating Systems. Travis is another distributed CI service that supports building and testing open source software projects. It encourages teamwork by tightly coupling to DevOps practices [63]. It can perform automatically scheduled tests with GitHub repositories. Docker is an open platform to build, ship and execute distributed software applications even on a virtual machine or a cloud environment [59]. The existence of microservices has enriched by tools including Docker. It has made the containers or objects that hold and transport data accessible easily.

Table II summarizes the features of the related studies in artefact traceability management in terms of change detection, CIA, consistency management, change propagation and CI. For instance, Zhang et al. [64], have addressed the change detection and impact analysis with a framework implemented in AspectJ programs and [65] is an architectural level artefact specific work. The workspace awareness tool in [66] has involved all the phases in continuous integration in an event-based approach, but it lacks automation. The tool Echo based on agile practice presented by [60], has addressed requirements and design related artefacts.

TABLE II. COMPARISON OF RELATED WORK ON TRACEABILITY MANAGEMENT

Rated studies	Traceability establishment	Change detection	Change impact analysis	Consistency management	Change propagation	Continuous integration
Impact propagation approach based on artefact change types [42].	A rule-based approach to detect dependency	-	Rule-based approach. Multi-level modelling.	Multi-perspective	Analyze dependency relations recursively	-
Requirements traceability tool for Agile methodologies [60].	Text annotations. Conversation-centric model.	Visualization.	Manually via visualization. Forward, backward traceability.	-	Use of elaboration activities.	Versioning.
CIA to locate failure reasons in AspectJ programs [64].		Use syntactic dependency.	Static AspectJ call graphs.	-	-	-
Architectural decisions change management to demystify architecture [65].	A template-based approach using architectural decision.	Decision-based approach.	Manual analysis.	-	Decision-based approach with manual monitoring.	-
Tool solution for early change detection and resolution on code conflicts [66].	Event-based approach.	Visualization.	Event-based approach. Binary measurements.	Manual visualizations.	YANCEES notification service.	Use a tool for workspace awareness
Artefact change management based on a feature-oriented hypothesis. [67].	Feature-oriented approach.	Feature-oriented manner.	Calculate artefact feature dependencies.	-	-	-
An artefact management tool to assess IR capabilities in traceability recovery [68].	Information retrieval methods.	Matrix-based using VSM.	Rule-based approach.	Traceability recovery using LSI.	-	-
Semi-automated traceability creation on requirements, design decisions and architecture [69].		An integrative approach by using LISA tool.	An integrative approach by integrating tool AREL.	-	-	-
Traceability based on event notifications in distributed development environments [70].	Event-based approach.	Publisher-subscriber technique.	Event-based approach. Event logs for artefacts.	-	Update artefact event logs.	-

VII. Discussion

A. Limitations in Existing Studies

One major limitation in the current state-of-art and research-level solutions for change detection, CIA, consistency management, change propagation and CI is being solely addressing the source code artefact [22][28][55][64]. Therefore, tracing heterogeneous artefacts corresponding to all the stages of SDLC with CIA remains challenging with respect to rapid changes. Otherwise, adapting to multiple separate tools or frameworks is one way out which leads to a higher traceability management cost. Another challenge in CI is identified as having continuous prioritization for integrations.

The influence of the CIA is important in the traceability management process in a DevOps practice as consistency management and change propagation depend on it. However, many studies have not addressed all the relevant artefacts and consider only a few numbers of CIA approaches, due to the heterogeneity nature of the artefacts. Thus, related studies are limited to either requirement and design level artefacts or entirely on source code artefact [17][47][48]. There is a lack of research in adequate successful attempts that addresses both development and operations level artefacts covering the entire SDLC. The existing research-level outcomes lack proper user interface and visualization features that are essential for a collaborative development environment such as DevOps for fast decision making. Moreover, some studies are domain specific such as for product line environments, modular software, safety critical context. [24][46][34]. Consequently, being dependent on a specific tool environment and integration incompatibilities with existing tools limits the practical usage of traceability tools in a wider range [49][69]. The rule-based techniques can resolve the heterogeneity nature of the artefacts, but requires human involvement to define newer rules [40]. Thus, automation becomes problematic that eventually increases the traceability cost. Table III summarizes the feature considerations of some of the existing tools.

