UNIVERSITY OF MORATUWA, SRI LANKA

TOOL SUPPORT FOR DevOps PROCESS ENHANCEMENTS

By

Dr. D.A. Meedeniya

A REPORT

SUBMITTED TO THE SENATE RESEARCH

COMMITTEE [**GRANT No. SRC /LT/2016/07**]

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY OF MORATUWA

2018

Contents

Abstract

Title of Project: **Tool Support for DevOps Process Enhancements**

Grant No: SRC/LT/2016/07

Software system engineering is rapidly growing to larger scales and software maintenance tends to be complex. The number of involving software artefacts increases with the growth of software systems. Thus, different software development approaches are getting introduced to ease the software management. Therefore, the notion of traceability management of software artefacts is given prominence along with continuous integration. DevOps based software development is in the rise among software development practitioners with the integration of developments and operations. DevOps improves software delivery and customer satisfaction by bringing together a set of activities, which can be repeated multiple times a day. Tool support for this level of continuous delivery is essential. Provision of traceability management tool support for DevOps process management remains unfulfilled at large, which we explored and addressed in this research.

The objectives of this research are

- To identify challenges in DevOps based software development.
- To design a prototype tool to address DevOps process challenges.
- To develop and evaluate the proposed tool for DevOps practice.

First, we have performed a context survey to identify the theoretical model (prescriptive process) of DevOps and the actual realization of the DevOps practice (descriptive process) in software development. With the gap analysis we identified the key features that support within the tool. Then, we have come up with an approach for managing traceability between software artefacts and the architecture is designed accordingly. Next, a prototype tool is developed to support key process steps of DevOps with continuous integration. This was evaluated with some case study applications such as POS and tour management system.

This research is developed a tool called SAT-Analyzer (Software artefact traceability analyzer), which is a prototype tool for establishing, managing and visualizing software artefacts in the software development life cycle with continuous integration. The case-study based evaluation shows positive accuracy results for the SAT-Analyser tool. The research output provides an original contribution to the field of software process management in general and tool support within DevOps in particular. With a usable tool support for DevOps practices can improve the DevOps process stability and performance. This will enable further extensions to the tool and conceptual model on DevOps usage into related areas such as software maintenance and quality management as needed in the software industry. This will be a main contribution to a number of research areas supporting software process management and enhanced software developed with rapid delivery.

Towards Traceability Management in Continuous Integration with SAT-Analyzer

I. D. Rubasinghe, D. A. Meedeniya, I. Perera
Department of Computer Science and Engineering,
University of Moratuwa, Sri Lanka
ireshar@cse.mrt.ac.lk, dulanim@cse.mrt.ac.lk, indika@cse.mrt.ac.lk

ABSTRACT

Software system engineering is rapidly growing to larger scales and software maintenance tends to be complex. The number of involving software artefacts increases with the growth of software systems. Thus, different software development methodologies, processes and practices are getting introduced to ease the software management. Consequently, the management of excessive software artefacts is also important towards a successful maintenance. Therefore, the notion of traceability management of software artefacts is given prominence along with continuous integration. This paper explores the existing traceability management approaches to propose an optimized framework that overcomes current limitations. Hence, the previous work of this research, SAT-Analyzer, which is a prototype tool, is extended to support continuous integration with DevOps practices.

CCS Concepts

• Software and its engineering → Software creation and management → Software post-development issues → Software evolution.

Keywords

Traceability management; continuous integration; change detection; impact analysis; DevOps.

1. INTRODUCTION

Software systems, in today's context, are considered as critical business assets. Change of a software system is inevitable and required to be updated continuously in order to maintain the value of these assets. Hence, software evolution is preferred over building completely new software systems due to the cost and time benefits [1]. Generally, software evolution occurs in a software system life cycle at a stage where it is in active operation and is evolving due to new requirements. The software evolution mainly depends on the type of software being maintained; involved in the development processes and continues within the software system lifecycle. The evolution is highly coupled with the components that are affected by the change; hence the cost and change impact can be estimated [2].

Software artefacts are the intermediate by-products used in each phase of the software development life cycle (SDLC) towards the intended software product. Changes in software artefacts are the primary motivation in software evolution [1]. It is crucial to maintain the consistency between the software artefacts, with the increasing scope of a software system. This is due to the rapid generation of information across a large information space. Thus, there is a requirement of the ability to describe and follow the artefact lifecycle. Without a well-defined traceability management between the software artefacts the consequences of different evolutions may result in expensive overheads in SDLC. Further, improper traceability management may lead to failures of a product. Therefore, traceability of software artefacts is important for the software evolution process. It strengthens the testability, maintainability and helps for system acceptance by providing consistent documentation [3]. The improper management and outdated artefacts can lead to inconsistency among artefacts, synchronization issues and lack of trust in artefacts by stakeholders. Thus, it is significant to maintain the traceability throughout the SDLC.

The concept of DevOps (Development-Operations) represents the integration of development environment and the operational environment that encourages developing systems rather than mere programs. DevOps ease the project management with communication, understandability, integration and bridging the gap between the development teams and operational teams. It increases the rate of change and deploys features into production faster [4]. There is a strong relationship between the quality of the software developed and the agility of the organization to the DevOps practices of software development [5]. Therefore, DevOps practices contribute to enhance these software quality attributes within continuous integration process.

SAT-Analyzer (Software Artefacts Traceability Analyzer) is a prototype tool developed previously, with the intension of traceability management [6] [7] [8]. It includes a core engine for traceability establishment and visualization. However, it mainly considers software artefacts such as natural language based requirements, UML class diagrams, and Java source code for traceability management as of now; the integration of DevOps practices along with continuous integration is explored. This paper mainly explores extensive related research and proposes an optimised framework for traceability management with continuous integration.

The paper is organized as follows: Section 2 presents related approaches in traceability management including change detection, impact analysis, change propagation and consistency management. Section 3 evaluates the literature and the proposed framework is elaborated in Section 4. Finally, Section 5 concludes the paper with future research directions.

ICCIP '17, November 24-26, 2017, Tokyo, Japan.
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5365-6/17/11...$15.00
DOI: https://doi.org/10.1145/3162957.3162985

2. TRACEABILITY APPROACHES

2.1 Terminology

A range of software artefacts is involved throughout the SDLC. Some of the early stage artefacts are Software Requirement Specification (SRS), design diagrams, architectural documents and quality attributes or the non-functional requirements reports and source code. Test scripts, walkthroughs, inspections, bug reports, build logs and test reports, configuration files, user manuals are important artefacts present in the latter stage of SDLC. Nevertheless, there is a relationship between the primary artefacts with the final deliverables of the software product. Thus the consistent management of software artefacts contains significant importance in fine-tuning the software products.

