TRACEABILITY MANAGEMENT IN A DEVOPS ENVIRONMENT WITH CONTINUOUS INTEGRATION

Iresha Dilhani Rubasinghe

(178020N)

Degree of Master of Philosophy

Department of Computer Science and Engineering

University Of Moratuwa
Sri Lanka

April 2019

TRACEABILITY MANAGEMENT IN A DEVOPS ENVIRONMENT WITH CONTINUOUS INTEGRATION

Iresha Dilhani Rubasinghe

(178020N)

Degree of Master of Philosophy

Department of Computer Science and Engineering

University Of Moratuwa
Sri Lanka

April 2019

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: ………………………… Date: ……………....

I. D. Rubasinghe

The above candidate has carried out research for the MPhil thesis under my supervision.

Name of the supervisor: Dr. D. A. Meedeniya

Signature of the supervisor: ……………………….. Date: ……………....

Name of the supervisor: Dr. G. I. U. S. Perera

Signature of the supervisor: ……………………….. Date: ……………....

Abstract

Software artefacts traceability is an important factor during the process of software development to analyse changes occur in software components. Traceability improves the quality attributes of software systems such that strengthens the testability, maintainability, reusability and helps for the system acceptance by providing consistent system documentation to the users. Meanwhile, the concept DevOps motivates towards the reduction of the gap between development and operations requiring considerable organizational changes. In a DevOps environment, significant software artefact changes are expectable rapidly where continuous integration is essential. Continuous integration is a cornerstone practice in DevOps that frequently merges developer working copies into a single shared branch. There is a requirement of determining and analysing the resulted impact of the traceability in order to make accurate change acceptance decisions during software development. Therefore, the core research problem addressed is determining a methodology for change detection and impact analysis together with software artefact synchronization to preserve consistency across all artefacts in a DevOps environment. A rule-based methodology is followed with visualization and analysis techniques applied on a proof-of-work traceability management prototype tool: SAT-Analyser 2.0. The evaluation results and industry-level user study results have shown the significant usefulness and suitability of the approach to a DevOps environment as well as to any software development process model.

Keywords:

Consistency management, Continuous integration, DevOps, Change impact analysis, Traceability management

Acknowledgements

I would like to offer my heartfelt gratitude to supervisors, Dr. D. A. Meedeniya and Dr. G. U. I. S. Perera for the immense guidance provided throughout the research project. Their valuable feedback and extremely kind advices motivated me to complete the research work as well as moulded my academic competencies. I ever appreciate their knowledge sharing and flexibility that made the research work an interesting and pleasant experience that would not have been possible without them.

I would also like to thank my project advisory panel members; Prof. N. D. Kodikara at University of Colombo School of Computing, Dr. A. C. De Silva at Department of Electronic and Telecommunication Engineering, University of Moratuwa, Dr. M. J. Walpola and Dr. K. Gunasekara at Department of Computer Science and Engineering, University of Moratuwa for the thoughtful comments and suggestions that helped me to achieve my research goals.

I highly appreciate the financial support received from Senate Research Council Grant, University of Moratuwa to complete this work.

I am thankful to my family and colleagues at the Department of Computer Science and Engineering, University of Moratuwa for the unfailing support.

Table of Contents

Declaration....i
Acknowledgements....iii
Table of Contents....iv
List of Figures....vii
List of Tables....xi
List of Appendices....xii
List of Abbreviations....xiii
Section 1....1
1 Introduction....1
1.1 Overview of the research....1
1.2 Motivation for the research....2
1.3 Problem statement....3
1.4 Research statement....4
1.5 Research objectives....4
1.6 Research outcome....5
Section 2....6
2 Literature review....6
2.1 Overview....6
2.1.1 Software artefacts in SDLC....6
2.1.2 Traceability....6
2.1.3 Software artefact traceability in a DevOps environment....9
2.2 Data pre-processing....10
2.3 Information retrieval methods....11
2.4 Traceability management....11
2.4.1 Evaluation of traceability support techniques....11
2.5 Change detection....13
2.5.1 Change detection techniques....14
2.6 Change impact analysis....15
2.6.1 Change impact analysis of heterogeneous software artefacts....16
2.6.2 Change impact analysis categories....17
2.6.3 Change impact estimation and analysis techniques....19
2.6.4 Change impact analysis related frameworks and models....20
2.7 Consistency checking and management....25
2.8 Change propagation in DevOps....26
2.8.1 Change propagation techniques....26
2.9 Continuous integration....27
2.9.1 DevOps practices....28
2.9.2 DevOps tools....30
2.9.3 DevOps related project management tools....32

2.10 Analysis of related work ... 34
2.11 Visualization of traceability links ... 36
2.12 Tool support for tractability management and continuous integration ... 38
2.13 Evaluation techniques of traceability management ... 41
2.13.1 Quality measures ... 41
2.13.2 Network analysis ... 43
2.13.3 Traceability testing techniques ... 44
2.13.4 Supported testing tools ... 44
2.14 Discussion ... 45
2.14.1 Limitations in current practices ... 46
2.14.2 Future challenges and research directions ... 46
Section 3 ... 47
3 Research methodology ... 47
3.1 System design ... 47
3.1.1 System overview ... 47
3.1.2 Research model ... 48
3.1.3 System architecture ... 52
3.1.4 Abstract system workflow ... 54
3.1.5 Detailed system workflow ... 55
3.1.6 System class structure ... 56
3.2 Traceability establishment ... 57
3.2.1 Data pre-processing of SAT-Analyser tool ... 57
3.2.2 Input to XML conversion ... 62
3.2.3 Traceability generation ... 63
3.3 Traceability visualization ... 66
3.3.1 Default SAT-Analyser informative traceability visualization ... 68
3.3.2 Analytical traceability visualization ... 71
3.3.3 Interactive traceability visualization ... 72
3.4 Impact analysis and change propagation ... 73
3.4.1 Identification of strengths of artefacts and relationships ... 73
3.4.2 Impact analysis process: workflow ... 78
3.4.3 Impact analysis process: pseudo code and implementation details ... 78
3.4.4 Impact analysis process: user modifiability ... 81
3.4.5 Change propagation of the impact ... 82
3.4.6 Notification approach ... 89
3.5 Traceability management ... 90
3.5.1 Change detection ... 91
3.5.2 Consistency management ... 98
3.5.3 Continuous integration ... 101
3.5.4 Multi-user supportability ... 109
3.6 Extended SAT-Analyser evaluation ... 110
3.6.1 Implementation of the accuracy analysis module ... 110
3.6.2 Implementation of the network analysis module ... 112
3.7 Tool performance analysis ... 113
3.8 Conclusion ... 115
Section 4 ... 116

4 Evaluation....116
4.1 Datasets and materials....116
4.1.1 Pre-defined categorization of change types....118
4.1.2 Evaluation environment specification....119
4.2 Experimental results: case study 1 (POS system)....119
4.2.1 Evaluation of traceability establishment component....121
4.2.2 Evaluation of continuous integration process....123
4.2.3 Performance analysis....125
4.3 Experimental results: case study 2 (Tour management system)....127
4.3.1 Evaluation of traceability establishment component....129
4.3.2 Evaluation of continuous integration process....130
4.3.3 Performance analysis....132
4.4 Experimental results: case study 3 (MyDrive multimedia library)....133
4.4.1 Evaluation of traceability establishment component....135
4.4.2 Evaluation of continuous integration process....136
4.4.3 Performance analysis....137
4.5 Experimental results: case study 4 (Disease management system)....139
4.5.1 Evaluation of traceability establishment component....141
4.5.2 Evaluation of continuous integration process....142
4.5.3 Performance analysis....144
4.6 Experimental results: case study 5 (E-School management system)....145
4.6.1 Evaluation of traceability establishment component....147
4.6.2 Evaluation of continuous integration process....148
4.6.3 Performance analysis....149
4.7 SAT-Analyser performance analysis....151
4.7.1 Traceability establishment performance....151
4.7.2 Accuracy evaluation of change detection component....152
4.7.3 Accuracy evaluation of impact analysis component....152
4.8 Usability of the extended SAT-Analyser tool....153
Section 5....156
5 Discussion....156
5.1 Feature analysis of SAT-Analyser in practice....156
5.2 Analysis of the usability study based evaluation....160
5.3 Mapping of the objectives and the methodology....161
5.4 Limitations....161
5.5 Future work....162
5.6 Conclusion....163
References....165

List of Figures

Figure 2-1 : Summary of traceability classification 9
Figure 2-2 : The software evolution process (Sommerville, 2010) 13
Figure 2-3 : Change impact analysis process (Li, Sun, Leung, & Zhang, 2013) 15
Figure 2-4 : Change impact analysis categorization 19
Figure 2-5 : Chianti tool architecture (Ren et al., 2005) 21
Figure 2-6 : Continuous integration process (Farcic, 2016) 27
Figure 2-7 : DevOps overview ("QASource DevOps Experts," 2018) 29
Figure 2-8 : Jenkins workflow 30
Figure 2-9 : Docker workflow 31
Figure 3-1: Extended SAT-Analyser system overview 47
Figure 3-2 : SAT-Analyser tool research model 49
Figure 3-3 : Extended-SAT-Analyser system architecture 52
Figure 3-4 : Extended SAT-Analyser abstract workflow 54
Figure 3-5 : Extended SAT-Analyser detailed workflow 55
Figure 3-6 : Extended SAT-Analyser class structure 56
Figure 3-7 : Semantic network for words 64
Figure 3-8 : Pre-defined relationship XML model 65
Figure 3-9 : Traceability link generation component 66
Figure 3-10 : Visualization component 68
Figure 3-11 : Default SAT-Analyser Traceability visualization menu 69
Figure 3-12 : Default SAT-Analyser visualization full graph view 70
Figure 3-13 : Default SAT-Analyser visualization requirement artefact filtered view 70
Figure 3-14 : General analytical traceability graph 72
Figure 3-15 : Analytical traceability graph in traceability validation 72
Figure 3-16 : Interactive traceability graph view 73
Figure 3-17 : Node-edge direct connectivity 75
Figure 3-18 : Node-edge scenario 1 76
Figure 3-19 : Node-edge scenario 2 77
Figure 3-20 : Impact analysis component workflow 78
Figure 3-21 : Code snippet of weight and influential factor calculators 81
Figure 3-22 : SAT-Analyser generated CIA results 82
Figure 3-23 : User altered CIA results 82

Figure 3-24 : Change propagation workflow ... 83
Figure 3-25 : Code snippet of change propagation implementation ... 86
Figure 3-26 : Impact analysis results window ... 87
Figure 3-27 : Change propagated analytical traceability graph view ... 88
Figure 3-28 : Change propagated interactive traceability graph view ... 88
Figure 3-29 : Trello change propagation card instance ... 90
Figure 3-30 : Trello board with change propagation notification ... 90
Figure 3-31 : Design diagram: change detection component ... 91
Figure 3-32 : Diffmk based change types declaration ... 96
Figure 3-33 : Change detection menu item ... 96
Figure 3-34 : Insufficient CI versions for change detection ... 97
Figure 3-35 : Change detection results xml_CD directory ... 97
Figure 3-36 : Change detection results in outcome window ... 97
Figure 3-37 : Consistency management workflow ... 99
Figure 3-38 : Scheduler workflow ... 102
Figure 3-39 : Scheduler code snippet ... 104
Figure 3-40 : Continuous integration configuration menu item ... 105
Figure 3-41 : Continuous integration configuration window ... 105
Figure 3-42 : Continuous integration menu item ... 106
Figure 3-43 : Continuous artefact integration window ... 106
Figure 3-44 : Continuous artefact integration window with disabled forward option ... 106
Figure 3-45 : Code snippet of version control management ... 108
Figure 3-46 : CI version 1 of a project ... 108
Figure 3-47 : CI version 2 of a project ... 108
Figure 3-48 : Multi-user accessible SAT-Analyser web version ... 109
Figure 3-49 : SAT-Analyser traceability establishment accuracy analysis window ... 110
Figure 3-50 : SAT-Analyser CIA accuracy analysis window ... 111
Figure 3-51 : Network analysis centrality measures code snippet ... 112
Figure 3-52 : SAT-Analyser network analysis main window ... 113
Figure 3-53 : Network analysis artefact information view ... 113
Figure 3-54 : SAT-Analyser traceability establishment performance analysis window ... 114
Figure 3-55 : SAT-Analyser CIA performance analysis outcome window ... 114
Figure 4-1 : Project scale ... 118

Figure 4-2 : POS system description 120
Figure 4-3 : POS system design diagram 120
Figure 4-4 : SAT-Analyser main artefact summary for POS system 120
Figure 4-5 : POS system Relations.xml instance 121
Figure 4-6 : Part of the traceability visualization graph - POS system 122
Figure 4-7 : Network analysis summary - POS system 122
Figure 4-8 : POS system change detection window 123
Figure 4-9 : POS system impact analysis window 124
Figure 4-10 : POS system change propagation instance 124
Figure 4-11 : CIA statistical analysis results: POS system 125
Figure 4-12 : CIA performance analysis results: POS system 126
Figure 4-13 : Tour management system description 127
Figure 4-14 : Tour management system design diagram 127
Figure 4-15 : SAT-Analyser main artefact summary for tour management system 128
Figure 4-16 : Tour management system Relations.xml instance 129
Figure 4-17 : Traceability visualization - tour management system 129
Figure 4-18 : Network analysis summary - tour management system 130
Figure 4-19 : Tour management system change detection window 131
Figure 4-20 : Tour management system impact analysis window 131
Figure 4-21 : Tour management system change propagation instance 131
Figure 4-22 : CIA statistical analysis results: tour management system 132
Figure 4-23 : CIA performance analysis results: tour management system 133
Figure 4-24 : MyDrive multimedia library system description 133
Figure 4-25 : MyDrive multimedia library system design diagram 134
Figure 4-26 : SAT-Analyser artefact summary for MyDrive multimedia library system 134
Figure 4-27 : Multimedia library system Relations.xml instance 135
Figure 4-28 : Traceability visualization - multimedia library system 135
Figure 4-29 : Network analysis summary - multimedia library system 135
Figure 4-30 : Multimedia library system change detection window 136
Figure 4-31 : Multimedia library system impact analysis window 137
Figure 4-32 : Multimedia library system change propagation instance 137
Figure 4-33 : CIA statistical analysis results: multimedia library system 138
Figure 4-34 : CIA performance analysis results: multimedia library system 138

Figure 4-35 : Disease management system description 139
Figure 4-36 : Disease management system design diagram 140
Figure 4-37 : SAT-Analyser main artefact summary for disease management system 140
Figure 4-38 : Disease management system Relations.xml instance 141
Figure 4-39 : Traceability visualization - disease management system 141
Figure 4-40 : Network analysis summary - disease management system 142
Figure 4-41 : Disease management system change detection window 143
Figure 4-42 : Disease management system impact analysis window 143
Figure 4-43 : Disease management system change propagation instance 143
Figure 4-44 : CIA statistical analysis results - disease management system 144
Figure 4-45 : CIA performance analysis results: disease management system 145
Figure 4-46 : E-School management system description 145
Figure 4-47 : E-School management system design diagram 146
Figure 4-48 : SAT-Analyser main artefact summary for E-School management system 146
Figure 4-49 : Relations XML format of traceability establishment - E-School system 147
Figure 4-50 : Traceability visualization – E-School management system 147
Figure 4-51 : Network analysis summary – E-School management system 148
Figure 4-52 : E-School management system change detection window 148
Figure 4-53 : E-School management system impact analysis window 149
Figure 4-54 : E-School management system change propagation instance 149
Figure 4-55 : CIA statistical analysis results: E-School management system 150
Figure 4-56 : CIA performance analysis results: E-School management system 150
Figure 4-57 : SAT-Analyser traceability establishment performance 151
Figure 4-58 : SUS positive responses analysis 154
Figure 4-59 : SUS negative responses analysis 155
Figure 4-60 : SAT-Analyser usability tag cloud 155
Figure 5-1 : Research objectives-methodology-results mapping 161

List of Tables

Table 2.1 : Evaluation of software artefacts traceability management techniques 12
Table 2.2 : Summary of change impact analysis techniques 20
Table 2.3 : Change impact analysis related work summary 23
Table 2.4 : Scope-based change impact analysis related work 25
Table 2.5 : Evaluation of related work on traceability management 35
Table 2.6 : Traceability visualization techniques 36
Table 2.7 : Evaluation of related work on traceability visualization techniques 37
Table 2.8 : Tool support for traceability management 38
Table 2.9 : Tool support for information retrieval 39
Table 2.10 : Tool support for traceability management 40
Table 3.1 : Analysis of existed SAT-Analyser 50
Table 3.2 : Analysis of the SAT-Analyser with DevOps extension 51
Table 4.1 : Dataset summary 116
Table 4.2 : Artefact categorization: POS system 121
Table 4.3 : Artefact categorization: tour management system 128
Table 4.4 : Artefact categorization: MyDrive multimedia library system 134
Table 4.5 : Artefact categorization: disease management system 140
Table 4.6 : Artefact categorization: E-School management system 146
Table 4.7 : Change detection component accuracy evaluation 152
Table 4.8 : Change impact analysis component accuracy evaluation 152
Table 5.1 : Comparison between Jenkins and SAT-Analyser for CI 158
Table 5.2 : Industry level traceability management vs. SAT-Analyser tool 158
Table 5.3 : Existing traceability management tools vs. SAT-Analyser 159

List of Appendices

Appendix A: Initial survey..173
Appendix B: User acceptance survey 178
Appendix C: Research tool configuration settings 180
Appendix D: SAT-Analyser 2.0 user guide overview 181
Appendix E: List of companies involved in the surveys/ interviews..........................186
Appendix F: Published papers 187

List of Abbreviations

AST	Abstract Syntax Tree
AIS	Actual Impact Set
ANTLR	ANother Tool for Language Recognition
AO	Aspect-Oriented
AR	Augmented Reality
AWS	Amazon Web Services
CIA	Change Impact Analysis
CD	Continuous Delivery
CI	Continuous Integration
CIP	Change Impact Prediction
CR	Change Request
DAG	Directed Acyclic Graph
DSL	Domain Specific Language
EDG	Entity Dependency Graph
EIS	Estimated Impact Set
GCT	Goal-Centric Traceability
GPS	Global Positioning System
HMC	Hidden Markov Chain
IR	Information Retrieval
IDE	Integrated Development Environment
IoT	Internet Of Things
IT	Information Technology
IQR	Interquartile Range
LSI	Latent Semantic Indexing
LTR	Likelihood To Recommend
MDD	Model Driven Development
MDE	Model Driven Engineering
ML	Machine Learning
NER	Named Entity Recognizer
NLP	Natural Language Processing
NPS	Net Promoter Score
PCA	Principle Component Analysis

PM	Project Management
POS	Part-Of-Speech, Point Of Sales
RCM	Requirement Change Management
RSSI	Really Simple Syndication
ROI	Return On Investment
SCM	Supply-Chain Management
SMS	Short Message Service
SVD	Singular Value Decomposition
SIG	Soft Goal Interdependency Graph
SDLC	Software Development Life Cycle
SRS	Software Requirement Specification
SUS	System Usability Scale
TF-IDF	Term Frequency–Inverse Document Frequency
TDD	Test Driven Development
UAT	User Acceptance Testing
UML	Unified Modeling Language
VM	Virtual Machine
VSM	Vector Space Model

Section 1
Introduction

1.1 Overview of the research

Software artefacts are the intermediate by-products used in each stage of the Software Development Life Cycle (SDLC) towards the successful outcome of the intended software product. That includes Software Requirement Specification (SRS), design diagrams, non-functional design reports, source code (Sommerville, 2010). Additionally, test cases, test scripts, walkthroughs, inspections, bug reports, build logs, configuration files, project plans, risk assessments and user manuals are important artefacts in the latter stages of the SDLC. Nevertheless, there is a relationship between the primary artefacts involved during the SDLC with the final deliverables of a software product. Thus, software artefacts play an important role in fine-tuning the software products.

The artefact management is essential to maintain adequate consistency in approaching towards a software product. Software artefact traceability has been defined as the ability to follow the life cycle of a particular software requirement both forward and backward and overcome the inconsistencies during software development (Cleland-Huang, Zisman, & Gotel, 2012). Thus, each alteration occurs in a particular artefact is traced among other artefacts and changed accordingly based on the impact. The relationship links among artefacts must be updated and maintained consistently.

Software traceability is required to handle changes during the process of Continuous Integration (CI). CI is known as a software development practice where the work is integrated frequently leading to multiple integrations per day (Duvall, Matyas, & Glover, 2007). The integration verification is done using build automation by detecting integration errors as early as possible. The proper application of CI can reduce integration problems and allows developing cohesive software rapidly.

The concept of Development-Operations (DevOps) represents the integration of development environment and the operational environment when developing software systems with continuous planning, CI, continuous delivery and continuous testing (Bass, Weber, & Zhu, 2015)(Kim, Debois, Willis, Humble, & Allspaw, 2016)(Ghantous & Gill, 2017). DevOps ease the project management with communication, understandability, integration and relationships among the development teams and operational teams by bridging the gap between them. It increases the rate of change and deploys features into efficient development.

1.2 Motivation for the research

Software systems, in today's context, are considered as critical business assets. A software system change is inevitable and hence must be updated continuously in order to maintain the value of these assets. Software evolution is preferred over building completely new software systems due to the cost and time benefits (Rajlich, 2014). Often, software evolution occurs in a software system life cycle at a stage where it is in active operation due to new requirements. Software evolution mainly depends on the type of software being maintained and cooperated development processes which continue the software system lifecycle. It is highly coupled with the components that are affected due to changes which allow the cost and impact of changes to be estimated (Pete & Balasubramaniam, 2015).

The improper or outdated software artefacts and their inconsistencies result in misleading the intermediate software system development processes due to the high coupling among artefacts. Hence, software development and maintenance become time-consuming with many issues such as higher cost and effort. Moreover, the proper artefact management is essential in integrating artefacts continuously. The changes must be accurately propagated in the integrations which is challenging to be automated. Thus, the auditability and traceability are classified as challenges in DevOps.

Therefore, changes in software artefacts cause software evolution (Rajlich, 2014). With the rapid generation of information, it is crucial to maintain the consistency between software artefacts. Well-Defined traceability management between

software artefacts is required to overcome the consequences of evolutions. Further, improper traceability management may lead to failures of a product. Thus, traceability management strengthens the software maintainability and helps for system acceptance (Cleland-Huang et al., 2012).

1.3 Problem statement

Among the existing traceability establishment systems, a prototype study 'Software Artefacts Traceability Analyser' (SAT-Analyser*)* (Kamalabalan et al., 2015)(Arunthavanathan et al., 2016) is selected to address the impact analysis during traceability in a DevOps environment with continuous integration. The existed traceability establishment system has addressed the traceability among textual requirements artefact, Unified Modeling Language (UML) class diagrams and Java source code artefact. Then, the traceability among them is established based on the attributes, methods using Natural Language Processing (NLP) and the results are visualized in a traceability graph.

The existing prototype lacks impact analysis, immediate change propagation capabilities and support towards the continuous integration in DevOps environments. Thus, it requires a mechanism to evaluate the impact of an artefact change prior to the change propagation and consistency management in remaining phases such as testing, configuration, deployment and maintenance. We have addressed these limitations with the integration of appropriate DevOps tools.

Accordingly, our study addresses the software artefact traceability for all the SDLC phases without expensive overheads. We have mainly focused on the notion of CI in software development with DevOps principles. Our methodology mainly consists of traceability establishment, change detection, impact analysis, change propagation, consistency management and visualization (Rubasinghe, Meedeniya, & Perera, 2017). Therefore, the traceability support for the entire SDLC is addressed in this research with automated tool support.

1.4 Research statement

Core Research Question: How to enable software artefact traceability management in a way the changes made to an artefact at any point of the development lifecycle will preserve consistency across all artefacts in a DevOps environment?

Research hypothesis: Current trend of emergent and changing requirements for software systems can be better supported by:

- Identifying the true links between heterogeneous artefacts in SDLC to establish traceability and
- Applying trace links, impact analysis for changes and synchronization by updating trace links to propagate artefact changes with impacts through all the phases in a DevOps environment.

1.5 Research objectives

The main limitation in the existing context of software traceability and continuous integration is the lack of sufficient tools and techniques. The current tools are limited to certain types of software artefacts and development environments depending on the used programming languages or the design notations. Therefore, the automation of traceability relations generation has become unachievable completely. The existing tools and techniques are identified to be containing strong semantic meanings and thus fail in satisfying requirements needed for system analysis in heterogeneous software artefacts. Further, the support of traceability and continuous integration is important to be available during the overall SDLC which is not completely preserved in current practices.

The prime objective of this research is to extend the initial SAT-Analyser tool as proof-of-work to integrate with the latter phases of the SDLC in terms of traceability management and continuous integration adhering to DevOps practices as a complete solution.

The objectives of this research are to:

- Identify, establish and maintain traceability links between all stages of SLDC
- Detect the changes in trace links between software artefacts in a DevOps environment
- Analyse the impact caused by the changes in software artefacts that interfere traceability
- Accurately determine and visualize the consequences of a change with impacts in software artefacts
- Enhance and visualize the traceability in a DevOps environment with continuous integration

1.6 Research outcome

The successful completion of this research would enable software artefact traceability support for all the key stages of SDLC such as requirement analysis, design, development, testing, configuration, deployment and operation. The intermediate software development process can be traced both backward and forward completely in a cycle of SDLC. Moreover, the successful DevOps tool integration into the traceability would facilitate continuous integration capabilities. Accordingly, the extended SAT-Analyser tool would be capable of applying into DevOps environments where the rapid changes, integrations and deployments are vital.

Therefore, the outcome of this research work is expected to be able to increase the efficiency of the software process by tracing the changes in artefacts effectively. It would contribute to making the developers' workload easier in order to proceed with software projects using the automated change tracing and impact analysing by maintaining an easy flow in software delivery pipeline.

Section 2
Literature review

2.1 Overview

This chapter summarizes important aspects of the research scope related to traceability management in a DevOps environment. The existing research solutions and state-of-the-practice in software traceability management with their background definitions are discussed. Mainly, an overview of software artefact traceability, change detection, impact analysis, consistency management and continuous integration are analytically discussed together with the existing related works on traceability techniques and tools.

2.1.1 Software artefacts in SDLC

A software system is a combination of several software artefacts that evolves through a certain software development process model. Software artefacts refer to the intermediate by-products used in different phases of the SDLC. Some of the artefacts can be named such as SRS, design diagrams, architectural documents and quality attributes or the non-functional design reports, source code, test scripts, walkthroughs, inspections, bug reports, build logs, test reports, project plans and risk assessments (Sommerville, 2010). There are relationships and dependencies between these software artefacts and it is essential to manage these software artefacts in order to maintain adequate consistency towards a software product. The improper management and outdated artefacts can lead to inconsistency among artefacts, synchronization issues and lack of trust in artefacts by stakeholders. Therefore, the software artefact traceability is essential for being capable of describing and following the artefact life cycle.

2.1.2 Traceability

Traceability facilitates a logical layer across artefacts throughout the various phases in software development. At the beginning of a software development process, the ability to track the consistency in requirements along with their sources is essential in order to confirm or revise the initial set of requirements (Sommerville, 2010). Thus, traceability is first used as a method of managing

requirements artefact during the Requirements Engineering phase. Generally, traceability is following the life cycle of any particular software requirement both forward and backwards to overcome the inconsistencies during software development (Cleland-Huang et al., 2012). Each alteration occurs in a given requirement is traced among other requirements and changed based on the impact. These traces are used in the requirement validation and verification processes.

The software artefact traceability definition of the professional body; 'Center of Excellence for Software and Systems Traceability (CoEST)' is declared as "*the ability to interrelate any uniquely identifiable Software Engineering artefact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process*". They have not been limited to requirement traceability and have declared traceability in terms of other artefact types including design documents, codes and test case files with the deployment of an experimental traceability environment for researchers called *TraceLab* (Keenan et al., 2012).

Mohan et al. (Mohan, Xu, Cao, & Ramesh, 2008) have defined traceability as the ability to discover the dependent entities within a software model and trace their corresponding artefact elements in other software models. As a result, currently, traceability is used not only in requirements management, but also for other artefact types in different software development methodologies like Model-Driven Development (MDD) (Sommerville, 2010). This wide range of adaptation of traceability shows its importance in improving software quality, maintenance, evolution and reuse activities.

For a given trace, there can be one or many possible trace paths, while each trace path has a source and target artefacts. In particular, an artefact may be a source for a given trace path and a target for another trace path simultaneously. A *trace link* or known by a *traceability link* is a relationship between a pair of artefacts. All such links generated in between two groups of software artefacts are referred as a *trace relation* (Cleland-Huang et al., 2012). A *trace set* is the sum of all generated traces and *traceability graph* is used to visualize all the relationships. A

traceability graph is a traceability network when the edges are directional or the nodes are embedded with a weight. Further, *traceability maintenance* is the consistency management of artefacts and trace updates for a given change.

Some of the categorizations based on the different dimensions of traceability exist such as requirement to design, requirements to code base and to test case files likewise. Among different traceability types, *requirement traceability* addresses the dependencies between requirements and among the requirements to design/ source codes. It can be subcategorized as *pre-requirements* and *post-requirements* specification that details the life cycle of a software requirement in forward and backward directions. *Design traceability* is the ability to trace design and requirements to design rationale for the verification and maintenance of architecture design accurately (Tang, Jin, & Han, 2007). Having the ability to trace design traceability can be useful to determine trace design evolution, root causes, to relate architectural design objects and also to analyse the cross-cutting concerns, especially in a DevOps environment.

Moreover, the different traceability classifications in the literature are shown in Figure 2-1. One such classification is automatic or manual, based on the automation level of the traceability process. Another classification is forward or backwards, based on the direction of the traceability path (Cleland-Huang et al., 2012). Forward tracing follows subsequent steps such that from requirements to code; whereas backward tracing follows antecedent steps such that code to design or requirements artefacts. Artefact-level is another criterion that classifies traceability as horizontal or vertical. Horizontal tracing considers homogeneous artefacts as such artefacts in the equivalent levels of abstraction like tracing between different versions of requirement artefacts (Mäder, Gotel, Kuschke, & Philippow, 2008). Further, this can be sub-classified based on the direction such that horizontal forward tracing or horizontal backward tracing. Tracing artefacts in different levels of abstraction; heterogeneous artefacts, such as the requirement to code is considered as vertical tracing, which can be either vertical forward tracing or vertical backward tracing. Proactive and reactive tracing is another categorization based on the stimuli behaviour. In reactive tracing, the traces are

created on demand in accordance to a stimulus for capturing traces. Whereas in proactive tracing, traces are generated in the background without explicit response to any stimulus (Cleland-Huang et al., 2012). The traceability link generation techniques for these categories are selected by considering the aspects such as the problem domain and the behaviour of the software system.

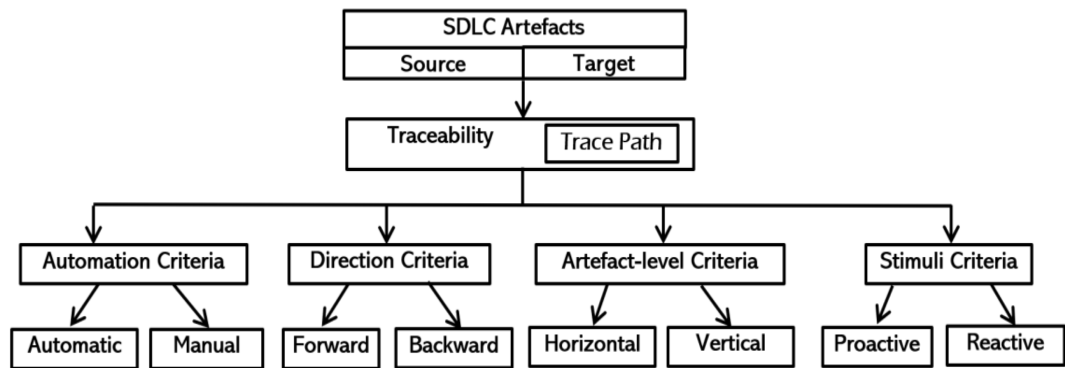


Figure 2-1 : Summary of traceability classification

A major challenge in tracing software artefacts is the heterogeneity in software artefacts, different abstraction levels and lack of defined data formats for software artefacts (Wijesinghe et al., 2014). Therefore, it is essential to identify the key artefact elements from a given artefact input in order to establish relationships.

2.1.3 Software artefact traceability in a DevOps environment

The concept of DevOps represents the integration of the development environment and the operational environment that encourages developing systems rather than software. Primarily DevOps ease the project team management with communication, understandability, integration and relationships among the development teams and operational teams by breaking the gap between them. It increases the rate of change and deploys features into production faster (Kim et al., 2016)(Ghantous & Gill, 2017). The demanding drivers for having DevOps can be identified as improving the quality of applications, enhancing customer experience, the ability for simultaneous deployment in different platforms and the need to reduce Information Technology (IT) costs. The technical benefits of a DevOps environment include identification of the problem earlier, reducing the error fixing time durations and reduction of the problem complexities due to its

cross-functionality behaviour. Similarly, the importance towards the business aspect is significant as DevOps shorten the development life cycle, increase the release velocity and improve the Return On Investment (ROI) by achieving a higher customer satisfaction (Bass et al., 2015). Furthermore, rich collaboration and performance-oriented culture encourage the ability to research and innovate within projects. However, the Internet of Things (IoT) and microservices architecture are identified to be challenging for DevOps.

In the software development process, DevOps is applicable to various phases of software delivery such as continuous planning, continuous integration, continuous delivery and continuous testing (Bass et al., 2015). Consequently, the ability to trace the changes made to the code is essential in providing feedback at any integration failures. Therefore, the artefact traceability is a key challenge in achieving CI. Furthermore, the need for techniques and tools to recover traceability links in legacy systems is important for a variety of software evolution tasks. These include maintenance tasks, impact analysis, program comprehension and encompassing tasks such as systematic reuse of traceability types and Reverse Engineering for redevelopment (Cleland-Huang et al., 2012).

2.2 Data pre-processing

Software artefacts consist of different formats such as the requirements in natural language, design artefacts in different UML notations and source code artefacts in programming languages. Thus, pre-processing and extracting the required data is an initial task towards the development of traceability links. The textual contents in artefacts provide descriptive details about their informal semantics. The frequently involved pre-processing steps for artefacts in requirements are the NLP activities such as tokenization, text normalization, anaphora analysis, morphological analysis and stemming (Cleland-Huang et al., 2012)(Arunthavanathan et al., 2016). It is assumed that if the textual contents of artefacts are similar, then those artefacts are conceptually related in resulting establishment of traceability links between them. The other types of artefacts can pre-process with different file readers, UML parsers and programming language specific parsers.

2.3 Information retrieval methods

Information Retrieval (IR) methods enable extracting and analysing the embodied textual contents in artefacts with less pre-processing effort. The cost of traceability link recovery can be minimized since no predefined vocabulary or grammar is involved (Cleland-Huang et al., 2012). The use of meaningful identifiers and comments in the source code of documentation can be found helpful in applying IR methods. The key steps in a generalized IR process that follows a pipelined architecture can be listed as; (1) document parsing, extraction and pre-processing, (2) corpus indexing with an IR method, (3) ranked list generation and (4) analysis of candidate links. Vector Space Model (VSM), Term Frequency-Inverse Document Frequency (TF-IDF) metric and Latent Semantic Indexing (LSI) techniques are the mostly used IR techniques (Y. Zhang, Wan, & Jin, 2016).

2.4 Traceability management

The cost of managing a larger number of artefact relationships whenever a change occurs is identified as a major reason for rarely using 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, ensuring the correctness of traceability over time is essential in traceability maintenance/ management and is a multi-step activity (Mäder & Gotel, 2012)(Maro, Anjorin, Wohlrab, & Steghöfer, 2016). The proper identification of a feasible traceability management approach could minimize the cost and effort during the SDLC.

2.4.1 Evaluation of traceability support techniques

Table 2.1 summarizes the features of traceability management techniques with a description, benefits and limitations.

Table 2.1 : Evaluation of software artefacts traceability management techniques

Technique	Functionalities	Methods/ techniques followed	Advantages	Limitations
Rule-based	Define rules in traceability links generation.	Rule set based on attributes of artefacts. Traceability management with rule re-evaluation (Mäder & Gotel, 2012).	Ideal for artefacts such as requirements, use cases and analysis of object models.	Weakness in recognition of structural changes (Cleland-Huang et al., 2012).
Hypertext-based	Support traceability maintenance.	XML. Markup specifications (Alves-Foss, Conte de Leon, & Oman, 2002).	Consider requirements and code artefacts (Cleland-Huang et al., 2012).	Weekly support for other types of artefacts.
Event-based	Automate trace link generation and maintenance.	Publish-subscribe relationship mechanism. Event-based subscriptions (Galvão & Goknil, 2007).	Ability to maintain dynamic links.	Scalability issues in maintaining the dynamicity of traceability.
Constraint-based	Support traceability maintenance.	Set of constraints are provided that should not get disobeyed by traceability links.	Artefact types can be viewed as constraints on one another (Fockel, Holtmann, & Meyer, 2012).	Difficulty in referencing all traceability links to constraints (Fockel et al., 2012).
Transformation-based	Support traceability maintenance.	Incremental transformation approaches. Graph- transformation based methodologies.	Beneficial for model-based software systems (Riebisch, Bode, Farooq, & Lehnert, 2011).	Not all artefacts are generated by model transformations (Maro et al., 2016).
Goal-Centric (GCT)	CIA over the non-functional software requirements.	Soft goal Interdependency Graph (SIG). Traceability matrix (Galvão & Goknil, 2007).	Finds the impact of functional changes over non-functional ones to ensure quality.	Lack of scalability and tool support (Galvão & Goknil, 2007).
Model-driven	Support traceability maintenance in MDD	Use of template-based models (Javed & Zdun, 2014).	Support for different artefact types including source code (Javed & Zdun, 2014).	Lack of support towards non-model-driven systems (Javed & Zdun, 2014).
Probabilistic model	Manage traceability with uncertainty handling.	Bayes' theorem ("Vector Space Model," 2017).	Simplicity and ability to evolve with data science methods.	Depend on probabilistic assumptions such as the artefacts are distributed differently.