The lack of automation capabilities and the need for adapting to multiple tool-chains have limited the adaptation of traceability in DevOps practice. Some of the existing solutions are limited to semi-automation, which is not sufficient to reduce the traceability cost in large-scale solutions. Thus, achieving automation for complete traceability management process model is important for a DevOps environment to cope with the frequent change integrations.

B. Future Research Directions

Addressing the heterogeneity of artefacts in change detection, CIA, consistency management, change propagation and CI is one prominent active research directions. This will help to reduce the traceability management cost in practice and encourage practical traceability adaptation. Machine learning and probabilistic methodologies are becoming a promising solution stack as well [7]. Thus, a possible suggestion is to incorporate machine learning based algorithms to establish traceability links, so that the human involvement in defining rules and monitoring can be avoided to cope with the new formats of artefacts. In addition, it will enable the transformation from semi-automation to automation.

Moreover, different methodologies can be integrated into a common platform to reduce the overhead of adapting to multiple tools or frameworks. This will reduce the cost associated with the tool-chain management and the inconsistencies among tools. Further, maintaining the research outcomes of the traceability management in the context of DevOps would be helpful to approach for stable solutions.

TABLE III. Feature Comparison of Existing Traceability Management Tools

Tools / Features	TraceME [71]	IBM DOORS [72]	TraceAnalyzer [73]	LDRA-TBmanager [74]	ArchEvol [75]	ReqView [76]	ArchStudio [77]
Requirement traceability	√	√		√		√	
Design level traceability	√		√		√		√
Heterogeneous artefacts	√		√	√			
Traceability visualization		√	√		√		√
Traceability validation							
CI/ scheduling/ versioning		√		√	√	√	
Change detection							
Change impact analysis	√	√					
CIA validation							
Change propagation visualization							
Consistency management, PM		√				√	√
DevOps tools stack supportability							
IDE independence						√	√
Performance analysis							

VIII. Conclusion

This survey explores the current approaches and techniques for achieving software artefact traceability in a DevOps environment. This paper mainly considered the tasks such as change detection, change impact analysis, consistency management and change propagation with continuous integrations in DevOps practice, that accommodate for feasible traceability management. Moreover, software artefacts traceability management can be implemented as a supportive tool in DevOps practice. Traceability links are generated by considering the dependencies among artefacts. In order to support the CICD pipeline, change detection is performed for each CI task that contains artefact changes in the forms of an edit, addition or removal. The main associated techniques are edit-history, tree-differencing and differencing algorithms. Then, the impact of a given changed is identified with different approaches such as call graphs, dependence graphs, program slicing, formal semantics, logical coupling, IR and rule-based approaches. Further, mathematical weight-based approaches are used for the CIA process. Heuristic rules and distance-based techniques are discussed in the change propagation that follows the impacted artefacts based on the CIA results.

However, most of the related studies have limitations that restrict their suitability for traceability management in a collaborative DevOps environment. Most of the existing methodologies have not addressed artefacts in all stages of SDLC and considered only the requirement, design level or source code artefacts. Although some studies have addressed a fully-automation process considering specific artefact types, there are limitations such as high traceability cost and efforts in a DevOps environment due to the frequent CI tasks. Thus, automated tool support for traceability management that addresses the overall SDLC in a DevOps environment is a need in practice. Accordingly, the requirement of having a generalized traceability solution to cope with the maximum types of artefacts, with a minimum cost and maximum level of automation is identified as a future research direction for an efficient traceability management process in DevOps practice.

Acknowledgment

The authors acknowledge the support received from the Senate Research Committee Grant SRC/LT/2017/12, University of Moratuwa, Sri Lanka in publishing this paper.

References

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

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

[3] S. A. Bohner and R. S. Arnold, Software change impact analysis. Wiley-IEEE Computer Society, 1996.

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

[5] W.T. Lee, W.Y. Deng, J. Lee, and S.J. Lee, "Change impact analysis with a goal-driven traceability-based approach," Int. J. Intell. Syst., vol. 25, no. 8, pp. 878–908, 2010.

[6] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," IEEE Trans. Softw. Eng., vol. 31, no. 6, pp. 429–445, 2005.

[7] C. Mills, "Towards the automatic classification of traceability links," in 32nd IEEE/ACM International Conference on Automated Software Engineering-ASE 2017, 2017, pp. 1018–1021.