Software artefact traceability, which is a key notion in the software evolution, refers to the ability of building and tracking the relationships among artefacts both backward and forward [3]. Traceability of different software artefacts can be among homogeneous, or heterogeneous such as requirement to design traceability and design to source code traceability, for example. Requirement traceability shows the dependencies between requirements and among the requirements and design/ code of a software system. Thus, the artefact management is essential to maintain adequate consistency in approaching towards a software product. Hence, the notion of software artefact traceability facilitates to overcome the inconsistencies in software artefacts.

DevOps concept motivates towards the reduction of the gap between development and operations teams [9]. In a DevOps environment, significant software artefact changes are expected rapidly. Thus, there is a requirement of determining and analysing the resulted impact of the traceability to make accurate change acceptance decisions in a DevOps environment [5].

2.2 Traceability Management

The major challenges in tracing software artefacts are due to different formats, abstraction levels and lack of defined data format for artefacts [10]. Extracting relevant data and analyzing the content of the artefact is one of the primary techniques towards the traceability link generation. When text is used to provide descriptive details of the informal semantics in artefacts, the frequently involved pre-processing steps can be identified as text normalization, identifier splitting and stop word removal.

Traceability provides a logical connection between artefacts of the software development process. The cost of maintaining a larger number of artefact relationships when a change occurs is identified as a major reason for the limited use of traceability in practice. Moreover, it is signified that the effort of maintaining artefact relations is considerably high though the number of artefacts is minimal. Hence, traceability maintenance, ensuring the correctness of traceability over time is significant to address [11]. Thus, proper identification of a feasible traceability maintenance approach could reduce the total cost and effort in the software development process.

The Rule-based approaches define rules based on the attributes of the artefacts to generate traceability links between different software artefacts. Then the traceability links maintenance is performed by re-evaluating the rules. Furthermore, the rule-based approaches can be combined with event-driven approaches. Thus the traceability maintenance can be conducted in two phases: recognizing changes based on events, and re-evaluating the rules that governing link updates [12].

The event-based approaches use the events occurring during software development activities to maintain traceability links. Accordingly, the deletion of an artefact can be made as a trigger to delete all the connected traceability links to it. Many related work has achieved this using similar conceptual techniques such as publish and subscribe mechanism for connecting traceability maintenance tasks to particular events [12]. The requirements and source code are classified as mandatory inputs to the hypertext-based traceability maintenance approaches, whereas conformance analysis is identified as complementary inputs [3]. This has used XML and the types of software artefacts are viewed as constraints on one another. A set of constraints are provided in the constraint-based approaches that must not be violated by any traceability link [13]. The traceability links that are not clearly referenced in any constraint are considered to be consistent by default. The transformation-based approaches have shown that artefacts generated through model transformations can be enriched to generate traceability links [12]. However, it is still found to be contradictory in practice. Furthermore, graph-transformation based methodologies are involved in to define, identify and maintain the traceability links in this domain [14].

Alternatively, Design Decision Tree (DDT) provides ability to connect requirements to architecture decision and design elements under traceability establishment. There is a model named 'Architecture Rationale and Elements Linkage (AREL)' that has targeted traceability in the design rationale modeling using the conceptual UML notations [15]. It can be used to capture relationships between only the two entities: architecture rationale and architecture elements.

2.3 Change Detection and Impact Analysis

Since software change is the central norm of today's mainstream SDLC, it is an utmost importance to cope with the changes properly to reduce cost regardless of the used software development model. A hypothesis-based change management with a traceability timeline in a feature-oriented manner is presented in [16]. They have mapped important requirements as features and a change is addressed in the feature level.

Change impact analysis (CIA) in software development detects the consequences of an artefact alteration on other parts of the software system. Generally, impact analysis is conducted before or/and after a change implementation [17]. The benefits of piloting impact analysis prior to a change are understandability, change impact prediction and cost estimations. Therefore, conducting impact analysis after an execution of a change can be beneficial in tracing ripple effects, selecting test cases and performing change propagation.

Different impact analysis methods are available in the literature. One such categorization is traceability-based and dependence-based [17]. The traceability-based CIA is narrowed in recovering the traceability links among software artefacts. Dependence-based CIA is defined as estimating the change effects of a proposed change. Another categorization of CIA techniques is static impact analysis and dynamic impact analysis. Static CIA techniques consider all possible behaviors and inputs [18]. Thus, contains a cost of precision though safe. Moreover, static CIA techniques analyze the syntax and semantic dependencies of a program code and construct intermediate representations using call graphs and program dependency graphs such as call graphs. Besides dynamic CIA techniques overcome this drawback by considering only a part of the inputs in practical use. Hence, their impact sets are identified to be more precise though less safe.

2.4 Change Propagation

Change propagation conducts after the sequence of tasks such as change detection, change impact analysis to trace ripple effects, selection of test cases, etc. [17]. When an alteration occurred, it is essential to ensure that other related artefacts are consistent as well. Change propagation considers the required new changes for other entities in the application to ensure the consistency within the system after an entity has been changed. Change propagation is mostly performed during the incremental changes.

An approach for change propagation in heterogeneous software artefacts by combining multi-perspective modeling and impact analysis is presented in [19]. They have introduced a recursive change propagation algorithm that restricts the change propagations across dependency relation regardless of the type and limit size of the impact sets to be computed. Another technique is the use of a distance measure to control the propagation of changes to indirectly related artefacts by either terminating the change propagation or by prioritizing the impact paths based on their depth [20]. Furthermore, there exist probabilistic models, such as Markov Chains and Bayesian Belief Networks that model change propagations based on mathematical theorems [21]. Thus, contribute in computing the probability of an entity being impacted by a change in an artefact.

2.5 Consistency Management

The changes and refinements that occur in artefacts are not guaranteed to happen in a same speed and pace. Therefore the consequences of each artefact change or refinement may not result in a uniform pattern. Some refinements may reflect and impact on other artefacts immediately. Thus, the stability among artefacts can become inconsistent and can fail in representing the expected software system solution. Consequently, that can lead to stakeholder dissatisfaction and system failure. Therefore, consistency management is essential to minimize efforts in software maintenance. Consistency management is the ability to preserve the synchronization among software artefacts along with the occurring changes [2]. Accordingly, an artefact alteration or the presence of outdated artefacts should consistently reflect on other affected artefacts before continuing in the software process.

A significant holistic artefact management framework that considers traceability in heterogeneous artefacts and the notions of change detection, change impact analysis and consistency checking has discussed in [2]. They have used different source code impact analysis techniques to support software artefacts such as requirements in natural language, UML class diagrams and Java source code. The presented prototype has emphasized any artefact inconsistencies with solution options. However, the work is limited for non-distributed development environments.