The rule-based and hypertext-based traceability support techniques are identified to be not applicable to all types of software artefacts rather than requirements and source code (Mäder & Gotel, 2012)(Cleland-Huang et al., 2012). Event-based and constraint-based methodologies along with publish-subscribe mechanisms have been widely involved in traceability maintenance while scalability is the main problem in them (Galvão & Goknil, 2007)(Fockel et al., 2012). The transformational and model-driven approaches can be identified as more environment-oriented such as for model-based software systems. Thus, it can be difficult to obtain a more generic traceability solution via them (Javed & Zdun, 2014). Moreover, the majority of IR related techniques such as VSM, LSI and TF-IDF are involved due to their better performance outcomes (Hayes et al., 2007)(Marcus, Xie, & Poshyvanyk, 2005). However, there is a lack of tool-support for majority techniques with the compatibility for all types of software artefacts. The scalability is the main issue that has been a limitation in most related works following existing techniques having the inability to cater to traceability management among a larger number of software artefacts.

2.5 Change detection

Change is always inevitable in any software development process. It is necessary to cope with the changes properly to reduce the cost regardless of the used software development model (Sommerville, 2010). Figure 2-2 illustrates the software evolution process. The impact of a change is assessed prior further propagating the change. The evolving software systems potentially support for dynamic modifications and extensions.

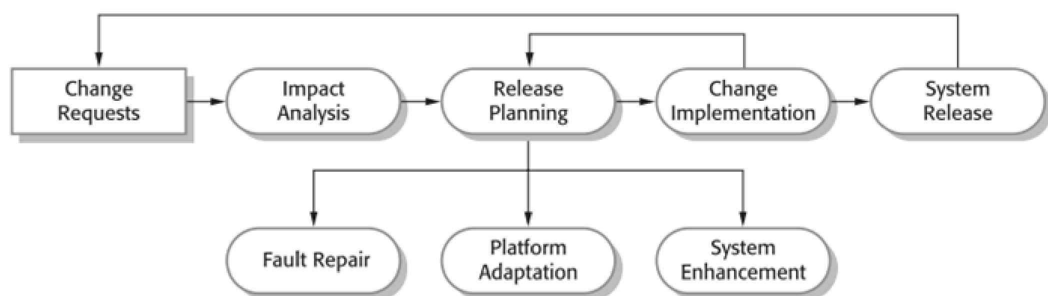


Figure 2-2 : The software evolution process (Sommerville, 2010)

2.5.1 Change detection techniques

A. Edit history

This approach keeps track of the alterations or the edits as a history. 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. The 'Undo/ Redo' and 'Restore' operators in most of the tools have used this technique. Accordingly, this has been used for traceability related change detections in the context of software artefacts as well (Omori & Maruyama, 2008)(Kitsu, Omori, & Maruyama, 2013). However, this technique is mostly used for change detection regarding the source code.

B. Tree differencing

Tree differencing represents elements as Abstract Syntax Tree (AST) and calculates the differences to extract detailed change information. AST is a tree representation of the abstract syntactic structure of source code, while each node represents a construct occurring in the code (Sager, Bernstein, Pinzger, & Kiefer, 2006). Even in an older related work in (Chawathe, Rajaraman, Garcia-Molina, & Widom, 1996), has used the idea of a matching and a minimum cost edit script that transforms one tree to another for hierarchically structured data. The authors have split the change detection problem such as '*Good Matching*' and '*Minimum Conforming Edit Script*'. However, the data format should be in a tree format for that algorithm and not in other formats such as graphs.

C. Differencing algorithms

The customized differencing algorithm is another technique for software artefact related change detection. It is used in software maintenance aspects such as program-profile estimation (stale profile propagation). Any type of software artefact can be generally taken as the input for a differencing algorithm though the source code artefact is heavily gone through this technique in related works (Apiwattanapong, Orso, & Harrold, 2004). However, implementing 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 and category of software artefacts.

2.6 Change impact analysis

The goal of Change Impact Analysis (CIA) in software development is detecting the consequences of an artefact alteration in other parts of the software system (Sommerville, 2010)(Lehnert, 2011). Traceability is a major supportive technique in the identification of affected artefacts and is a key notion in the software maintenance process. In areas such as Model-Driven Engineering (MDE), before changing a metamodel, it is crucial to measure the impact of the changes among the artefacts to understand whether the evolution is sustainable or not.

Figure 2-3 illustrates the iterative process of CIA in software development. It starts with an analysis of a change request in source code to initially identify the set of changes in which the artefacts could be affected. It is also called as *concept location* or *feature location* with the meaning of finding a place in the source code that an initial change needs to be made. Then, the change impact analysis is conducted 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 modified. The AIS is not considered to be unique for a particular change request as a change can be implemented in several ways.

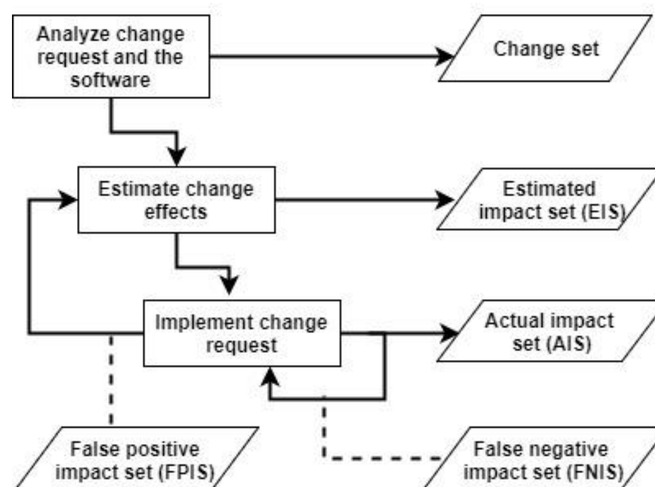


Figure 2-3 : Change impact analysis process (Li, Sun, Leung, & Zhang, 2013)

Generally, impact analysis is conducted before or/and after a change implementation. The advantages of conducting impact analysis prior to a change can be listed as for 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 (Li et al., 2013).

2.6.1 Change impact analysis of heterogeneous software artefacts

The heterogeneous software artefacts in different stages of SDLC are always following different types and formats. For instance, the requirement artefact can be in a natural language provided in a text file while the source code artefact in Python programming language as a set of *.py* files. Therefore, a change occurred in one artefact does not directly reflect for other artefacts due to that type and format mismatches. Due to this complexity, generally, an artefact change comes as a request for change without a direct action of alteration. Currently, a responsible resource person is in charge for coordinating change requests either by accepting the change or declining the change request depending on his/ her manual change interpretations which are subjected to human error.

The artefact traceability acts as the main pillar for artefact change management. The inter-relationships and intra-relationships are established via the traceability establishment process by linking each other based on the dependencies. Therefore, the changes can be handled using those traces and paths. The traceability visualization techniques such as traceability matrix or traceability graphs that follow node edge connectivity can be used effectively. The graph theory techniques and algorithms are involved in finding the change impacts among traces (W.-T. Lee, Deng, Lee, & Lee, 2010). Moreover, the use of IR techniques and Machine Learning (ML) are trending in the research level to manage artefact changes with CIA (Zimmermann, Zeller, Weissgerber, & Diehl, 2005)(Dantas, Murta, & Werner, 2007).

In considering the graph-based traceability results, the artefact changes can be mapped to the nodes and the change propagation would be mapped via the

connected links. However, all the endpoints of the links may not be subjected to changes. Thus, the impact of the propagated change at the linked endpoints has to be measured in finding the actual victims of the change. A CIA algorithm has to be implemented depending on the addressed artefact types in calculating the occurred impacts (Tóth, Hegedűs, Beszédes, Gyimóthy, & Jász, 2010). The features of the initial change would be highly influenced in the impact calculation since the linked endpoints have to be compared with the initial change.

Accordingly, the proper identification of the initial change is essential. The IR techniques are associated to this aspect in the related works (W. Wang, He, Li, Zhu, & Liu, 2018)(Y. Zhang et al., 2016). Thus, based on the artefact type of the initial change the required data such as the scope of the change and the keywords in the change has to be identified. Then, the CIA algorithm must imply the consequences of the change in calculating the impact values in comparison with linked endpoint nodes. The probabilistic theorems such as association rules, Bayes theorem and Change History are widely involved in the literature (Lehnert, 2011)(Mens, Buckley, Zenger, & Rashid, 2005). However, addressing the change ripple effects becomes a challenge after the initial impacted endpoint identification since the changes can propagate continuously from those endpoints too.

2.6.2 Change impact analysis categories

Change impact analysis methods are categorized as *traceability-based CIA* and *dependence-based* in determining the change effects in the literature (Li et al., 2013). 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. It is relatively more biased towards analysing program syntax relations and in performing CIA of artefacts in the same level of abstraction such as in the level of software design or within the level of code. The higher level 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 of determining change impacts of the final software product with improved precision as directly analysing the implementation details.

Another categorization of CIA techniques is *static impact analysis* and *dynamic impact analysis*. The static CIA techniques encounter all behaviours and inputs (Sun, Li, Tao, Wen, & Zhang, 2010). Thus, contains a cost of precision though safe. Moreover, static CIA techniques analyse the program code syntax and semantic dependencies to construct intermediate representations using call graphs and program dependence graphs. Then, perform CIA on those representations resulting larger impact sets that are problematic to use in practice. Thus, lower precision remains a main drawback in the static impact analysis techniques. Besides, dynamic CIA methods overcome this disadvantage by considering only a partial set of the inputs. Hence, these impact sets are more precise although lack of safety. Furthermore, the impact sets computation process in dynamic impact analysis techniques depend on the types of analysis of the gathered data such as execution traces details, execution relation and coverage related information.

Two of the sub-techniques in dynamic CIA are *Coverage Impact* and *Path Impact* (Apiwattanapong, Orso, & Harrold, 2005). Path Impact computes impact sets in the method level using compressed program execution traces. It processes forward and backward traces to determine the impact of changes. The forward traces determine all methods called after the changed method(s), while the backward traces identify methods into which the execution can return. The coverage impact technique uses the coverage information to identify the executions that traverse a minimum of one method in the changeset and marks the covered methods in each execution. Next, it computes a static forward trace from each change by considering the marked methods. Thus, the methods in computed traces become the impact set. Moreover, it is identified that the path impact technique is more precise compared to the coverage impact technique analytically due to the use of traces rather than the coverage (Apiwattanapong et al., 2005).

However, in comparison, the time and space overhead in the path impact technique is high. The time consumption in path impact technique is dependent on the size of the analysed traces, while the coverage impact requires 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 proportional to the size of the traces in the path impact technique.

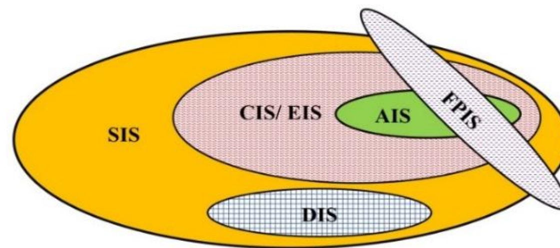


Figure 2-4 : Change impact analysis categorization

Figure 2-4 shows the categorization of change impact sets. Starting Impact Set (SIS) indicates the initially affected set of entities by a change. Candidate or Estimated Impact Set (CIS or EIS) is a subset of SIS, that denotes the potentially impact entities. However, that subset may or may not be the actual change impacted subset, which has to be clarified where AIS would be the outcome. The CIS tends to coincides with the AIS. Due to the challenging effect of artefact type mismatches, developer mistakes and artefact naming inconsistencies there may be artefacts that are actually impacted by the change, but have not been identified by CIS. Those are categorized as Discovered Impact Set (DIS). The manual or a knowledge-based identification can be useful in that aspect (Czibula, Czibula, Miholca, & Marian, 2017). False Positive Impact Set (FPIS) shows the artefacts that are overestimated as belong to the CIS, but which are not actually impacted.

2.6.3 Change impact estimation and analysis techniques

Table 2.2 presents s summary of techniques that can be applied for CIA. Among the techniques, the *call graphs*, *dependence graphs* are widely used in handling the changes that enable the backtracking ability for easier debugging. Most of the artefact types including design, source code and test cases are influenced by these call graphs related techniques and IR based: LSI, VSM and TF-IDF techniques. The *formal semantics*, *first-order logic* have mostly addressed the requirement artefacts. However, most of these techniques are semi-automated.

Table 2.2 : Summary of change impact analysis techniques

Category	Technique	Description
Statistical analysis	Data flow analysis, relational language, program slicing, static call graphs (Oliva, Gerosa, Milojicic, & Smith, 2013)(Maule, Emmerich, & Rosenblum, 2008)	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 (Kama, 2013)(Li et al., 2013)(Galbo, 2010)(De Lucia, Oliveto, & Tortora, 2008)(Y. Zhang et al., 2016)(Kabeer, Nayebi, Ruhe, Carlson, & Chew, 2017)(Déhoulé, Badri, & Badri, 2017)(Kchaou, Bouassida, & Ben-Abdallah, 2017)(Czibula et al., 2017)(Shahid & Ibrahim, 2016)(Borg, Wnuk, Regnell, & Runeson, 2017)	Results certify that existing algorithms require enhancements and effective mechanisms to facilitate automated tools for 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 (Sharafat & Tahvildari, 2007)	Maintenance of object-oriented mission critical systems is addressed. Limited for object-oriented software.
	Call graphs, Entity Dependency Graph (EDG) (Oliva et al., 2013)(Ibrahim, Idris, Munro, & Deraman, 2005)(Ibrahim, Munro, & Deraman, 2005)(Yiheng Wang, Zhang, & Fu, 2017)(Kchaou et al., 2017)(Ren, Ryder, Stoerzer, & Tip, 2005)	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 (Goknil, Kurtev, van den Berg, & Spijkerman, 2014). Logical dependencies and classification criteria (Lehnert, 2011)(Duarte, Duarte, & Thiry, 2016)(M. Lee & Offutt, 2002)(Rempel & Mader, 2017)	Removal of false positive impacts and consistency checking. Adds valuable information. Restricted for particular change and relation types.
	Rule-based (Lehnert, 2015)(Yiheng Wang et al., 2017)(Lehnert, Farooq, & Riebisch, 2013)	Allows developers to smoothly retrace the changes.
	Data mining, Apriori algorithm (Zimmermann et al., 2005)	Useful in change predictions.
History-based	Historical co-change analysis, change history (Sharafat & Tahvildari, 2007)	Use version histories to identify logical/ evolutionary couplings between entities. Predict impact files after a change.
	ML (Mills, 2017)(Czibula et al., 2017)(W. Wang et al., 2018)	Classification models to predict the validity of the candidate links. Use unsupervised learning to identify hidden dependencies. Less human involvement
	Logical coupling (Wong, Cai, & Dalton, 2011)	Use logical coupling with a Markov model. Better accuracy.

2.6.4 Change impact analysis related frameworks and models

Figure 2-5 shows an architecture of a Java source code impact analysis tool called '*Chianti*' which is a plugin in Eclipse Integrated Development Environment (IDE) (Ren et al., 2005). There are three main submodules in this tool. Initially, derive atomic code changes from pair of Java source code versions which is done via pairwise AST comparisons. Another module reads the test *call graphs* for original source code and edits code snippets. Also, it computes affecting code changes and

affected tests. Besides, the third module visualizes the obtained change impact details to the user. Thus, this plugin model is mainly based on the call graphs and does not involve calculations for each impact in a quantitative value.

Figure 2-5 : Chianti tool architecture (Ren et al., 2005)

Arnold and Bohner present another impact analysis model in traceability (De Lucia, Fasano, & Oliveto, 2008). They consider the changes occurred in documentation and source codes in the identification of SIS. The *dependency graphs* are used to identify the CIS using direct impacts and reachability graph representations for indirect type of impacts. They have suggested to apply this incrementally for identifying CIS to minimize the false positive rate.

Acharya and Robinson (Acharya & Robinson, 2011) present a static CIA framework that is developed as a tool named '*Imp'*. The scope has been the source code artefact impact analysis during frequent builds. Being a mathematical model, this work has used forward slicing consisting of three criteria namely; (1) range, (2) dependences and (3) summary edges to calculate the impact sets. Moreover, the *Andersen's algorithm* along with *pointer analysis* is applied. The algorithm of this framework consists of two variations one for *high setting impact analysis* which is expensive and another for *low setting impact analysis* which can be performed more frequently faster with a lower cost.

Another static CIA technique is presented in (Jashki, Zafarani, & Bagheri, 2008) that generates clusters of associated code files according to the *co-modification history records* in the source code repository. It has decreased the complexity to

accelerate the CIA with the aid of *dimensionality reduction* techniques. Initially, it mines the changed repository to find co-occurring source files and develops a matrix containing the degree of closeness in each pair of files. Then, it has performed an *intrinsic dimensionality* method based on *Eigenvalues* for estimation on that matrix and gets a low dimensional matrix. Further, Principle Component Analysis (PCA) is used for reduction. Finally, the matrix rows are taken as coordinates of files and distance between each pair of files is measured and passed to five different clustering methods. It identifies the clusters of associated files from source code modules and creates the impact sets. However, any quantitative measure is not adapted in measuring the severity of the impacts in this model.

Table 2.3 summarizes some related work with their methodology, advantages and drawbacks of each. However, 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.

Table 2.4 summarizes related work on CIA according to their scope. Many studies have based on estimating impact among homogeneous artefacts in the same level such as either on requirements, UML designs or source code artefacts. Among them, the majority of the studies have addressed requirement and source code artefacts. The Java programming language or the object-oriented aspects are the considered programming category in them. The *call graphs*, *dependence graphs* and *treemaps* are mainly used in requirements artefact, UML designs and source code artefacts while minor has involved data mining algorithms such as *Apriori algorithm* in source code artefact. A few have addressed the impact analysis between heterogeneous artefacts including requirements to test cases artefact.

Accordingly, one of the major limitations is being restricted to one or two types of homogeneous artefacts mostly requirements or source code. The work that has addressed heterogeneous artefacts are also limited only up to test cases artefact without considering remaining stages artefacts such as build scripts, configuration files and user manuals. Also, the visualization aspects are stated as future works to be addressed in some of these existing related works.

Table 2.3 : Change impact analysis related work summary

Reference	Addressed scope	Description	Advantages	Limitations
(Lehnert, 2015)	Address heterogeneous software artefacts from different development stages.	Based on a set of predefined CIA propagation rules. Heterogeneous artefacts are mapped on a common meta-model, dependencies are extracted as traceability links. A set of impact propagation rules recursively executed to compute the impact. Implemented as the prototype tool *EMFTrace*.	Forecast the impacts prior to implementation and address a multitude of different change operations. Maintain the consistency in architecture and the code of the test system.	Requirement artefact is not included.
(Y. Zhang et al., 2016)	Automatic recovery of requirement to code trace links.	A tool; *R2C* is implemented which concatenates features to recovery links in requirements to source code. *WordNet* is used to find synonyms of terms. Part-Of-Speech (POS) tagging, parsing, extracting verb object phrases and stemming applied. Comments are also used in the tracing process. Compare the text similarities based on IR techniques: VSM, TF-IDF.	Traceability link recovery is addressed for requirement-to-code artefacts.	Only requirement-to-code traceability links recovery is considered. Tool fails to recover all links. CIA is not considered.
(Duarte et al., 2016)	A body of knowledge on traceability is build named *TraceBoK*.	Requirements are classified based on the target domain. Available as a web-based open source on internet to access.	The transferring of the findings of researches on traceability from academia to the software industry is achieved via this knowledge body.	Limit for requirements. CIA methods are not discussed. Not a straightforward impact analysis tool.
(Goknil, Kurtev, & Berg, 2016)	CIA between requirements and architecture.	Formal semantics in requirements relations used. Implemented as an extension for existing tool called *TRIC*.	Provide precise CIA in software architecture which is able to mitigate false positives.	Only the requirements artefact is considered.
(Rodrigues, Lencastre, & Filho, 2016)	A user interactive tool for traceability visualization.	Used visualizing techniques: Sunburst and tree in radial layout, graphs, matrix and hierarchical.	Evaluate traceability allowing domain independent data. Provide various visualization options via single tool.	Only the requirements artefact is considered. Impact analysis is not addressed.
(Shahid & Ibrahim, 2016)	Prototype tool, *HYCAT* to support CIA.	First traceability matrix is generated between requirements and test cases. CIA integrates both types; static impact analysis and dynamic analysis together.	Results have shown high accuracy and efficiency.	Only a case study evaluation is performed on the tool application.
(Yiheng Wang et al., 2017)	A rule-based CIA method for software lifecycle objects is designed.	5 types of entity dependency and changes were defined and the corresponding change propagation rules were designed. CIA is based on change propagation rules. Explain the concept of two dependency states, (1) persistent relationship state and (2) immediate relationship state in change propagation.	Experiments have shown the effectiveness of the introduced algorithm.	No GUI support.

(Kabeer et al., 2017)	Evaluates the applicability of textual similarity techniques for CIA following Bag of Words with topic modelling and file coupling.	Finds the impact of textual similarity on altered files. A corpus is created using the summary of the change requests mined via *Jira*. Cosine similarity is applied to get the textual similarity between documents. TF-IDF is used to express Change Requests (CR) in the vector space. Used Leave-One-Out Cross Validation (LOOCV) to obtain model performance.	The effort in CIA for can be minimised by extending its applicability to many dimensions such that to impacted files and duration.	Existing CIA techniques are involved. No straightforward tool with a GUI.
(Déhoulé et al., 2017)	CIA model addresses AspectJ programs.	Change Impact Model for Java (CIMJ) is involved, 41 impact rules defined relevant to AspectJ programming rules as well. Three change impact categories were identified: (1) object code impacts on AspectJ code, (2) AspectJ code on AspectJ code and (3) AspectJ code on object code. Used precision and recall to evaluate.	Allows better support for cascading impact analysis. Evaluation shows higher accuracy.	Limited for source code artefact in Aspect-Oriented Programming (AOP) language.
(Kchaou et al., 2017)	Impact analysis in UML class and sequence diagrams.	Uses structural and semantic dependencies within and inter-UML diagrams. Uses graphs to map structural dependencies. IR techniques; TF-IDF and LSI used for semantic traceability.	Have gained precision of 84% and a recall of 91% in the requirements CIA and management.	Limited to design artefact in UML notation.
(W. Wang et al., 2018)	An approach to combine multiple existing IR techniques to facilitate CIA.	Approach integrates a bag-of-words based IR technique and a neural network based IR technique to derive couplings from the code. Extract all identifiers, comments and other artefacts from the code and generates a corpus. Transform the corpus and change request into their corresponding matrix and vector forms by IR techniques. Use LSI and doc2vec. Employ a learning paradigm to generate a similarity metric.	Results provide statistically significant improvements in accuracy across several cut points. A new method is introduced for measuring the similarity between source code and change request based on a learning paradigm in overcoming drawbacks associated with IR techniques.	Only source code artefact is considered.

Table 2.4 : Scope-based change impact analysis related work

Reference	Artefact level				
	Requirements	***Design***	***Source code***	***Testing***	***Other***
(Maule et al., 2008)		X			
(Spijkerman, 2010)	X				
(Oliva et al., 2013)			X		
(Li et al., 2013)			X		
(W.-T. Lee et al., 2010)	X				
(Phetmanee & Suwannasart, 2014)				X	
(Goknil et al., 2014)	X				
(Lehnert, 2015)		X	X	X	
(Y. Zhang et al., 2016)	X	X	X		
(Duarte et al., 2016)	X				
(Goknil et al., 2016)	X	X			
(Rodrigues et al., 2016)	X				
(Shahid & Ibrahim, 2016)	X			X	
(Yiheng Wang et al., 2017)					X
(Kabeer et al., 2017)	X				
(Déhoulé et al., 2017)			X		
(Borg et al., 2017)					X
(Galbo, 2010)					X
(Kchaou et al., 2017)		X			
(Mills, 2017)					X
(Czibula et al., 2017)					X
(Rempel & Mader, 2017)	X				
(W. Wang et al., 2018)			X		
(Wong et al., 2011)			X		
(Lehnert et al., 2013)		X	X	X	
(Sharafat & Tahvildari, 2007)		X	X		
(Zimmermann et al., 2005)			X		
(Tóth et al., 2010)					X
(Dantas et al., 2007)		X			

2.7 Consistency checking and management

In software development, different artefacts process through various phases of the SDLC. The changes and refinements that occur in artefacts do not happen at 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 solutions. That can lead to stakeholder dissatisfaction and system failure. The

consistency management in software domain is defined as the capability to preserve the synchronization among artefacts along with the occurring changes (Pete & Balasubramaniam, 2015). Accordingly, an artefact alteration or the presence of outdated artefacts should consistently reflect on all the other affected artefacts before they are used in the software process.

2.8 Change propagation in DevOps

The change propagation is conducted after the sequence of activities; change detection and change impact analysis to monitor ripple effects and for the selection of test cases respectively (Li et al., 2013). When new alterations are made, it is essential to confirm that remaining software artefact elements in a system are synchronized and consistent.

Thus, change propagation is the new changes necessary in a software system in order to validate the consistency of assumptions in the system after an artefact has been changed. This is mostly conducted during the incremental software changes. Firstly, CIA is done to predict the change effects before checking whether they need modifications. The tool '*JTracker*' is popular for assisting change propagation along with CIA. When a programmer changes a class, it marks the potentially impacted neighbouring classes. The propagation is terminated if the changes of neighbouring classes are not necessary. Furthermore, '*JRipples*' is another significant tool for change propagation throughout the incremental artefact changes (Lehnert, 2011)(Li et al., 2013)(Rajlich, 2014).

2.8.1 Change propagation techniques

A. Heuristic rules

Heuristic rules use to aggregate the detected changes to propagate and to obtain an optimal solution with high performance (Cleland-Huang et al., 2012)(Cleland-Huang, Gotel, Hayes, Mäder, & Zisman, 2014). Hence, it is essential to determine the best path for propagating any change in the context of software artefact traceability. There exist specific aggregation algorithms which have been based on the heuristic rules and the related work in (Kitsu et al., 2013) has discussed one for source code change propagation.

B. Distance-based

In the distance-based option, the temporal distance and spatial distance are mainly involved. The time taken among changes and the location distance among two modifications are considered in propagating a change. The use of ASTs and other representations are involved in determining these distances (Kitsu et al., 2013).

2.9 Continuous integration

Continuous integration is the repetitive integration process of building software implementations and testing them during specifically an Agile software development process. It elaborates frequent merging of the sole components of a software system into a shared branch by preserving the healthiness of the code. The importance of CI is significant in reducing most of the risks in software development such as lack of deployable software, late discovery of defects and lower project visibility (Duvall et al., 2007)(Meyer, 2014)(Kim et al., 2016). The automation of the CI process has given significant importance in the literature. In CI, the working code is committed to the version control repositories by developers. And make build scripts on those frequently pushed code in the CI servers to integrate new changes to the software. Figure 2-6 illustrates the CI workflow conceptually.

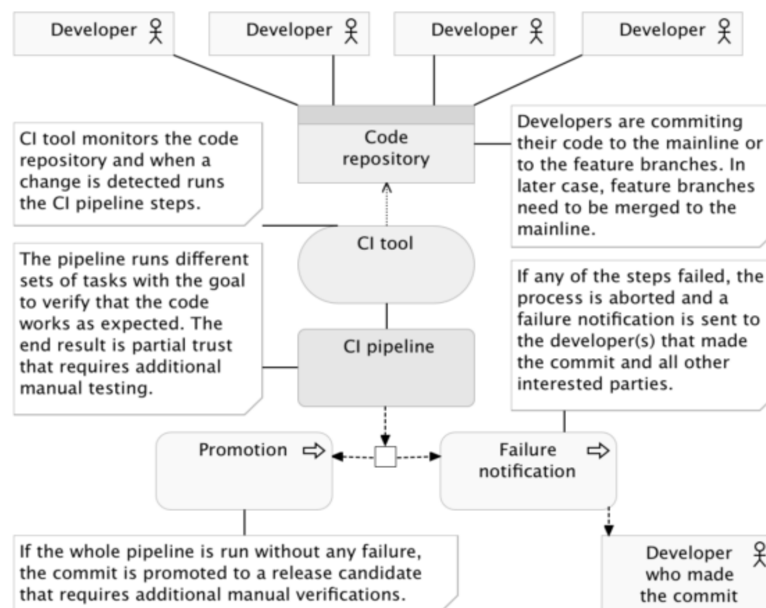


Figure 2-6 : Continuous integration process (Farcic, 2016)

The principal '*Single Source Point*' is encouraged via version control repositories such as CVS, Subversion, Perforce and Visual SourceSafe that allow accessing all source codes from a single primary location. After each build script execution in CI servers, the feedback mechanism notifies the status of the build. The Short Messaging Service (SMS), Really Simple Syndication (RSSI), E-mail and Project Management (PM) tools are the main techniques used for feedback. Fixing the discovered pipeline failures sooner without delaying is recommended to continue well with CIs. Furthermore, CI and testing with Test-Driven Development (TDD) are intricately linked together (Eck, Uebernickel, & Brenner, 2014)(Farcic, 2016). The rationale of version controlling using the scripts to control the code rather than the individual commands is a key method in tracing the artefacts. The '*Echo*' approach is an evolving tool-based solution that addresses traceability in requirements as tracking the artefacts using static or manual documentation is impractical in an Agile environment (C. Lee, Guadagno, & Jia, 2003).

2.9.1 DevOps practices

Being a cultural aspect DevOps broadens the view of Software Engineering paradigm by defining metrics that are understood across teams, sharing measurement methods, tools and by making performance part of Agile stories. *CALMS* approach is the principal notion followed in DevOps that describes to start with people (*Culture*), bring in *Automation*, stay in *Lean*, *Measure* everything and to *Share* among team members respectively. DevOps practices give equal priority to the operations team in the development environment while making developers responsible for incident handling towards faster code repairing, enforcing the deployment processes used by Devs or Ops, adhering to continuous deployment and developing infrastructure code as deployment scripts (Bass et al., 2015). The major four dimensions of the DevOps practices remain as plan/ track, dev/QA, release/deploy and monitor/optimize. This strengthens the Agile software development methodology that stands as an umbrella for many software process models such as SCRUM, XP, Lean and many more.

DevOps engineer is a prominent role in a DevOps environment that can be an individual, team or even handled at an organizational level (Ghantous & Gill,

2017). The responsibility of a DevOps engineer is to manage the tool support that which automation, version control, configuration and maintenance tools to be used, when to use based on their performance and contribution for productivity. Thus, the level of automation in the development and deployment pipeline of a DevOps environment is basically controlled by the DevOps engineer.

The coordination of human resources in a DevOps environment is important to maintain the manageability of collaborative nature. Similarly, there are separate team coordination mechanisms defined. They are human processes and automated processes. The frequently used stand-up meetings in Agile is an example of human processes-based team coordination while automated processes involve version control, configuration management and continuous integration to fasten the feedback to developers.

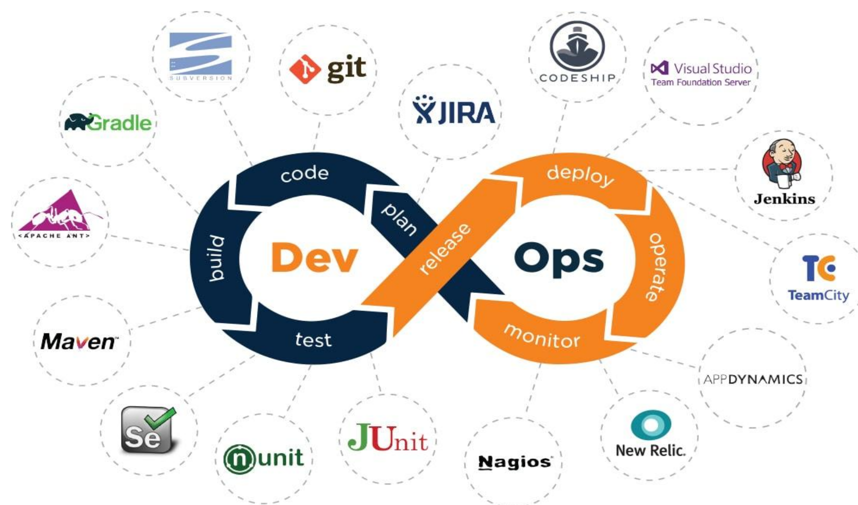


Figure 2-7 : DevOps overview (“QASource DevOps Experts,” 2018)

There are stages of a DevOps cycle with respect to SDLC phases as depicted in Figure 2-7. Some sources have defined as 5C’s of DevOps and some as 6C’s. Those include continuous planning, CI, continuous testing, Continuous Delivery (CD), continuous deployment and continuous monitoring (Kim et al., 2016)(Ghantous & Gill, 2017). CD refers to the product-level software system releases through the continuous process of building, test and continuous deployment automation. Technically, the best practice to keep a successful DevOps environment is following CICD (Continuous Integration Continuous Delivery) pipeline (Farcic, 2016)(“DZone DevOps,” 2018) that combines DevOps

practices together. It ensures to have faster integrations, accelerates product delivery with more frequent deployments and releases. That eventually contributes to increase the productivity by bringing the best plus points in adapting to DevOps.

2.9.2 DevOps tools

DevOps is classified as a new way of testing strategies that heavily contribute to increase organization throughput. It has been a powerful selection for quality results and in speeding up even customer level query processing due to the evolving DevOps tool support. The tool support in a DevOps environment majorly helps in maintaining CI and traceability. Jenkins, Travis, Ansible, Docker, Sonar, Maven and OpenStack are few among many (Ghantous & Gill, 2017). The existing higher level plugins such as 'Hudson post-build scripts' enable automated analysis of CI operations carried out in CI tools like Jenkins.

A common fact on most of these existing tools is that they have only concentrated on source code artefact integrations regardless of other artefact integrations such as a design diagram modification, test case alteration and a requirement addition. The reason for that is DevOps is emphasizing the practices on source code by assuming that a source code change is done only after considering other earlier stages artefact modifications such as design or requirements changes. Thus, the DevOps tools are performing on source code artefact according to CICD practices.

A. Jenkins

Jenkins being a leading build automation server is a prominent DevOps tool that supervises regularly executed jobs. It is an open source rapid CI server, which generates a scenario where errors can be captured at a very early stage in the SDLC. Figure 2-8 illustrates the basic workflow of building a software project on Jenkins automation server as a job.

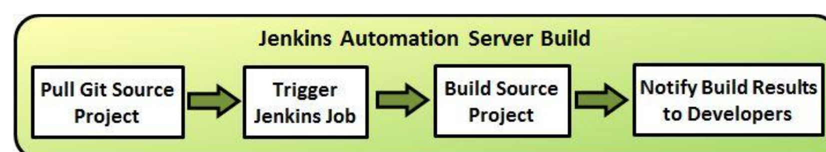


Figure 2-8 : Jenkins workflow

The functionality of the Jenkins server is conducting a definite set of activities or tasks invoked via a trigger. The trigger can be a change happened in a linked version control system or a temporal trigger such that a build in each 10 minutes. The possible tasks include performing a build with Maven or Gradle for instance, executing a pre-written shell script, archiving the build outcomes and starting any integration tests. Currently, Jenkins focuses on building or testing software systems continuously and supervising executions of jobs even though those are running on a remote machine. The simple configuration through the web-based GUI, the capability of deploying at a larger scale environment and the ability to call slaves from the cloud by adhering to a slave topology can be identified as major advantages of adhering to Jenkins (Berg, 2015). In addition, it offers a huge bundle of plugins to enhance the capabilities to support the CICD pipeline. Those plugins are usually integrated with other existing DevOps tool stack features such as for instance Jenkins has Docker deployment-related plugins where Docker itself is another DevOps tool.

B. Docker

Docker is an open platform for building, shipping and executing distributed software applications even on a Virtual Machine (VM) or a cloud environment. The existence of microservices is enriched by tools like Docker. It has made the containers/ objects that hold and transport data easily (Farcic, 2016)(Ghantous & Gill, 2017). Docker containers are happened to replace VMs as the preferable way to create immutable deployments due to the higher usage of it in the industry. The powerful utilization of Docker reduces the deployment efforts.

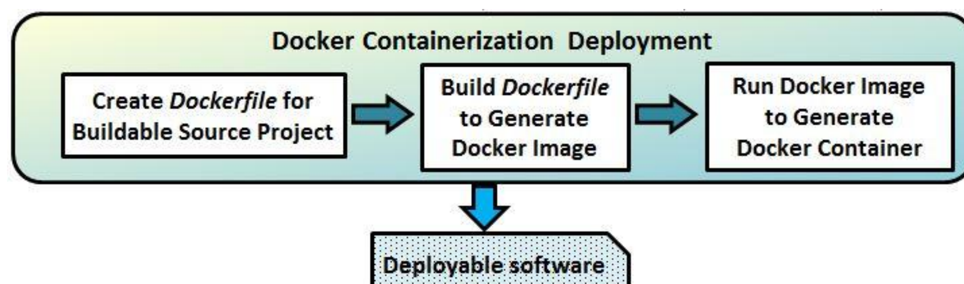


Figure 2-9 : Docker workflow

Figure 2-9 shows the workflow of Docker. *Dockerfile* is the primary element in this process that encapsulates the instructions required to build a source project with its dependencies and depending on environmental features. The execution of *Dockerfile* results in a *Docker Image* that is a file comprised of a number of layers. Finally, Docker runs that *image* to obtain the outcome that is *Docker Container* known as a standardized software capable of delivering ("Docker," 2018). They are accessible and easily usable to everyone. The relationship between a *Docker image* and a *Docker container* is similar to the difference between an object-oriented class and an object where *Docker image* is depicting the class and *Docker container* representing the runtime instance or the object out of the *image*.