[8] M. Lee and A. J. Offutt, "Algorithmic analysis of the impacts of changes to object-oriented software," in Technology of Object-Oriented Languages and Systems, 2002, pp. 61–70.

[9] G. Tóth, P. Hegedűs, Á. Beszédes, T. Gyimóthy, and J. Jász, "Comparison of different impact analysis methods and programmer's opinion: an empirical study," in 8th International Conference on the Principles and Practice of Programming in Java - PPPJ '10, 2010, pp. 109–118.

[10] W. Wang, Y. He, T. Li, J. Zhu, and J. Liu, "An Integrated Model for Information Retrieval Based Change Impact Analysis," Sci. Program., vol. 2018, no. Article ID 5913634, pp. 1–13, 2018.

[11] Y. Zhang, C. Wan, and B. Jin, "An empirical study on recovering requirement-to-code links," in 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing - SNPD, 2016, pp. 121–126.

[12] S. Lehnert, "A taxonomy for software change impact analysis," in 12th International Workshop and the 7th Annual ERCIM Workshop on Principles on Software Evolution and Software Evolution - IWPSE-EVOL '11, 2011, pp. 41–50.

[13] T. Mens, J. Buckley, M. Zenger, and A. Rashid, "Towards a Taxonomy of Software Evolution," Journal of Software Maintanance and Evolution, vol. 17, no. 5, pp. 309–332, 2005.

[14] A. Aryani, I. D. Peake, and M. Hamilton, "Domain-based change propagation analysis: An enterprise system case study," in 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1–9.

[15] F. M. A. Erich, C. Amrit, and M. Daneva, "A qualitative study of DevOps usage in practice," Journal of Software Evolution and Process, vol. 29, no. 6, p. e1885, 2017.

[16] A. Eck, F. Uebernickel, and W. Brenner, "Fit for continuous integration: how organizations assimilate an agile practice," in 20th Americas Conference on Information Systems - AMCIS '14, 2014, pp. 1–11.

[17] M. Rath, D. Lo, and P. Mäder, "Analyzing requirements and traceability information to improve bug localization," Proc. 15th Int. Conf. Min. Softw. Repos. - MSR '18, pp. 442–453, 2018.

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

[19] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom, "Change detection in hierarchically structured information," ACM SIGMOD Rec., vol. 25, no. 2, pp. 493–504, 1996.

[20] E. Kitsu, T. Omori, and K. Maruyama, "Detecting Program Changes from Edit History of Source Code," in 20th Asia-Pacific Software Engineering Conference (APSEC), 2013, pp. 299–306.

[21] B. Fluri and H. C. Gall, "Classifying Change Types for Qualifying Change Couplings," in 14th IEEE International Conference on Program Comprehension (ICPC'06), 2006, vol. 2006, pp. 35–45.

[22] J.-R. Falleri, F. Morandat, X. Blanc, M. Martinez, and M. Montperrus, "Fine-grained and accurate source code differencing," in 29th ACM/IEEE International Conference on Automated Software Engineering - ASE '14, 2014, pp. 313–324.

[23] T. Sager, A. Bernstein, M. Pinzger, and C. Kiefer, "Detecting similar Java classes using tree algorithms," in International Workshop on Mining Software Repositories - MSR '06, 2006, pp. 65–71.

[24] H. Cho, J. Gray, Y. Cai, S. Wong, and T. Xie, "Model-Driven Impact Analysis of Software Product Lines," in Model-Driven Domain Analysis and Software Development, IGI Global, 2011, pp. 275–303.

[25] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," Softw. Test. Verif. Reliab., vol. 23, no. 8, pp. 613–646, Dec. 2013.

[26] I. G. Czibula, G. Czibula, D. L. Miholca, and Z. Marian, "Identifying Hidden Dependencies in Software Systems," Stud. Univ. Babeș-Bolyai Inform., vol. 62, no. 1, pp. 90–106, 2017.

[27] S. Wong, Y. Cai, and M. Dalton, "Change Impact Analysis with Stochastic Dependencies," PA, USA, 2011.

[28] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," in 33rd International Conference on Software Engineering - ICSE '11, 2011, pp. 746–755.

[29] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," in 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering - PASTE '08, 2008, pp. 84–90.