2.6 Continuous Integration

Continues Integration (CI) is the repetitive integration process of building and testing in a software process. It elaborates the frequent merging of the sole components of an application into a shared branch by preserving the healthiness of the code.. The impact of CI is significant in reducing the risks in software development such as lack of deployable software, late discovery of defects and lower project visibility [22]. Here, the code commits to the version control repositories are frequently pushed into the CI servers and applied build scripts to integrate new changes. The principal *Single Source Point* is encouraged via having version control repositories such as CVS, Subversion, Perforce and Visual SourceSafe that allows to access all source codes from a single primary location [22]. Also there is a feedback mechanism involved after each build script execution by CI servers to notify the [illegible] fixing the [illegible] pipeline failures without delaying is recommended best practice to preserve CI. Moreover, the rationale of version controlling using the scripts to control code rather than individual commands is a key methodology in tracing software artefacts.

DevOps broadens the view of software engineering paradigm by defining metrics that are understood across teams, sharing measurement methods and tools, bring in automation, measure everything to share among team members and by making performance part of agile stories [23]. DevOps is an approach in testing strategies that increases the organization throughput. It has been a powerful selection for better results and in speeding up customer query processing due to the evolving tool support.

Jenkins is a prominent DevOps tool that supervises regularly executed jobs. It is an open source, rapid, continuous integration server that generates a scenario where errors are being detected at an early stage in the SDLC. The basic functionality of Jenkins server is to conduct a list of steps supported by a trigger [24]. Puppet is another configuration tool in DevOps, that deploys micro-services [25]. There is a central configuration server that is polled by clients for making changes to the configuration [26]. The configurations are described using a set of scripts defined in a Domain Specific Language (DSL). Docker is another open platform for building, shipping and executing distributed software applications even on a virtual machine or a cloud environment. The existence of microservices has enriched by tools including Docker. It has made the containers or the objects that hold and transport data accessible for everyone easily [25]. Thus, the powerful utilization of Docker has reduced the deployment efforts in microservices. Travis [27] is a recognized distributed continues integration service that supports building and testing open source software projects. It encourages team workings by tightly coupling to DevOps practices. Further, it performs automatic scheduled tests with GitHub repositories.

3. TRACEABILITY IN PRACTICE

3.1 Traceability Support Techniques

Figure 1, illustrates a combination of existing techniques and approaches in the domain of traceability management, change detection, impact analysis, consistency management and continuous integration. It emphasizes the lack of specific techniques in traceability management in CI rather than theoretical principles such as DevOps, probabilistic practices.

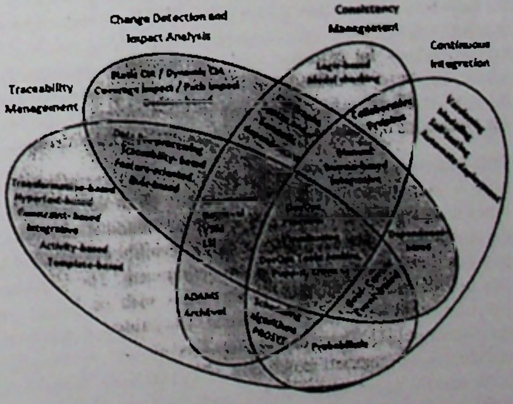

Figure 1. Traceability support techniques

34

Table 1. Evaluation of traceability support techniques

Technique	Functionalities	Methods/ techniques followed	Advantages	Limitations
Rule-based	Define rules in traceability links generation	Rules based on artefact attributes. Traceability maintenance is based on rule re-evaluation [11]	Ideal for artefacts such as requirements, use cases and object models. [11]	Weakness in recognition of structural changes. [3]
Hypertext-based	Support traceability maintenance	XML. Markup specifications. [28]	Supports requirements and source code artefacts. [3]	Weekly support for other types of artefacts.
Event-based	Automate trace link generation and maintenance.	Publish-subscribe relationship mechanism. Event-based subscriptions. [29]	Ability to maintain dynamic links. [29]	Scalability issues when maintaining the dynamicity of the traceability. [29]
Constraint-based	Support traceability maintenance	Set of constraints are provided that must not be violated by any traceability link. [13]	Most artefacts types can be viewed as constraints on one another. [13]	Difficulty in referencing all traceability links to constraints. [13]
Transformation-based	Support traceability maintenance.	Incremental transformation [12] Graph- transformation based methodologies. [14]	Beneficial for model based software systems. [12]	Not all software artefacts are generated by model transformations. [12]
Goal-centric (GCT)	Manage change impact of non-functional requirements.	Soft goal Interdependency Graph (SIG). Traceability matrix. [29]	Maintain the quality by assessing the impact of functional changes upon non-functional requirements. [29]	Lack of scalability and tool support. [29]

A comparison of traceability management techniques is given in Table 1. The major limitations are being restricted for few types of artefacts and insufficient tool support. Many techniques addresses only the requirements and design level software artefacts. Thus, the artefacts in later phases of SDLC such as test reports and configuration files are not extensively addressed.

3.2 Challenges in Traceability Management

The current software industry is still reluctant in adapting the traceability aspects in to the environments due to the above identified limitations. The major challenge is in building an automated tool for traceability support with a wide range of customizability and scalability [29]. It is important to consider most of the artefact types and development environments [12]. Also it is challenging to visualize traceability management in a flexible way [30]. Many existing work lacks tangible direct advantages of traceability management in software development. Further, maintaining traceability links during continuous software evolution is challenging, as it is an endless and error prone task.

4. PROPOSED FRAMEWORK

We propose a framework to capture traceability management in a continuous integration environment with DevOps practices and the high-level view is illustrated in Figure 2. The previous work of this research [6] [7] [8], SAT-Analyzer, is primarily involved in this framework for extending with the proposed enhancements, which are shown in dashed line. Yet, the existing components of the SAT-Analyzer, which are shown in filled colour are still need enhancements to cater new software artefacts and considerations.

This framework mainly considers software artefacts in CI process such as configuration files and test scripts. With the scheduler a scheduling algorithm will be implemented to automatically trigger the continuous integration along with traceability management by providing automation in a DevOps environment. The CI process can be integrated with the DevOps tools such as Jenkins that supports build automation, versioning, triggering and distributed development [31]. Therefore, enables DevOps with rapid changes, collaborations, constant monitoring, CI and delivery. Thus, the CI component is compromised with change detection, change impact analysis, change propagation through the dependent artefacts and consistency management among the affected artefacts prior to the visualization of the traceability links. Correspondingly, the DevOps practices can be achieved in this framework.