C. Puppet

Puppet is a configuration tool in DevOps environments, based on deploying microservices with less time (Farcic, 2016). Puppet comprised of a centralized configuration server accessed by clients (Ghantous & Gill, 2017). The configurations are described in the form of scripts defined in a Domain Specific Language (DSL). Puppet provides a unified platform for activities such as initiating system services or organizing packages that need various tasks in heterogeneous operating systems.

D. Travis

Travis is classified as a recognized distributed continuous integrations service that supports building and testing open source software projects. It encourages team workings by tightly coupling to DevOps practices. It can automate test scheduling with GitHub repositories (Redmiles et al., 2007)("Travis CI," 2018).

2.9.3 DevOps related project management tools

Project management tools have a significant contribution in any software development model especially in DevOps where collaboration is maximum. Hence, managing a larger number of smaller teams, tracking software changes and tasks allocation among teams are keep recorded using a PM tool by any software organization. Therefore, there is a huge number of PM tools available to fulfil the organizational and personal PM needs. Few of the most prominent PM tools having different capabilities are discussed in this section.

A. Trello

Trello (“Trello,” 2018) is a prominent, web-based PM application currently owns by the company *Atlassian* which provides many software tools. It follows a *board*, *list* and *card* structure with a simpler GUI using JavaScript to manage tasks, assign among team members with deadlines, priorities and progress. It provides most of the functionalities freely being a reason for its wider usage in industry level and academia for projects management. Further, Trello is enriched with open source APIs to integrate with various environments and cloud-based integration services.

B. Jira

Jira is another leading product by *Atlassian* company for Agile PM tasks and issue tracking (“JIRA Software,” 2018). Being a proprietary tool, it is comprised of three main packages namely *Jira Core* for generic PM features, *Jira Software* specifically for Agile PM features and *Jira Service Desk* for IT/ business service desks. The organizations such as Skype, Twitter and NASA also rely on this tool due to its cross-platform supportability.

C. Slack

Slack (“Slack,” 2018) is a cloud-based team collaboration tool which stands for ‘*Searchable Log of All Conversation and Knowledge*’ launched in 2013. It provides persistent chat rooms for software environment communication that can be organized by topic and searchable including files. Slack is a proprietary tool that provides basic functionalities free with cross-platform capabilities.

D. Zoho sprints

Being similar to Trello in structure, Zoho Sprints (“Zoho Sprints,” 2018) is a proprietary Agile PM tool that is built specially for Scrum teams to plan the workloads in Sprints. Thus, it provides features to add user stories to backlogs estimate and prioritize work items following a board-based GUI.

E. Bitrix24

This is categorized as a leading free cloud and mobile PM solution that provides extended features proprietarily. Tasks, Gantt charts, task dependencies, resource planning and invoice management can be conducted with Bitrix24 (“Bitrix24,” 2018) in many languages such as English, Spanish, Russian and German.

2.10 Analysis of related work

Table 2.5 summarizes some related work on traceability management. The work presented by Tyree (Tyree & Akerman, 2005) has used the decision-based traceability on the architectural artefact. Passos (Passos et al., 2013) has addressed up to the implementation level artefacts and its development environment is limited to feature-oriented software projects. A rule-based approach with multi-level dependency modelling considering many artefacts including the testing phase is presented in (Lehnert et al., 2013). It has applied impact analysis over heterogeneous artefacts and has achieved significant precision and recall results though lacking support for dynamic UML models. Zhang (S. Zhang, Gu, Lin, & Zhao, 2008) has addressed the change detection and impact analysis with a framework implemented in *AspectJ* programs. The workspace awareness tool in (Sarma, Redmiles, & Van Der Hoek, 2012) has involved all the phases in continuous integration in an event-based approach, but it lacks the automation.
The IR techniques VSM and LSI are used in (Lucia, Fasano, Oliveto, & Tortora, 2007) though the change propagation and continuous integration are not addressed. However, this work has been not limited to a specific artefact type and has semi-automated traceability recovery. The tool *Echo* presented in (C. Lee et al., 2003) which is based on Agile practices have addressed requirements and design artefacts. Another event-based notification approach that supports heterogeneous and distributed development environments is used in (Cleland-Huang, Chang, & Christensen, 2003) though it lacks CI support.

Although there are a considerable amount of research has been done, most of the literature has certain limitations such as being addressing only a few artefact types, not focusing on complete SDLC, lack of support towards continuous integration and lack of automation. Further, it is observable that the IR methods are involved in requirement traceability, whereas event-based and rule-based approaches are used in change detection, impact analysis and change propagation. Accordingly, the lack of traceability management to cope with continuous integrations for the entire SDLC can be identified in the existing related works.

Table 2.5 : Evaluation of related work on traceability management

Related work	Traceability establishment	Change detection	CIA	Consistency management	Change propagation	CI
(Tyree & Akerman, 2005)	Template-based approach using architectural decision templates.	Decision-based approach.	Manual analysis by humans.	-	Decision-based approach require manual monitoring.	-
(Passos et al., 2013)	Feature-oriented approach.	Feature-oriented manner.	Calculate feature dependencies in artefacts.	-	-	-
(Lehnert et al., 2013)	Rule-based approach. Dependency detection, Dependency relations.	-	Rule-based approach. Multi-level modelling. Depend on change propagation rules.	Multi-perspective consistency checking.	Analyse dependency relations.Recursive algorithm.	-
(S. Zhang et al., 2008)	-	Atomic change representation, syntactic dependencies.	Static AspectJ call graphs.	-	-	-
(Sarma et al., 2012)	Event-based approach.	Visualization.	Event-based approach. Binary measurements.	Manual visualizations.	*YANCEES* notification service.	Workspace awareness tool.
(Lucia et al., 2007)	Information retrieval methods.	Matrix-based using VSM.	Rule-based approach.	Traceability recovery using LSI.	-	-
(C. Lee et al., 2003)	Text annotations. Conversation-centric model.	Visualization.	Manually via visualization. Forward, backward traceability.	-	Use of elaboration activities.	Versioning.
(Cleland-Huang et al., 2003)	Event-based approach.	Publisher-subscriber.	Event-based approach. Event logs for artefacts.	-	Update artefact event logs.	-
(Alves-Foss et al., 2002)	Data pre-processing XML, HTML.	Visualization.	Manually via visualization.	Integrative approach.	-	-

2.11 Visualization of traceability links

Visualizing software artefact traceability is useful in building, recovering the artefact relationships and in decision making. It is challenging to visualize a large number of traceability links and paths among artefacts in real time with inter-relationships due to scalability and visual clutter related issues.

Table 2.6 : Traceability visualization techniques

Technique	Features	Advantages	Limitations	Related work
Lists	Represent data in a single dimension sequentially.	Efficiency due to simplicity.	Limited for a smaller amount of data due to single dimension.	(Merten, Jüppner, & Delater, 2011)
Traceability matrix	Store data with two-dimensional grid structure.	Capable of displaying artefacts in two dimensions. Recommended for a smaller number of artefacts.	Impractical to represent a larger number of traceability relationships.	(Chen, Hosking, & Grundy, 2012)
Cross-reference	Represent data in a table structure.	Capable of providing a list of relevant trace links for artefacts.	Inability to provide an inclusive structure of traces and to find individual trace links since strictly adhered to a table structure. Lack of scalability.	(Chen et al., 2012)
Treemap	A tree data structure to represent data in a 2D manner.	Display a large tree by using display space effectively.	Inability in communication with the hierarchical structure. Complex for a larger number of links.	(Shneiderman, 1992)
Hierarchical tree	Represent data in hierarchical structure with node-link style.	Provide detailed dependency information about traces. Simplicity, understandability.	Visual clutter in an excessive number of trace links.	(Holten, 2006)
Traceability graph	Graph representation, data in nodes and relationships in edges.	Higher ability in visualizing structured data with relations.	Limit the viewing of graph for excessive nodes. Performance issues.	(Herman, Melancon, & Marshall, 2000)
Sunburst and Netmap	A radial layout. Alternative for matrices and graphs.	Effective in browsing and navigation with better user orientation.	Not filtering the visualization links.	(Merten et al., 2011)

There are visualization techniques and tools that enable analysing large temporal data. The selection of an optimal technique depends on the various properties in trace links. Table 2.6 presents a summary of visualization techniques. Among different visualization techniques, *traceability matrix* is mostly used for requirements artefact with NLP aspects (Thommazo, Malimpensa, De Oliveira, Olivatto, & Fabbri, 2012)(Chen et al., 2012). The graph-based, tree-based and other techniques are also used in some related work (Chen et al., 2012)(Rodrigues et al., 2016).

Table 2.7 analyses the related work on traceability visualization techniques. Most of them have slightly considered model driven features. It is a limitation in supporting to a range of software types (Kugele & Antkowiak, 2016)(Santiago, Vara, De Castro, & Marcos, 2014). Many works have addressed the visual clutter and scalability issues (Merten et al., 2011)(Filho & Lencastre, 2012) and several tools are integrated with a specific IDE. Moreover, many studies have considered only a certain type of artefacts such as either requirements or source code. Thus, there is a need for a generic software artefact visualization methodology.

Table 2.7 : Evaluation of related work on traceability visualization techniques

Related work	**Visualization technique**							
	Lists	***Traceability matrix***	***Cross-reference***	***Tree map***	***Hierarchical tree***	***Traceability graph***	***Sunburst and Netmap visualization***	***Other***
(Merten et al., 2011)	√						√	
(Chen et al., 2012)		√	√					
(Holten, 2006)				√	√			
(Rodrigues et al., 2016)		√		√		√	√	
(Filho & Lencastre, 2012)		√		√			√	
(Thommazo et al., 2012)		√						
(Kugele & Antkowiak, 2016)						√		metaphor-based
(Santiago et al., 2014)						√		MDE-oriented

2.12 Tool support for tractability management and continuous integration

One of the approaches for maintaining traceability is tool-based approaches where a specific tool is used for tracing purpose of that particular artefact. The tool support for artefact traceability and continuous integration is an evolving area with the use of existing and novel techniques. The representation and visualization of the identified traceability results is a major challenge for proper artefact management. Some existing traceability tools support the representation while some remain with limitations as stated in Table 2.8.

Table 2.8 : Tool support for traceability management

Tool	Usefulness	Limitations
TraceME (Bavota et al., 2012)	Artefact traceability visualization in traceability dependency graphs.	Limited to Eclipse IDE as a plugin. Research-level.
ADAMS Re-Trace (De Lucia, Oliveto, et al., 2008)	Heterogeneous artefact traceability management and recovery.	Limited to be used within Eclipse IDE as a plugin.
Caliber-RM (Capterra, 2019)	Allow stakeholder collaboration with versioning. Impact identification and visualization of requirements.	Proprietary. Limited for requirements artefact. Platform dependent with Windows OS.
Cradle ("3SL," 2018)	Designed for Agile development. Scalable and multi-user accessible.	Lack of CIA, Proprietary tool. Limited for requirements.
RequisitePro ("Rational RequisitePro," 2017)	A collaborative requirements management tool. Support use case generation.	Limited for requirements artefact. Proprietary. Lack of tool maintenance in updates.
YAKINDU ("YAKINDU Traceability," 2019)	Support tool integration with the applicable artefacts. Visualize query and generate traceability coverage reports and impact analysis results.	Limited for requirements artefact. Proprietary and patent pending tool.
Palantír (Sarma et al., 2012)	Notify artefact changes, CIA. Graphically display in a configurable and non-obtrusive way. Enforce continuous coordination.	Changes related information is captured at the file level and user notification of conflicts at the code entity level.
ReqView ("ReqView," 2017)	Present structured requirements in a tabular way and visualize in a traceability matrix.	Limited for requirements artefact. Proprietary tool.

Table 2.9 summarizes the tool support for information retrieval techniques. The tool *TraceME* has addressed all the main artefact types and stands as an Eclipse plugin (Bavota et al., 2012). The tool *RETRO* can be identified as more towards a

case study biased to the requirements artefacts (Hayes et al., 2007). *ADAMS Re-Trace* is another Eclipse plugin that has addressed the main types of artefacts and has used the LSI technique for IR (De Lucia, Oliveto, et al., 2008).

Table 2.9 : Tool support for information retrieval

Tool	**Artefacts**	**Information retrieval technique**			
		VSM	***TF-IDF***	***LSI***	***Other***
IBM RequisitePro ("Rational RequisitePro," 2017)	Requirements				document-based
TraceME (Bavota et al., 2012)	All	X			
RETRO (Hayes et al., 2007)	Requirements, design, bug reports	X	X	X	
ReqAnalyst ("SERG :ReqAnalyst," 2017)	Requirements			X	Extract-Query-View
ADAMS Re-Trace (De Lucia, Oliveto, et al., 2008)	All			X	
TraceTool (Mischler & Monperrus, 2014)	SRS		X	X	

Wider use of VSM, TF-IDF and LSI techniques for the purpose of information retrieval can be seen in this summarized commercial traceability related tools in Table 2.9. However, still, the major software artefact that most of the tools have addressed is only the requirements artefact where the test scripts, configuration files sort of artefact types are hardly addressed.

The use of traceability management support along with CI, change impact analysis and consistency management in the existing tools is summarized in Table 2.10. Some tools are platform dependent such as *Caliber-RM* is only supporting Windows environment (Borland, 2006). *TraceMaintainer* is an independent tool that supports any CASE tools in any heterogeneous environment. However, it is limited for the support towards the requirements and design artefacts (Mäder et al., 2008). *LDRA-TBmanager* is a significant tool that has addressed the artefacts related to testing activities in SDLC and it supports the applications developed using any programming language ("LDRA," 2018). *TraceME* and *ArchEvol* are integrative tools with the Eclipse IDE as a plugin. An object-oriented Supply-Chain Management (SCM) infrastructure is contained in the tool *MolhadoArch* that has addressed majority types of software artefacts (Nguyen, Munson, &

Boyland, 2004). Accordingly, a lack of tool support addressing all the types of heterogeneous artefacts together in SDLC with a minimum of dependencies such as depending on a particular IDE or a platform can be identified.

Table 2.10 : Tool support for traceability management

Tool	**Artefacts**	**Traceability management approaches**			**Continuous integration approaches**				**CIA**	**Consistency management**
		Rule-based	***Hypertext-based***	***Integrative***	***Versioning***	***Collaboration***	***visualization***	***Modeling***		
IBM DOORS (“IBM-Rational DOORS,” 2017)	Requirements		X	X	X	X	X	X	X	X
RequisitePro (“Rational RequisitePro,” 2017)	Requirements			X		X	X	X	X	
Caliber-RM (Borland, 2006)	Requirements			X	X	X	X		X	X
Cradle (“3SL,” 2018)	Requirements			X					X	
TraceMaintainer (Mäder et al., 2008)	Requirements, structural UML	X		X						
TraceAnalyzer (Egyed, 2001)	UML designs, test cases, code			X			X	X		
TraceME (Bavota et al., 2012)	All			X					X	
LDRA-TBmanager (“LDRA,” 2018)	Requirements, regression suites, test scripts				X	X				
ReqView (“ReqView,” 2017)	Requirements			X	X	X				
ArchEvol (Nistor, Erenkrantz, Hendrickson, & van der Hoek, 2005)	Architectural aspects, code			X	X		X			
ArchStudio (“ArchStudio,” 2018)	Architecture			X		X	X	X		
MolhadoArch (“Molhado Project,” 2017)	All		X		X		X			

2.13 Evaluation techniques of traceability management

2.13.1 Quality measures

A. Traceability coverage

The correctly identified trace links in a traceability recovery process is known by *traceability coverage*. It is advantageous to determine the quality of established artefact traces such that *well traced* and *poorly identified traces* which helps to improve the traceability. Traceability coverage can be defined as in equation (2.1) (Cleland-Huang et al., 2012).

$$\text{Traceability coverage a} = \frac{|\text{links_a (targets)}|}{|\text{targets}|} \quad (2.1)$$

Where, *targets* is the target artefacts and $links_{_a}(targets)$ denotes the links traced between a particular artefact *a* with the artefacts in the target set.

B. Correctness measures

Precision, recall and F-Measure are the highly applied accuracy measures. The accurate instance count among all the obtained instances regardless of their relevance is known by *precision* (2.2) and helps to save time when finding changes. *Recall* (2.3) that is also referred by *sensitivity* expresses the accurate instance count among the obtained related instances and contributes to confirm whether proposed changes are all considerable or not (Zeugmann et al., 2011).

$$\text{Precision} = \frac{|\text{EIS} \cap \text{AIS}|}{|\text{EIS}|} \quad (2.2)$$

$$\text{Recall} = \frac{|\text{EIS} \cap \text{AIS}|}{|\text{AIS}|} \quad (2.3)$$

Where, EIS represents estimated impact set and AIS denotes actual impact list.

In relation to both precision and recall, *F-Measure* (2.4) is defined by the weighted harmonic mean of both of them in a test having the values in range [0,1] which highlights the association of precision and the recall (Zeugmann et al., 2011). F-Measure is also known by *F1 score*.

$$\text{F} - \text{Measure} = 2 * ((\text{precision} * \text{recall}) / (\text{precision} + \text{recall})) \quad (2.4)$$

C. Reliability

Reliability of traceability is important in safety-critical and high-reliability systems. The Hidden Markov Chain (HMC) algorithm can be used in measuring software reliability (J. Lee, Cho, Youn, & Lee, 2009)(Vrignat, Avila, Duculty, & Kratz, 2015). The UML artefact based reliability prediction in traceability also contains a significant value in the literature (Trung & Thang, 2009)(J. Lee et al., 2009). Moreover, the proper maintaining of a traceability matrix is considered useful to preserve the reliability with respect to requirements artefact.

D. Usability

Usability is concerned with the user experience and the interactivity based on evolving user expectations. The usefulness, ease of use, learnability and likeability are treated as the general concepts of the usability (Winkler, 2008). A larger number of users sample is mainly considered in measuring the usability aspects of a traceability tool in the related works (Faulkner, 2003). Moreover, the degree of automatization by reducing human effort in reducing the trace link generation time and user interface improvements are considered as usability features (Sünnetcioglu, Brandenburg, Rothenburg, & Stark, 2016).

There exist multiple criterion methods to evaluate usability quantitatively and qualitatively such as System Usability Scale (SUS), Likelihood to Recommend (LTR), Net Promoter Score (NPS) and the use of Tag Clouds. SUS is a Likert scale methodology found by John Brooke in 1986 to measure usability level of a software tool with the involvement of a set of users (Brooke, 2013). It provides a questionnaire consists of ten standard questions each with five options. The options which are scale-based ranging from 'strongly agree' to 'strongly disagree' remain the same for all questions and each contains a quantitative weight. Thus, the final outcome of SUS is a numerical value called *SUS score*. It assigns a usability level to the tool such as *Average* or *Above Average*.

NPS is another user satisfaction evaluation method, but with a single question to the users that question *how likely a user would recommend the tool to someone* (Keiningham, Aksoy, Cooil, Andreassen, & Williams, 2008). It is known as LTR and is also defined as SUS score divided by 10 in relation with SUS methodology

("MeasuringU," 2018). The answerable 11 scale options range from 'not at all likely' to 'extremely likely' similar to SUS and results in a quantitative value as the final NPS value. In addition, a *tag cloud* is a novel visualization technique to represent weighted keyword-based textual contents (Sinclair & Cardew-Hall, 2008). It can be used to represent user feedback with their response frequencies.

2.13.2 Network analysis

Network analysis combines several centrality measures to specifically assess network graphs. Thus, they are applicable to evaluate the accuracy of traceability links in traceability networks that are represented in a form of visualization graphs. The centrality measures include (Knoke & Yang, 2008), *degree centrality*: denotes the status of a node based on the number of adjacent links; (2.5) *closeness centrality*: gives the most nearer node to a maximum number of nodes; (2.6) *betweenness centrality*: states the number of times a node act as a bridge along the shortest path to others; *Eigenvector Centrality (EVC)*: gives the most influential element. EVC measure is used to analyse the accuracy of the artefact traceability establishment (Borgatti, 2005).

$$C(x) = \frac{N}{\sum_y d(x,y)} \quad (2.5)$$

$$g(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (2.6)$$

EVC is defined as the principal Eigenvector of the adjacency matrix defining the network and has been used to analyse accuracy in previous work (Perera, Miller, & Allison, 2017). If an EVC measure of the artefact has a higher value, it can influence many other artefacts. An artefact is more influential if it affects other highly influential artefacts than an artefact which affects the same number of less influential artefacts (Perera et al., 2017). Thus, EVC is useful to find how the artefacts influence the linked artefacts, without necessarily being restricted to the shortest path etc in a node connectivity (Borgatti, 2005).

2.13.3 Traceability testing techniques

A. Unit testing

The unit testing verifies the fulfilment of the specification by each unit (Sommerville, 2010). The concept of *Test-first development* encourages to have automated unit test for each functionality, before the implementation of the function. It is essential to perform unit tests at each integration of the continuous integration process.

B. Regression testing

The regression test is important in confirming that the previously run tests and alterations have not introduced new defects (Sommerville, 2010). Hence, it is essential in applying for a traceability management tool as the traceability itself heavily gets changed based on artefact element alterations. Besides, regression testing is the main testing method in Agile-based software development. Traceability matrix technique can be used in regression testing (Athira & Samuel, 2011).

C. User acceptance testing

The User Acceptance Testing (UAT) is one perspective in the final stage system testing that verifies the intended behaviour of the final software product and is a black box test (Hambling & Goethem, 2013). The version controlling features can be used to maintain the requirements and acceptance tests. UAT can be applied to different case studies and user samples. The *alpha* and *beta tests* are two subtypes of UAT.

2.13.4 Supported testing tools

A. Selenium

Selenium is an open source, portable test automation suite that is capable of test management and reporting (“Selenium,” 2018). This can be applied for continuous integration servers, where the machine-readable test reports are essential to evaluate the accuracy of integrations.

B. JUnit

JUnit stands as a prominent Java-based unit test automation tool which is integrated with most of the IDEs such as NetBeans (Sommerville, 2010). Therefore, it can be used in testing the Java involved source code artefact traceability.

2.14 Discussion

Software systems in every domain become highly complex and competitive requiring the ability to perform in high reliability in order to sustain without being replaced by a newer software system. The development of these systems requires strong traceability and consistency management for the correct functioning and maintenance of the product.

Different types of intermediate software artefacts are involved during the development process. The main software artefacts include requirements, designs, source codes, test scripts, build scripts, configuration files and many more. The Agile software development model is identified to be the most evolving one in trend due to its highly collaborative and cost-effective nature. It is comprised of practices such as DevOps, continuous integration and continuous delivery. DevOps reduce the gap between development and the operations, whereas the continuous integration referrers frequent merging of developer working copies. The resulting rapid changes of artefacts are required to be traced in order to preserve the maintainability in DevOps. Furthermore, change impact analysis plays a significant role before and after each change detection process.

The evaluation measures of traceability and impact analysis are more towards efficiency, performance and correctness measures. Evaluating the quality attributes is vital for the betterment of traceability management in a rapidly changing DevOps environment. Correspondingly, the uniqueness and the usefulness of the core research problem identified to be addressed in this research work; *determining an approach of impact analysis for artefact traceability in a DevOps environment* is justified in this conducted literature study.

2.14.1 Limitations in current practices

The main limitation in the existing context of software traceability and continuous integration is the lack of sufficient tools and technique. The existing tools are limited to certain artefacts and development environments (C. Lee et al., 2003)(Burgaud, 2006). Also, there is a lack of CIA methods associated with traceability especially in a quantitative approach. Moreover, traceability visualization and validation covering heterogeneous artefacts are hindrances in literature. Thus, the automation of traceability establishment has become unachievable and inapplicable into DevOps environments. Although the support of traceability and CI is important to be available during the overall SDLC, it is not completely preserved in the current practices (Chang, 2005).

2.14.2 Future challenges and research directions

The current software industry is still reluctant to adapt the traceability aspects into the environments due to the above-identified limitations and challenges. It is challenging to build a general framework that supports traceability management with a wide range of customizability. Another challenge is that currently, traceability does not provide tangible direct advantages to software development. Thus, there is a need of a tool that supports all the artefact types and development environments in managing traceability. On the other hand, DevOps practices support great collaboration between many functions engaged in the current software development processes. Therefore, a technically sound and feasible approach to manage software artefact traceability with impact analysis for a DevOps environment having continuous integrations is essential for software development as well as for maintenance (Rubasinghe et al., 2017).

Section 3

Research methodology

The solution space of the addressed research question is presented in this chapter. The system designs of extended SAT-Analyser system and the technical aspects of functionalities with their implementation are explored in this chapter.

3.1 System design

3.1.1 System overview

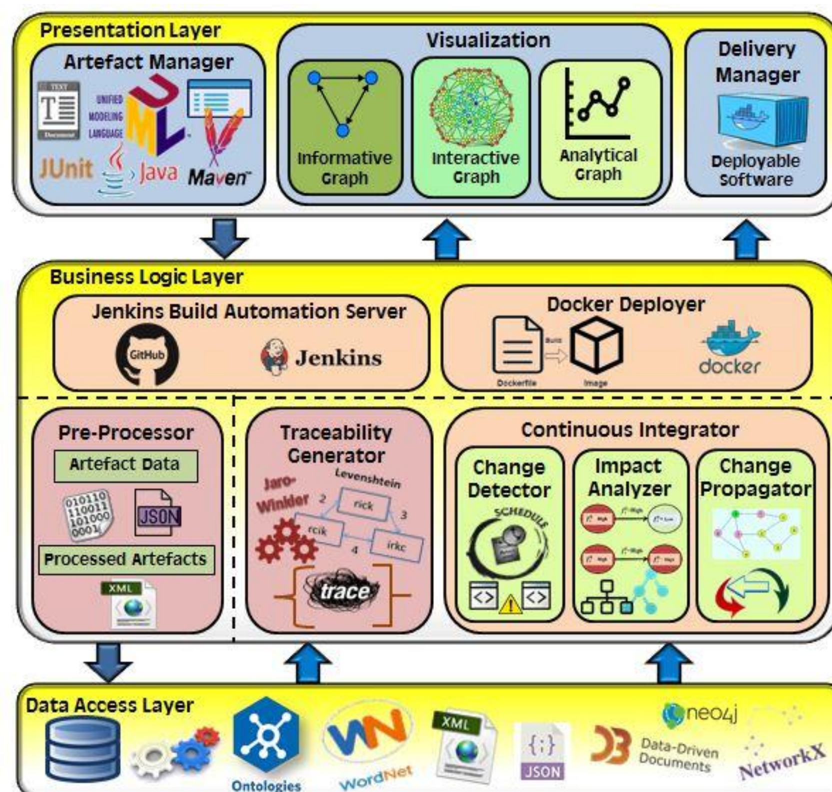


Figure 3-1: Extended SAT-Analyser system overview

Figure 3-1 depicts the abstract system overview of extended SAT-Analyser tool. The heterogeneous artefacts are handled by the presentation layer as inputs to the system. The business logic layer *pre-processor* component is responsible for the data pre-processing of the acquired artefacts, data extraction and storing the extracted items. Major functions related to overall traceability management; traceability establishment is handled by *traceability generator* component and

change detection, change impact analysis, change propagation and consistency management are managed by the *continuous integrator* component. Most of the algorithmic workflow is divided within the business logic layer for these functions. The traceability CI component is defined to be triggered with the continuous deployments in a DevOps environment. Hence, at a deployment task, *pre-processor* obtains the latest source code and build script artefacts via the Jenkins automation server's most recent successful build. The data management required by the business logic layer is stored in the bottom data access layer. Finally, the results visualization in three enhanced methods such as informative, analytical and interactive graph is a responsibility of the presentation layer's *visualization manager* while providing a notification back to *Docker Deployer*. The *delivery manager* in presentation layer then proceeds to complete the deployment task with deployable software prior to CD.

3.1.2 Research model

In designing the extended SAT-Analyser, an industry level survey is conducted among the DevOps practitioners. The purposes of the survey are to;

- Identify software artefacts that are highly subjected to changes,
- Continuous integration techniques and frequencies,
- Traceability management methods used in practice,
- Visualization mechanisms in DevOps environments,
- Change detection, change impact analysis, change propagation and methods used to ensure artefact consistency.

Accordingly, unit test scripts are selected with respect to source code artefact based on the analysis of obtained survey responses. The major activity of the configuration phase within a DevOps environment is to build the implemented codebase. Build automation is associated to compile the source code and to transform into a binary code form. Build automation ensures that the tested code is executable. The tools like Jenkins, buildbot, Apache Ant, Ninja, MSbuild and Puppet support it with many options such as make-based, make-build, build script generation and make-incompatible. Correspondingly, the build script generation approach is used in this research. The Java-based Apache Maven is selected as the

configuration artefact while Jenkins as the build automation server. In the build scripting method, the source code related dependencies, dependant plugins and repositories are provided in a scripting format which is executed when a build is necessary that can be either continuously or periodically through Jenkins.

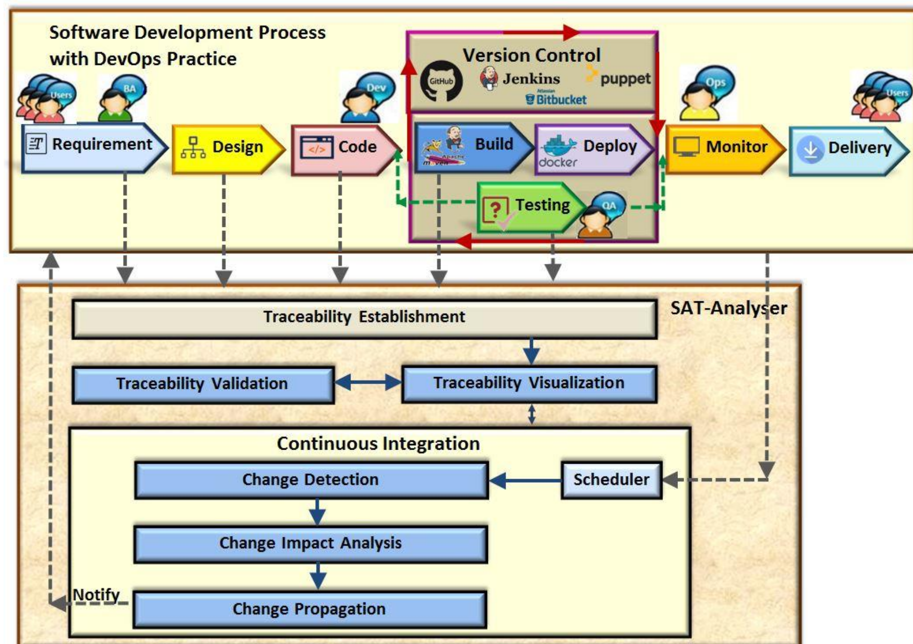


Figure 3-2 : SAT-Analyser tool research model

Figure 3-2 illustrates the research model of the extended SAT-Analyser. It shows the integration of SAT analyser with a practical software development process in DevOps practice. A DevOps environment differs from a traditional development process by the means of continuity in testing, integrating, deploying, delivering and due to the collaborative workforce of different roles together. The developers who are denoted as *Devs*, Quality Assurance (QA) engineers who perform testing and other operational level team members denoted by *Ops* who are responsible for monitoring, deployment, delivery and maintenance tasks work hand in hand than performing their duties in isolation.

This research model is designed to perform the artefact traceability in CICD pipeline. Initially, SAT-Analyser obtains all artefacts; requirement, design diagram, source codes, test scripts, build script and establishes a traceability model among them with visualization and validation. Then, SAT-Analyser's invokes

scheduler during continuous integration. To ensure the traceability before delivering, SAT-Analyser obtains the latest source code and build script via the most recent successful build from Jenkins server. Accordingly, SAT-Analyser re-establishes a traceability model based on obtained new source code and build script artefact with change detection, change impact analysis, change propagation and visualizes the traceability results. Simultaneously, notifies the teams via the project management tool Trello about the change propagation. Consequently, DevOps teams can decide on proceeding with CD. SAT-Analyser supports the synchronization within the software process in this approach. Table 3.1 lists the addressed software artefacts and the features existed in the initial SAT-Analyser prototype tool. The limitations in the initial tool and the possible improvements for the considered aspects in this research work are also summarized.

Table 3.1 : Analysis of existed SAT-Analyser

Existed SAT-Analyser			**Considerations for the tool extension**
SDLC phase	***Software artefacts***	***Possible improvements for limitations***	
Requirement analysis	Natural language requirement description	The data pre-processing is currently error-prone. Only the requirements given in the simplest raw text is considered in a .txt format. The other types of artefacts can be considered. NLP data pre-processing can be enriched with IR techniques for better data extraction. Does not support continuous integration.	Will not be considered as the area is more into Natural Language Processing and information retrieval. Not within the research scope.
Design	UML class diagram	Only the structural view class diagrams are considered. Can be extended to behavioural models. In a class diagram, only the inheritance is considered. The aggregation, composition types of relationships have not considered.	Other UML design diagrams are not within the research scope, as we focus on CICD pipeline.
Implementation	Java source code	Only the source files in Java language are processed using Java Grammar 8. Can be extended to other Object-Oriented or functional programming languages such as C++, Python. Impact analysis is not included significantly after the change detection.	Other programming languages are not within the research scope. Source code will be considered as it is used in CI to integrate the code to a shared repository.

Table 3.2 summarises the software artefacts considered with respect to phases in DevOps, the importance of the selected artefact types and possible techniques for the proposed SAT-Analyser tool.

Table 3.2 : Analysis of the SAT-Analyser with DevOps extension

Proposed research for SAT-Analyser with DevOps environment			
Phases	***Artefacts***	***Description***	***Possible techniques***
Development	Java source code	The source code changes are prominent in continuous integration. Many changes occur in the codebase and the proper consistency management is essential.	Java Grammar 8 and ANother Tool for Language Recognition (ANTLR) are used to pre-process Java code. The version controlling can be done via GitHub. Jenkins can be integrated for change management.
Testing	Unit test script	Unit tests automate the testing process by verifying individual units. It refactor easily instead of changing already tested codes which is costly and risky.	Event-based traceability can map with the previous code version. Tree-differencing with Edit history can use for change detection.
Configuration	Dependency files in Maven	Maven repository build automation by concatenating artefact dependencies. All prominent CI servers are supportive to the packaging structure used in the Maven dependency management and Maven is highly supported within the IDEs involved in industry level.	Maven files follow an XML data structure such as pom.xml file. XML based data pre-processing can be done using DOM parser. Tree-differencing with Edit history can apply for change detection. Tools such as Puppet and Chef can be used for configurations and Jenkins can be used to automate the source code repository compilation into an executable code base.
Deployment	Deployment scripts	The cloud technologies are used for efficient productivity in DevOps. The deployment scripts can be used for cloud integrations with cloud hosts. Thus, the build-deploy-test-release pipeline in CD aspect of DevOps can be facilitated.	The deployment scripts can be created based on the build outcomes using Docker that create a deployment script called Dockerfile and containerize it to deploy as a standard software unit to a cloud or local repository.

3.1.3 System architecture

Figure 3-3 depicts the extended SAT-Analyser system architecture that follows a layered behaviour having presentation, business logic and a data access layer. The *artefact manager* provides the input interaction by intaking requirement in textual, design in UML, source code in Java, test script in JUnit and build script in Maven files. The *visualization manager* provides the output interaction with different graph representation types such as informative, interactive and analytical by involving JSON, JavaScript and Python to facilitate flexibility for decision making. The *delivery manager* fulfils the deployment with deployable software before continuous delivery.

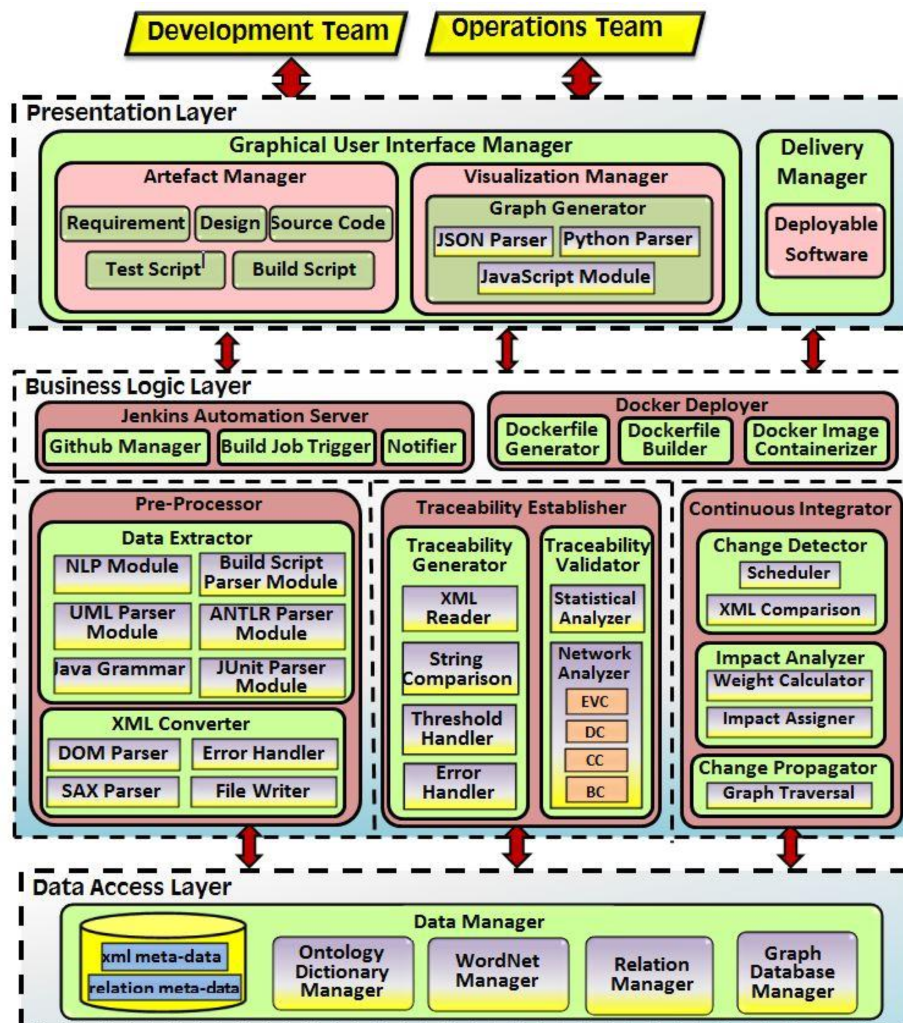


Figure 3-3 : Extended-SAT-Analyser system architecture

The business logic layer is responsible for artefact data conversion into processed artefact elements, traceability establishment and continuous integration management. The *CI manager* is associated with the Jenkins automation server and Docker deployer. The *pre-processor* component has a separate data extraction module for each type of artefacts due to the heterogeneity of them. Once pre-processed, *XML converter* brings them into a single common XML format using XML parsers like DOM and SAX parser. Traceability generation with the aid of XML readers, string comparison and traceability results validation using both statistical and network analysis techniques are responsibilities of the *traceability establisher* component.