[30] G. A. Oliva, M. A. Gerosa, D. Milojicic, and V. Smith, "A change impact analysis approach for workflow repository management," in IEEE 20th International Conference on Web Services, ICWS 2013, 2013, pp. 308–315.

[31] S. J. Kabeer, M. Nayebi, G. Ruhe, C. Carlson, and F. Chew, "Predicting the Vector Impact of Change - An Industrial Case Study at Brightsquid," in ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2017, pp. 131–140.

[32] M. Shahid and S. Ibrahim, "Change impact analysis with a software traceability approach to support software maintenance," in 13th International Bhurban Conference on Applied Sciences and Technology (IBCAST), 2016, pp. 391–396.

[33] F. Déhoulé, L. Badri, and M. Badri, "A Change Impact Analysis Model for Aspect Oriented Programs," in 12th International Conference on Evaluation of Novel Approaches to Software Engineering, 2017, pp. 144–157.

[34] M. Borg, K. Wnuk, B. Regnell, and P. Runeson, "Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context," IEEE Trans. Softw. Eng., vol. 43, no. 7, pp. 675–700, 2017.

[35] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in 27th International Conference on Software Engineering - ICSE '05, 2005, pp. 432–441.

[36] Y. Wang, J. Zhang, and Y. Fu, "Rule-Based Change Impact Analysis Method in Software Development," in 2nd International Conference on Computer Engineering, Information Science & Application Technology - ICCIA, 2017, vol. 74, pp. 396–403.

[37] D. Kchaou, N. Bouassida, and H. Ben-Abdallah, "UML models change impact analysis using a text similarity technique," IET Softw., vol. 11, no. 1, pp. 27–37, 2017.

[38] X. Ren, B. G. Ryder, M. Stoerzer, and F. Tip, "Chianti: a change impact analysis tool for Java programs," in 27th International Conference on Software Engineering - ICSE '05, 2005, pp. 664–665.

[39] A. M. D. Duarte, D. Duarte, and M. Thiry, "TraceBoK: Toward a Software Requirements Traceability Body of Knowledge," in 24th International Requirements Engineering Conference, 2016, pp. 236–245.

[40] A. Goknil, I. Kurtev, and K. van den Berg, "A Rule-Based Change Impact Analysis Approach in Software Architecture for Requirements Changes," eprint arXiv:1608.02757, pp. 1–44, 2016.

[41] S. Lehnert, "Multiperspective Change Impact Analysis to Support Software Maintenance and Reengineering," University of Hamburg, 2015.

[42] S. Lehnert, Q. U. A. Farooq, and M. Riebisch, "Rule-based impact analysis for heterogeneous software artifacts," in European Conference on Software Maintenance and Reengineering - CSMR, 2013, pp. 209–218.

[43] H. Kagdi, "Improving change prediction with fine-grained source code mining," in 22nd IEEE/ACM International Conference on Automated Software Engineering - ASE '07, 2007, pp. 559–562.

[44] A. R. Sharafat and L. Tahvildari, "A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems," in 11th European Conference on Software Maintenance and Reengineering - CSMR'07, 2007, pp. 27–38.

[45] T. H. D. Nguyen, B. Adams, and A. E. Hassan, "Studying the impact of dependency network measures on software quality," in 2010 IEEE International Conference on Software Maintenance, 2010, pp. 1–10.

[46] F. Angerer, H. Prahofer, and P. Grunbacher, "Modular Change Impact Analysis for Configurable Software," in 2016 IEEE International Conference on Software Maintenance and Evolution - ICSME, 2016, pp. 468–472.

[47] B. Dit et al., "ImpactMiner: a tool for change impact analysis," in 36th International Conference on Software Engineering - ICSE Companion 2014, 2014, pp. 540–543.

[48] L. Zhang, M. Kim, and S. Khurshid, "FaultTracer: a change impact and regression fault analysis tool for evolving Java programs," in ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12, 2012, pp. 40:1-40:4.

[49] H. C. Gall, B. Fluri, and M. Pinzger, "Change Analysis with Evolizer and ChangeDistiller," IEEE Softw., vol. 26, no. 1, pp. 26–33, 2009.

[50] S. Wong and Y. Cai, "Predicting change impact from logical models," in IEEE International Conference on Software Maintenance, ICSM, 2009, pp. 467–470.