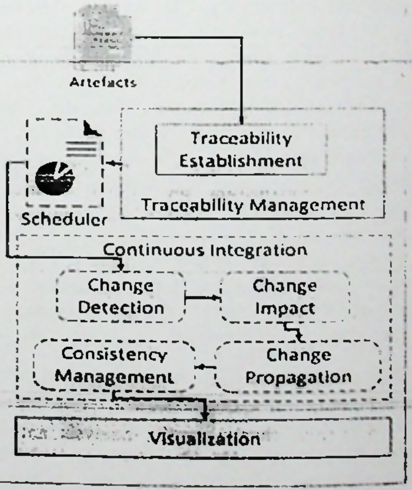

Figure 2. High-level view of the SAT-Analyzer extension

5. CONCLUSION

Traceability management in a continuous integration environment is an important aspect in SDLC due to the risk of conflicts and the growth of software maintenance cost. This paper explores literature on traceability management, change detection, impact analysis, change propagation, consistency management and continuous integration. The main limitation in existing context is lack of sufficient tools and techniques. The existing tools are limited to certain types of software artefacts and development environments depending on the used programming languages or the design notations. Thus, the automation of traceability relations generation has become unachievable completely. Moreover, the support for traceability and continuous integration is important to be available throughout the SDLC, which is not completely preserved in current practices. Thus, the necessity of a framework for traceability management and continuous integration to cover SDLC with DevOps practices is identified. Further, this paper proposed an extended framework for the existing SAT-Analyzer tool. The proposed tool will be beneficial in the long run of software development in terms of traceability management in continuous integration.

6. ACKNOWLEDGMENTS

The author acknowledges the support received from the LK Domain Registry in publishing this paper. The conclusions and recommendations in this paper are those of the author and may not necessarily reflect the views of the LK Domain Registry.

7. REFERENCES

[1] Rajlich, V. and Václav 2014. Software evolution and maintenance. *In Proceedings of the on Future of Software Engineering (FOSE 2014)*. ACM, New York, USA, (2014). 133–144.

[2] Pete, I. et al. 2015. Handling the differential evolution of software artefacts: A framework for consistency management. *In Proc.of the 22nd Int. Conf. on Software Analysis, Evolution, and Reengineering*. (2015), 599–600.

[3] Cleland-Huang, J. et al. 2012. *Software and systems traceability*. Springer.

[4] Kim, G. 2011. Top 11 Things You Need to Know About DevOps. *IT Revolution Press*. (2011).

[5] Perera, I. et al. 2016. Evaluating the impact of DevOps practice in Sri Lankan software development organizations. *In Proceedings of the 16th Int.Conf.on Advances in ICT for Emerging Regions*, (2016), 281–287.

[6] Wijesinghe, D.B. et al. 2014. Establishing traceability links among software artefacts. *In Proceedings of the 14th Int. Conf. on Advances in ICT for Emerging Regions*. (2014). 55–62.

[7] Kamalabalan, K. et al. 2015. Tool Support for Traceability of Software Artefacts. *In Proceedings of the Moratuwa Engineering Research Conference*, (2015), 318-323.

[8] Arunthavanathan, A. et al. 2016. Support for traceability management of software artefacts using Natural Language Processing. *In Proceedings of the 2nd Int. Moratuwa Engineering Research Conference*, (2016), 18–23.

[9] Pfleeger, P.C. et al. 2015. *DevOps A Software Architect's Perspective*.

[10] Al-Ani, B. et al. Continuous coordination within the context of cooperative and human aspects of software engineering, *In Proc.of the Int. workshop on Cooperative and human aspects of software engineering*, ACM, NY, (2008), 1-4.

[11] Mader, P. and Gotel, O. 2012. Towards automated traceability maintenance. *Journal of Systems and Software*. 85, 10 (2012), 2205–2227.

[12] Maro, S. et al. Traceability Maintenance: Factors and Guidelines. *In Proceedings of the 31st IEEE/ACM Int. Conf. on Automated Software Engineering (ASE 2016)*. ACM, USA, 1313- 1322.

[13] Fockel, M. et al. 2012. Semi-automatic establishment and maintenance of valid traceability in automotive development processes. *In Proc. of the 2nd Int. Workshop on Software Engineering for Embedded Systems*, (2012), 37–43.

[14] Schwarz, H. et al. 2010. Graph-based traceability: a comprehensive approach. *Software & Systems Modeling*. 9, 4 (2010), 473–492.

[15] Tang, A. et al. 2007. A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*. 80, 6 (2007), 918–934.

[16] Passos, L. et al. 2013. Feature-Oriented Software Evolution Categories and Subject Descriptors. *In Proc. of the Int. worksop on Variability Modelling of Software Intensive Systems (VaMoS)*, ACM. (2013). 17:1-17:8.

[17] Li, B. et al. 2013. A survey of code-based change impact analysis techniques. *Software Testing Verification and Reliability*. 23, 8 (2013), 613–646.

[18] Sun, X. et al. 2010. Change impact analysis based on a taxonomy of change types. *In Proc. of the Int. Computer Software and Applications Conference* (2010), 373–382.

[19] Lehnert, S. et al. 2013. Rule-Based Impact Analysis for Heterogeneous Software Artifacts. *In Proceedings of the 17th European Conference on Software Maintenance and Reengineering* (2013). 209–218.

[20] Di Rocco, J. et al. 2013. Traceability Visualization in Metamodel Change Impact Detection. *In Proceedings of the 2nd Workshop on Graphical Modeling Language Development*. (2013). ACM. NY, USA. 51–62.

[21] Lehnert, S. 2011. A review of software change impact analysis. (2011).

[22] Duvall, P. et al. 2007. *Continuous integration: improving software quality and reducing risk*. Addison-Wesley. 2007. 1-272.

[23] Gottesheim, W. et al. 2015. Challenges, benefits and best practices of performance focused DevOps. *In Proceedings of the 4th ACM/SPEC Int. Workshop on Large-Scale Testing*.(2015), ACM, NY, USA. 3-3.

[24] Mullaguru, S. 2015. Changing Scenario of Testing Paradigms using DevOps–A Comparative Study with Classical Models. *Global Journal of Computer Science and*. 15, 2 (2015).

[25] Viktor, F. 2016. The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices. 2nd ed. Victor Farcis. (2016), 397.

[26] Schäfer, A. et al. 2011. Collaborative Administration in the Context of Research Computing Systems. *October*. II, (2011), 1–6.

[27] Travis CI - Test and Deploy Your Code with Confidence: *https://travis-ci.org/*. Accessed: 2017-07-05.

[28] Alves-Foss, J. et al. 2002. Experiments in the use of XML to enhance traceability between object-oriented design specifications and source code. *In Proc.of the Annual Hawaii Int. Conf. on System Sciences.*, (2002), 3959–3966.

[29] Galvao, I. and Goknil, A. 2007. Survey of Traceability Approaches in Model-Driven Engineering. *In Proceedings of the 11th IEEE Int.Enterprise Distributed Object Computing Conference*, (2007), 313–313.

[30] Biehl, J.T. et al. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. *In Proceedings of the 2010 ACM SIGCHI Int.Conference on Human Factors in Computing Systems, ACM, USA, 1313- 1322.*

[31] Berg, A.M. 2012. Jenkins Continuous Integration Cookbook. I, PACKT publishing, (2012), 344.