The *continuous integration* component is considerably important as it is responsible for change detection and impact analysis that involves mathematical models such that scheduler algorithms, XML comparison algorithms, weight calculation, impact assignment and change propagation using graph traversal algorithms. It is triggered by the *delivery manager* at a continuous deployment activity that could be twice a day or more frequent. Then, *pre-processor* obtains the latest development artefact; source code and associated build script artefact via the Jenkins latest successful build job. Jenkins server is comprised of a relationship with multiple source code management systems like GitHub, a *build job triggering* to perform project build activities and a *notifier* to inform the build results whether a success or a failure. After completing the traceability and CI process with change propagation results visualization, SAT-Analyser notifies to teams via project management tool Trello. Next, the Docker deployer can proceed with deployment by creating *Dockerfile*, *Docker image* and containerization to let the *delivery manager* in presentation layer have deployable software.

The bottom layer is providing the database storage and access for all the purposes such as for artefact storage, graph storage involving *ontology dictionary manager*, *WordNet manger*, *relation manager* and *graph database manager*.

3.1.4 Abstract system workflow

Figure 3-4 illustrates the abstract workflow design of extended SAT-Analyser tool. It starts by input artefacts such that requirements in the text, design in UML class diagram, source code in Java, unit test in JUnit and build script in Maven pom.xml. Then, preprocesses each type, converts into an intermediate XML format, generates traces, visualizes and analyses them. The remaining CI part has a scheduler to initiate the CI process and detects changes based on XML versions of artefacts by managing versions. Thereafter, calculates impact and propagates changes accordingly. Finally, updates changed artefact XMLs and notify deployer via a project management tool to bring the system into a stable stage.

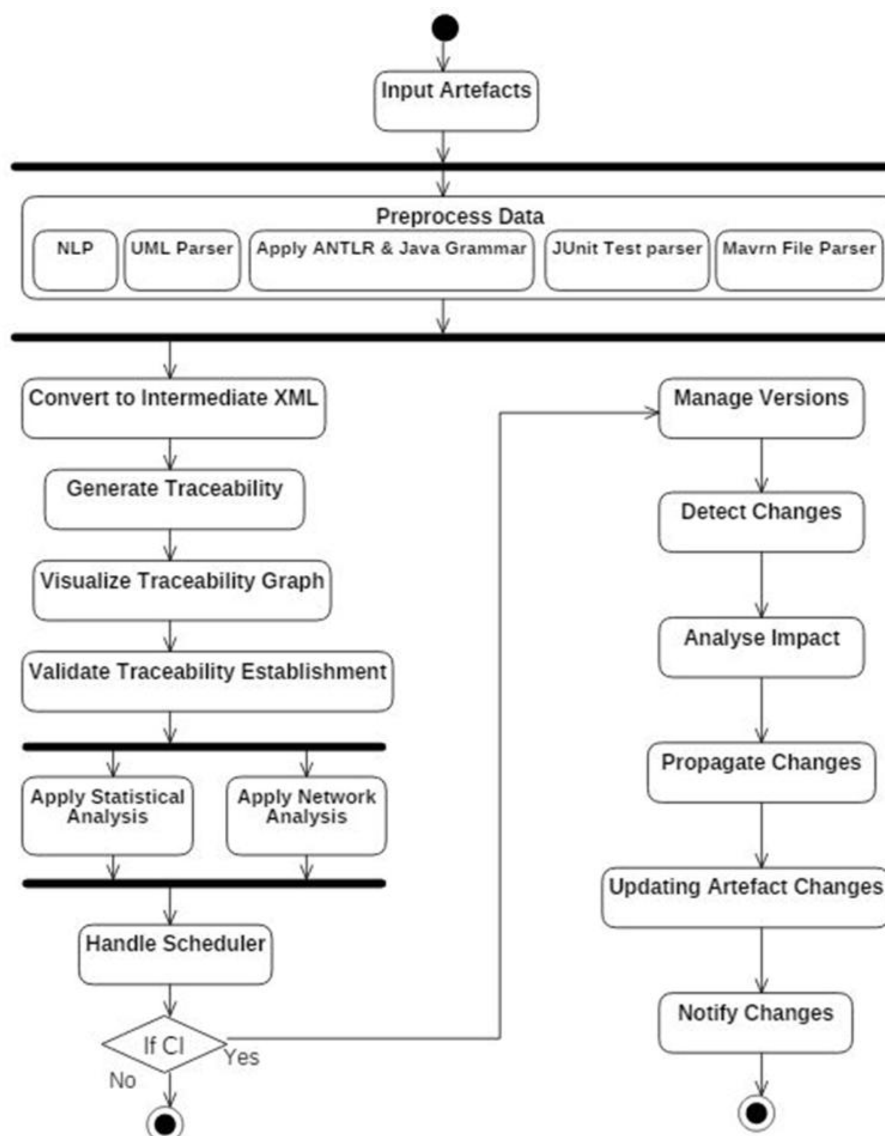


Figure 3-4 : Extended SAT-Analyser abstract workflow

3.1.5 Detailed system workflow

The detailed system workflow of extended SAT-Analyser is shown in Figure 3-5. The application of the technical aspects shown and discussed in Figure 3-3 is illustrated in this diagram. The leftmost side shows the data/ information elements such as artefacts, Java Grammar, JSON parser, artefact elements, XML writer, WordNet, dictionary ontology, thresholds, Neo4j graph database and Gephi open graph platform. The activities are shown by the other type of rectangular shapes while arrows depict the activity flow with directions. The notations IN, V1/2/3, CP, CIA, CD, CM represent inputs, versions, change propagation, change impact analysis, change detection and consistency management to categorize the activities for better readability. The workflow ends when consistency is managed with traceability project stability.

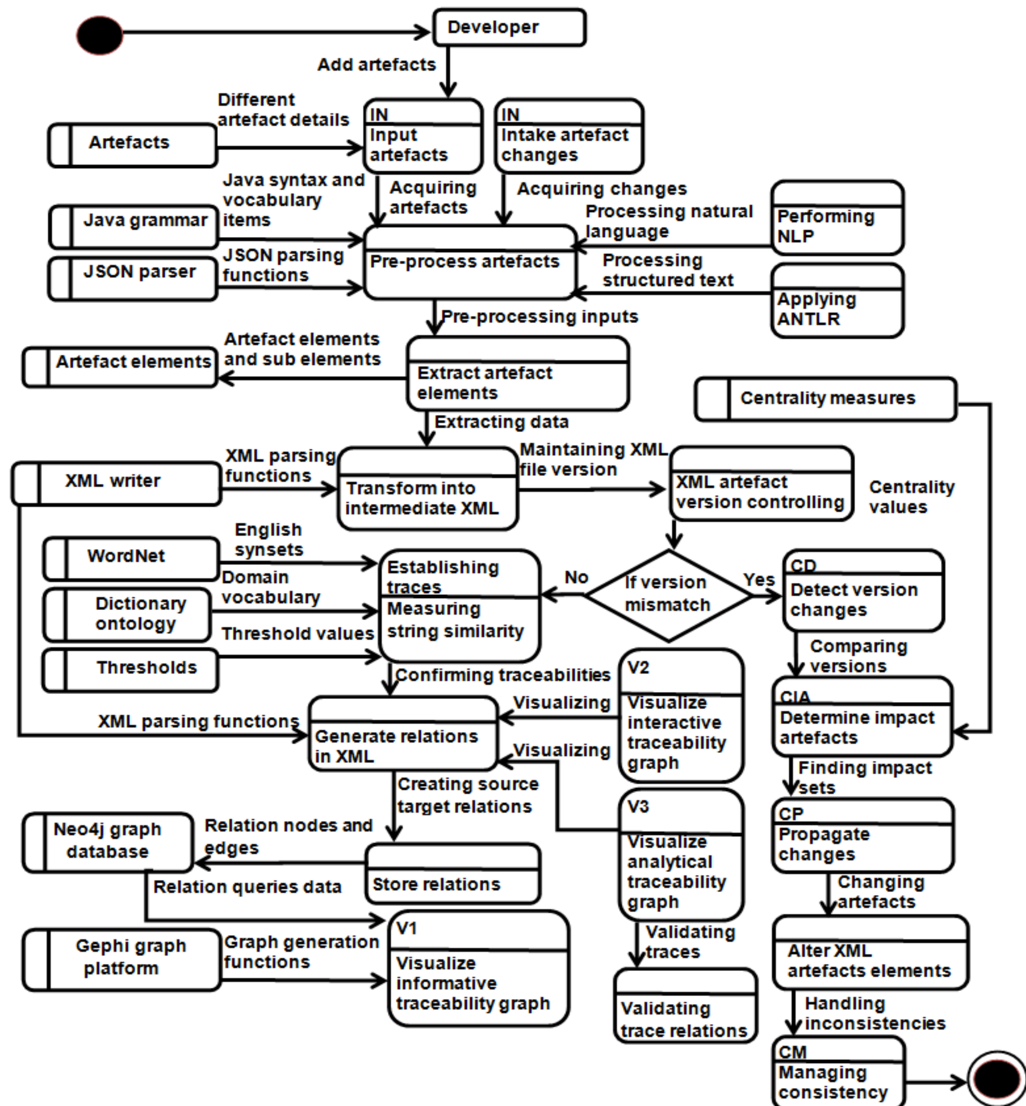


Figure 3-5 : Extended SAT-Analyser detailed workflow

3.1.6 System class structure

The class diagram of SAT-Analyser tool is shown in Figure 3-6. The classes *GUI_Manager* and *DB_Controller* handle user interface and database, respectively. The superclass *Artefact_Manager* intakes the artefacts from the user and initiates the data *Pre_Processor* and *Data_Extractor* classes. There exist subclasses inherited from *Artefact_Manager* for each type of artefacts. There can have many pre-processor and extractor sub-modules as the inputs are in heterogeneous. The extracted artefact elements and sub-elements are handled by the *Artefact_Elements* class. Traceability establishment is done by class *Traceability_Generator* which has a composition relationship with *Relation_Manager* which manages the established trace relations among artefact elements. Traceability visualization is based on the established relations and is provided in three different kinds of views namely; informative view, interactive view and analytical view. Traceability evaluation is performed by class *Trace_Validator* that is related with analytical visualization type. The *Change_Detetor* class monitors the database changes of artefact elements. It triggers *Impact_Management* class whenever a change is identified for impact analysis of affected items. *Change_Propagator* is used to propagate the changes for affected artefacts by mandatorily using the class *Graph_Traversal*.

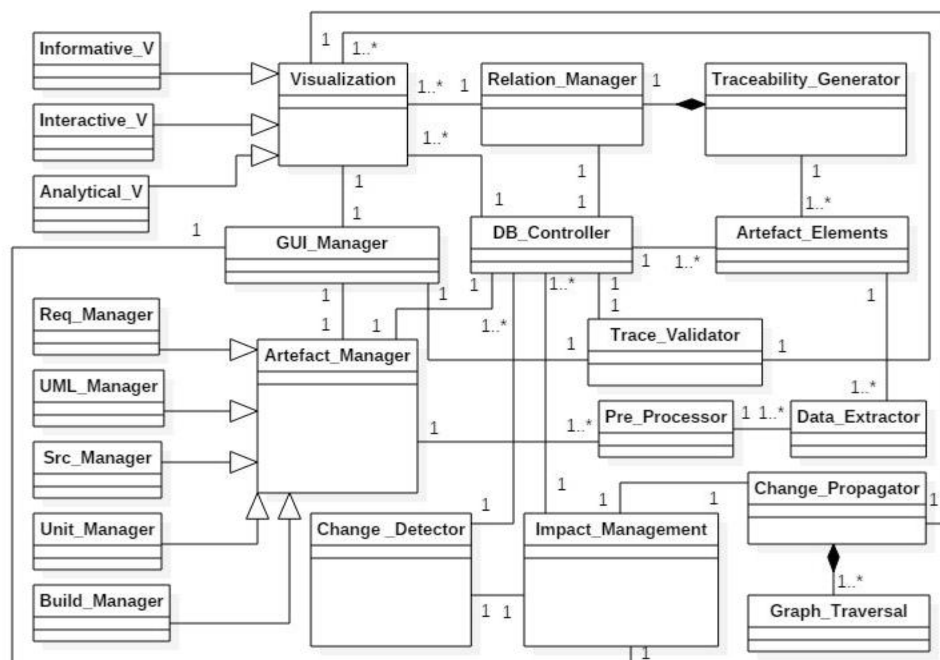


Figure 3-6 : Extended SAT-Analyser class structure

3.2 Traceability establishment

3.2.1 Data pre-processing of SAT-Analyser tool

The requirements, design, source code, unit test script and build script artefacts are considered for the artefact traceability process of this system. Therefore, those artefacts are addressed in the data pre-processing component as the input items. The, requirement documents are in document format (.docs) or text file format (.txt), design diagrams in metadata-JSON file format (.mdj) following UML notation, source codes in Java programming language (.java), unit test artefact in JUnit unit test script files (.java) and Maven build script is in a pom file (.xml). The pre-processing of up to source code artefact is from the existed initial version of the SAT-Analyser tool.

The NLP module is responsible for pre-processing data and extracting information from requirement documents, story cards that are written in generic natural language English. It is designed to extract the artefact elements such as classes, methods, attributes and relationships from the stated requirements. The *Stanford CoreNLP* is used to process the natural language statements to produce a base form of text. Thus, its submodules Part-of-Speech tagger, parser, Named Entity Recognizer (NER) and Anaphora analysis (coreference resolution) are also involved for better pre-processing of the requirement data.

Initially, the NLP module tokenizes the pronouns of a given requirement document as the task of tokenization. Then, Anaphora analysis is conducted to identify the coreferences in given sentences before extracting the artefact elements. Consequently, the extraction of nouns is performed in order to detect the artefact elements among the processed requirement statements. A parse tree is internally generated with the aid of used *Stanford CoreNLP* to obtain detailed granularities of sentences by using POS tagging.

Correspondingly, the classes, methods, attributes and the relations among them are extracted as the major artefact elements. A set of identification rules are involved since differentiating among classes and attributes is problematic as both are nouns.

Therefore, if a verb phrase is following noun phrases in a sentence those nouns are extracted as class names. Thus, attributes are extracted from nouns and adjectives if the nouns and adjectives in a sentence are not following a verb phrase. Also, methods are extracted from the noun phrases associated with class names. Moreover, the relationship identification is defined to extract the association and generalization type of relationships. Accordingly, a rule-based approach is followed in the requirements data extraction. Then, the morphological analysis is performed to convert the contents into a root form for redundancy elimination purpose. Hence, the stemming analysis and redundant elimination are conducted in retrieving a unique set of requirement data.

The design tools *StarUML* and *Modelio* are selected based on their ability to store design diagram files as a model file (.mdj) and the export capability to generate XMI (.xmi) and UML (.uml) formats respectively. Thus, it can be used to data extraction via a JSON reader since diagram information and class diagram concepts are well stored in a JSON format in those selected design tools. Additionally, two pre-defined dictionaries are also integrated with this parsing module in order to fine-tune the data extraction process by eliminating non-realistic extractions in the context of class diagrams.

The source code parsing module pre-processes the source code artefact data from the project workspace. The tool ANTLR is used to generate lexers, tokens and relevant listener classes for the Java language. An abstract syntax tree of a source code file is generated by ANTLR and is further processed using the Java Grammar. The source code data extraction is done by traversing through the syntax trees using the tree walker integrated with ANTLR in identifying class declarations, methods, attributes, generalization and association relationships. Moreover, they are designed to be stored in a temporary Neo4j database.

The goal of Algorithm 3:1 is to pre-process the input artefact resources and extract the data accurately. The supportive input artefact types are requirements, UML design class diagram, Java source code, JUnit test files and Maven build script at this stage of the SAT-Analyser.

Algorithm 3:1 Data pre-processing

Require: Software artefacts (requirements, design, source code, test script, build script)

Ensure: associating input data to a traceability project

```
1.  input: artefact: a
2.  if (a== requirements)
3.      a_req = NLP_module(a)
4.  if (a== design)
5.      a_uml = UML_parser(a)
6.  If (a== source code)
7.      a_src= SRC_parser(a)
8.  If (a== unit test)
9.      a_ut= UT_parser(a)
10. If (a== build script)
11.     a_bs= BS_parser(a)
12. a_xml = Convert_to_XML(a)
13. If (all 5 a_xml exists)
14.         Build project structure module
15.         Make folder structure
16.         Initiate graph files
17. Else
18.         Notify failure
19. output: new artefact traceability management project
```

Accordingly, if the type is 'requirements', the artefact is forwarded to process via NLP_module algorithm and if the artefact type is 'design' it is forwarded to UML_parser algorithm. Else, the artefact is forwarded into the SRC_parser if the input artefact type is 'Java source code'. Similarly, if the artefact is a 'unit test' type as a set of JUnit test script files, then UT_parser is used and if the artefact is a build script in the form of a Maven pom.xml file, BS_parser is invoked. The extracted artefact data are processed through the Convert_to_XML algorithm to convert the data into a common format using XML writers. A new SAT_Analyser project is created only if all artefact elements containing XML files are successfully found. Thus, the expected final outcome in this algorithm is the creation of a new SAT-Analyser project.

The goal of Algorithm 3:2 is to process the requirements artefact data provided in English natural language via the *.txt* or the *.doc* file formats. The input is processed through NLP activities to extract the requirements related artefact elements.

Algorithm 3:2 NLP_module

Require: Software artefacts: requirements in natural language
Ensure: pre-process requirements artefact data

```
1.  input: requirements artefact: a
2.  while (a)
3.        tokanization
4.        Anaphora analysis
5.        Data extraction
6.              Return classes, methods and attributes
7.  if (classes, methods, attributes exists)
8.        morphological analysis
9.        Stemming analysis
10. Redundant elimination
11. output: pre-processed requirements artefact
```

The tokenization is performed to clear the statements by segmenting the running text into words and sentences. The anaphora analysis is done to achieve coreference identification. Thus, the pronouns are identified and re-organise the requirement statements. The names of classes, methods, attributes and relationships are extracted. A rule-based approach is designed such as class rules, method rules, attribute rules and relationship rules. Once a certain set of elements are collected, the morphological analysis along with stemming analysis is conducted on the outcome. The motive is to transform the extracted requirements artefact elements into a further base form to eliminate redundancies due to plurality.

Algorithm 3:3 UML_parser

Require: Software artefacts: design in UML class diagram
Ensure: pre-process design artefact data

```
1.  input: design artefact: a
2.  if (a== UML class file)
3.        process via StarUML reader
4.        Process via Modelio reader
5.  Data extraction
6.        Return classes, methods and attributes
7.  output: pre-processed design artefact
```

The motive of Algorithm 3:3 is to pre-process the design artefacts such that the class diagrams designed in UML notation. Only the diagrams origin from the tools

StarUML and *Modelio* are processed since they highly contained the class diagram details in JSON or the model based formats which eases the processing. Therefore, a StarUML reader and a Modelio reader are designed to identify encoded details in a class diagram file. The identified details such as class names, methods and attributes are extracted as the design elements.

Algorithm 3:4 SRC_parser

Require: Software artefacts: source code in Java programming language
Ensure: pre-process source code artefact data

1. **input:** source code artefact: a
2. if (a== Java source files)
3. process via ANother Tool for Language Recognition
4. Process via Java grammar
5. Data extraction
6. Return object-oriented classes, methods and attributes
7. Store in Neo4j DB
8. **output:** pre-processed source code artefact

The pre-processing of source code artefacts is the goal of the Algorithm 3:4 and the input must be a set of Java source code files. The ANTLR tool is involved to generate Java Grammar based syntax trees, to traverse the tree using its tree walker and to make use of the listeners for tracking. Hence, the class declarations, methods, attributes are extracted with the aid of the ANTLR capabilities. The extracted source code elements are stored in a Neo4j graph database temporarily.

Algorithm 3:5 UT_parser

Require: Software artefacts: unit test in JUnit test scripts
Ensure: pre-process unit test artefact data

1. **input:** unit test artefact: a
2. if (a== JUnit test script)
3. process via ANother Tool for Language Recognition (ANTLR)
4. Process via Java and JUnit grammar
5. Data extraction
6. Return JUnit classes, methods and attributes
7. Store in Neo4j DB
8. **output:** pre-processed unit test artefact

The pre-processing of unit test artefact provided in JUnit test scripts is the motive of the Algorithm 3:5. The input to this algorithm must be a set of JUnit test script

files. Similar to the SRC_parser, ANTLR tool is involved in the process to generate Java Grammar. The extracted unit test artefact elements are stored as the output of this algorithm in a Neo4j graph database.

Algorithm 3:6 BS_parser

Require: Software artefacts: build script in Maven dependency pom.xml file
Ensure: pre-process build script artefact data

1. **input:** build script artefact: a
2. if (a== build script file)
3. process using XML data extraction
4. Return build script name, plugin dependency names
5. Store in Neo4j DB
6. **output:** pre-processed build script artefact

The pre-processing of build script artefact in Maven dependency file as a pom.xml file is the intention of the Algorithm 3:6. The Maven build script; pom.xml files are in a *.xml* tag structure. Therefore, directly the XML data extraction is performed on pom.xml file to extract build script (project) name and dependencies/ plugins names as data. Then, the extracted build script artefact elements are stored as the output of this algorithm in a Neo4j graph database.

3.2.2 Input to XML conversion

The all five artefact processing modules write the pre-processed and extracted artefact data in XML format using XML writers separately. A new traceability project is created only if all processed requirement, design, source code, test script and build script artefact XML formats are available. The extracted artefact data are processed through the Convert_to_XML algorithm in order to convert the data into a common format using XML writers.

The primary motive of this Algorithm 3:7 is the common format conversion of pre-processed and extracted artefact related data. Therefore, the input to this algorithm is designed to be the pre-processed requirements, design, source code, unit test and build script artefact elements. The XML format is selected as the common conversion format as XML structures are helpful in building complex graphs with readability over others. Hence, all pre-processed artefact element data

are written using XML writers. The outcome of this algorithm is a separate XML file for each artefact type that contains relevant extracted artefact data.

Algorithm 3:7 Convert_to_XML

Require: pre-processed artefact data

Ensure: Convert pre-processed software artefact to a common format

1. **input:** pre-processed artefact: a
2. if (a== requirements OR design OR source code OR unit test OR build script)
3. XML writer (pre-processed classes, methods, plugins, attributes)
4. Return a.xml
5. **output:** XML conversion of an artefact

3.2.3 Traceability generation

The pre-processed and extracted artefact elements are used for the traceability link building among the addressed artefact types that are software requirements, design, source code, unit test and build script. The *WordNet* is heavily involved in the mapping purpose in this trace process (Kamalabalan et al., 2015). Moreover, a self-generated dictionary is used for similarity calculations between artefact elements which are helpful to manage the traceability links. The similarity is calculated among two strings at a time where the strings represent the extracted artefact element data and data stored in the *WordNet*. The Levenshtein algorithm is applied for that purpose and it outputs a distance value called 'Edit Distance Value'. It denotes the minimum number of edit operations required for transforming a string into the other string which signifies the similarity among two strings. The most prominent edit operations performed include the insertion of a character into a string, deletion of a character from a string and character replacements. Accordingly, a threshold value is defined as 0.85 for the similarity calculation based on the edit distances. Thus, the artefact elements that exceed the defined threshold are considered as having a higher similarity and are mapped together. Here the above mentioned self-generated dictionary fine-tunes the performance of the matching artefact elements. The threshold-based mapping refers to the relationship building process where the traceability links are generated and established.

Correspondingly, a semantic network is created for word matching through the build relationship module of this traceability link generation component of the SAT-Analyser. The distance between nodes is measured in the semantic network to identify the matching percentage. Importantly, the process so far is designed to be automated. Also, the nodes are allowed to be manually adjusted if an inappropriate lower matching percentage is achieved among two artefact elements. Thus, a user can generate a new traceability link for the appropriate relevant elements manually in the interface level. Nevertheless, the manual link creation in every project is time-consuming and inefficient. Hence, the self-generated dictionary is triggered to resolve this issue. It keeps track of the artefact element words in the built semantic network continuously. For an example, the following network shown in Figure 3-7 would be created in considering the words *Bank, Library, Online, Offline, Student* and *Cashier*.

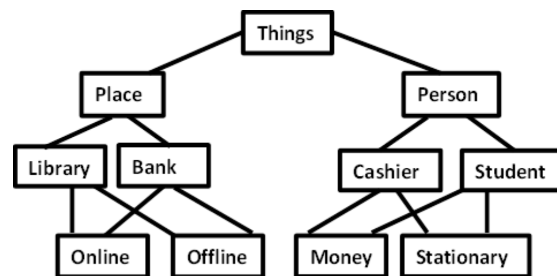


Figure 3-7 : Semantic network for words

Each word is stored with its relevant similar words and properties. The properties include name-value pairs and a word's parent class information. An API provided by the *Apache Jena* Library is involved to build the ontology model in implementations. Next, RDF is identified as a data format that more accurately describes a metadata data model. Thus, it is used to record information as one of the building block standards of the semantic web. However, RDF can be represented in various different formats like JSON and XML. Hence, the artefact specific XML file conversions are also mapped into a pre-defined relationship XML model as shown in Figure 3-8.

```
<Relations>
    <Relation id="1">
        <SourceNode>...</SourceNode>
        <TargetNode>...</TargetNode>
    </Relation>
    <Relation ...>
        ...
    </Relation>
</Relations>
```

Figure 3-8 : Pre-defined relationship XML model

The XML artefact models are separately generated for all supported three types of software artefacts namely, requirements, UML class diagrams, Java code, JUnit test files and Maven build script. The relationships among artefact elements are recorded and modified based on change detections and change propagation results during the software development.

Algorithm 3:8 Traceability link generation

Require: Software artefacts
Ensure: Building relationships among artefacts

```
1.   input: artefacts: a
2.   for (a )
3.      get synonyms from WordNet
4.      String comparison for classes, attributes, methods, relationships
5.     matchDistance = Jaro Winkler algorithm similarity (element1,element2)
6.         If (matchDistance > = 0.8 and < = 1.0)
7.             Build trace link among two artefact elements
8.         Else
9.             editDistance= Levenshtein Distance algorithm
10.                distance (element1,element2)
11.            matchDistance = 1 - editDistance
12.            If (matchDistance > = 0.8 and < = 1.0)
13.                Build trace link among two artefact elements
14.     XML Writer (nodes, links)
15.  output: XML conversion of artefact traceability links (Relations.xml)
```

Algorithm 3:8 handles the pre-processed artefact data towards the traceability link generation. It ensures the relationship creation among the extracted artefacts that are input for the algorithm. Then, a string similarity computation is performed via the Jaro-Winkler algorithm (“Jaro Winkler Distance,” 2017) and Levenshtein Distance algorithm (“Levenshtein-Algorithm,” 2017) using the *WordNet* synonyms and pre-defined dictionary ontology. The Jaro-Winkler algorithm considers that the differences in the start of the strings are more significant than differences close to the end of the strings, while Levenshtein algorithm computes

the number of modifications needed to transform a string to another. Therefore, the Jaro-Winkler algorithm is selected due to its efficiency compared to the Levenshtein distance algorithm. 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.

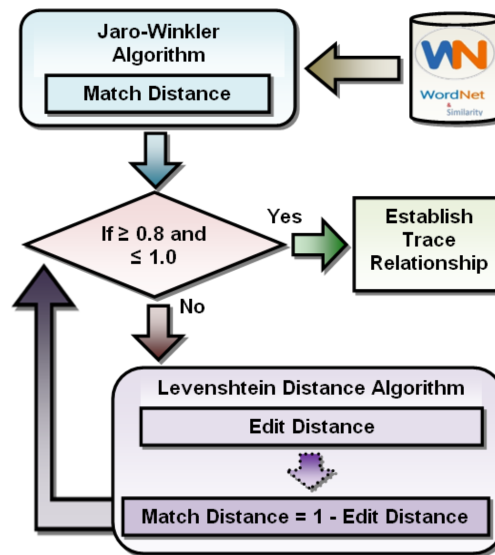


Figure 3-9 : Traceability link generation component

Figure 3-9 illustrates the abstract processes involved in the traceability establishment process. 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 (Olsson, 2015) and converted into a predefined XML structure.

3.3 Traceability visualization

Visualization of the established traceability links is essential in decision making during the SDLC. It allows users to browse, explore and manage the relationships among software artefacts which are useful in recovering from artefact degradation. However, the number of artefact elements to be represented is a major challenge in this context of visualization due to high visual clutter. Another challenge is the instant modification facility of a built visualization schema based on the change detection and change propagation outcomes.

Algorithm 3:9 Visualization

Require: Software artefact traceability links

Ensure: Visualise and represent artefact traceability links

1. **input:** artefact traceability links: k
2. Store relation nodes in Graph DB
3. while (DB is Not Null)
4. Graph generator (Graph DB)
5. Visualize default traceability graph
6. XML_R =Obtain Relations.xml
7. $JSON_R$=XML to JSON convertor (XML_R)
8. D3 visualization graph generation ($JSON_R$)
9. $Python_R$=XML to Python list convertor (XML_R)
10. Python visualization graph generation ($Python_R$)
11. **output:** Artefact traceability graphs

The visualization Algorithm 3:9 is responsible for generating the representation outcomes of established traceability links among artefacts. The outcomes of the traceability link generation component are mainly involved as the input for this component along with the change detection components results. Thus, the artefact relationship links are obtained as the direct input. The extracted and traceability established artefacts and relationships are stored as nodes and links in a Graph Database. The Neo4j database is selected in this purpose as it is supportive for graph-based representations. The graph generator module is triggered to process the relation nodes obtained from the graph database into graph-friendly formats such as Gexf (Graph Exchange XML Format) files. The visualization module represents the artefact elements as nodes and the relationships or the built traceability links among them as edges among the nodes. Concurrently, the relations XML version that consists of all traceabilities is obtained and converted into a JSON format and Python list format for two variations of visualizations.

The visualization component is responsible for providing the output of the system. It is based on the graph representation techniques following the insights from literature. The traceability visualization enhancement in this research is performed in two additional aspects such as analytical and interactive representations apart from the existed static informative visualization type in the initial version of SAT-Analyser. The component is modularized as depicted in Figure 3-10.

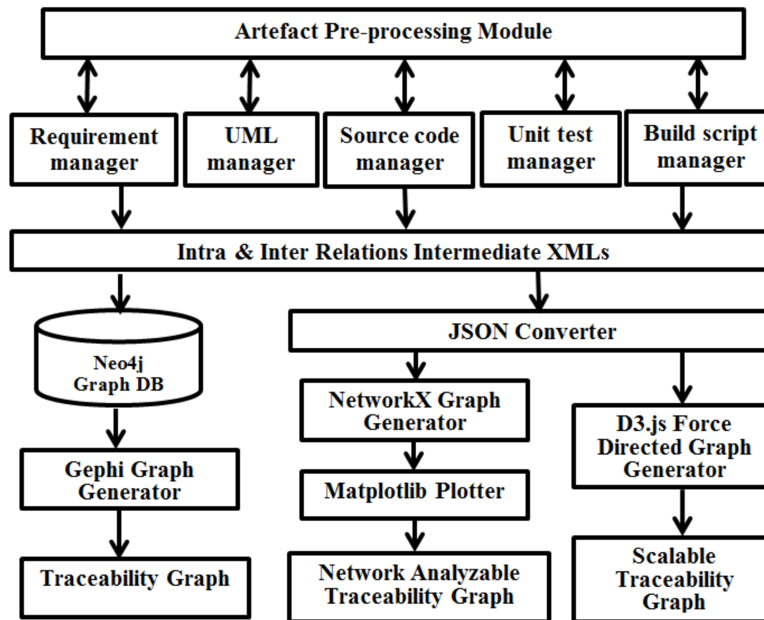


Figure 3-10 : Visualization component

The established traceability links in the traceability link generation component are used in this visualization component. The inner modules are designed to manage each type of artefact intra-relations. Thus, relationship management is defined for each requirement, UML class diagram related design, source code, test files and build script. Finally, all the finalized relation nodes that consist of the artefact elements relationships are stored in the Neo4j graph database. Additionally, the relations are stored in JSON format for the use of enhanced visualization methodologies such that one for the interactivity purpose and one for traceability analysis purpose that are described in this section.

3.3.1 Default SAT-Analyser informative traceability visualization

The default traceability visualization of SAT-Analyser contains the Neo4j graph database ("Graph Visualization-Neo4j," 2018) and Gephi graph generation platform ("Gephi," 2017). The Neo4j database is selected for this purpose due to its support towards graph-based visualizations. It is identified to be capable of handling a larger number of nodes, relationships among them with properties. Moreover, the graph structure of it is flexible and not a defined schema that follows a semi-structured schema. It follows a simple set of rules in a key-value pair based manner. The graph generator module then obtains the relation nodes and converts them into graph-friendly formats including Gexf files. The *Apache*

Lucene Indexing API is involved for the purpose of searching and locating nodes and edges among relation nodes in the Neo4j graph database since the Neo4j lacks the ability of indexing. Then, the visualization is performed with the node and links using the Java library called *Gephi-toolkit* API ("Gephi," 2017).

This view facilitates a general representation with colour codes for nodes and edges to reduce the scalability issues provided with a separate information pane on the right-hand side of the window to elaborate details of each node. Hence, this can be categorized more as an informative visualization type. The naming conventions used in this traceability graph visualization are as follows; RQ-Requirement, D -Design, SC-Source Code, UT-Unit Test, BS-Build Script, _M-Method/ Function, _F-Field/ Attribute. Each artefact type is illustrated and used with a unique number next to each artefact element or sub-element for unique referencing. In addition, results can be filtered based on artefact types or edge types from the menu as shown in Figure 3-11. Those options are as follows;

- Full graph view with artefacts and their links.
- Edge filtered view for the relationship among the identified classes, attributes, operations for each of the artefacts in requirements, design, code, test script and build script.
- Artefact filtered views for each one of 5 artefact types separately.

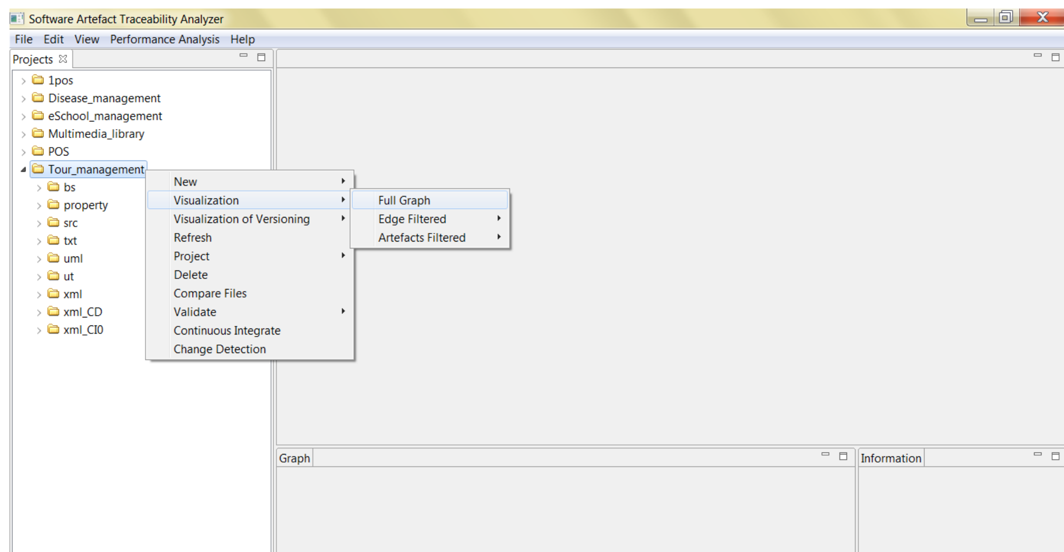


Figure 3-11 : Default SAT-Analyser Traceability visualization menu

The artefact level representations are offered with each artefact. Different filtered views are facilitated to avoid huge visual clutter. Figure 3-12 illustrates a selected section of the generated graph view. The length of the edges denotes the strength of the similarity between every two nodes. Larger the string comparison value means shorter the length of the corresponding edge. For example, in Figure 3-12, 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 comparatively lengthy as the UML class diagram artefact has used the class name with naming conventions. Figure 3-13 shows a portion of artefact filtered view for requirement artefact type that illustrates a particular requirement and its associated methods, fields' relationships.