[51] L. C. Briand, Y. Labiche, and L. O'Sullivan, "Impact analysis and change management of UML models," in International Conference on Software Maintenance, ICSM 2003, 2003, pp. 256–265.

[52] M. Sherriff and L. Williams, "Empirical software change impact analysis using singular value decomposition," in 1st International Conference on Software Testing, Verification and Validation, ICST, 2008, pp. 268–277.

[53] L. Hattori, G. dos Santos Jr, F. Cardoso, and M. Sampaio, "Mining software repositories for software change impact analysis: a case study," in SBBD '08 23rd Brazilian symposium on Databases, 2008, pp. 210–223.

[54] A. De Lucia, F. Fasano, and R. Oliveto, "Traceability management for impact analysis," in Frontiers of Software Maintenance, 2008, pp. 21–30.

[55] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," in 2012 34th International Conference on Software Engineering - ICSE, 2012, pp. 430–440.

[56] J. Cleland-Huang, O. C. Z. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in Future of Software Engineering - FOSE 2014, 2014, pp. 55–69.

[57] I. Pete and D. Balasubramaniam, "Handling the differential evolution of software artefacts: A framework for consistency management," in IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering - SANER, 2015, pp. 599–600.

[58] M. L. Lee, "Change impact analysis of object-oriented software," George Mason University, Virginia, 1998.

[59] V. Farcic, The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices, 1st ed. CreateSpace Independent Publishing Platform, 2016.

[60] C. Lee, L. Guadagno, and X. Jia, "An agile approach to capturing requirements and traceability," in 2nd International Workshop on Traceability in Emerging Forms of Software Engineering, 2003, pp. 1–7.

[61] J. Hembrink and P.-G. Stenberg, "Continuous integration with Jenkins," Coach. Program. Teams - EDA 270, pp. 1–8, 2013.

[62] A. M. Berg, Jenkins Continuous Integration Cookbook, 2nd ed. Packt Publishing, 2015.

[63] "Travis CI," Travis CI, 2018. [Online]. Available: https://travis-ci.org/. [Accessed: 05-Jul-2017].

[64] S. Zhang, Z. Gu, Y. Lin, and J. Zhao, "Change impact analysis for AspectJ programs," in IEEE International Conference on Software Maintenance, 2008, pp. 87–96.

[65] J. Tyree and A. Akerman, "Architecture decisions: Demystifying architecture," IEEE Softw., vol. 22, no. 2, pp. 19–27, 2005.

[66] A. Sarma, D. F. Redmiles, and A. Van Der Hoek, "Palantír: Early detection of development conflicts arising from parallel code changes," IEEE Trans. Softw. Eng., vol. 38, no. 4, pp. 889–908, 2012.

[67] L. Passos, S. Apel, C. Kästner, K. Czarnecki, A. Wasowski, and J. Guo, "Feature Oriented Software Evolution," in 7th International Workshop on Variability Modelling of Software-intensive Systems - VaMoS '13, 2013, pp. 17:1-17:8.

[68] 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.

[69] G. Buchgeher and R. Weinreich, "Automatic Tracing of Decisions to Architecture and Implementation," in 9th Working IEEE/IFIP Conference on Software Architecture, 2011, pp. 46–55.

[70] 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.

[71] 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.

[72] IBM, "IBM-Rational DOORS", Available https://www.ibm.com/us-en/marketplace/rational-doors, [October 14, 2018].

[73] A. Egyed, "A scenario-driven approach to traceability", In 23rd International Conference on Software Engineering, ICSE, 2001, pp. 123–132.

[74] LDRA, "TBmanager", Available https://ldra.com/industrial-energy/products /tbmanager/tbmanager/, [January, 10, 2019].

[75] E. C. Nistor, J. R. Erenkrantz, S. A. Hendrickson and A. van der Hoek, "ArchEvol: versioning architectural-implementation relationships", In 12th International Workshop on Software Configuration Management, SCM, 2005, pp. 99–111.

[76] "ReqView," 2018. [Online]. Available: https://www.reqview.com/. [Accessed: 07-May-2018].

[77] E. Dashofy, H. Asuncion, S. Hendrickson, G. Suryanarayana, J. Georgas, and R. Taylor, "ArchStudio 4: An Architecture-Based Meta-Modeling Environment," In 29th International Conference on Software Engineering - ICSE, 2007, pp. 67–68.