Software Artefact Traceability Analyser: A Case Study on POS System

I. D. Rubasinghe
ireshar@cse.mrt.ac.lk

D. A. Meedeniya
dulanim@cse.mrt.ac.lk

G. I. U. S. Perera
indika@cse.mrt.ac.lk

Department of Computer Science and Engineering,
University of Moratuwa, Sri Lanka

ABSTRACT

Software traceability is a key notion in the software development. The paper explores the previously developed research-based Software Artefact Traceability Analyser tool called 'SAT-Analyser'. The workflow and capabilities of SAT-Analyser tool are described and evaluated using a case study of a Point of Sale system. Phases such as software artefact identification, data pre-processing, data extraction and traceability establishment methodologies used in the tool SAT-Analyser are presented with graph-based traceability outcome. The case-study based evaluation shows positive accuracy results for the SAT-Analyser tool. Moreover, the proposed traceability management framework for the entire software development life cycle is presented.

CCS Concepts

• **Software and its engineering → Software creation and management → Software post-development issues → Software evolution**

Keywords

Traceability establishment; Visualization; Traceability graph, SAT-Analyser tool.

1. INTRODUCTION

Software system development is challenging due to the changes occur in requirements, business organizations, legal rules and improper use of tools and technologies. Managing these changes is difficult and affects the success or the failure of a software system. Thus, it is essential to have appropriate solutions to handle the changes during the Software Development Life Cycle (SDLC). The changes can occur at any phase to any intermediate software outcomes, which are called artefacts. An alteration to a single artefact can affect one or more other artefacts in one to many phases with different severities. Therefore, identification of the changes, affected artefacts, severity and the consequences is important to manage artefact traceability throughout the SDLC. Accordingly, the notion of software traceability has been evolved to enable tracing capabilities among software artefates.

Consequently, today different software traceability support tools and frameworks can be identified [1][2]. However, most solutions are research-based due to the challenging limitations.

Software Artefact Traceability Analyser (SAT-Analyser) tool described in this paper is one such software traceability support tool. It is capable of establishing traceability among software artefacts in requirement, design and source code level and to visualize the traceability graph for a given software application. Thus, this paper describes a Point of Sale (POS) based case study demonstrating the process, workflow of the SAT-Analyser and evaluates the accuracy of the traceability establishment process.

The paper is structured as follows. Section 2 explores set of related work and Section 3 describes the case study application. The accuracy of the tool is evaluated in Section 4 and Section 5 concludes the paper with possible future extensions.

2. BACKGROUND STUDY

Software traceability is the ability to track artefact behaviors during the software development process by providing a logical connection among artifacts. Software traceability process consists of several sub processes such as establishing traceability links among artefacts and traceability maintenance [3].

An architecture-centric, stakeholder-driven, industry-oriented and open hypermedia traceability approach influenced by e-Science technologies, is presented in [4]. It has addressed the multi-faceted traceability problem by integrating the implementation to a traceability tool named ArchStudio. They have followed a rule-based classification approach for establishing artefact-link relationships and n-ary first class links for trace relationships. A facet-based approach by Grammel [5], has narrowed the traceability scope into model-driven software development. They have traced the model transformations using a domain specific language (DSL) called trace-DSL for data extraction. A work on source code and test case artefacts traceability using gamification technologies is presented in [6], with a proof-of-concept prototype called GamiTraci. It is highly influenced by the similar previous work [7], that has used slicing and conceptual coupling techniques in establishing test to code traceability.

The accuracy of the traceability results is a major challenge. The reliability of the traceability is addressed in [8], that can be applied for safety critical software systems. It can be identified as a light-weight results-oriented solution. This trace link model separates the untrusted links and conducts a remediation process continuously. However, there is a limitation of trace maintenance facilities. Traceclipse [9], is a research-based Eclipse plugin targeted for traceability link recovery and maintenance. Its link recovery process is influenced by Information Retrieval (IR)

ICCBN 2018, February 24–26, 2018, Singapore, Singapore ,
© 2018 Association for Computing Machinery
ACM ISBN 978-1-4503-6360-0/18/02...$15.00
https://doi.org/10.1145/3193092.3193094

techniques and limited for source code artefact in Java. A similar work on semi-automated traceability recovery with the use of IR and classification is presented in [10]. An ontology-based attempt has been conducted in [11], by mapping domain concepts and artefact indexes into an ontology. But the traceability support of it is limited only for Unified Process based software development. Although, there exist limitations and challenges in achieving traceability within the SDLC, traceability management has been an active research area in modern software development [12][13].

Our previous work [14], has evaluated different traceability and consistency management techniques. We have proposed an extended framework for SAT-Analyzer to be compatible with traceability management and continuous integration for DevOps environments. Another research on managing traceability in self-adaptive systems is presented in [15], which is a generic toolkit with a interlink visualizer for inconsistency detection. Further, considering the software artefacts in later stages of software development with DevOps practices is discussed in [16] providing continuous integration capabilities by using Jenkins.

3. SAT-ANALYSER

3.1 Design Considerations

SAT-Analyser is a traceability management tool capable of tracing software requirement artefact, Unified Modelling Language (UML) class diagram artefact and the Java source code artefact [17][1][18]. Thus, it can be used for traceability in requirements, design and development stages of the SDLC. The system design is shown in Figure 1.

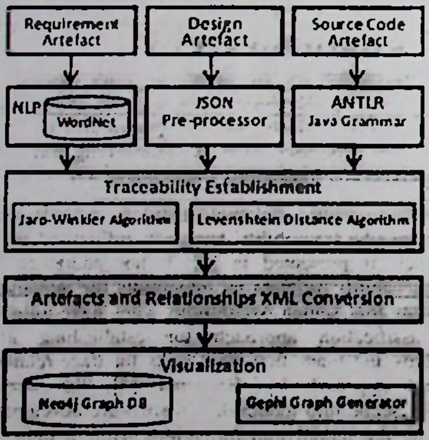

Figure 1. SAT-Analyser system design

SAT-Analyser tool considers three main artefacts; natural language based requirements, UML (Unified Modelling Language) class diagram as design artefact and Java source code for the implementation phase artefact. Initially, data pre-processing techniques are applied for all three types of artefacts, retrieve necessary information and transform them into a common format in XML. Then traceability links are established between the dependent artefacts and visualized the traceability relationships in a traceability graph.

3.2 Workflow: Point of Sale Application

A case study based evaluation is performed using the tool SAT-Analyser. The selected case study is a Point of Sale (POS) system, where a customer can place orders consist of items. An order can be either a special order having the online ordering feature or a normal order having only the cash on delivery facility. These requirements are stated in the software requirement specification in natural language. Figure 2, shows a section of the natural language requirements considered for this study.

The corresponding design in UML class diagram is shown in Figure 3. The main classes are identified as Customer, Order and Item. An Order is specialized into SpecialOrder and NormalOrder. Since the entity Order is composed of set of Item entities, there is an aggregation relationship. Similarly, the association between the entities, Customer class and Order class, is a composition relationship, which is a strong aggregation. Thus, if the Customer entity is deleted, then Order (part) entity is deleted as well.