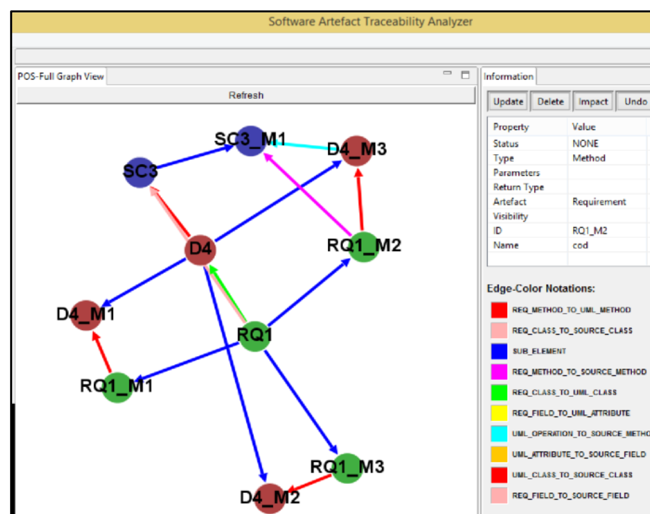


Figure 3-12 : Default SAT-Analyser visualization full graph view

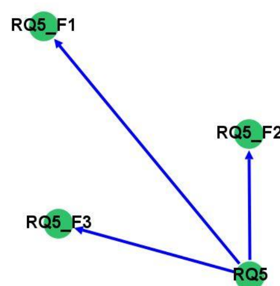


Figure 3-13 : Default SAT-Analyser visualization requirement artefact filtered view

3.3.2 Analytical traceability visualization

The analytical traceability visualization is mainly targeted for the purpose of traceability outcome validation that is in detail described in the subsection 3.6.2. Traceability results validation is important to proceed with decision making. On the other hand, visualization is vital to convey validation results when the number of items to be validated increases. Therefore, analysing the traceability outcomes and visualizing the analysed traceability outcomes is added to the SAT-Analyser.

The existed SAT-Analyser default traceability visualization mechanism was not supportive to traceability validation techniques used in this research such as network analysis. Thus, a newer traceability visualization module is added as Python-based analytical traceability visualization. The XML relations file is involved in this variation. The network analysis functions in Python *NetworkX* libraries ("NetworkX," 2018) that are used for the traceability results validation in this work are mapped with *matplotlib.pyplot* libraries to render into a graph in Python. *NetworkX* is widely used for the creation, manipulation and analysing structure dynamics, and function of complex networks due to its ability of painlessly slurp in large non-standard data sets. Also, *Matplotlib* is a recognized Python 2D plotting library capable of producing quality figures and the *pyplot* module provides a MATLAB-like interface via a set of functions familiar to MATLAB users ("Matplotlib," 2018).

This analytical visualization is also used in the change impact analysis and change propagation process (see section 3.4) as it's based on the network analysis techniques involved in traceability validation. Figure 3-14 shows a general analytical graph view obtained during a SAT-Analyser validation task. The basics of artefact category naming convention of default informative visualization technique and the colour codes for nodes are preserved in this view too. In addition, zooming, recording zoom levels, moving, saving as image features are facilitated in this view. The more analytical aspects of this view based on the network analysis results are described in the subsection 3.6.2 and chapter 4. Figure 3-15 shows a visualization of traceability results based on Eigenvector centrality measure by applying heat maps. The darker nodes depict lesser important nodes

and lighter nodes represent the higher importance nodes in the network. Figure 3-27 in subsection 3.4.5 illustrates an example of change propagated analytical traceability graph view based on impact values.

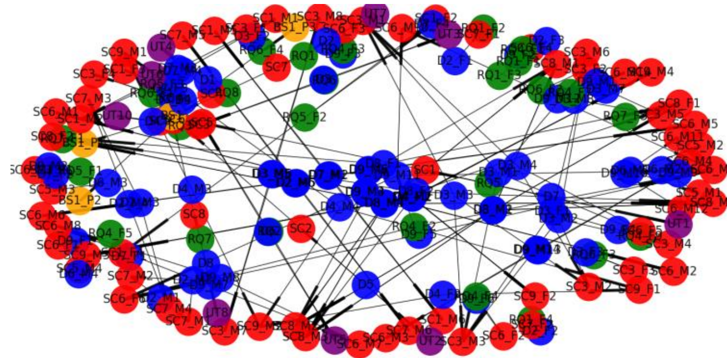


Figure 3-14 : General analytical traceability graph

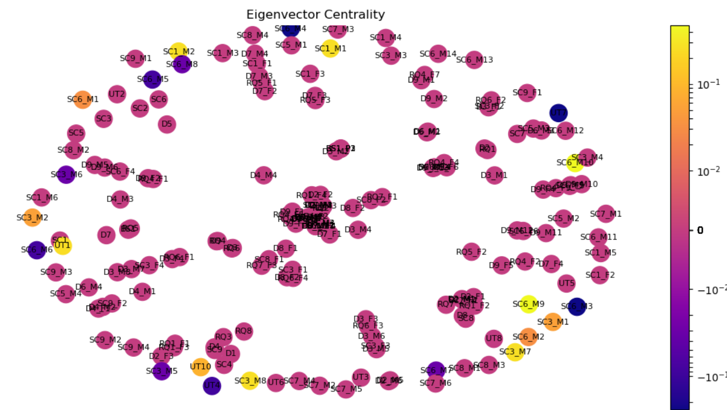


Figure 3-15 : Analytical traceability graph in traceability validation

3.3.3 Interactive traceability visualization

The purpose of another lightweight interactive traceability visualization module is as an enhancement for the SAT-Analyser tool. The XML relations file is involved in this variation by converting it into JSON format. The JavaScript library *D3.js* (Data-Driven Documents) technology, well recognized for manipulating documents based on data is used in generating the interactive behaviour on a browser view (“D3.js,” 2018). It visualizes data using HTML, SVG, and CSS by emphasizing on web standards. It provides full capabilities of modern browsers by combining powerful visualization components using DOM manipulation based on a data-driven approach instead of depending on a proprietary framework.

Figure 3-16 represents a part of an interactive traceability graph view that has preserved basics of colour codes and artefacts category naming conventions to maintain the consistency among all visualization types in SAT-Analyser. Mainly, the hovering features on nodes and edges that encapsulates more details of nodes/ edges without colliding with the view, double-clicks on nodes to highlight neighbours for better readability, re-positioning of nodes/ edges on the network via dragging are facilitated in this view. This interactive visualization is also facilitated at CIA, change propagation results representation and at traceability results validation results. Additionally, Figure 3-28 illustrates a change propagated interactive traceability graph view in subsection 3.4.5.

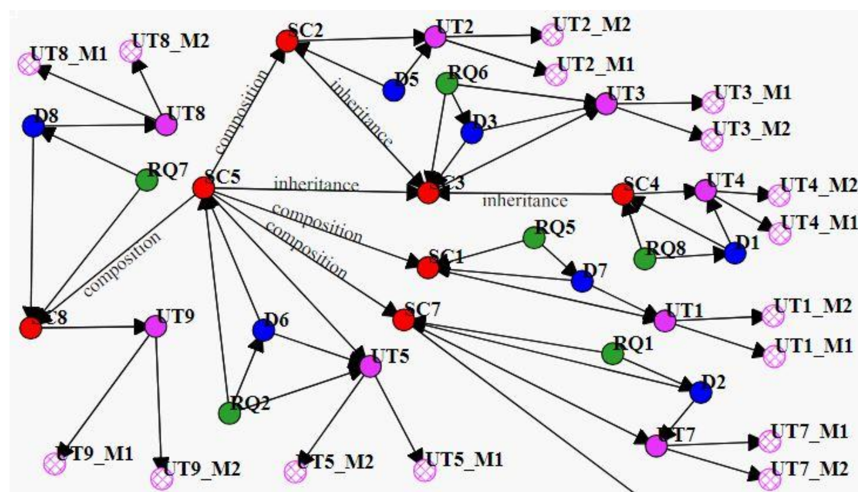


Figure 3-16 : Interactive traceability graph view

3.4 Impact analysis and change propagation

3.4.1 Identification of strengths of artefacts and relationships

The list of affected nodes can be obtained from the outcome of the change detection process (see subsection 3.5.1). For, example if a new node is added, then there can be new links created with it and other nodes. Thus, for each change, there can be impacts in different degrees. The impacts can be explored using graph representations.

There can be highly affected artefact elements which are highly impacted and lesser ones. Thus, a measure is required to identify that. For that purpose, the graph nodes and edges can be assigned with weights. In assigning weights, a particular measurement is required with a pre-defined static or a dynamic one that

differs from one traceability project to another according to the node-link count. The concept of centrality is widely used in Social Network Analysis and has found different realizations regarding proper measures. One of the centrality measures; Eigenvector centrality also known as Eigencentrality (Borgatti, 2005) that expresses the influence level or the importance of nodes in a network is identified as a useful measure in this purpose (Jashki et al., 2008). Google search engine also uses this to rank the search results (Fernández, 2008). In definition, the EVC for node *i* as in the equation (3.1),

$$Ax = \lambda x \tag{3.1}$$

Where A represents the adjacency matrix of the graph network G having Eigenvalue λ. There is an identical solution if Eigenvalue λ is the largest associated with the Eigenvector of the adjacency matrix A according to the Perron-Frobenius theorem (Newman, 2010).

A fixed scale is defined with three margins using the EVC values of nodes and edges in designing the weight system of CIA component of SAT-Analyser (Iresha D. Rubasinghe, Meedeniya, & Perera, 2018). The designed weight assignment system consists of two sections; one for nodes using an *Influential Factor* and *weights* for edges using that *Influential Factor*. The base for this mathematical model is EVC.

$N = Node$ $\quad$ $E = Edge$

$n = count$ $\quad$ $f = Influencial\ Factor$

$w(N_i) = weight\ of\ i^{th}\ Node$ $\quad$ $w(E_i) = weight\ of\ i^{th}\ Edge$

f_i^N=$Influencial\ Factor\ of\ i^{th}\ Node$ $\quad$ f_i^E=$Influencial\ Factor\ of\ i^{th}\ Edge$

$$w_{min}(N) = \ min\ [\forall N, EVC(N)] \tag{3.2}$$

$$w_{max}\ (N) = \ max[\forall N, EVC(N)] \tag{3.3}$$

$$w_{avg}\ (N) = \frac{\sum_{i=1}^{n} EVC(N_i)}{n} \tag{3.4}$$

A node's weight is defined based on its EVC value. The lowest weight on the scale is the minimum EVC value across all the nodes (3.2). Similarly, the maximum weight is the largest EVC value (3.3). Also, the average on the scale is decided by considering the average EVC value with respect to the total EVC value

of all nodes and the node count (3.4). Thus, for any i^{th} node, the influential factor is assigned *low* if having the weight in the range of $[w_{min}\ ,\ w_{avg})$ and otherwise, the influential factor can be assigned *high* if the weight is within the range of $[w_{avg}\ ,\ w_{max}]$ as derived in the conditional equation (3.5).

$$
\begin{aligned}
&\forall i, \\
&\quad If\ (w(N_i) \geq w_{min}\ and\ w(N_i) < w_{avg}) \\
&\quad\quad f_i = \text{'Low'} \\
&\quad If\ (w(N_i) \geq w_{avg}\ and\ w(N_i) \leq w_{max}) \\
&\quad\quad f_i = \text{'High'}
\end{aligned}
\tag{3.5}
$$

Similarly, the weight system of the trace links or the edges is based on the influential factor definition of nodes. Each edge is mandatorily associated with a source node (starting point) and a target node (endpoint) as the traceability management of the tool SAT-Analyser is output as a directed graph. An assumption is made based on the directed behaviour of the traceability declaration.

Assumption 01: The weight of an ith edge (E_i) is identical to the weight of the source node (N_i^{source}) of that particular edge as shown in Figure 3-17 and equation (3.6).

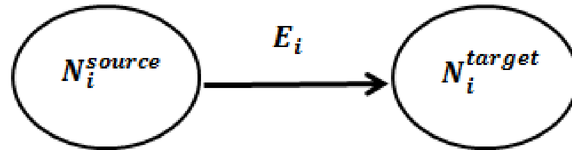


Figure 3-17 : Node-edge direct connectivity

$$\forall i, w(E_i) = w(N_i^{source}) \tag{3.6}$$

Thus, the *Influential Factor* of edges can be defined as the conditional equation (3.7).

$$
\begin{aligned}
&\forall i, \\
&\quad If\ (f_i^N = '\ High')\\
&\quad\quad f_i^E = \text{'High'} \\
&\quad If\ (f_i^N = '\ Low')\\
&\quad\quad f_i^E = \text{'Low'}
\end{aligned}
\tag{3.7}
$$

The influential factor of any edge can be obtained by its source node's influential factor where the edge starts from. If any node has a higher influential factor, its outgoing edges have a high influential factor value. Consequently, the scale system for the edge weight can be obtained similarly to nodes in determining the weight of edges (3.8), (3.9), (3.10).

$$w_{min}(E) = min\,[\forall E, EVC(E)] \quad (3.8)$$

$$w_{max}\,(E) = max[\forall E, EVC(E)] \quad (3.9)$$

$$w_{avg}\,(E) = \frac{\sum_{i=1}^{n} EVC(E_i)}{n} \quad (3.10)$$

There can be at most two scenarios for any pair of artefact nodes that are associated with a traceability edge.

I Scenario 1: Outgoing traceability edge from a low influential artefact node

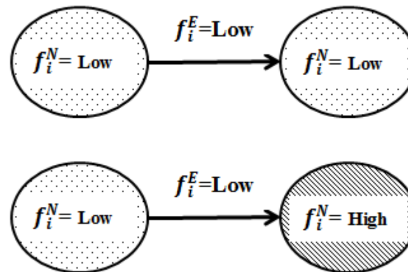


Figure 3-18 : Node-edge scenario 1

In the scenario illustrated in Figure 3-18, the target artefact can become either a low influential or a high influential node. Adhering to the stated assumption 01, whenever the starting point is a low influential node, then the outgoing edges of it get a low influential factor resulting a lower impact. For instance, if the source artefact node is a Maven build script file which certainly would hold a low EVC value, the outgoing trace links from it are certainly the declared dependant plugins which also have a low influential factor value. In that case, a traceability link between the Maven build script file and it's any of the plugins contains a low impact such that any change applied to a build script is not crucial to forward propagation. In contrast, if the target artefact node is high, then an incoming change from a low influential artefact cannot generate a significant impact. Thus, the traceability link gets assigned a low influential factor in accordance with the assumption 01 which becomes true for this scenario 1.

II Outgoing traceability edge from a high influential artefact node

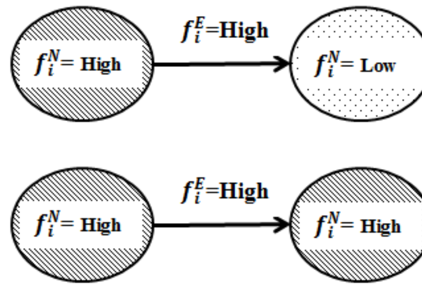


Figure 3-19 : Node-edge scenario 2

In the second scenario illustrated in Figure 3-19, the target artefact is either a low influential or high influential node. Adhering to assumption 01, the trace links starting from a high influential artefact node contain a high influential factor value with a considerably higher impact. For instance, if the starting node is type of a source code artefact class which most probably holds a high value of EVC, the outgoing trace links from it can reach to source code artefact attributes/ methods/ unit test artefact/ build script artefact which would mostly have a high influential factor value by creating a parent-child dependant nature. Hence, the traceability link between this source code class artefact and its attribute/ method/ unit test artefact/ build script artefact gets a significant high impact because any change applied to the source code class is having a higher possibility of affecting to its dependant natured endpoint artefacts. Similarly, if the target artefact is a high node, then an incoming change from a high influential artefact can considerably impact on it. Thus, the trace link can be assigned with a high influential factor according to the stated assumption 01 that becomes true for this scenario 2 as well.

Accordingly, the impact is designed to be measured and would propagate forward through direct edges until a low impact node is reached. As the outgoing edge of a lower node is also low, the change impact propagation would terminate there without further moving forward.

3.4.2 Impact analysis process: workflow

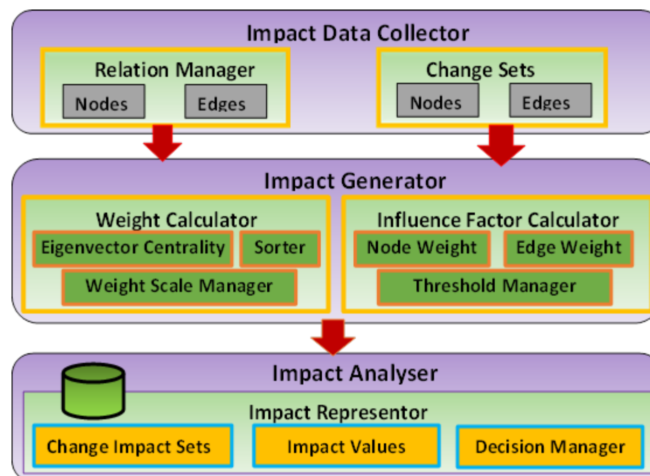


Figure 3-20 : Impact analysis component workflow

Figure 3-20 represents the designed change impact analysis component of the SAT-Analyser tool. This resides as a sub-component within the CI component with the change detector and change propagator. The outcome of the change detector that is changesets becomes an immediate input for this CIA component. Also, the outcome of the traceability establishment component within the business logic layer is another input for this where the relations with their source and target are obtained. Accordingly, the *Impact Data Collector* gathers all the required sets of nodes and edges. *Impact Generator* is the core of this CIA overall component where the mathematical model is handled. The *Weight Calculator* assigns a weight to each node and edge using Eigenvector centrality value of each node and edge. The *sorter* module finds the minimum and maximum valued EVCs and based on that the *Weight Scale Manager* declares the weight scale. The *Influence Factor Calculator* provides a two-level influence factor for each node and edge that is solely considered for impact analysis. *Impact Analyser* performs the impact results representation as change impact sets and their respective values. Also, the *Decision Manager* module triggers that impacted results to Change Propagator component to further navigate the changes to other remaining nodes and edges.

3.4.3 Impact analysis process: pseudo code and implementation details

Impact analysis process can be initiated only through change detection as they are sequential activities in this problem domain. Therefore, once change detection is

performed the impact analysis option is available in the changesets window. If the change set is null, still the impact analysis option helps to make the system to a more consistent level by producing an updated Relations.xml file.

Algorithm 3:10 elaborates the flow of events in the impact analysis process among the modules in the impact analysis workflow diagram. A changeset must exist for this to proceed. Then, the Relations.xml that concatenates all types of artefact relationships together in the SAT-Analyser must be prepared for the current version of the integration.

Algorithm 3:10 Impact analysis

Require: Detected change set among two versions
Ensure: Impact value of a change in changeset

```
1.  input: A changeset
2.  If change set node is not null
3.      Relation Manager(current version)
4.          If change set contain Additions
5.              Prepare Relations.xml_NEW_VERSION via traceability re- establishment
6.          Else
7.              Start preparing Relations.xml_NEW_VERSION using Relations.xml_PREVIOUS_VERSION
8.          Generate analytical traceability graph (G_node,edge)
9.      Weight Calculator (nodes)
10.         If change set node not in G (nodes)
11.             Add change set node to G (nodes)
12.         For each node in G
13.             Weight_node=EigenvectorCentrality_node
14.     Max_weight= max from all Weight_node
15.     Min_weight= min from all Weight_node
16.     Avg_weight= Sum of all weight_node / Node count (G)
17.     Influential Factor Calculator (nodes, edges)
18.         For each node in G
19.             If (Weight_node >= Min_weight And Weight_node < Avg_weight)
20.                 InflencialFactor_node= Low
21.             Else if (Weight_node >= Avg_weight And Weight_node < = Max_weight)
22.                 InflencialFactor_node= High
23.         For each edge (source, target) in G
24.             If (InflencialFactor_source == Low)
25.                 InflencialFactor_edge = Low
26.             Else
27.                 InflencialFactor_edge = High
28.     Obtain influential factor of change set nodes
29. output: Influential factor of each change in changeset
```

In this case; if the current integration has contained only a few types of artefacts, there would be only those types of intermediate XML versions prepared. Thus, establishing a proper Relations.xml is not possible. Therefore, the remaining missing artefact types' XML versions that have not been affected in the current integration are copied from the previous version of integration. Once all types of artefact intermediate XML files are collected and if the change set contains the change type '*Additions'*, the SAT-Analyser's traceability re-establishment is performed in the back-end to prepare the new Relations.xml. Otherwise, if the change set does not contain any '*Additions*' and only include '*Modifications'* and/or *'Deletions'*; then the previous version's Relations.xml is taken. Thereafter, it is altered via the change propagation process based on the CIA results.

Once the Relations.xml is prepared successfully for the current integration the nodes and links are extracted from it and fed into the analytical traceability graph type which is powered by Python *NetworkX* libraries. There, the *Weight Calculator* gets triggered and applies the Eigenvector centrality measure on all nodes and assigns a value for each node as its weight. The minimum, average and maximum weight values are calculated as the weight scale by analysing all the values of all nodes. Thereafter, the *Influential Factor Calculator* gets invoked and converts the node weights into either a *low* or a *high* value according to the criteria. Also, assigns a *low* or *high* value to each edge based on an edge's source node's influential factor value.

Figure 3-21 evidently shows a code snippet of weight and influential factor calculator implemented in Python. The nodes in the changeset are located in the analytical traceability graph and obtain the influential factor of change set nodes to decide which ways to start propagating the changes depending on the impact of change set nodes. For instance, when a changeset node holds a *low* influential factor value, then the outgoing trace links of that artefact node are discarded.

```
...
evc=nx.eigenvector_centrality_numpy(UG)
nx.set_node_attributes(UG, evc, 'EVC')
node_labels = nx.get_node_attributes(UG,'EVC')
weights=nx.get_node_attributes(UG,'EVC')
Wmax=evc[max(evc, key=evc.get)]
Wmin=evc[min(evc, key=evc.get)]
for node in UG:
   Wtot=Wtot+weights[node]
Wavg=Wtot/len(UG)
InfluenceFactor=[]
for node in UG:
   if (weights[node]>=Wmin and weights[node]<Wavg):
      UG.node[node]['IF'] = 'Low'
   elif (weights[node]>=Wavg and weights[node]<=Wmax):
      UG.node[node]['IF'] = 'High'
node_if=nx.get_node_attributes(UG,'IF')
for s, t, d in UG.edges(data=True):
   if (UG.node[s]['IF'] == 'Low'):
      d['IF'] = 'Low'
   else:
      d['IF'] = 'High'
edge_if=nx.get_edge_attributes(UG,'IF')
...
```

Figure 3-21 : Code snippet of weight and influential factor calculators

3.4.4 Impact analysis process: user modifiability

In the practical scenario, the CIA results are subjected to vary beyond the described SAT-Analyser CIA calculations. For instance, a change on a least important artefact node may cause the whole project to be failing which will be only among the awareness of developers and operations team members who are actively involved in that particular project. Thus, the CIA results are provided with user modifiability capability to strengthen the accuracy and to avoid inconsistencies. The user can alter the impacted nodes by adding newer, modifying and deleting if any unnecessary node(s) that do not require change(s) to be propagated.

Figure 3-22 and Figure 3-23 show the SAT-Analyser tool automatically identified CIA results and the user altered CIA results respectively. Then, the final altered CIA result is considered in propagating the changes further in visualization, PM and CIA validation.

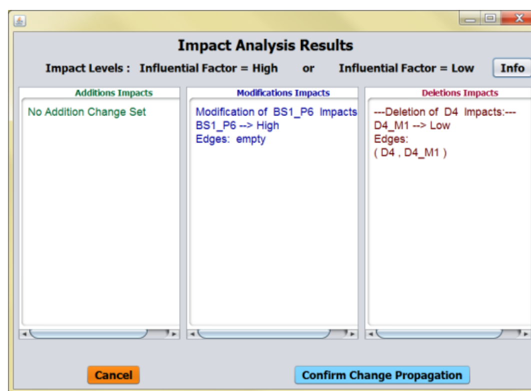


Figure 3-22 : SAT-Analyser generated CIA results

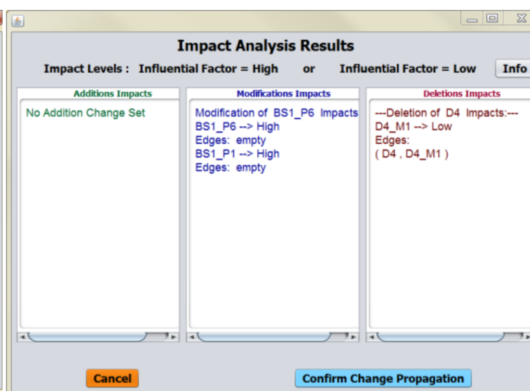


Figure 3-23 : User altered CIA results

3.4.5 Change propagation of the impact

Firstly, the change impact analysis is declared as the first level dependencies such that for a given node; its connected intermediate nodes are considered for the impact set. Secondly, the impact analysis is presented with a quantitative impact value based on their assigned node and edge weights. Thus, the graph traversal is minimal for this one level consideration.

However, the changes are possible to continue affecting remaining nodes until reach a leaf node or root. Therefore, the change propagation is required to be managed from first level impact analysis onwards. The weight system is applied in order to provide a quantitative value for them too.

The graph traversal is highly required at this stage to identify the nodes that are subjected to propagate changes. The graph traversal algorithms in finding paths can be applied for this purpose. The algorithms; Dijkstra algorithm, Bellman-Ford algorithm and Floyd & Warshall algorithm are identified to be having a potential in applying for this SAT-Analyser tool suitably. The Dijkstra algorithm can be used for stepwise routing with weights. Bellman-Ford algorithm which is powerful in handling negative edge weights and Floyd & Warshall algorithm that supports both negative and positive weights are useful in propagating changes across the graph representation of traceabilities.

A. Graph traversal model for change propagation

Once the changeset is obtained by the *Impact Analyser* component it assesses the weights and influential factor of each node and edge for the items in the given changeset. Thus, the changeset items and all the remaining nodes and edges contain a certain impact value depending on each other relationships. Identifying the impact of each changeset item on other remaining nodes and edges is the primary task of this change propagation model. This works in collaboration with the *Impact Analyser* component as the impact value of each node is required to be accessed during the propagation.

The Dijkstra algorithm is selected for the graph traversal as the weights are non-negative. For each item in changeset, graph traversal model gets applied. However, if there exist any '*Addition*' change type items in the changeset, then the change propagation results are only for the displaying purpose. Because according to the constraint defined in the impact analysis process if a CI activity includes any artefact '*additions*' the developer must update all the other artefact types and insert at the same moment. Otherwise, if the changeset only includes '*Modifications*' and/or '*Deletions*' the change propagation results are used to alter the overall Relations.xml that concatenates all the artefact types' relationships.

B. Change propagation process: workflow

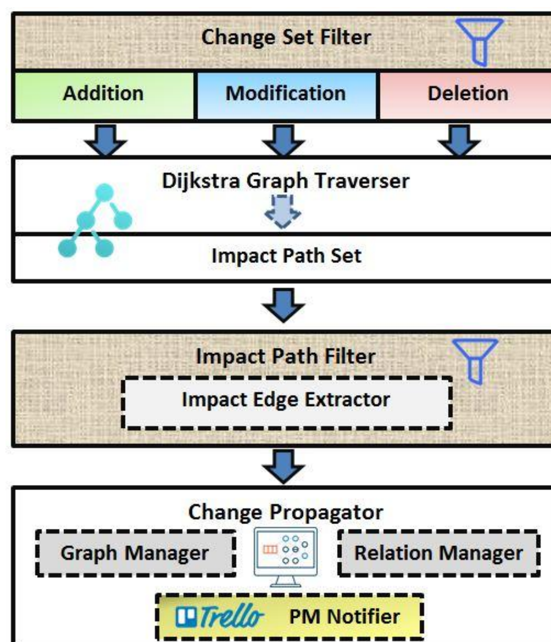


Figure 3-24 : Change propagation workflow

Figure 3-24 illustrates the designed workflow of the change propagation model in this SAT-Analyser tool. The *Change Set Filter* module pre-processes the changeset that is associated with impact values assigned in Impact Analyser based on the change types such that *additions*, *modifications* and *deletions*. Then, each type of change set is parsed into the *Dijkstra Graph Traverser* to identify the complete impact path sets until reaches a leaf node. The *Impact Path Filter* is responsible for applying the conditional constraints on those complete impact path sets based on change artefact types and filters the relevant portion of impact path from each complete impact path. Each of those filtered impact path is parsed through the *Impact Edge Extractor* submodule to express path in edges to be helpful in change propagation. Finally, *Change Propagator* module applies that impact path edge sets on *Graph Manager* to update traceability graph and to *Relation Manager* sub-module to update Relations.xml file for the continuation of SAT-Analyser tool. Simultaneously, the DevOps teams get notified about the change propagation via the project management tool Trello for each change propagation activity as a separate card in associated company Trello board.

C. Change propagation process: pseudo code and implementation details

Algorithm 3:11 describes the change propagation model associated with Dijkstra graph traversal algorithm and artefact type based path filtering conditional process. The single source Dijkstra algorithm helps to find the cheapest path of a given starting node which is each node item in changeset in this scenario. The Python analytical *NetworkX* library's single source Dijkstra function is used in the implementation purpose ("NetworkX," 2018).

The weights are supposed to be in numerical form for the Dijkstra traversal and hence the textual influential factor system in *High* and *Low* levels is converted temporarily into 1 and 0 respectively. Then, an artefact type oriented conditional algorithm is applied for each complete impact path obtained through Dijkstra algorithm path traversal. That captures the level of affecting based on artefact type; such as if the changeset item is a sub-element like a method, attribute or plugin, then only that node is declared to be considered as affected. If the changeset item is a main requirement element the design, source code and unit

test, then the paths up to unit test are considered as the relevant impact path portion.

Algorithm 3:11 Change Propagation

Require: Impact value assigned changeset
Ensure: Change propagation of changeset

1. **input:** An impact assigned changeset
2. For each item in impacted changeset
3. Dijkstra single source graph traverser (G, item, influential factor value)
4. Impact path set = $Impact_Path_{Complete}$
5. For each $Impact_Path_{Complete}$ in Impact path set
6. Impact path filter ($Impact_Path_{Complete}$)
7. If $item_{change_set}$ = a sub element
8. $Impact_Path_{Relevant} = item_{change_set}$
9. If $item_{change_set}$ = a requirement element
10. $Impact_Path_{Relevant} = item_{change_set}$ & $item_{req_sub}$ & $item_{design}$ & $item_{source}$ & $item_{unittest}$
11. If $item_{change_set}$ = a design element
12. $Impact_Path_{Relevant} = item_{change_set}$ & $item_{design_sub}$ & $item_{source}$ & $item_{unittest}$
13. If $item_{change_set}$ = a source code element
14. $Impact_Path_{Relevant} = item_{change_set}$ & $item_{source_sub}$ & $item_{unittest}$
15. If $item_{change_set}$ = a unittest element
16. $Impact_Path_{Relevant} = item_{change_set}$ & $item_{unittest_sub}$
17. If $item_{change_set}$ = a build script element
18. $Impact_Path_{Relevant} = item_{change_set}$ & $item_{buildscript_sub}$
19. Propagate changes ($Impact_Path_{Relevant}$ Set)
20. Extract edges ($Impact_Path_{Relevant}$ Set)
21. Update graph manager
22. Update relation manager
23. Project management notifier
24. **output:** Relevant Impact path sets

Similarly, if the changeset item is a design element; its sub-elements along with source code and unit test are taken as affected. If the changeset item is a source code element only the unit test items are declared to be affected along with that particular source code elements' sub-elements if any. Also, if the changeset item is of type unit test or build script, then only their sub-elements are declared as affected. Accordingly, the relevant impact set paths are captured from the complete impact paths and are extracted into edges format in order to update the traceability graph and Relations.xml.

Figure 3-25 shows a Python *NetworkX* involved code snippet that contains the Dijkstra algorithm and conditional algorithm applications.

```
...
for delitem in CDdeleteNodeList.CDdeleteNodeList:
    delete_node=delitem
    path2 = nx.single_source_dijkstra(UG, delete_node,weight='IF')
    nodesetdict=path2[0]
    pathsetdict=path2[1]
    for i in nodesetdict:
        if UG.node[i]['IF'] == 0:
            nodesetdict[i]="Low"
            #print i,nodesetdict[i]
        elif UG.node[i]['IF'] == 1:
            nodesetdict[i]="High"
            #print i,nodesetdict[i]
    print "---------Deletion of ",delete_node," Impacts:---------"
    ##for SUBELEMENTS ##
    if ('_' in delete_node):
        for i in nodesetdict:
            if '_' in i:
                print i ,"-->", nodesetdict[i]
                f2.write("\""+i+"\",")
        print "Edges:"
        for i in pathsetdict:
            if '_' in i:
                ind=0
                for s in range(len(pathsetdict[i])):
                    if ind+1 <=len(pathsetdict[i])-1:
                        print "(",pathsetdict[i][ind],",",pathsetdict[i][ind+1] ,")"
                        f.write("("+pathsetdict[i][ind]+","+pathsetdict[i][ind+1] +"),")
                        ind=ind+1
    ##for DESIGN  ELEMENTS ##
    if ('D' in delete_node and '_' not in delete_node):
        for i in nodesetdict:
            if delete_node+"_" in i or 'SC' in i or 'UT' in i:
                print i ,"-->", nodesetdict[i]
                f2.write("\""+i+"\",")
        print "Edges:"
        for i in pathsetdict:
            if delete_node+'_' in i or 'SC' in i or 'UT' in i:
                ind=0
                for s in range(len(pathsetdict[i])):
                    if ind+1 <=len(pathsetdict[i])-1:
                        print "(",pathsetdict[i][ind],",",pathsetdict[i][ind+1] ,")"
                        f.write("("+pathsetdict[i][ind]+","+pathsetdict[i][ind+1] +"),")
                        ind=ind+1
    ##for SOURCECODE  ELEMENTS ##
    if ('SC' in delete_node and '_' not in delete_node):
...
```

Figure 3-25 : Code snippet of change propagation implementation

The GUI level impact analysis results are shown in Figure 3-26. It shows the changeset, affected nodes with their influential factor and the edges of identified paths. The user has the option to edit the impact results in the window as necessary and the '*Info*' button helps with the artefact details for ease of alteration. Once the confirmation button in that window is clicked, the graph updating and relations manager updating get triggered by completing the change propagation process.

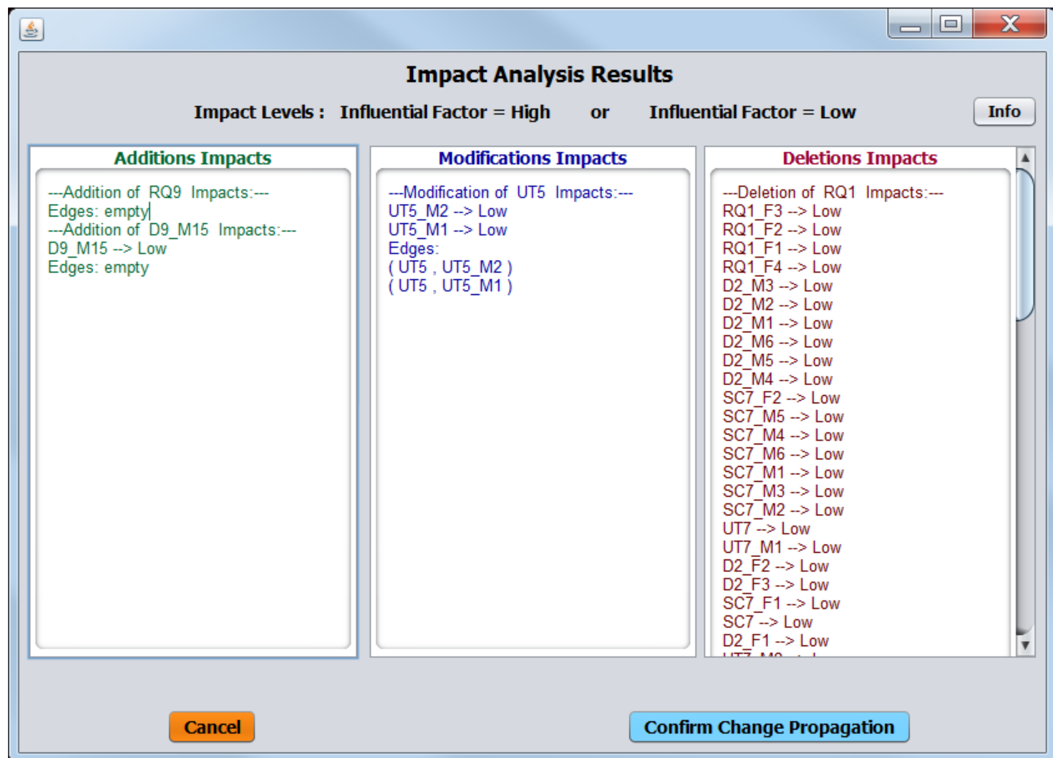


Figure 3-26 : Impact analysis results window

Accordingly, when the button 'Confirm Change Propagation' is invoked, it propagates the displayed impact results to *Graph Manager* and *Relations Manager* to update artefacts. Thus, the updated traceability graph displayed in both interactive graph mode and analytical graph mode. Figure 3-27 shows an example of the overall updated analytical traceability graph. Figure 3-28 shows the interactive graph preview based on D3.js and localhost server. This view is provided as optional since the localhost (*Apache wampserver*) is required to be started. The modified node and impacted nodes by modification are shown with larger node size for better readability while the deleted node and impacted nodes by deletion are completely deleted from both of these graph views. Additionally,

the influence factor values of edges are shown on top of edges and influential factor of nodes can be seen by keeping the cursor on any nodes (hovering) in the interactive graph mode in Figure 3-28. Moreover, the neighbourhood highlighting facility for any particular node is facilitated by double-clicking any node for better interactivity in this interactive graph mode. Simultaneously, the notification approach gets triggered to make awareness about the change propagation to project teams as described in subsection 3.4.6. Thus, relevant requirement engineers, design teams, developers and QA teams can update their responsible raw artefact types such as requirements document, design diagrams, source codes, unit test scripts and/or build script files.