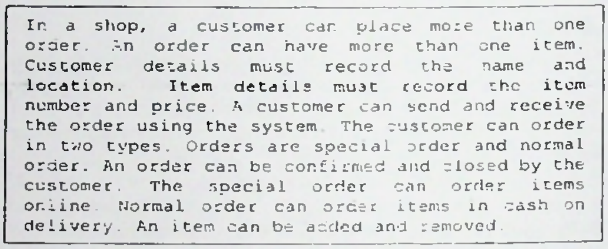
In a shop, a customer can place more than one order. An order can have more than one item. Customer details must record the name and location. Item details must record the item number and price. A customer can send and receive the order using the system. The customer can order in two types. Orders are special order and normal order. An order can be confirmed and closed by the customer. The special order can order items online. Normal order can order items in cash on delivery. An item can be added and removed.

Figure 2. POS requirements in natural language

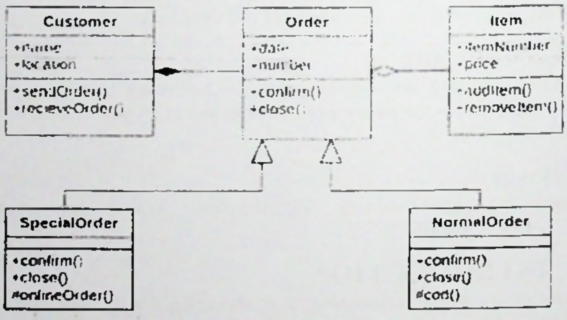

Figure 3. POS UML class diagram

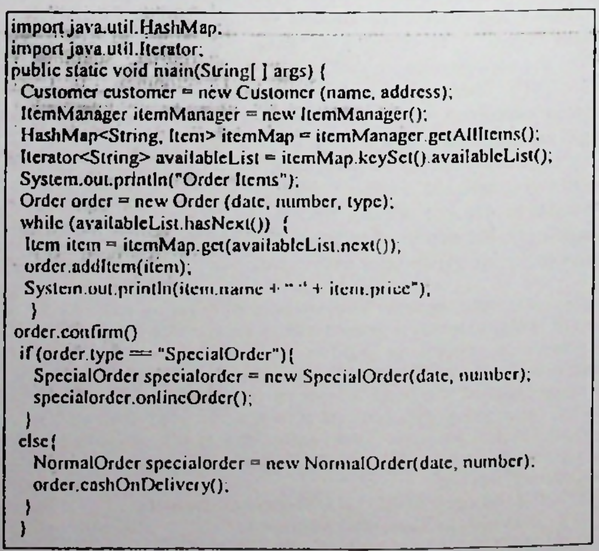
```
import java.util.HashMap.
import java.util.Iterator;
public static void main(String[ ] args) {
 Customer customer = new Customer (name, address);
 ItemManager itemManager = new ItemManager();
 HashMap<String, Item> itemMap = itemManager.getAllItems();
 Iterator<String> availableList = itemMap.keySet().availableList();
 System.out.println("Order Items");
 Order order = new Order (date, number, type);
 while (availableList.hasNext())  {
  Item item = itemMap.get(availableList.next());
  order.addItem(item);
  System.out.println(item.name + " " + item.price"),
   }
order.confirm()
 if (order.type == "SpecialOrder"){
   SpecialOrder specialorder = new SpecialOrder(date, number);
   specialorder.onlineOrder();
  }
 else{
   NormalOrder specialorder = new NormalOrder(date, number).
   order.cashOnDelivery();
  }
 }
```

Figure 4. POS source code

The relevant source code artefacts are given in Java programing language as a set of class files. A part of the main class of the Java source code of this POS system is shown in Figure 4. The corresponding classes for object creation and method calling are implemented separately, and considered as the code artefacts.

Requirements, design and development related artefacts are given as the inputs to the SAT-Analyser in their raw formats, namely requirements document in .txt, UML class diagram in either .mdj or .xmi format and the source code in one or more .java files. Then, SAT-Analyser performs the artefact extraction in the data pre-processing stage. Since the inputs are in three different formats, (1) requirements are processed using the Stanford Core NLP libraries [19] and WordNet lexical database; (2) design class diagram follows a JSON structure at the backend of its .mdj and .xmi formats; (3) source code is processed using the ANother Tool for Language Recognition (ANTLR) [20] Java 8 Grammar to identify the required artefact sub elements. The artefact elements include the requirements, classes, methods, attributes and the relationships inheritance, association and generalization. Next the extracted artefacts are listed and initiate the traceability establishment process. The traces are generated and mapped based on a string comparison as give in Algorithm 1.

Algorithm 1 Traceability link generation

```
Require: Software artefacts
Ensure: Building relationships among artefacts
input: artefacts a
for (a )
    get synonyms from WordNet
    String comparison for names of classes, attributes,
    methods and relationships
        matchDistance = Jaro Winkler algorithm
                        similarity (element1,element2)
        If (matchDistance > = 0.8 and < = 1.0)
            Build trace link among two artefact elements
        Else
            editDistance= Levenshtein Distance algorithm
                          distance (element1,element2)
            matchDistance = 1 - editDistance
            If (matchDistance > = 0.8 and < = 1.0)
                Build trace link among two artefact elements
    XML Writer (nodes, links)
output: XML conversion of artefact traceability links
        (Relations.xml)
```

Algorithm 1, handles the pre-processed artifact data towards the traceability link generation. It ensures the relationship building among the extracted artefact elements that are input for the algorithm. Then using the WordNet synonyms and pre-defined dictionary ontology, a string similarity computation is performed using the Jaro-Winkler algorithm [21] and Levenshtein Distance Algorithm [22]. Jaro-Winkler algorithm is selected prominently due to its efficiency than Levenshtein algorithm [23]. The former algorithm considers that, the differences near the start of the strings are more significant than differences close to the end of the strings, while Levenshtein algorithm computes the number of edits needed to convert one string to another. Fixed threshold values are associated for both algorithms and Levenshtein is used for deep comparison if the Jaro-Winkler similarity measure is not in the range of 0.8 and 1.0. Additionally, the WordNet synonym selection is done using the Levenshtein Distance algorithm with a threshold of 0.85.