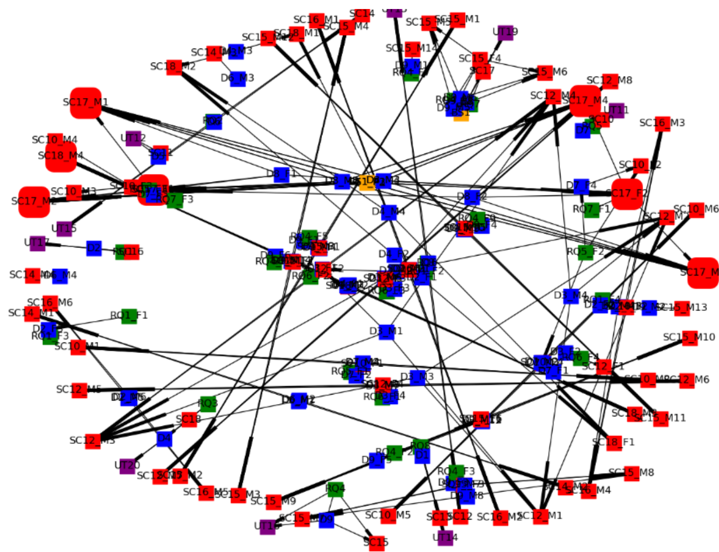


Figure 3-27 : Change propagated analytical traceability graph view

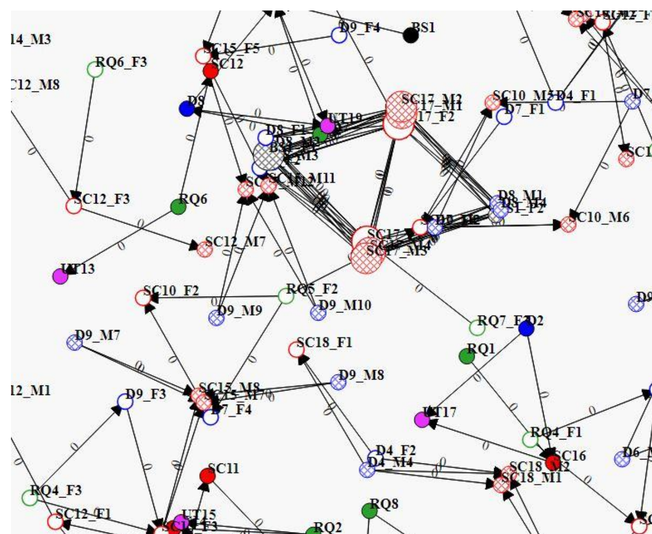


Figure 3-28 : Change propagated interactive traceability graph view

Simultaneously, the *Relations Manager* also gets triggered at the back-end. The artefact XML files of changes propagated artefact types get updated with deletion and/or modification impact results and set them as the artefact XML files of the current version. However, according to the constraints in this extended SAT-Analyser, if there are any '*Addition*' type changesets, then the overall traceability gets re-established. Thus, the overall graph is shown in the SAT-Analyser's default graph format without requiring above-described change propagation steps. In that case, all artefacts' XML files along with the Relations.xml file also gets re-generated during that traceability re-establishment without requiring any separate XML file updating as described above.

3.4.6 Notification approach

SAT-Analyser's each traceability change propagation result is notified to DevOps teams via one of the industry-level project management applications Trello ("Trello," 2018). Trello is selected for the SAT-Analyser integration due to its open source availability and industry level popularity as a PM tool.

The Trello Java API is used to integrate it with the SAT-Analyser tool to signify the SAT-Analyser's ability to integrate with industry level PM tools. For each change propagation confirmation, a newer Trello card is created automatically in a dedicated list in the Trello board. The Trello card name is generated with the particular change propagated traceability project name along with the date and time for unique identification as shown in Figure 3-29. The CIA results that contributed to that particular change propagation activity are also embedded into each Trello card in its card description. Once, the change propagation is confirmed the overall Trello board is automatically loaded in the browser with the new card instance as shown in Figure 3-30. Accordingly, the teams get notified about the SAT-Analyser change propagation for them to alter their responsible raw artefacts that are affected by the change propagation based on traceability.

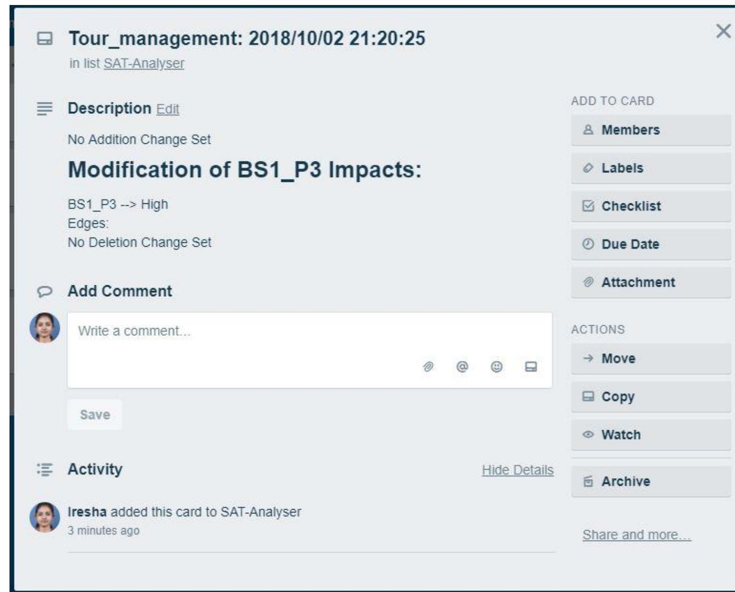


Figure 3-29 : Trello change propagation card instance

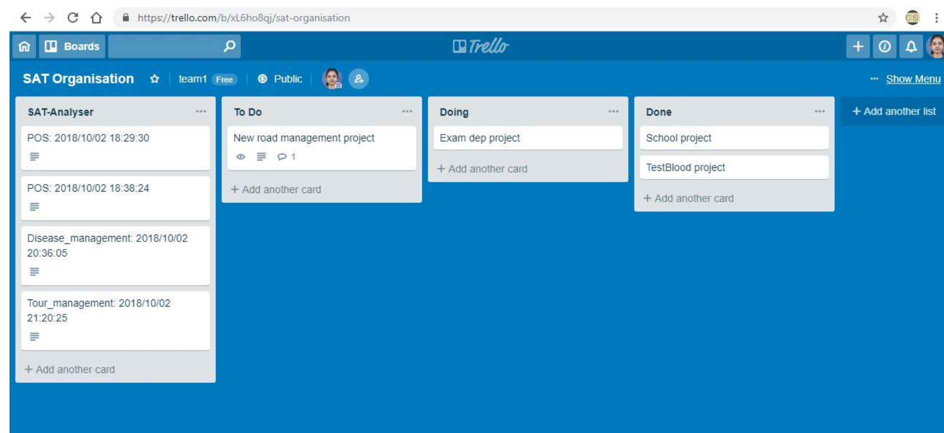


Figure 3-30 : Trello board with change propagation notification

3.5 Traceability management

A scheduler is implemented to trigger the artefact changes to the extended traceability management tool SAT-Analyser. The scheduler is designed to get triggered based on the CD timelines prior to a CD activity on a project such that along with a continuous deployment task. Thus, once the scheduler is triggered it displays a window to fetch all the types of artefact changes corresponding to all phases in the form of an input window except the source code and build scripts which are fetched automatically via the Jenkin's latest successful build job as described in the section 3.5.3: Continuous Integration. The remaining process items of the traceability management process in this extended SAT-Analyser except CIA are discussed in this section that consists of change detection, consistency management and CI.

3.5.1 Change detection

The change is vital during the software development process throughout all the stages. All the phases in SDLC such as requirements engineering, design, implementation, testing and maintenance can be changed in different frequencies. The changes occurred in one phase of the development process can evolve through all or most of the other phases based on the dependencies among them. Accordingly, the changes are evolved via the artefacts involved in each phase. The change detection component relies on the established traceability links among those artefacts in the traceability link generation component.

The extracted artefact elements are stored and maintained in a common XML format based on a predefined XML relation model using customized tags. Therefore, the comparison of artefacts is performed via those common format versions of them, specifically as an XML comparison. The relation model is compared among each and checked whether the artefact elements are compatible with each other. A change can be in three types, edit, deletion and addition.

A. Change detection process: workflow

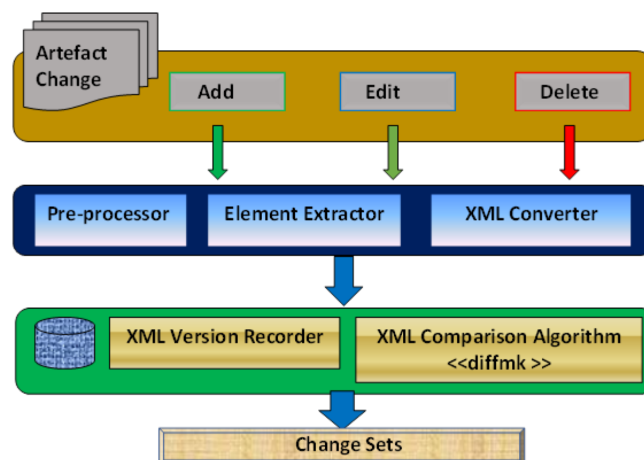


Figure 3-31 : Design diagram: change detection component

Figure 3-31 illustrates the module and subcomponent organization of the change detection component. The artefact changes are allowed to occur in any type of five artefacts supported in the tool SAT-Analyser such that requirement changes, design diagram changes, source code changes, unit test case changes or build script changes. A developer can integrate any one or many types of artefacts in the

form of a continuous integration task. That new artefact integration may contain either element *additions*, element *alterations* of *removals*. Whenever a new artefact input is received by the tool SAT-Analyser, it starts its intermediate XML format generation task for that particular artefact category using the data pre-processor, element extractor and XML converter modules. As a result, an XML format representation is created for that particular artefact(s) integrations. For instance, if the integration included a new requirement document and a new source code, then two different intermediate XML files are generated one for requirement and one for source code.

Then, as depicted in Figure 3-31, the bottom layer holds a database to store the newly created intermediate XML versions corresponding to a performed continuous integration task. The *XML Version Recorder* module is responsible for adding the correct version suffix to those newly created XML versions based on the previously generated XML version suffix numbers. The actual change detection is initiated there onwards in the *XML Comparison* module that compares the newly found intermediate XML files and the last previous XML version of those corresponding artefact types. Regarding the above stated example, the old requirement XML file versus new requirement XML file and the old source code XML file versus newly created source code XML file would be used by this XML comparison module to encounter the occurred changes. Finally, the identified changes are the outcome of this change detection component.

B. Change detection process: pseudo code

The change detection Algorithm 3:12 is designed to ensure the identification of artefact related changes. The traceability management must be altered based on the changes since each phase of the SDLC is lightly or tightly coupled with other phases. That leads to a change in one artefact affects the other dependent artefacts. Therefore, the change detection having high performance is crucial in order to minimise the number of concurrent conflicts that can lead to artefact inconsistencies. The input considered to this change detection algorithm can be any type of artefact change such as an *edit*, *addition* or *deletion*.

Algorithm 3:12 Change Detection

Require: Software artefact intermediate XML versions
Ensure: Identify changes in continuous artefact integrations

1. **input:** artefact integrations
2. If input=Requirement OR Design OR Source Code OR Test case OR Build Script
3. Change Detection Component
4. Invoke artefact pre-processor (input artefact)
5. Artefact element extractor
6. XML Convertor (pre-processed artefact input)
7. Return XML $_{new}$ version
8. Store XML $_{new}$
9. If XML version recorder(XML $_{new}$) fetches corresponding XML $_{old}$
10. Invoke diffmk comparison engine
11. $Diff_{artefact}$=XML comparison (XML $_{new}$, XML $_{old}$)
12. Return XML_{diff}
13. Store XML_{diff}
14. XML extractor (XML_{diff})
15. Return all changes
16. String pre-processor (changes)
17. Return Changes $_{Added}$, Changes $_{Modified}$, Changes $_{Deleted}$
18. Display changesets
19. **output:** Detected changesets

The scheduler can be triggered either to invoke in any given specific time slot or to invoke whenever the SAT-Analyser traceability tool is executed which means a change has occurred. For an example, if a class called 'shop' is identified in the requirements artefact element, it must be available as an artefact element in both other UML class diagram and source code related artefacts. Accordingly, if the 'shop' class is removed from the requirements specification or from the XML relation model, that and all dependent items such as 'bookshop', 'bakery shop' must not be available in the other two types (design, source code) of artefact related files. If an existence is identified, it is marked as an incompatibility situation. Hence, a change is declared and the change detection points are triggered.

Accordingly, the change detection is technically based on XML version comparison in the SAT-Analyser system model. The XML comparison algorithms such as BULD (Bottom-Up Lazy-Down propagation) and Diff (Cobena, Abiteboul, & Marian, 2002) that match nodes and construct a delta in a linear

time, X-Diff algorithm (Yuan Wang, DeWitt, & Cai, 2003) that does comparison by generating trees with a minimum cost edit script and Johnson's algorithm that detects changes of documents are in active research. As the SAT-Analyser is mainly on a Java-based platform, performance wise it is ideal to stay in the same technological domain. Therefore, the Java friendly XML comparison modules such as XMLUnit ("XMLUnit," 2018) are especially experimented to find useful in this process.

C. Change detection process: implementation details

Studying the existing open source XML comparison algorithms, frameworks and research works, the generalized tool for XML named '*diffmk*' ("diffmk," 2018) from *Oracle Sun* developers is selected to be incorporated with the SAT-Analyser tool. It is provided with the license type BSD-3-Clause and is allowed for productivity or publishing. The origin of this tool *diffmk* is from the tool '*diff*' and is in the language Perl though currently, Java supported binary versions are also available.

The *diffmk* operates in the sequence domain as it encodes changes by annotating the input document. It expresses *diffs* by inlining them into the d_1 document, so no size comparison is available for that tool and is actively involved in the XML based research works (Suzuki, 2002)(Lindholm, Kangasharju, & Tarkoma, 2006). *Diffmk* compares the previous version of a file with the current version and creates a file that includes nroff/troff 'change mark' commands. Accordingly, *diffmk* generates markfile which contains all the lines of the current file plus inserted formatter 'change mark' requests. When markfile is formatted, changed or inserted text is shown by a | character at the right margin of each line. The position of the deleted text is shown by a single *. If the characters | and * are inappropriate, a copy of *diffmk* can be edited to change them as the original version of *diffmk* is a shell script.

Considering the limitations of *diffmk*, it does not differentiate between changes in text and changes in formatter request coding. Thus, file differences involving only formatting changes with no change in the actual text can produce change marks. But regarding the tool SAT-Analyser, its XML intermediate files are properly

generated according to a predefined format structure. Therefore, unnecessary formatting changes do not exist by not being affected with this limitation of *diffmk*. As *diffmk* uses *diff*, it has the same limitations on file size and performance that *diff* may impose. In particular, the performance is nonlinear with the size of the file and very large files (over 1000 lines) may take longer to process. Also, *diffmk* uses the '*ed'* editor ("GNU 'ed,'" 2018). If the file is too large for *ed*, *ed* error messages may be embedded in the file. As a precaution for these limitations, breaking the file into smaller pieces is technically advisable. However, a single artefact file such as a corresponding to single requirement file, design diagram may not exceed 1000 lines practically in a normal Agile based software project where non-critical projects are addressed. Therefore, this limitation is also not affecting the purpose of SAT-Analyser tool in incorporating the *diffmk* engine for XML comparison module.

Consequently, the *diffmk* based XML comparison module implemented in Java is more aligned with the tool SAT-Analyser. The two latest versions of XML files are considered such as the current version and the latest previous version. It checks for the mutual XML artefact file types in two selected version directories. For an instance, if both directories have requirement artefact's XML files, source code artefact type XML files; then those artefact types from each version directory are taken as the input in .xml file format. Then, the *diffmk* starts its comparison process for two files and creates another XML file with the content of newer XML file content and the change points marked as *changed (modified)*, *added* and *deleted* as evidently shown in the following code snippet in Figure 3-32.

The outcome XML file that contains the marked change points is used in the remaining process to extract those changes as suitable to the SAT-Analyser tool. Once a traceability project is created, initial traceability is established and should have at least single continuous integration activity occurred to be eligible for proceeding with this change detection process.

A menu item is available for each traceability project in SAT-Analyser tool to invoke that as shown in Figure 3-33.

```
public void setChanged(Element node) {
    //node.setAttribute(attrName, changed);
    node.setAttributeNS("http://diffmk.sf.net/ns/diff","diffmk:change","changed");
}

public void setAdded(Element node) {
    //node.setAttribute(attrName, added);
    node.setAttributeNS("http://diffmk.sf.net/ns/diff","diffmk:change","added");
}

public void setDeleted(Element node) {
    //node.setAttribute(attrName, deleted);
    node.setAttributeNS("http://diffmk.sf.net/ns/diff","diffmk:change","deleted");
    // FIXME: add a switch for id attribute name
    if (node.hasAttribute("id")) {
        node.setAttribute("id", "DEL." + node.getAttribute("id"));
    }
    if (node.hasAttribute("xml:id")) {
        node.setAttribute("xml:id", "DEL." + node.getAttribute("xml:id"));
    }
}
```

Figure 3-32 : Diffmk based change types declaration

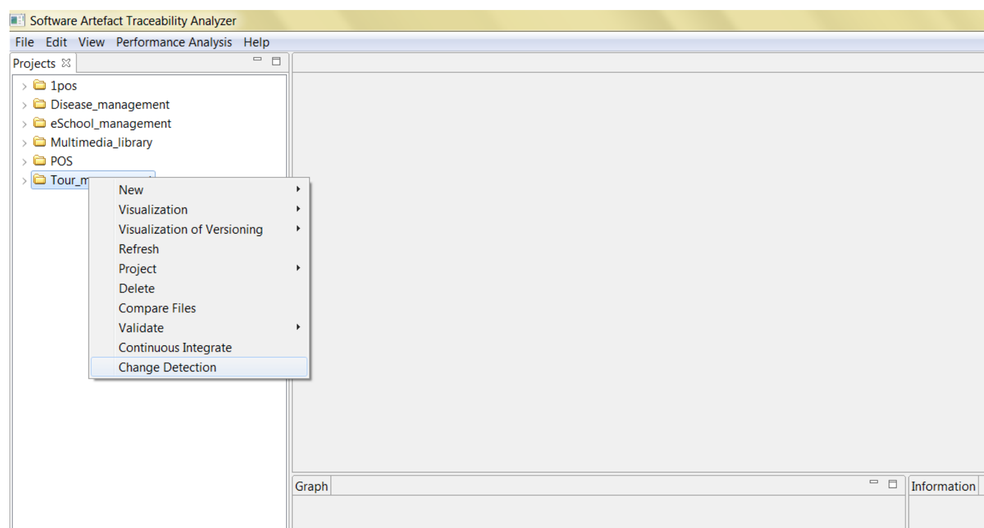


Figure 3-33 : Change detection menu item

There must be at least one integrated CI version to proceed and else an error is shown indicating insufficiency of integrations as shown in Figure 3-34. Otherwise, a new directory named ‘xml_CD’ gets created to the folder structure as shown in Figure 3-35. That xml_CD directory holds all the *diffmk* change points marked XML files. The *XML Extractor* sub-module is called for each of those files in the xml_CD directory corresponding to artefact types and all the marked change points that are either *changed*, *added* or *deleted* are extracted as string from each XML file. Then, that data string is pre-processed and categorized as *additions*, *modifications* and *deletions* to be more user-friendly and readable. Finally, the categorized change detection results are presented as shown in Figure 3-36.

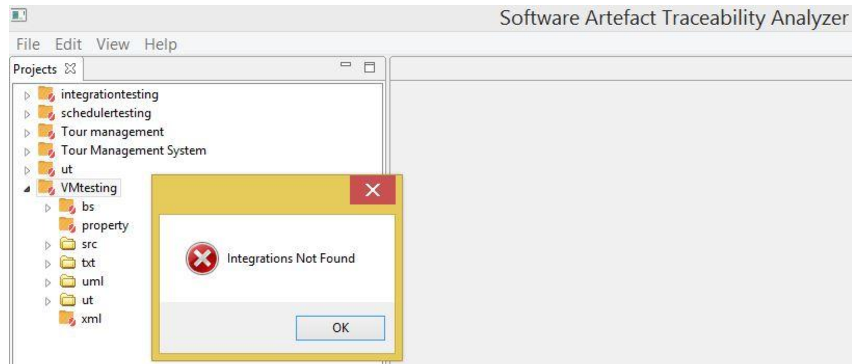


Figure 3-34 : Insufficient CI versions for change detection

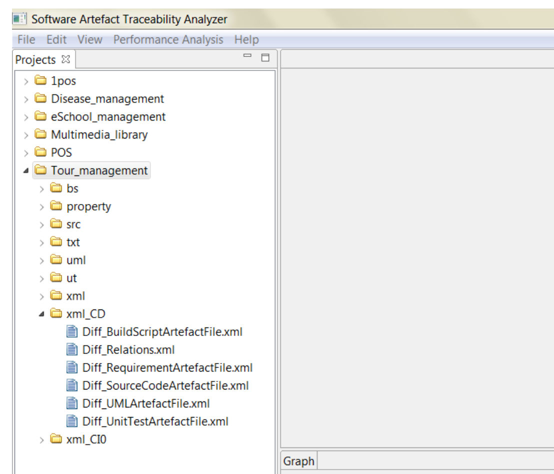


Figure 3-35 : Change detection results xml_CD directory

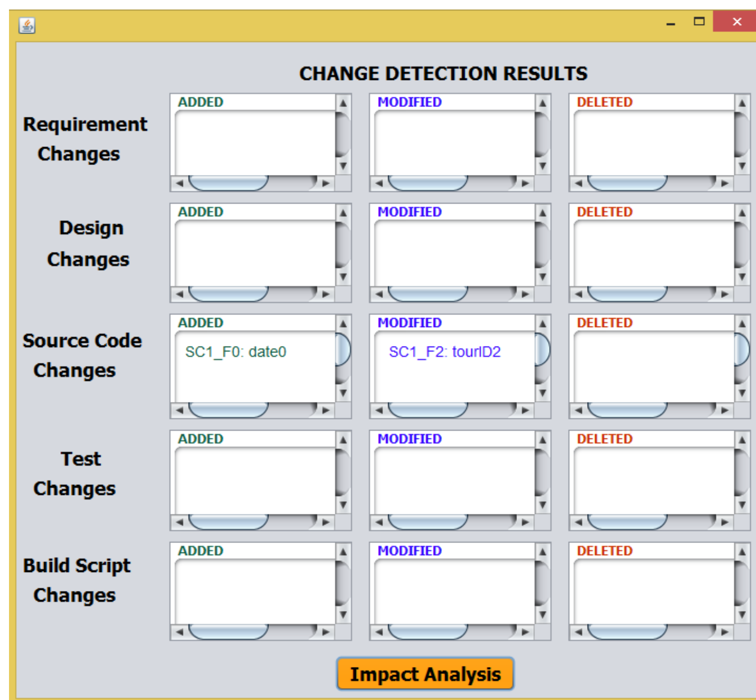


Figure 3-36 : Change detection results in outcome window

The continuous integrations may or may not have all types of artefact integrations. For example, a CI may only integrate a set of code files to a source code repository in a particular integration resulting in a newer version of intermediate XML file generated only for source code artefact. Thus, the change detection will be conducted only for the source code artefact in that scenario as in above Figure 3-36. The unique artefact ID and the artefact element or sub-element name are displayed as the change content. The colour code scheme of green for *added* changes, blue for *modified* changes and red for *deleted* changes is adapted for better readability. Thereafter, the *Impact Analysis* can be initiated from this window to identify the encountered impacts due to the detected changes.

3.5.2 Consistency management

In frequent changes during a DevOps environment, the risk of artefact inconsistency tends to be high. The changes may need to be propagated to maintain the consistency level, but should be propagated based on the impact analysis outcomes. Because propagating a non-impacted change across the artefacts can become an unnecessary overhead.

SAT-Analyser establishes traceability with traceability visualization. Whenever an artefact alteration; add/ edit/ delete occurs, the inconsistencies arise. The SAT-Analyser's process of change detection, change impact analysis, change propagation is designed in handling this inconsistency issue.

A. Consistency management process: workflow

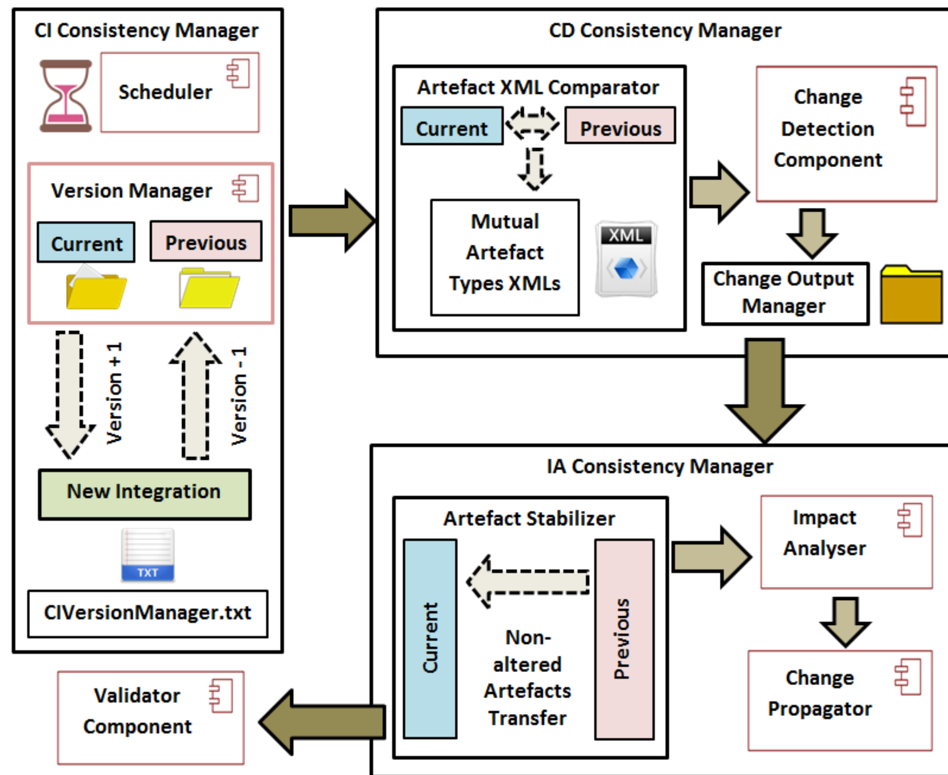


Figure 3-37 : Consistency management workflow

A rule-based consistency management approach is followed in this extended SAT-Analyser system to ensure consistency. Figure 3-37 shows the inter-component and intra-component wise handling of consistency management. The version management is a major part in ensuring the consistency related to continuous integration which is the core of this extended SAT-Analyser in order to cope with DevOps environments. There, the version numbers are separately handled in a textual file format to maintain consistency. Also, the version directory structure creation is monitored and roll backed properly in any unsuccessful integration attempt to avoid inconsistencies. In the change detection, a module called *Artefact XML Comparator* performs to ensure a consistent input to *Change Detector* component. The outcome of the *Change Detector* is also handled in a separate directory structure named ‘CD’ by this consistency manager. A module named *Artefact Stabilizer* executes during the impact analysing to make the current version always stable by transferring non-altered artefact types XML files from the immediate previous version to the current version. That highly helps to *Impact Analyser, Change Propagator* and *Traceability Validator* components.

B. Consistency management process: pseudo code and implementation details

Algorithm 3:13 Consistency Management

Require: SAT-Analyser to be started
Ensure: Consistency of SAT-Analyser

1. **input:** Scheduler strike
2. For each successful Continuous Integration
3. Invoke Version Manager
4. prepare CI directory structure (version)
5. $CI_{Current}$= version
6. $CI_{Previous}$=version-1
7. CI_{New}=version+1
8. Create CIVersionManager.txt to xml directory
9. Write to CIVersionManager.txt ($CI_{Current}$)
10. Create xml_CI($CI_{Previous}$) directory
11. $CI_{Current}$= CI_{New}
12. $CI_{Previous}$= $CI_{Current}$
13. For each Change Detection initiation
14. Invoke artefact XML comparator ($CI_{Current}$, $CI_{Previous}$)
15. Mutual_XMLs=Extract mutual artefact types XMLs current and previous versions
16. Continue Change Detection ($Mutual_XMLs_{Current}$, $Mutual_XMLs_{Previous}$)
17. Create xml_CD directory
18. Store/ replace detected changes output files
19. For each Impact Analysing initiation
20. Invoke artefact stabilizer ($CI_{Current}$, $CI_{Previous}$)
21. If $CI_{Current}$ does not contain all artefact types
22. Transfer missing artefact type from $CI_{Previous}$
23. Continue Impact Analysing
24. Change propagation
25. Confirm consistency of validator
26. **output:** Maintenance of consistency in SAT-Analyser

Algorithm 3:13 describes the workflow of rule-oriented consistency management approach. This ensures the consistency of Continuous Integration, Version Manager, Change Detector, Impact Analyser, Change Propagator and Traceability Validator components throughout the SAT-Analyser execution.

Artefact XML Comparator intakes the current XML directory content and previous XML directory at the moment and identifies the mutual XML artefact file types. Because there must be similar artefact type XML files available in both versions to avoid any inconsistencies during the change detection process. Mainly the *Artefact Stabilizer* module ensures the current XML directory's stability before

proceeding of the impact analysis process. Because if all artefact types XML files does not exist in the current XML directory before the impact analysis, that would lead to an incomplete Relations.xml where some artefact types are missing, thus the system can become inconsistent there onwards.

3.5.3 Continuous integration

Due to the frequent software changes, the continuous integrations can occur in dynamic frequencies in a DevOps software development environment. According to the survey conducted among industry experts, they are currently coping with no traceability support or schedulers. Hence, it is identified that scheduling traceability is required to be managed in this SAT-Analyser tool to make it synchronized with the usual DevOps workflow of the environment without being an overhead. Therefore, a scheduling algorithm is needed to invoke the traceability process at CI activities. The initially identified options are;

- Fixed intervals: office starting time (8.00AM) and office finishing time (6.00PM) or
- User-defined intervals in a customizable manner or else
- Whenever continuous deployment occurs in CICD pipeline prior delivery (CD).

to trigger starting the traceability management process.

A. Scheduler: workflow

The fixed intervals scheduling option is provided at one point in the SAT-Analyser as most of the local software development companies are still not functioning 24 hours continuously. Therefore, the office starting time and the ending time that is approximately after about 10 hours is applied in the scheduler design. Thus, if the office start time is 8.00AM, then once the SAT-Analyser is opened, first integration triggering will be prompted. After 10 hours which means at 6.00PM on the same working day, another triggering will occur automatically. Accordingly, mandatorily two CI traceability triggering are designed to be occurred automatically with the option either to proceed with or to cancel if there is nothing to be integrated for traceability monitoring based on the productivity of the company teams on a particular day.

Also, it is allowed to manually trigger at any time enhancing the immediate CI traceability capabilities apart from the defined scheduling intervals. Therefore, it is possible to look for traceability whenever continuous deployment occurs in CICD pipeline prior to delivery (CD).

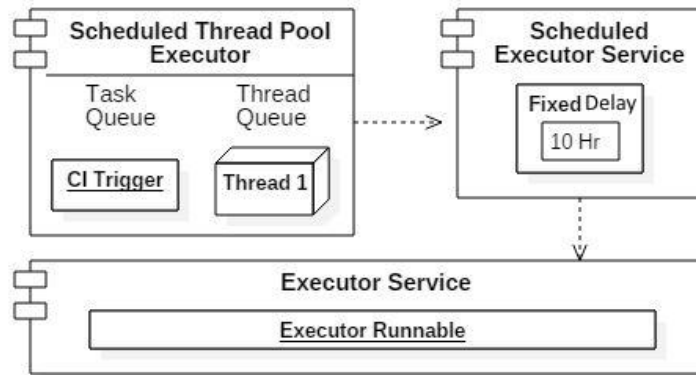


Figure 3-38 : Scheduler workflow

The scheduler is designed based on an executor framework. Figure 3-38 represents the workflow of the scheduler that consists of three main components. There is a scheduled thread pool where the number of tasks and the number of threads is defined as necessary. As there is only one task (task of triggering CI) to be triggered for the requirement of SAT-Analyser, only one task and one thread are queued. The *CI Trigger* object in Figure 3-38 represents the functionality of CI artefact fetching window. The *Scheduled Executor Service* component handles the periodical behaviour of the scheduler. A fixed delay of each 10 hours is applied there as the scheduler frequency to invoke *CI Trigger* via the thread. The *Executor Service* component represents the used executor framework that holds the runnable interfaces corresponding to threads.

B. Scheduler: pseudo code and implementation details

Algorithm 3:14 elaborates the CI scheduler process pseudo code that ensures the CI automation. As it is highly coupled with the main tool SAT-Analyser, the execution of SAT-Analyser tool is essential. When the SAT-Analyser is started at the beginning of the working day, simultaneously a responsible person can confirm the stable projects that are ready to accept integrations continuously.

Algorithm 3:14 Continuous Integration Scheduler

Require: SAT-Analyser to be started
Ensure: Continuous Integration Automation

```
1.  input: Project Stability Confirmation
2.  If project is stable after first traceability establishment
3.      CI initial = Invoke first CI task
4.      Scheduler Starts (CI initial)
5.          Thread starts
6.              Timer = T0
7.              CI version number generator starts
8.              Task ()  = CI window prompts
9.              Do
10.                 Runnable Timer 10hours
11.                 If Timer = T10hours
12.                     CI automatic = Invoke CI task
13.                     Assign CI version number
14.                     Task ()
15.             Loop Until SAT-Analyser shuts down
16.         Thread shuts down
17.     Scheduler terminates
18. output: Automated Continuous Integration window prompting
```

Accordingly, the responsible person in-charge can initiate the first CI task (CI $_{initial}$) as a way of confirming that particular project created in SAT-Analyser is capable of successfully accepting continuous integrations. That CI $_{initial}$ automatically becomes an input to the scheduler to start immediately. Then, the thread starts soon after initiating the timer (T_0) to the current time and the task of prompting CI window to input artefact integrations executes. The CI version management subprogram also starts parallel and generates version numbers starting from version 1 to each project. Thereafter, the runnable interface of that started thread runs with the timer counting to 10-hour intervals from T_0 until the SAT-Analyser shuts down. After each successful CI task submission, when the timer reaches a 10-hour interval ($T_{10hours}$), automatically the CI task gets invoked, get assigned a CI version number and CI window prompts to input artefact integrations. The running thread and scheduler terminate when the SAT-Analyser tool shuts down.

The implementation of this CI scheduler algorithm is performed using Java as the core development of the SAT-Analyser depends on Java. The Java Executor

framework's Scheduled Executor Service is adapted in implementation that supports interfaces and methods for scheduled or repeated periodic executions of tasks ("Priority Blocking Queue," 2018). Figure 3-39 is an evident code snippet regarding the implementation of this CI scheduler in SAT-Analyser.

```
package com.project.traceability.Scheduler;
import java.util.concurrent.TimeUnit;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.Executors;
import com.project.traceability.GUI.ProjectIntegrateWindow;
public class SchedulerService {
   public static int versionNumber;
   public static void main(String ProjectPath, int PrevVersionNumber) {
      ScheduledExecutorService execService = Executors.newScheduledThreadPool(1);
      execService. scheduleWithFixedDelay (new Runnable() {
      //The repetitive task
          @Override
          public void run() {
           //Continous Integration Version Number Generation
           versionNumber=PrevVersionNumber+1;
           Flag(ProjectPath, versionNumber);}
           }, 10, 10, TimeUnit.SECONDS); }
public static void Flag(String ProjectPath, int integrationVersion) throws IOException{
      ProjectIntegrateWindow.main(ProjectPath, integrationVersion);
      System.out.println("CI V"+integrationVersion+" for "+ProjectPath+" Triggered at: "+ new
     java.util.Date()); }}
```

Figure 3-39 : Scheduler code snippet

The Java concurrent libraries associated with the *Executor* framework are included to use their methods and runnable interfaces. Only one thread pool is declared since it is guaranteed not to be reconfigurable to use additional threads as only one task to be executed. The *scheduleWithFixedDelay( )* method creates and executes a periodic task that runs periodically until cancelled. It becomes enabled first after the given initial delay, that is declared as 10 as we want the scheduler to start after 10 hours from initial CI task. Then, it runs subsequently with the given period which is 10 hours. Thus, executions will commence after initialDelay+period, next initialDelay + 2 * period and so on. The task of prompting CI window and the integration version number generation is defined inside a runnable as a repetitive task. CI window prompting is declared in a separate method by enhancing code modularization where the integration number and project path is passed as parameters. That helps to bind the generated version number of that particular integration with the opening CI window and its file intakes.

Further, a log is printed each time a CI is triggered for later inspections. The code is configured to pass the corresponding project location and name to this scheduler once an initial CI task is invoked by a person-in-charge. Thus, the scheduler gets triggered for each project separately by invoking CI windows separately for each project. The corresponding relevant project path and the time snippet that CI is triggered is included in each CI log entry.

After a traceability project creation, can synchronise each project with the DevOps environment's CI process by configuring the local source code, unit test script and build script paths that are used for versioning via the menu item shown in Figure 3-40. Figure 3-41 depicts the intermediate configuration settings to set those paths to obtain the latest source code, test, build script artefacts from the DevOps process's involved Jenkins build servers and versioning tools such as GitHub.