Consequently, a similarity is marked if either threshold is met by triggering a relationship among those two artefact elements. Next, the artefacts and their established trace links are parsed through the Document Object Model (DOM) parser [24] and converted into a predefined XML structure. Figure 5 shows a section of the structure of the generated intermediate XML file for the UML class diagram artefact; Customer and Order class.

```
<?xml version="1.0" encoding="UTF-8"?>
<Artefacts>
 <Artefact type="UMLDiagram">
  <ArtefactElement id="D1" name="Customer" type="Class">
   <ArtefactSubElement id="D1_F1" name="name"
     type="UMLAttribute" variableType="" visibility="public"/>
   <ArtefactSubElement id="D1_F2" name="location"
     type="UMLAttribute" variableType="" visibility="public"/>
   <ArtefactSubElement id="D1_M1" name="sendOrder"
     parameters="" returnType="" status=""
     type="UMLOperation" visibility="public"/>
   <ArtefactSubElement id="D1_M2" name="recieveOrder"
     parameters="" returnType="" status=""
     type="UMLOperation" visibility="public"/>
  </ArtefactElement>
  <ArtefactElement id="D2" name="Order" type="Class">
   <ArtefactSubElement id="D2_F1" name="date"
type="UMLAttribute" variableType="" visibility="public"/>
   <ArtefactSubElement id="D2_F2" name="number"
type="UMLAttribute" variableType="" visibility="public"/>
   <ArtefactSubElement id="D2_M1" name="confirm"
parameters="" returnType="" status="" type="UMLOperation"
visibility="public"/>
   <ArtefactSubElement id="D2_M2" name="close" parameters=""
returnType="" status="" type="UMLOperation" visibility="public"/>
  </ArtefactElement>
```

Figure 5. UML Artefact XML file

Accordingly, the classes are considered as the major artefact elements and are given a unique id. The corresponding attributes and the methods are listed as the artefact sub elements for each artefact element with a unique identifier starting with the id of the parent artefact element. For an example, the customer is identified as a class name and the attributes of it are the name and location, while the methods are sendOrder and receiveOrder.

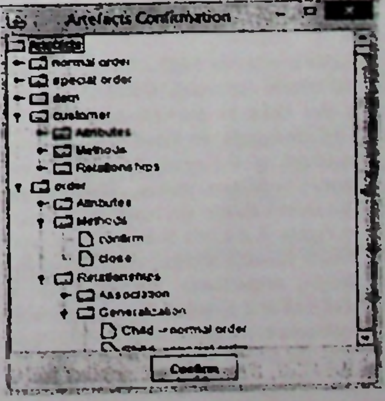

Figure 6. Artefact Extraction Confirmation Window

At the end of these backend data pre-processing, data extraction and traceability establishment, the results are presented in a tree structure by the artefact confirmation option of the tool as shown in Figure 6. The use of the DOM parser is benefitted, since it is capable of loading the full XML documents into a tree structure.

Hence, the user can alter, delete or add any misinterpreted artefact elements prior the confirmation.

The generated intermediate XML files would be modified accordingly and soon after the traceability project is created to the user. Afterwards, all these set of XML files are converted into an array format that follows a key-value pair structure using DOM parser and the Simple API for XML (SAX) parser's exception handling capabilities [25] to store in the Neo4j graph database [26]. Then the open graph visualization platform Gephi [27] is used for the graph generation using the nodes and links stored in Neo4j. Consequently, the SAT-Analyser visualizes the traceability links among artefacts or any selected artefact sub elements. The set of visualization filtering are as follow.

- Full graph view with artefacts and their links.
- Edge filtered view for the relationship among the identified classes, attributes, operations for each of the artefact in requirements, design and code.
- Artefact filtered views for each one of 3 artefacts separately.

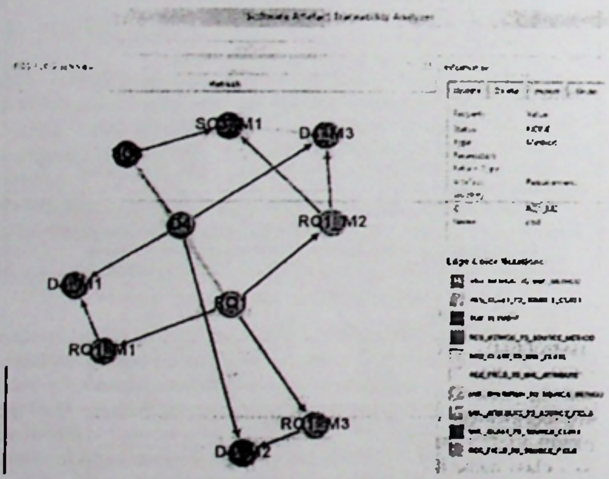

Figure 7. Full graph view of traceability in POS system

Figure 7 illustrates a selected section of the obtained full graph view of this POS system case study. Color codes are used for each type of nodes and links in the representation. Moreover, the details of each selected node are listed in the information section separately. The length of the edges denotes the strength of the similarity between each two nodes. Thus, larger the string comparison value means shorter the length of corresponding edge. For example, in Figure 7, the edit distance value among RQ1 and D4 is 0.916 which denotes normal order class in requirement artefact and design, respectively. Similarly, the value among RQ1_M2 and D4_M3 is 1.0, which represents cash on delivery method in requirements artefact and UML design artefact, respectively. Thus, the length of the edge between RQ1 and D4 is bit lengthy as the UML class diagram artefact has used the class name with naming conventions.

4. EVALUATION

The evaluation of the applied POS system is conducted using correctness measures based on the artefact, relationship extraction shown in Figure 6, since proper artefact and relationship identification is crucial towards the final traceability outcomes. Accordingly, the metrics precision and recall are applied as information retrieval accuracy measurements [28]. The artefact and relationship extraction results are evaluated as follows.

$$\text{Artefact, relationship extraction precision} = \frac{\text{number of correctly identified artefacts, relations}}{\text{total numberofidentified artefacts, relations}}$$

Similarly, the recall is measured as follows.

$$\text{Artefact, relationship extraction recall} = \frac{\text{number of correctly identified artefacts, relations}}{\text{total numberofactual artefacts, relations}}$$

Moreover, F-measure (F1 score), which is the weighted average of the obtained precision and recall; is derived as follows.

$$F1 = 2\,\frac{\text{prescion} \cdot \text{recall}}{\text{prescion} + \text{recall}}$$

Table 1. Evaluation of traceability support techniques

Traceability Establishment	Artefact	Precision	Recall	F-Measure
Requirement – Design	Class	1	0.8	0.8
	Attribute	1	0.5	0.6
	Method	1	0.5	0.6
Design – Source code	Class	1	1	1
	Attribute	1	0.3	0.4
	Method	0.8	0.5	0.6
Requirement – Code	Class	1	1	1
	Attribute	1	0.6	0.7
	Method	1	0.6	0.7

Traceability establishment accuracy among similar artefacts in different phases of the SDLC is shown in Table 1. The precision denotes positive results for the generated trace links, while the lower recall signifies that there exist missing links among attributes and methods. It is observed that the inaccurate artefact elements extraction and identification with NLP that contain different naming conventions and less meaningful names in requirement artefacts, have led to the lack of accuracy. However, the overall F-measures are biased towards 1 and requirement to code traceability has shown a high accuracy.