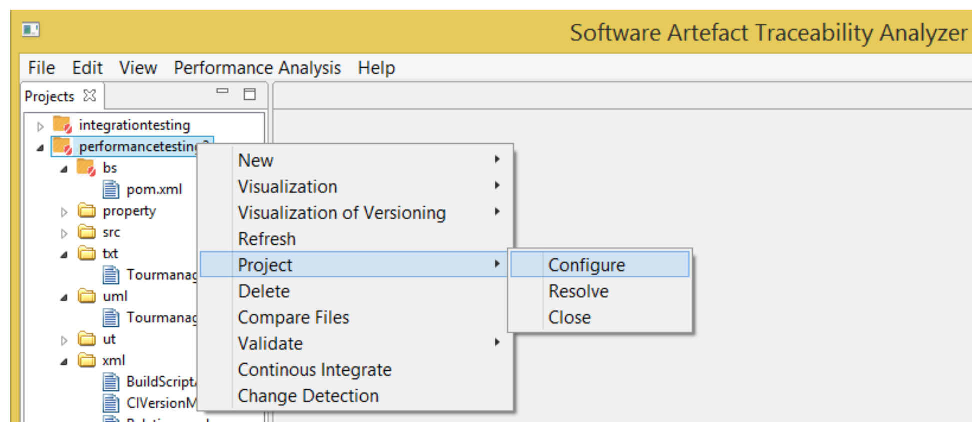


Figure 3-40 : Continuous integration configuration menu item

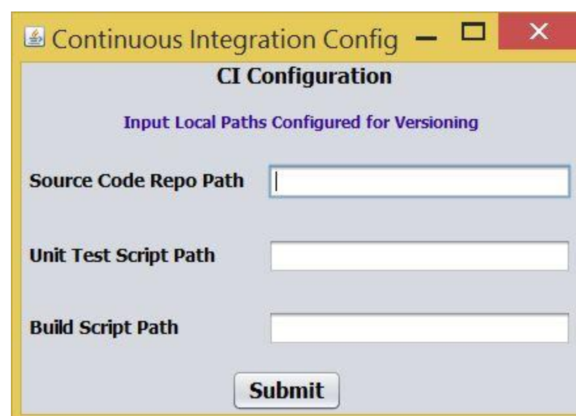


Figure 3-41 : Continuous integration configuration window

Figure 3-42 shows the first CI task ((CI $_{initial}$)) initiation towards the confirmation for a stable project that is ready to accept integrations continuously. By right-clicking the project and selecting '*Integrate*' option, Figure 3-43 is prompted with CI window. This gets triggered by scheduler that accepts all types of input artefact integrations. The corresponding project name, path, CI task and the integration number are shown on the top of that window for accuracy when there are multiple traceability projects being monitored through the SAT Analyser. There includes a toggle button to indicate whether the integration includes any new artefact element '*additions*' or not. A click on it considers as only artefact '*modifications*' and '*deletions*' would be included in the particular integration task. Figure 3-44 represents the artefact '*addition'* mode where that all artefact inputs must be provided to enable the '*Finish*' button. Also, the artefact '*modification'* or '*deletion'* attempts, at least one artefact must be input to proceed.

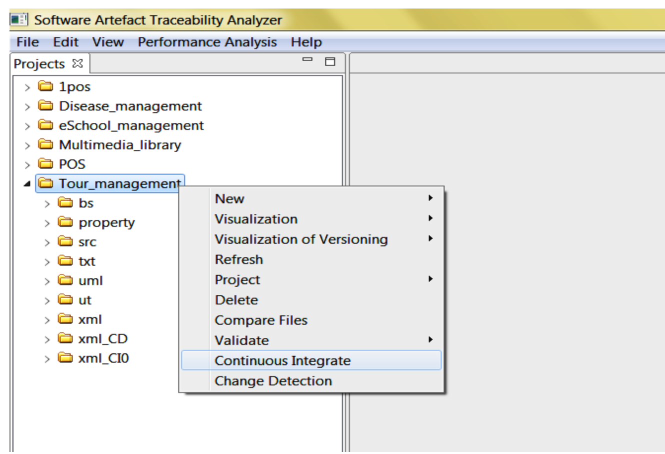


Figure 3-42 : Continuous integration menu item

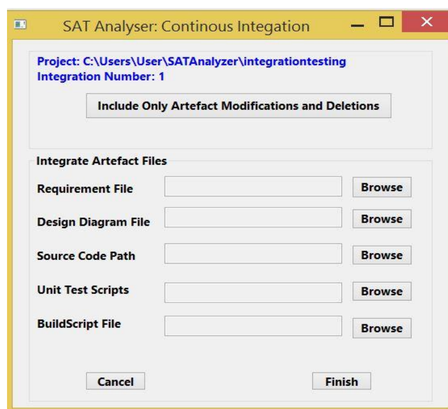


Figure 3-43 : Continuous artefact integration window

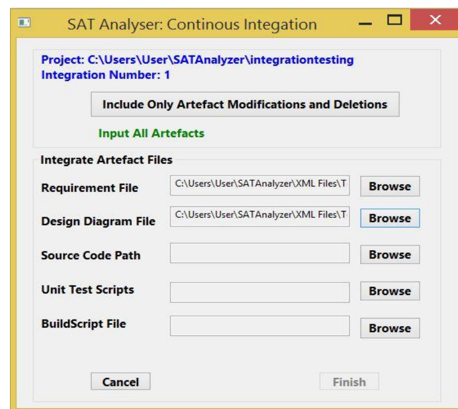


Figure 3-44 : Continuous artefact integration window with disabled forward option

C. Version control system: implementation details

Along with a scheduler having an integration version management mechanism is essential to save and keep track of each individual CI task. Therefore, the CI version management is done using a folder structure. The CI version number generated through the scheduler is linked with this version control for unique identification of each integration. The raw artefact contents in each CI task such as requirement and design always replace the previous corresponding artefact items. In the existed initial version of SAT-Analyser, the intermediate XML formats of all the artefacts are generated into a separate directory called 'xml'. Thus, version control is designed around that. The corresponding version number of each integration is physically saved in a text file dedicated for that particular version content to avoid version number resetting when the SAT-Analyser is restarted.

Initially, the *Version Control Manager* looks for 'CIVersionManager.txt' file containing a version number. If does not exist, that indicates it as the traceability project's first CI task where the version number is one. If exists, reads that file's version number, increments it by one and labels the current CI version with that number, while writing it into a new version text file. Considering, the CI content management of these each integration; a new directory is created in the same level of default 'xml' directory with the name 'xml_CIn' where '*n*' denotes the current version number minus one. The existed content inside the 'xml' directory is moved to the new directory by leaving the 'xml' directory empty. Thus, always the current CI version's intermediate XML file content is saved into 'xml' directory. The previous version can be found in the 'xml_CIn' directory.

Figure 3-45 evidently presents a part of the code used in creating this *Version Management* module. It shows how each previous version storage directory gets created and move the existed content into that by leaving 'xml' directory empty and by deleting each moved file. Although this gets executed whenever a CI task is triggered, this moving and deletion get rollbacked if a CI task is cancelled without being a successful submission. Thus, each file is moved back into 'xml' directory and deletes the created previous version 'xml' directory.

```
...
File oldxmlFolder = new File(ProjPath+File.separator+"xml_CI"+(versionNumber-
1)+File.separator);
        File currentxmlFolder = new File(ProjPath+File.separator+"xml"+File.separator);
        oldxmlFolder.mkdir();
        if(currentxmlFolder.isDirectory()) {
          File[] contents = currentxmlFolder.listFiles();
          for(int k = 0; k < contents.length; k++) {
            File sourceFile=new File
(ProjPath+File.separator+"xml"+File.separator+contents[k].getName());
            Path sourceFilepath=
FileSystems.getDefault().getPath(ProjPath+File.separator+"xml"+File.separator+contents[k].ge
tName());
            File destFile = oldxmlFolder;
            FileUtils.copyFileToDirectory(sourceFile, destFile);
            Files.delete(sourceFilepath); } }
```

Figure 3-45 : Code snippet of version control management

Figure 3-46 and Figure 3-47 show the *Version Management* directory structure in the GUI level. In Figure 3-46, the first CI task is performed with UML and build script artefacts by moving default traceability project's initial XML file content to 'xml_CI0' folder. The generated XML files of UML and build script files with version number text file are stored in the 'xml' directory. Similarly, after the given interval (10 hours) when the 2nd CI occurs and submitted successfully with a build script artefact integration, the 1st CI task's content in 'xml' directory moved into 'xml_CI1' directory. The XML content and version number text file of 2nd CI task get stored in 'xml' directory as shown in Figure 3-47.

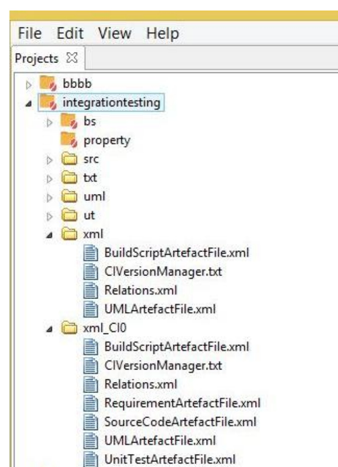


Figure 3-46 : CI version 1 of a project

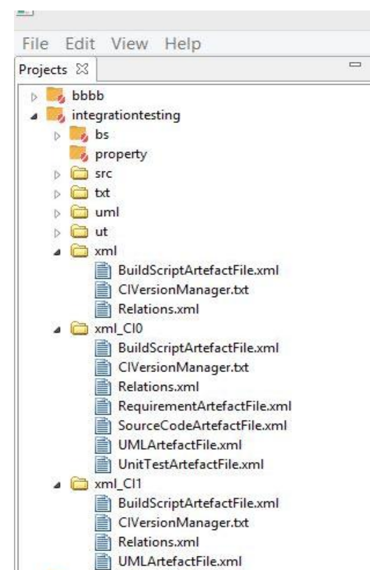


Figure 3-47 : CI version 2 of a project

3.5.4 Multi-user supportability

DevOps environments function in combination of small-size teams in achieving faster developments and delivery simultaneously. Therefore, the DevOps tools stack mostly consists of multi-user features with shared access through a dashboard in order to facilitate team coordination and work allocations properly. Accordingly, the SAT-Analyser tool also powered with multi-user accessibility through a web-based version in addition to the stand-alone desktop version with equal features discussed in previous sections.

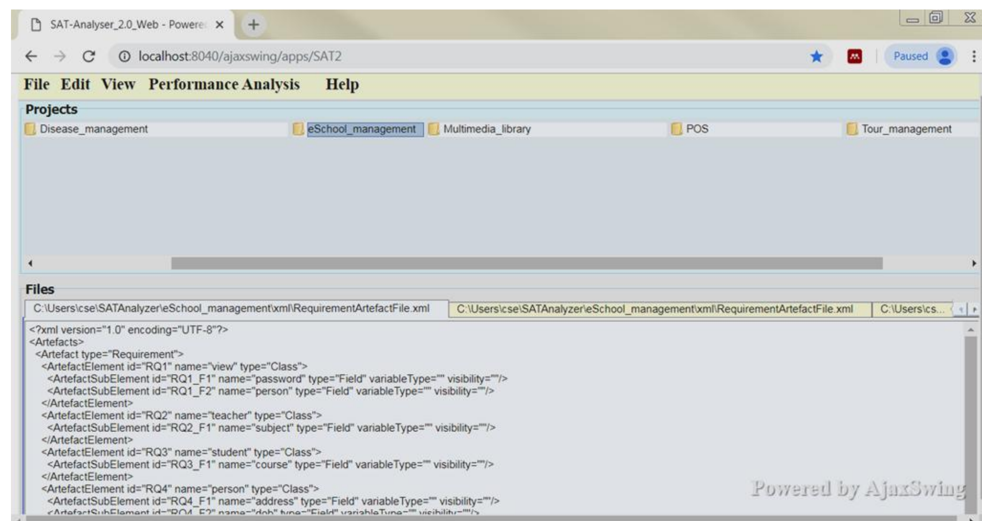


Figure 3-48 : Multi-user accessible SAT-Analyser web version

Figure 3-48 shows the implemented main GUI of the web-based SAT-Analyser prototype. The web deployment platform *AjaxSwing* ("AjaxSwing," 2018) is integrated into the implementation that creates HTML and JavaScript at runtime by transforming Java Swing to HTML. It performs with the open-source Java Servlet container *Apache Tomcat* server. Thus, SAT-Analyser is featured with cross-browser compatibility where one machine in the DevOps environment can act as the server while other team members can access the SAT-Analyser in real-time using their own client device browsers. User session timeouts, update intervals and auto-refreshing are defined to enable dynamic multi-user accessibility in similar to DevOps tools stack. The features; traceability visualization, validation, CI, change detection, CIA with user modifiability using asynchronous monitor updates for controlling single session input at a time, change propagation, PM notification and SAT-Analyser performance monitoring all equally render in the browser itself.

3.6 Extended SAT-Analyser evaluation

The SAT-Analyser tool validation components are implemented into the tool itself as three different modules for the purposes of measuring resource utilization, traceability accuracy analysis and network analysis of traceability results. The implementation details of the three modules are described in this section.

3.6.1 Implementation of the accuracy analysis module

Traceability establishment and CIA accuracy analysis module is based on statistical measures precision, recall and F-measure (Zeugmann et al., 2011)(Rubasinghe, Meedeniya, & Perera, 2018b). The traceability establishment process of SAT-Analyser depends on the artefact elements and sub-elements extraction results since the traceability is generated according to string comparison among extracted artefact outcomes. Thus, the accuracy of artefact extraction results is implemented to be measured specifically under the traceability establishment accuracy. The accuracy of CIA process is implemented around the outcomes obtained in the CIA as described in section 3.4. The mathematical calculations of precision, recall and F-measure following their definitions are applied using the Java calculations into the development. The example output windows of implemented accuracy measure application are shown in Figure 3-49 and Figure 3-50 for traceability artefact extraction and CIA respectively.

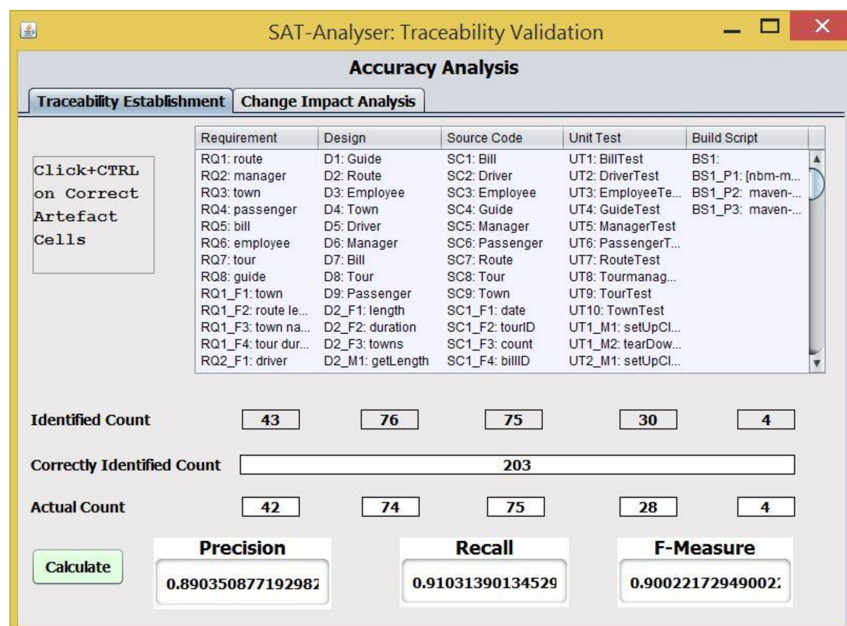


Figure 3-49 : SAT-Analyser traceability establishment accuracy analysis window

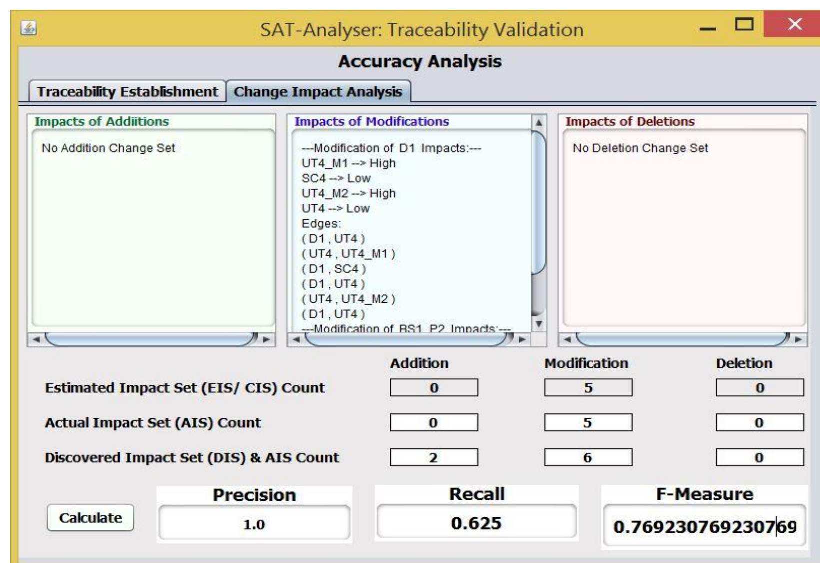


Figure 3-50 : SAT-Analyser CIA accuracy analysis window

The outcome of artefacts extraction in a current version of a project for each artefact type is shown in Figure 3-49. The identified count denotes the automated count of the items in the table for each artefact type. The user can select the correctly identified items with left mouse click and CTRL key on the keyboard together on a correct cell. Similarly, clicking twice on the same cell deselects a selection. Accordingly, the correctly identified count is a subset of the total identified count. Then, the actual total count of artefacts elements/ sub-elements is extracted based on expert knowledge and inserted manually in the text boxes next to the actual count. A click on the '*Calculate*' button automatically counts the number of selected items in the table and computes the accuracy measures.

The latest CIA outcome of a selected software traceability project containing impacted artefact element/ sub-elements for the three change types '*addition*', '*modification*' and '*deletion*' are displayed to increase the usability as shown in Figure 3-50. The CIA accuracy calculation uses the CIA categorization sets (see Subsection 2.6.1) EIS/ CIS, AIS and DIS. Thus, the count of estimated or the candidate artefact items shown in text areas is automatically displayed. The remaining AIS count which is a subset of EIS count has to be manually identified by an expert from the results shown in text areas and manually entered. Similarly, the total real impact set count which is the sum of DIS count and AIS count is entered based on expert knowledge before proceeding with the '*Calculate*' button.

3.6.2 Implementation of the network analysis module

The traceability validation module is developed by applying network analysis concepts over the traceability visualization graph. This is implemented using Python *NetworkX* libraries ("NetworkX," 2018) with Java-based GUIs. Python *Matplotlib* based and JavaScript D3.js based two traceability graph visualization extensions are also integrated into this module as described in the traceability visualization Subsection in 3.3.2 and 3.3.3 (Rubasinghe, Meedeniya, & Perera, 2018a). Figure 3-51 shows a Python code snippet used for obtaining centrality measures summary from a traceability graph.

```
...
import networkx as nx
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import numpy as np
UG=nx.read_gexf(GexfPathForValidation.path, node_type=None, relabel=True)
ddd=nx.degree(UG)
bt=nx.betweenness_centrality(UG)
ebt=nx.edge_betweenness_centrality(UG)
ec=nx.eigenvector_centrality_numpy(UG)
cc=nx.closeness_centrality(UG)
dc=nx.degree_centrality(UG)
idc=nx.in_degree_centrality(UG)
odc=nx.out_degree_centrality(UG)
print "Traceability Graph Info: #Nodes=",
nx.number_of_nodes(UG),' #Edges=',nx.number_of_edges(UG)
print "Max degree centrality:",
max(dc.iterkeys(), key=(lambda key: dc[key])),'=',dc[max(dc, key=dc.get)]
print "\t Max in-degree centrality:",
min(idc.iterkeys(), key=(lambda key: idc[key])),'=',idc[min(idc, key=idc.get)]
print "\t Max out-degree centrality:",
max(odc.iterkeys(), key=(lambda key: odc[key])),'=',odc[max(odc, key=odc.get)]
print "Max closeness centrality:",
max(cc.iterkeys(), key=(lambda key: cc[key])),'=',cc[max(cc, key=cc.get)]
print "Min betweenness centrality node:",
min(bt.iterkeys(), key=(lambda key: bt[key])),'=',bt[min(bt, key=bt.get)]
print "Max betweenness centrality edge:",
max(ebt.iterkeys(), key=(lambda key: ebt[key])),'=',ebt[max(ebt, key=ebt.get)]
print "Max eigenvector centrality:",
max(ec.iterkeys(), key=(lambda key: ec[key])),'=',ec[max(ec, key=ec.get)]
...
```

Figure 3-51 : Network analysis centrality measures code snippet

The main network analysis window facilitating analytical, interactive traceability visualization extensions, separate centrality measure options, overall measure summaries and centrality measure visualizations is shown in Figure 3-52. The output of the button '*Centrality Measures Textual Summary'* that summarizes the

maximum and minimum values of degree centrality, closeness, betweenness and Eigenvector centrality is shown in that example. Similarly, in detail analysis of each individual centrality measure is embedded into the left side centrality button series. The '*Info*' button provides the artefact details of involved traceability project graph as shown in Figure 3-53.

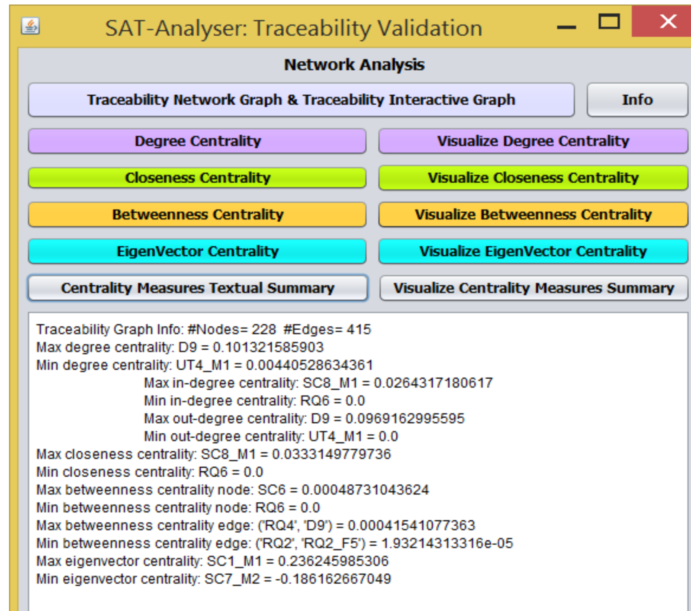


Figure 3-52 : SAT-Analyser network analysis main window

Artefacts Information

Requirement	Design	Source Code	Unit Test	Build Script
RQ1: route	D1: Guide	SC1: Bill	UT1: BillTest	BS1:
RQ2: manager	D2: Route	SC2: Driver	UT2: DriverTest	BS1_P1: [nbm-m...
RQ3: town	D3: Employee	SC3: Employee	UT3: EmployeeTest	BS1_P2: maven-...
RQ4: passenger	D4: Town	SC4: Guide	UT4: GuideTest	BS1_P3: maven-j...
RQ5: bill	D5: Driver	SC5: Manager	UT5: ManagerTest	
RQ6: employee	D6: Manager	SC6: Passenger	UT6: PassengerT...	
RQ7: tour	D7: Bill	SC7: Route	UT7: RouteTest	
RQ8: guide	D8: Tour	SC8: Tour	UT8: Tourmanag...	
RQ1_F1: town	D9: Passenger	SC9: Town	UT9: TourTest	
RQ1_F2: route le...	D2_F1: length	SC1_F1: date	UT10: TownTest	
RQ1_F3: town na...	D2_F2: duration	SC1_F2: tourID	UT1_M1: setUpCl...	
RQ1_F4: tour dur...	D2_F3: towns	SC1_F3: count	UT1_M2: tearDo...	
RQ2_F1: driver	D2_M1: getLength	SC1_F4: billID	UT2_M1: setUpCl...	
RQ2_F2: bill	D2_M2: setLength	SC1_M1: getdate	UT2_M2: tearDo...	
RQ2_F3: payment	D2_M3: getDurati...	SC1_M2: setdate	UT3_M1: setUpCl...	
RQ2_F4: main task	D2_M4: setDuration	SC1_M3: getCount	UT3_M2: tearDo...	
RQ2_F5: tour	D2_M5: getTowns	SC1_M4: setCount	UT4_M1: setUpCl...	
RQ3_F1: route	D2_M6: setTowns	SC1_M5: getBillID	UT4_M2: tearDo...	
RQ3_F2: details	D3_F1: empCode	SC1_M6: setBillID	UT5_M1: setUpCl...	
RQ3_F3: overnig...	D3_F2: name	SC3_F1: name	UT5_M2: tearDo...	

Figure 3-53 : Network analysis artefact information view

3.7 Tool performance analysis

We have measured the SAT-Analyser tool performance for its core functionalities traceability establishment and CIA process. The resource utilization is monitored in terms of elapsed time, memory consumption and CPU processing power consumption, where the tool contains a menu item for performance analysis. The in-built Java library classes; *runtime* and *ManageemntFactory* are used for the

implementation. The output is provided in both textual and graphically for the elapsed time, memory usage and CPU usage percentage. The Java-based *JFreeChart* (“JFreeChart,” 2018) is used for graph creation with features such as zooming, saving and manual alterations are provided for better analysis.

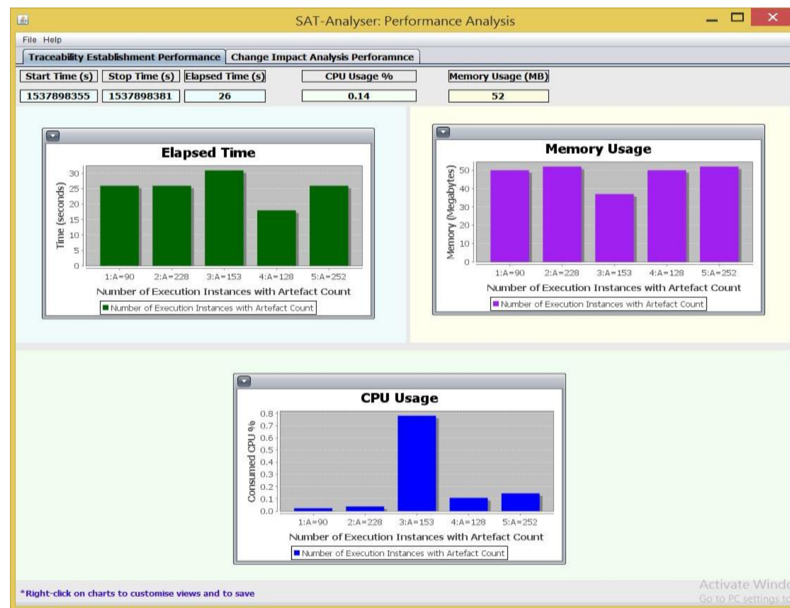


Figure 3-54 : SAT-Analyser traceability establishment performance analysis window

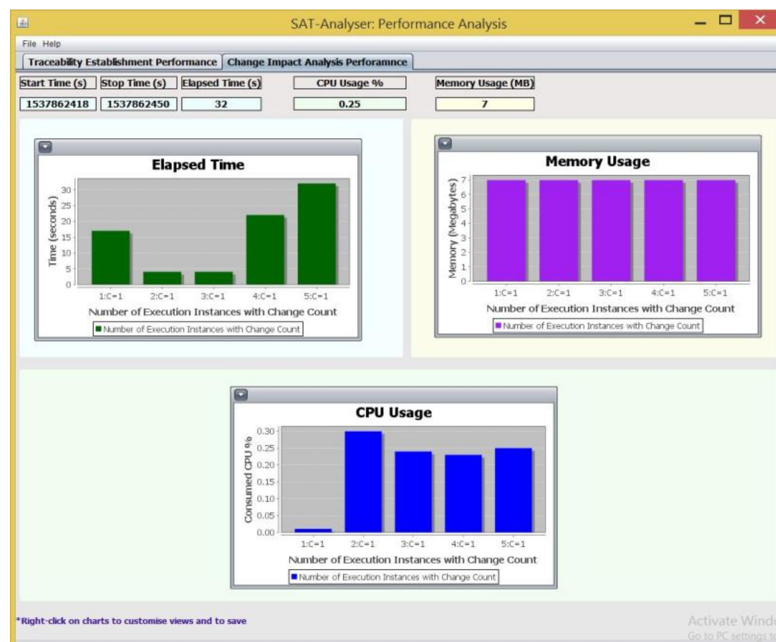


Figure 3-55 : SAT-Analyser CIA performance analysis outcome window

An example performance analysis output for traceability establishment activities of software projects and CIA activities are provided in Figure 3-54 and Figure 3-

55 respectively. Evidently, the initial execution of the tool has consumed a larger elapsed time in both traceability establishment and CIA as initially, it requires collecting dependencies, repositories and libraries.

The elapsed time, memory consumption and CPU usage percentage depend on the performance of used machine such that the SAT-Analyser would be executed more smoothly when the machine is in an idle state without any other heavy applications and background processes running. Furthermore, the size and complexity level of the software project used in SAT-Analyser for traceability establishment and CIA also affect the resource utilization such that tool execution would consume a larger amount of elapsed time, memory and CPU if the project contains a larger amount of artefact elements/ sub-elements/ trace relationships.

3.8 Conclusion

The context of software artefact traceability is strengthened with continuous integration capabilities in order to be compatible with the evolving DevOps environments. The existed SAT-Analyser tool is extended to support DevOps environments by addressing software artefacts related to the remaining phases of SDLC such as testing and maintenance phases that were not included in the initial tool. The artefact data pre-processing, traceability establishment and visualization is performed for DevOps related software artefacts based on the justifications obtained from current industry level employee feedback and literature. The existed traceability visualization is further enhanced with two additional variations for better interactivity and for the purpose of traceability analysis with better usability. Besides, change impact analysis model is designed with change detection and change propagation with the aid of a mathematical and weight system mainly based on the centrality measure; Eigenvector centrality and graph traversal algorithms. Moreover, multi-user accessibility is featured in the SAT-Analyser tool as a web-based version to improve the team-based usability in DevOps environments. The implementation of change impact analysis model and traceability validation along with traceability establishments are further evaluated on a heterogeneous case study basis in chapter 4.

Section 4

Evaluation

A case study based approach is used for evaluation of the proposed methodology, using a data set of heterogeneous software projects. The remaining subsections describe the data set, experimental results and analysis for the evaluation metrics.

4.1 Datasets and materials

A data set of 20 software engineering projects where the underlying technology is Java programming language is selected in different domains and scales as the dataset. Table 4.1 provides an overview of the considered software projects.

Table 4.1 : Dataset summary

	Project title	**Description**	**Software product measures**				**Scale**
			#Req.	***#Design classes***	***LOC***	***function calls***	
S1	Virtual historical site guide	Application to guide historical sites in Sri Lanka using virtual reality.	15 Large	9 Medium	3185 Medium	742 Medium	Medium
S2	Workout manager	Application to manage exercise routine using smartwatch and gamification.	9 Medium	11 Large	1333 Small	313 Small	Small
S3	Employee performance tracker	Mobile application to measure employee performance during professional travelling duties.	9 Medium	12 Large	2415 Medium	515 Medium	Medium
S4	Medical appointment manager	An android application to manage doctor/patient medical appointments.	9 Medium	6 Small	2362 Medium	529 Medium	Medium
S5	Task planner - PlanIt	Personal daily-tasks organizing system.	8 Small	4 Small	2977 Medium	662 Medium	Small
S6	Interactive book reader	A mobile app with augmented reality to visualize characters and scenarios in kid's books.	12 Medium	4 Small	5491 Large	939 Large	Large
S7	MyDrive multimedia library	Personal media content management system.	7 Small	7 Medium	2646 Medium	571 Medium	Medium
S8	E-School manager	An MIS to ease the activities of students and teachers.	10 Medium	6 Small	3460 Medium	490 Small	Medium
S9	Graphical password strategy	A system to maximize the user password space using memorable information.	6 Small	5 Small	1466 Small	475 Small	Small
S10	Hotel management Android app	An Android application to handle all hotel activities via a mobile.	19 Large	8 Medium	5579 Large	1141 Large	Large
S11	Expenses tracker	Mobile application to track daily income and expenses.	17 Large	12 Large	3355 Medium	782 Medium	Large
S12	Online developer	A system to generate a complete insight of a software	8 Small	10 Mediu	2269 Small	162 Small	Small

	profile analyser	developer based on the profile.		m			
S13	Computer-based psychotherapy	A system to identify and reduce the effects of mental health disorders with self-guided treatment.	16 Large	7 Medium	3712 Large	602 Medium	Large
S14	Child monitoring system	A monitoring system with play sound, voice over, listen, watch the child and call a neighbour.	10 Medium	12 Large	2288 Small	502 Medium	Medium
S15	PDF content search system	Desktop application to search through PDF files.	8 Small	7 Medium	3638 Large	873 Large	Large
S16	Disease management system	Integrated digital health system to manage patients with chronic disease, remotely using SMSs.	12 Medium	10 Medium	3700 Large	802 Large	Large
S17	HTTP2 support for Apache JMeter	Software plugin to adopt and implement HTTP2 support for performance measuring of JMeter application.	18 Large	19 Large	2649 Medium	628 Medium	Large
S18	Point of sales system	System for customer and order management in sales.	5 Small	5 Small	97 Small	16 Small	Small
S19	GuideME - smart tour guide	Tour guide system to display locations, accommodations, routes in Sri Lanka.	7 Small	11 Large	3208 Medium	676 Medium	Medium
S20	Tour management system	Tour booking management system for passengers and drivers and guides.	8 Small	9 Medium	2298 Medium	494 Small	Medium

The software product measures are prominent aspect in measuring software projects scale. Especially, Line of Code (LOC) and number of associated function calls of a software project are two common measures (Hattori, Guerrero, Figueiredo, Brunet, & Dam, 2008)(Li et al., 2013). However, these cannot be solely used as a metric in deciding a scale of an overall software project as they both are associated only with source code. For example, basic software problem having a smaller number of requirements may be complex to implement due to lack of technologies, coding abilities and refactoring methods, which may eventually increase the LOC or/and function calls count. Thus, the functional requirements count, number of classes in UML class diagram and LOC along with the number function calls of a project is considered as software product measures when deciding the project scale ("Measuring Requirements," 2018).

We have followed the Interquartile Range (IQR) methodology to scale the projects based on the overall median (Q2), the median in the lower half of data (Q1) and median in the upper portion of data (Q3) with respect to each selected software

product measure. Thus, a minimum, Q1, Q2, Q3 and a maximum value exist for requirements count, design classes count, LOC and function call count. Firstly, each project is assigned a subscale (small/ medium/ large) for each of that product measure such as a single project gets four subscales as described in Table 4.1.

The subscale is determined as small, if the value is greater than or equal to the minimum and less than or equal to Q1. Similarly, subscale medium is defined, if the value is greater than Q1 and less than Q2 while subscale is assigned as large if the value is greater than or equals to Q3 and also less than or equals to the maximum. Figure 4-1 boxplot illustrates the scale ranges according to Q1, Q2 and Q3 measures. The final scale of the project is obtained based on the highest subscale probability. Further, if any two subscales are similar and the remaining two subscales also similar which results in an equal probability; the final scale is decided manually based on project area, scope and codebase development effort.

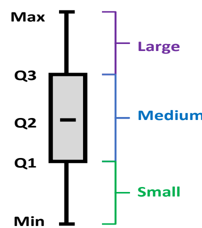


Figure 4-1 : Project scale

4.1.1 Pre-defined categorization of change types

We have defined 17 change types for artefact changes and three-to-five unique change types are applied for a given case study, covering all 17 change types for the testing purpose. These changes are selected based on the possibilities in a practical software development. Based on the survey among DevOps practitioners, currently, there exists no mechanism to track the heterogeneous artefact level changes covering every phase of SDLC. Following are the defined change types.

- **C1:** Add a main requirement
- **C2:** Add a moderate importance requirement
- **C3:** Add a low importance requirement
- **C4:** Modify a requirement
- **C5:** Delete a requirement

- **C6:** Add a design component
- **C7:** Modify a design component
- **C8:** Delete a design component
- **C9:** Add a source code artefact
- **C10:** Modify a source code artefact
- **C11:** Delete a source code artefact
- **C12:** Add a unit-test artefact
- **C13:** Modify a unit-test artefact
- **C14:** Delete a unit-test artefact
- **C15:** Add a configuration artefact
- **C16:** Modify a configuration artefact
- **C17:** Delete a configuration artefact

4.1.2 Evaluation environment specification

The SAT-Analyser tool performance depends on the execution environment as any other software tool. Thus, the evaluation results presented in this chapter depends on the used environment parameters. SAT-Analyser is evaluated in an environment specification with Core i5-321M CPU @ 2.50GHz processor, 700GB storage, 4GB RAM and Windows 8.1 Pro operating system.

4.2 Experimental results: case study 1 (POS system)

This section presents an overview of the selected case study, S18: Point of Sales system for a shop, where a customer can place orders consisting 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. The system records the customer details with name and location for delivery purposes. Also, the system facilitates the ability to record item details with an item number and price. A customer can send and receive orders using the system. These requirements are stated in the software requirement specification in natural language. Figure 4-2 shows the natural language requirements considered for this study. The corresponding design in UML class diagram is shown in Figure 4-3. 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 a set of Item entities, there is

an aggregation relationship. There is a composition relationship as a strong aggregation between the classes Customer and Order. 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 4-2 : POS system description

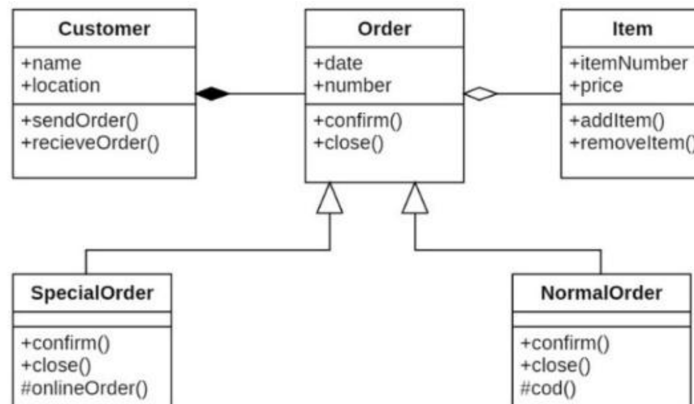


Figure 4-3 : POS system design diagram

The relevant source code artefacts are given in Java programming language as a set of class files and unit test scripts are provided in JUnit test files. Further, as configuration file, a Maven build script file used for building the POS system is considered. The artefact files are provided in the SAT-Analyser tool web site ("SAT-Analyser," 2018).