5. CONCLUSION

Software traceability is essential to ensure the proper synchronization among software artefacts during the software development process. There exist various software traceability related solutions; however most of them have certain limitations. SAT-Analyser tool presented in this paper is one such tool support software requirement, design and source code artefacts. This paper highlights the accuracy of the traceability establishment process of SAT-Analyser tool using a POS based application.

Requirement, design and development related artefacts in their raw formats are fed to the tool as text, UML class diagram file and Java source code files, respectively. SAT-Analyser pre-processed the input data and extracts the relevant artefact elements. The traceability links among the artefacts are established based on a similarity calculation algorithm. Moreover, the traceability relationships are visualized using traceability graphs for developer decision making. The tool allows manual artefact trace alterations and updates the graphs accordingly.

SAT-Analyser is evaluated using the accuracy measures precision, recall and F-measure based on the established traceability links among artefacts in the considered case study. Significant positive results have been obtained and identified possible improvements [illegible] data extraction [illegible] with [illegible] establishment algorithm. Furthermore, the integration of continuous-integration support for the tool-with DevOps principles would be an important future work to cope with the agile based software development environments.

6. ACKNOWLEDGMENTS

The author acknowledges the support received from the Senate Research Committee Grant SRC/LT/2016/07, University of Moratuwa, Sri Lanka in publishing this paper.

7. REFERENCES

[1] Kamalabalan, K. et al. 2015. Tool Support for Traceability of Software Artefacts. In *Proceedings of the Moratuwa Eng. Research Conf. (MERCon)*. (2015). IEEE. 318-323.

[2] Satish, C. J. et al. 2016. A Review of Tools for Traceability Management in Software Projects. *Int. Journal for research in emerging science and technology*. 3. 3 (2016), 6–10.

[3] Mader, P. and Gotel, O. 2012. Towards automated traceability maintenance. *Journal of Systems and Software*. 85. 10 (2012), 2205–2227.

[4] Hazeline U. Asun ion. 2008. Towards practical software traceability. In *Companion of the 30th international conference on Software engineering* (ICSE Companion '08). ACM, NY, USA. 1023-1026.

[5] Grammel, B. and Kastenholz, S. 2010. A generic traceability framework for facet-based traceability data extraction in model-driven software development. In *Proceedings of the 6th ECMFA Traceability Workshop* (ECMFA-TW '10). ACM, NY, USA, 7-14.

[6] Parizi, R. M. On the gamification of human-centric traceability tasks in software testing and coding. In *Proceedings of the 2016 IEEE 14th Int. Conf. on Software Eng. Research, Management and Applications* (SERA). IEEE, 193–200.

[7] Qusef, A. et al. 2011. SCOTCH: Test-to-code traceability using slicing and conceptual coupling. In *Proceedings of the 2011 27th IEEE Int. Conf. on Software Maintenance* (ICSM), IEEE, 63–72.

[8] Cleland-Huang, J. et al. 2014. Achieving lightweight trustworthy traceability. In *Proceedings of the 22nd ACM SIGSOFT Int. Symposium on Foundations of Software Eng.* (FSE 2014). ACM, NY, USA, 849-852.

[9] Klock, S. et al. 2011. Traceclipse: an eclipse plug-in for traceability link recovery and management. In *Proceeding of the 6th Int. workshop on Traceability in emerging forms of Software Eng.* (TEFSE '11). ACM, NY, USA, 24-30.

[10] Mills C. Automating traceability link recovery through classification. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Eng.* (ESEC/FSE 2017). ACM, NY, USA, 1068-1070.

[11] Noll, R. P. and Ribeiro, M. B. Enhancing traceability using ontologies. In *Proceedings of the 2007 ACM symposium on Applied computing* (SAC '07). ACM, NY, USA, 1496-1497. DOI=http://dx.doi.org/10.1145/1244002.1244322.

[12] Cleland-Huang, J. Traceability research: taking the next steps. In *Proceeding of the 6th Int. workshop on Traceability in emerging forms of Software Eng.* (TEFSE '11). ACM, NY, USA, 1-2.

[13] Poshyvanyk, D. et al. 6th Int. workshop on traceability in [illegible] forms of software engineering [illegible] *Proceedings of the 33rd International Conference on Software Engineering* (ICSE '11). ACM, NY, USA, 1214-1215.

[14] Rubasinghe, I. D. et al. 2017. Towards Traceability Management in Continuous Integration with SAT-Analyser. In *Proceedings of the 3rd Int. Conf. on Communication and Information Processing*. (2017). ACM. Tokyo.

[15] Perera, I. et al. 2015. A Traceability Management Framework for Artefacts in Self-Adaptive Systems. In *Proceedings of the 10th Int. Conf. on Industrial and Information Systems(ICIIS)*. (2015). IEEE. 37–42.

[16] Palihawadana, S. et al. 2017. Tool support for traceability management of software artefacts with DevOps practices. In *Proceedings of the Moratuwa Eng. Research Conf. (MERCon)*. (2017). IEEE. 129-134.

[17] Wijesinghe, D.B. et al. 2014. Establishing traceability links among software artefacts. In *Proceedings of the 14th Int. Conf. on Advances in ICT for Emerging Regions*. (2014). IEEE. 55–62.

[18] Arunthavanathan, A. et al. 2016. Support for traceability management of software artefacts using Natural Language Processing. In *Proceedings of the 2nd Int. Moratuwa Eng. Research Conf. (MERCon)*. (2016). IEEE. 18–23.

[19] Manning, C. D. et al. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Baltimore, Maryland. 55–60.

[20] ANTLR: *http://www.antlr.org/*. Accessed: 2017-07-21.

[21] JaroWinklerDistance (LingPipe API): *http://alias-i.com/lingpipe/docs/api/com/aliasi/spell/JaroWinklerDistance.html*. Accessed: 2017-08-14.

[22] Efficient Implementation of the Levenshtein-Algorithm, Fault-tolerant Search Technology, Error-tolerant Search Technologies: *http://www.levenshtein.net/*. Accessed: 2017-10-14.

[23] Christen P. 2006. A Comparison of Personal Name Matching: Techniques and Practical Issues. In *Proceedings of the IEEE Sixth Data Mining Workshop* (ICDM '06). IEEE, Hong Kong, China.

[24] Le Hors, A. et al. 2004. *Document Object Model (DOM) Level 3 Core Specification*. W3C Technical Report. Massachusetts Institute of Technology, Cambridge, MA.

[25] Parser: *http://www.saxproject.org/apidoc/org/xml/sax/Parser.html*. Accessed: 2017-10-15.

[26] Graph Visualization for Neo4j: Tools, Methods and More: *https://neo4j.com/developer/guide-data-visualization/*. Accessed: 2017-07-23.

[27] Gephi - The Open Graph Viz Platform: *https://gephi.org/*. Accessed: 2017-10-14.

[28] Zeugmann T. et al. 2011. Precision and Recall. In *Encyclopedia of Machine Learning*. Springer US, Boston, MA, 781–781.