Artefacts Information

Requirement	Design	Source Code	Unit Test	Build Script
RQ1: normal order	D1: Customer	SC1: Customer	UT1: CustomerTest	BS1: POSSystem
RQ2: special order	D2: Order	SC2: Item	UT2: ItemTest	BS1_P1: [selenium-java
RQ3: item	D3: SpecialOrder	SC3: NormalOrder	UT3: NormalOrderTest	BS1_P2: operadriver
RQ4: customer	D4: NormalOrder	SC4: Order	UT4: OrderTest	BS1_P3: junit
RQ5: order	D5: Item	SC5: SpecialOrder	UT5: SpecialOrderTest	
RQ1_F1: items.an	D1_F1: name	SC1_F1: name	UT1_M1: setUpClass	

Figure 4-4 : SAT-Analyser main artefact summary for POS system

The identified main artefact elements by the tool SAT-Analyser are listed in Figure 4-4 followed by the tool generated unique identifier of each artefact. Further, there exists artefact sub-elements for methods, attributes (fields) and

plugins as partially shown in Figure 4-4 with _F, _P and _M notations. Table 4.2 summarises the manual artefact identification and categorization of the POS system based on expert knowledge such as by a requirement engineer.

Table 4.2 : Artefact categorization: POS system

Artefact type	Low	Medium	High
Requirement	RQ1, RQ2	RQ3	RQ4, RQ5
Design	D1, D2	D3	D4, D5
Source code	S1, S2	S3	S4, S5
Test script	UT1, UT2	UT3	UT4, UT5
Configuration files	-	-	BS1

4.2.1 Evaluation of traceability establishment component

Figure 4-5 represents a part of the final traceability established relations file in XML format. It contains a source to target format depicting directed traceability relationships. For example the D4: Normal Order is connected to SC1: Customer as one of the traces, showing that any alteration occurred in Normal Order design class would affect the Customer source code class.

```
<?xml version="1.0" encoding="UTF-8"?>
<Relations>
    <Relation id="1">
        <SourceNode>D4</SourceNode>
        <RelationPath>UMLClassToSourceClass</RelationPath>
        <TargetNode>SC1</TargetNode>
    </Relation>
    <Relation id="2">
        <SourceNode>D4_F1</SourceNode>
        <RelationPath>UMLAttributeToSourceField</RelationPath>
        <TargetNode>SC1_F1</TargetNode>
    </Relation>
    <Relation id="3">
```

Figure 4-5 : POS system Relations.xml instance

Figure 4-6 provides a section of the full traceability graph for the POS system. Nodes denote the heterogeneous software artefacts and edges represent the traceability relationship links. Colour codes are applied for each category of nodes to enhance the usability aspects. The node BS1 in black shows the Maven pom.xml build script file and each source code (SC) class is visualized in red coloured nodes. The notations D, RQ, UT stand for the design diagram,

requirement item and unit test class item respectively. Moreover, this interactive traceability graph is customizable.

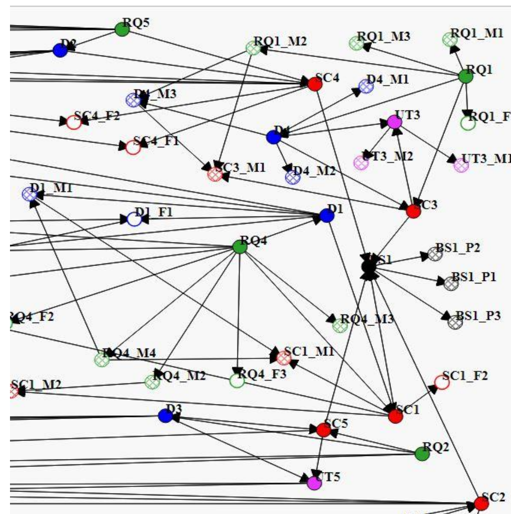


Figure 4-6 : Part of the traceability visualization graph - POS system

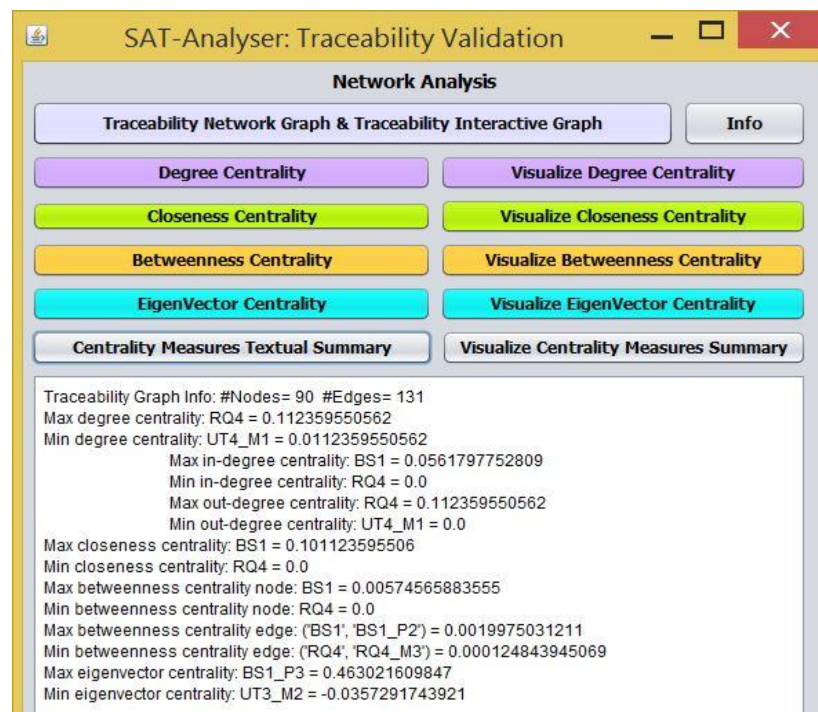


Figure 4-7 : Network analysis summary - POS system

Figure 4-7 states the SAT-Analyser tool computed centrality measures summary for the network analysis based traceability validation. The Maven build script artefact (BS1) holds the maximum betweenness and closeness centrality measure values since this POS case study has one Maven build script file that is linked with every source class artefact by verifying the centrality result as accurate.

4.2.2 Evaluation of continuous integration process

Among the defined 17 change types for the impact analysis process, five change types are applied to this POS case study project as follows.

- **C4:** Modify a requirement
- **C9:** Add a source code artefact
- **C10:** Modify a source code artefact
- **C12:** Add a unit-test artefact
- **C15:** Add a configuration artefact

Figure 4-8 shows the detection of the five change types in the tool's Change Detection results window for each particular artefact category listed according to *addition*, *modification* and *deletion*. The corresponding impact analysis results of the five change types are calculated and summarized in the Impact Analysis Results window as shown in Figure 4-9. The impacted nodes/ edges are listed using the influential factor values obtained through EVC. For example, the addition of unit test artefact element (UT5_M3: InvokeTest method) has not affected any other since its own influential factor has been a *low* value.

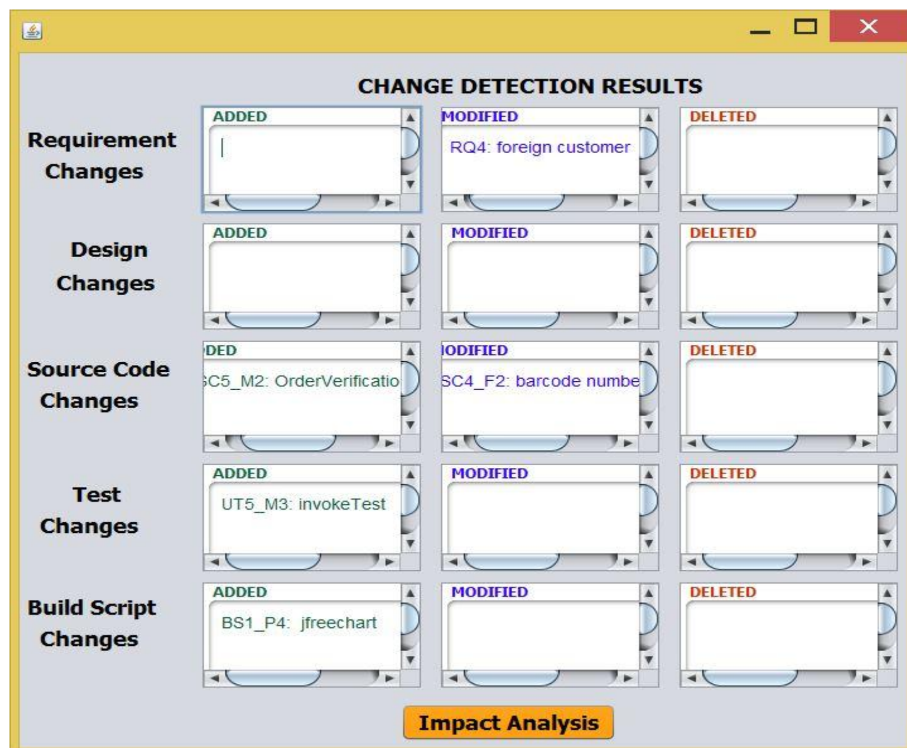


Figure 4-8 : POS system change detection window

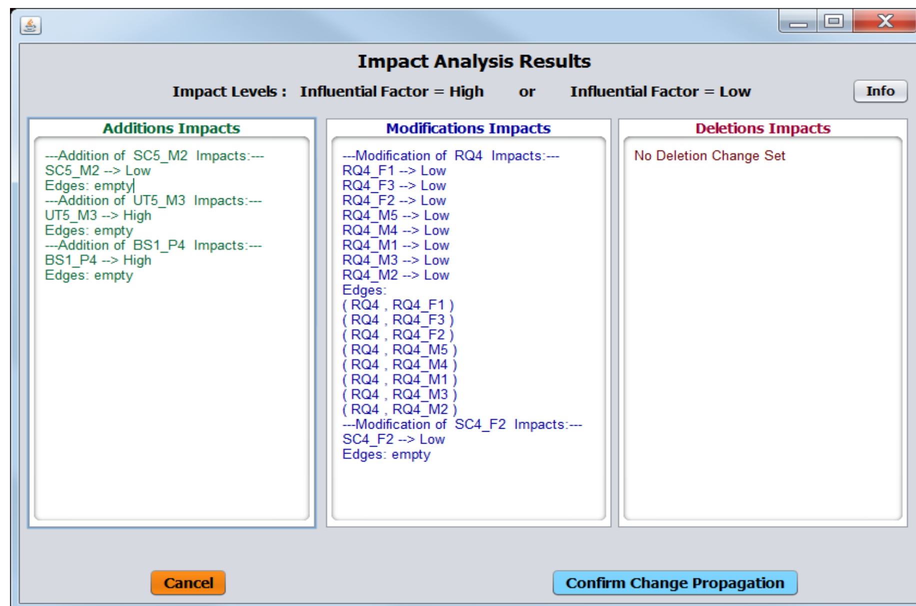


Figure 4-9 : POS system impact analysis window

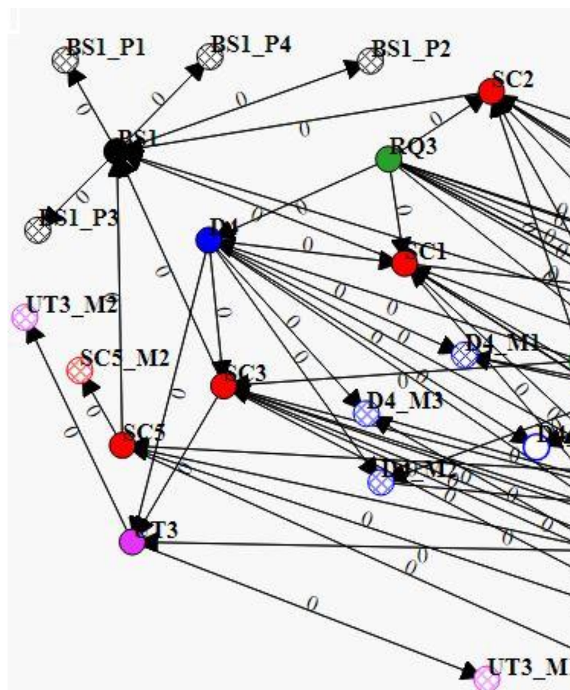


Figure 4-10 : POS system change propagation instance

The propagation of changes is visualized in the traceability graph with impact analysis results such as 1 for *high* and 0 for *low*. The node impact analysis results are shown when hovered on each node in real-time. In this artefact change example, the scenario is relevant to one constraint defined in the SAT-Analyser tool CIA process. As this scenario contains artefact *additions* as change types (source code addition, test script element addition, build script element addition),

all artefact types must be submitted with relevant affecting results of newer artefact additions. Thus, the system considers this type of a change integration as a re-establishment of traceability. Figure 4-10 shows a part of the change propagated traceability graph where the newly added build script artefact plugin (BS1_P4: jfreechart) can be seen as a new node linked with its mother artefact node BS1.

4.2.3 Performance analysis

Figure 4-11 provides the statistical analysis results for the CIA process conducted on the POS system. The AIS count is completely same as the EIS count based on SAT-Analyser's impact analysis, signifying the identified impacts are accurate. However, there are two impacts that have not been identified by the tool that is relevant to the DIS set. The *addition* of SC5_M2 must impact on a corresponding unit test (UT) item and the modification of SC4_F2 may impact on a UT item which is missing in the obtained EIS. Thus, the recall and F-measure are more than 0.95 while precision is 1.0 successfully.

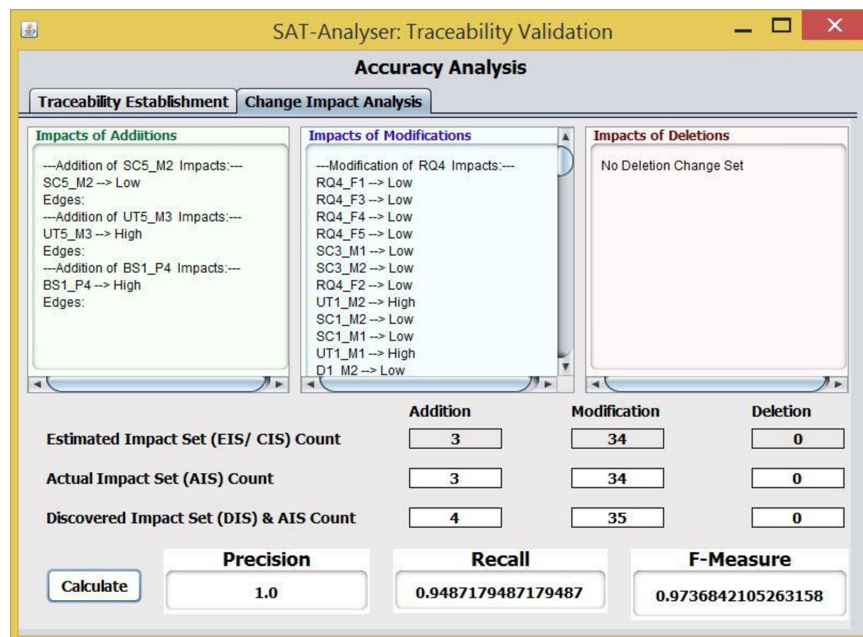


Figure 4-11 : CIA statistical analysis results: POS system

The performance of CIA process of the case study POS with respect to time consumption, CPU and memory consumption is shown in the Figure 4-12 for the above demonstrated 5 changes one at a time such as C4, C9, C10, C12 and C15 respectively. According to the variations in the results, it is observable that the

performance of the CIA process depends on the change type and that particular changed artefact item's nature on the traceability network.

Thus, the time, memory and CPU consumption is higher for the C4 in this scenario which represents a modification done to a requirement artefact item such that RQ4: Customer has been modified into RQ4: the Foreign Customer. It has been occupied more resources since it is having a higher number of trace links with design, source code and unit test artefacts that result in having a larger number of affected items. According to the defined CIA rule-based Algorithm 3:11, a requirement artefact is supposed to check the maximum number of paths in calculating the impact sets. Therefore, the graph traversal consumes a higher resource amount during the CIA process of such a change done on a significant artefact item. Remaining four changes C9, C10, C12 and C15 have occupied lesser similar amounts of resources since they all are later stages artefacts like source code, unit test and build script that are having a lesser number of relationships. According to these results, the *modifications* are requiring a considerably higher amount of resources compared to artefact *additions*.

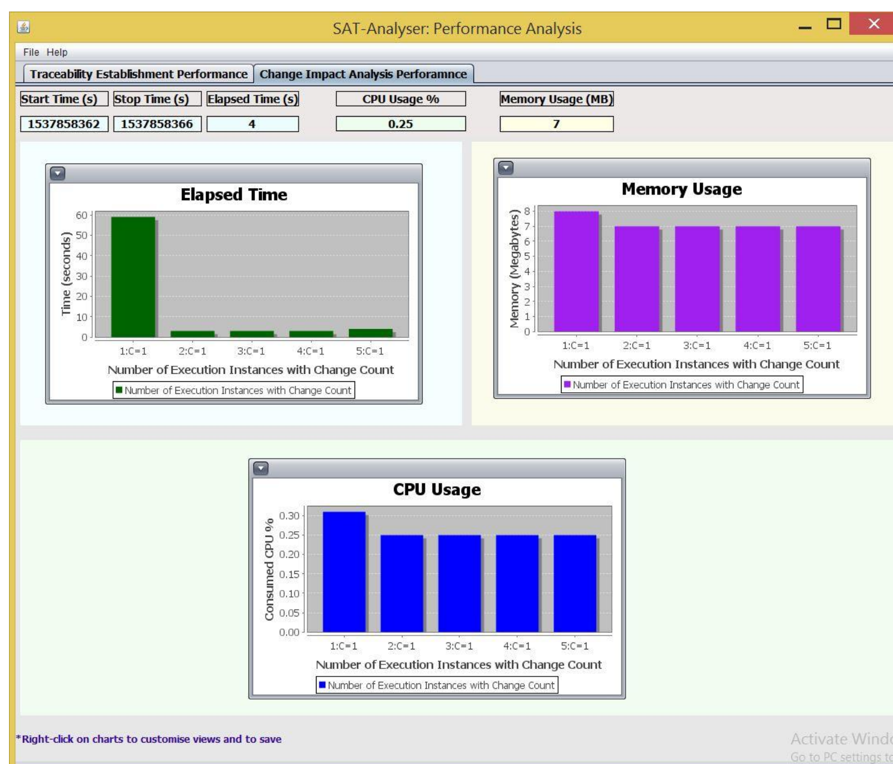


Figure 4-12 : CIA performance analysis results: POS system

4.3 Experimental results: case study 2 (Tour management system)

This section presents the considered artefacts and the evaluation results obtained by SAT-Analyser. The selected case study, S20: Tour Management System manages tours that mainly address the types of employees, namely manager, driver, tour guide and a passenger who books a tour. The system records both employees' details and passenger details. The system provides a list of available tours along with a date. The manager can reserve a tour for a passenger, can assign the route to a tour, assign a driver for each tour, create a bill to the passenger and a passenger can book a tour using this application. Figure 4-13 represents the requirements description of this Tour Management case study.

> In a tour management system there are three types of employees, namely manger, driver and guide. An employee must record the employee code, name, address and a contact number. A tour is identified by a unique tour ID and a date. The manager reserves a tour for a passenger. This is one of the main requirements of the system. When a passenger registers for a tour, he/ she provide the name, address, contact number, birth date, gender and preferences. Another main task of the manager is that manager assigns route to a tour. A route has a route length, tour duration and town names. Moreover, manager assigns a driver for each tour. Additionally, when a passenger makes the payment for a tour, the manager creates bill to the passenger. A bill consists of the date, passengers count and tour ID. Furthermore, a guide elaborates each tour for the passengers during a tour. Further, a route has one or more towns. For each route, a town records its overnight stay details.

Figure 4-13 : Tour management system description

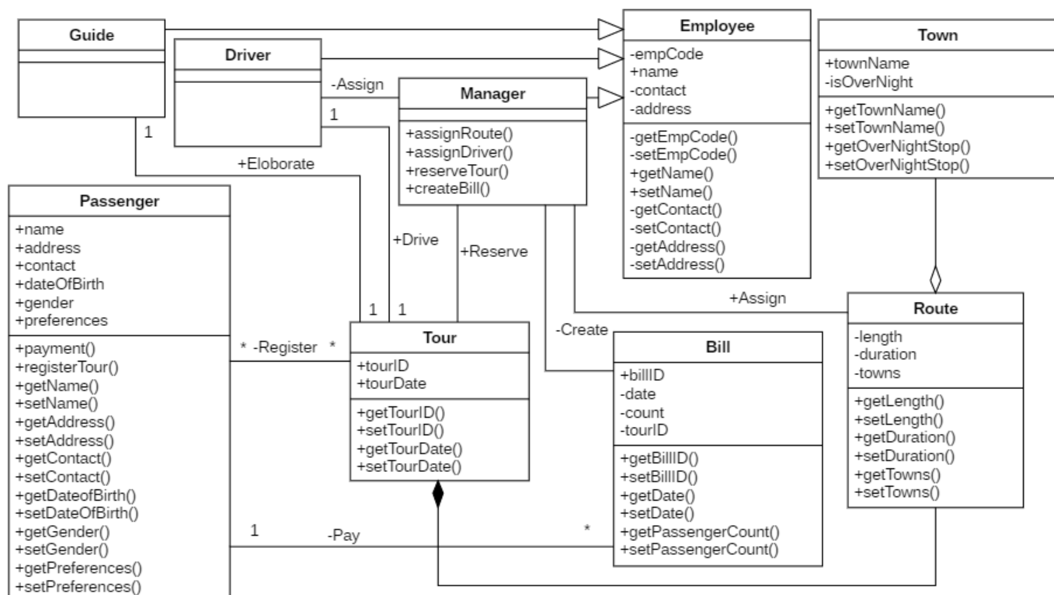


Figure 4-14 : Tour management system design diagram

Figure 4-14 shows the corresponding UML class diagram of the Tour Management case study consisting of nine classes. There is an inheritance relationship in Guide, Driver and Manager with Employee class. An aggregation relationship between Town and Route classes and a composition between Tour and Route classes exist in the design with other association relationships. The relevant Java source code artefact, JUnit test artefact and the used Maven build script artefact file of the case study are provided in the SAT-Analyser tool web portal (“SAT-Analyser,” 2018).

The identified main artefact elements by the tool SAT-Analyser are listed in Figure 4-15 followed by the tool generated unique identifier of each artefact. Further, there exists artefact sub-elements for methods, attributes (fields), and plugins as partially shown in Figure 4-15 with _F, _P and _M notations. Table 4.3 summarises the manual artefact identification and categorization of Tour Management system based on expert knowledge such as by a requirement engineer/ software engineer involved in the project.

Artefacts Information

Requirement	Design	Source Code	Unit Test	Build Script
RQ1: route	D1: Guide	SC1: Bill	UT1: BillTest	BS1: TourManagementSystem-parent
RQ2: manager	D2: Route	SC2: Driver	UT2: DriverTest	BS1_P1: [nbm-maven-plugin
RQ3: town	D3: Employee	SC3: Employee	UT3: EmployeeTest	BS1_P2: maven-compiler-plugin
RQ4: passenger	D4: Town	SC4: Guide	UT4: GuideTest	BS1_P3: maven-jar-plugin
RQ5: bill	D5: Driver	SC5: Manager	UT5: ManagerTest	
RQ6: employee	D6: Manager	SC6: Passenger	UT6: PassengerTest	
RQ7: tour	D7: Bill	SC7: Route	UT7: RouteTest	
RQ8: guide	D8: Tour	SC8: Tour	UT8: TourmanagementTest	
RQ1_F1: town	D9: Passenger	SC9: Town	UT9: TourTest	
RQ1_F2: route length	D2_F1: length	SC1_F1: date	UT10: TownTest	

Figure 4-15 : SAT-Analyser main artefact summary for tour management system

Table 4.3 : Artefact categorization: tour management system

Artefact type	Low	Medium	High
Requirement	RQ1, RQ3	RQ2, RQ6, RQ8	RQ4, RQ5, RQ7
Design	D2, D4	D1, D3, D5, D6	D7, D8, D9
Source code	S7, S9	S2, S3, S4, S5	S1, S6, S8
Test script	UT7, UT9, UT10	UT2, UT3, UT4, UT5	UT1, UT6, UT8, UT9
Configuration files	-	-	BS1

4.3.1 Evaluation of traceability establishment component

```
<Relation id="155">
      <SourceNode>RQ2_F2</SourceNode>
      <RelationPath>ReqFieldToUMLOperation</RelationPath>
      <TargetNode>D9_M11</TargetNode>
</Relation>
<Relation id="156">
      <SourceNode>RQ1</SourceNode>
      <RelationPath>ReqClassToUMLClass</RelationPath>
      <TargetNode>D6</TargetNode>
</Relation>
```

Figure 4-16 : Tour management system Relations.xml instance

A part of the tool generated traceability relations wrote down in the XML format is shown in Figure 4-16. For instance, a relation between RQ1: Route to D6: Manager can be seen as a directed relationship since Manager is the person who assigns a Route to each Tour. Figure 4-17 provides a section of the full traceability graph for the Tour Management system.

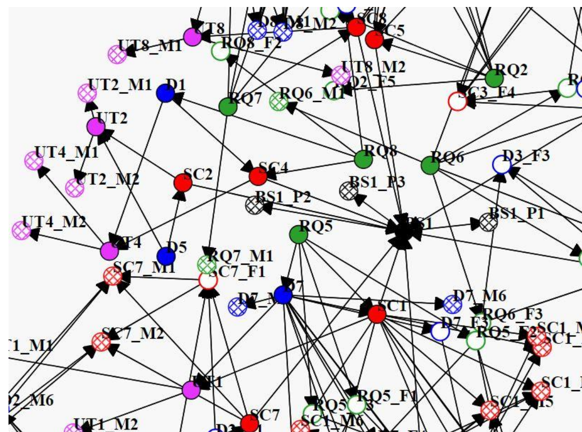


Figure 4-17 : Traceability visualization - tour management system

Figure 4-18 states the SAT-Analyser tool computed centrality measures summary for the network analysis based traceability validation. The build script artefact (BS1) holds the maximum values for betweenness and closeness centrality measures. This case study has only single Maven build script and it is related with each and every source class artefact, hence the result is acceptable. One of the maximum Eigenvector centrality is held by the node SC6_M4 that denotes the method setPreferences () in the Java Tour class which can be considered as one of the highly important artefacts in this case study.

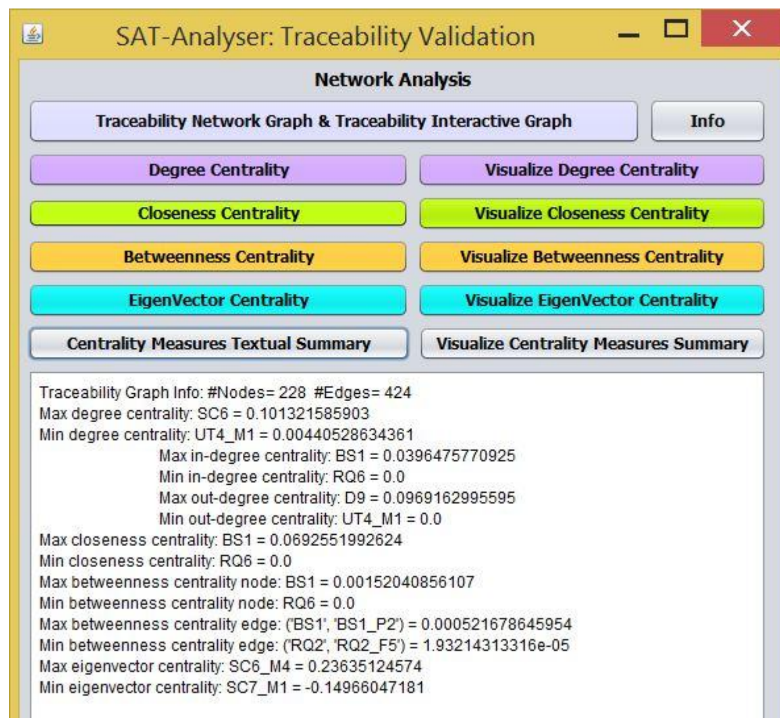


Figure 4-18 : Network analysis summary - tour management system

4.3.2 Evaluation of continuous integration process

From the defined 17 change types for the impact analysis process, another five change types are applied to this Tour Management case study.

- **C2:** Add a moderate importance requirement
- **C5:** Delete a requirement
- **C6:** Add a design component
- **C13:** Modify a unit-test artefact
- **C17:** Delete a configuration artefact

Figure 4-19 shows the corresponding change detection results obtained by the SAT-Analyser tool. The performed five changes are accurately detected by displaying the affected artefact ID and name. The performed CIA results are shown in Figure 4-20. For example, for C13, the modified unit test artefact (UT5: ManagerTest) has impacted on its two child nodes UT5_M1:setUpClass method and UT5_M2:tearDownClass method which has a lower impact value. The propagated changes are re-visualized and a part of the traceability graph is shown in Figure 4-21. For instance, the newly added D9_M15 is newly represented in the graph while BS1_P2 has removed and earlier BS1_P3 has become BS1_P2 by making the IDs consistent.

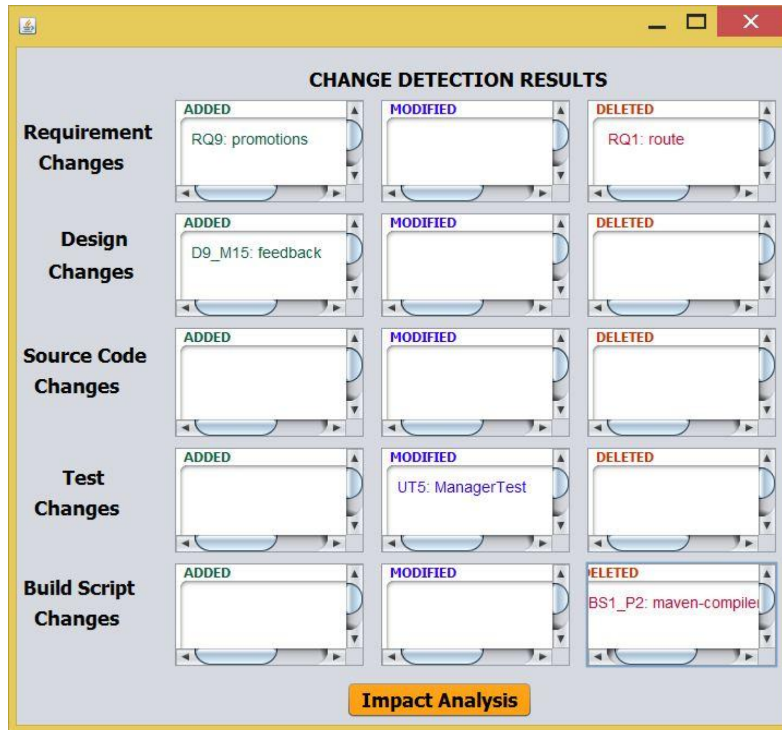


Figure 4-19 : Tour management system change detection window

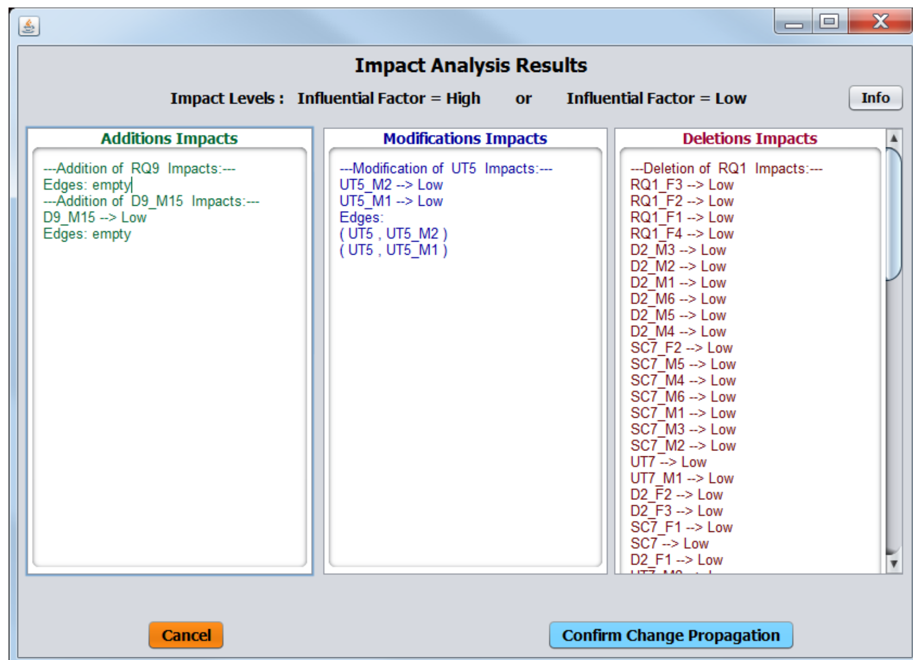


Figure 4-20 : Tour management system impact analysis window

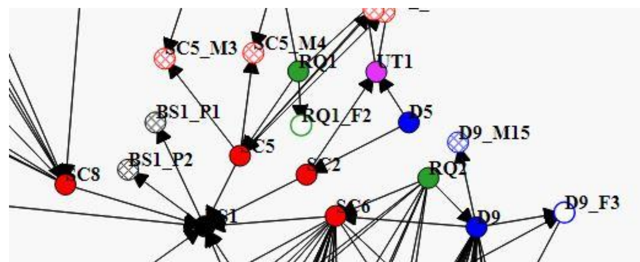


Figure 4-21 : Tour management system change propagation instance

4.3.3 Performance analysis

The CIA accuracy of the Tour Management example is shown in Figure 4-22. The *modification* and *deletion* change types related impacts are completely identified by the tool. However, there are five missing impact items in the *addition* change type since the corresponding artefact elements are not modified according to the added changes during CI. Thus, the addition of RQ9 must impact on a design (D), source code (SC) and a UT item while the addition of D9_M15 must impact on an SC sub-element and may impact on a UT item. The CIA process has obtained 0.86 recall, 0.93 F-measure and 1.0 precision.

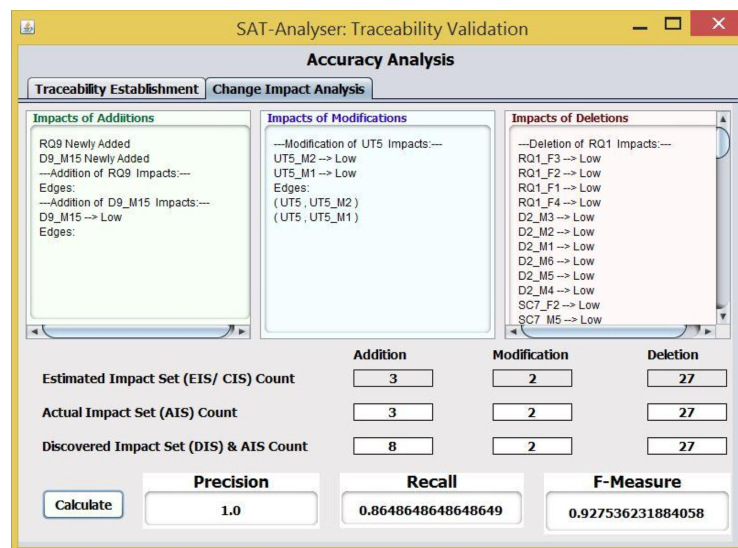


Figure 4-22 : CIA statistical analysis results: tour management system

The resource consumption of the CIA process for each change type is shown in Figure 4-23. The bar instances in each graph show five changes C2, C5, C6, C13 and C17. The memory consumption of each change is the same, since all are *additions* and *deletions* while C13 has been a *modification,* but on a later stage unit test artefact has fewer trace relations. Moreover, the highest CPU consumption is taken by the C5 where a requirement RQ1: Route has been deleted which affects a larger number of related trace links. The second highest CPU consumption occurs for the other artefact deletion C17 where a build script artefact item is deleted. It is observable that *modifications* and *deletions* tend to consume more resources compared to *additions*.

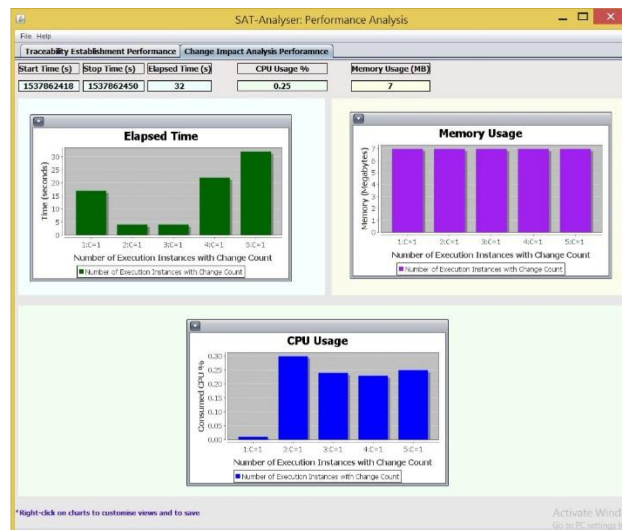


Figure 4-23 : CIA performance analysis results: tour management system

4.4 Experimental results: case study 3 (MyDrive multimedia library)

As the third case study, we have selected, S7**:** MyDrive Multimedia Library, which is a personal media content management system. A user can store and manage own favourite media contents such as music, video or pictures. It ensures user privacy rather than storing in any content management system. The requirements of the Multimedia Library system are provided in a text format as shown in Figure 4-24 where it describes the major functionalities required such as managing multimedia file contents and altering metadata of files by a user.

> User has to create user accounts in the system and login to the system. User can edit profile and logout anytime. Each multimedia file contains a file ID, original name, publicity and file type. Then user can upload multimedia files, search multimedia files, manage uploaded files, download and delete files. Multimedia files can be in three kinds such as image, video or an audio. Each multimedia file has at least one metadata associated. A user's all multimedia files have a folder to storage. That folder must have a folder ID and some metadata. Further, user can edit metadata of a multimedia file.

Figure 4-24 : MyDrive multimedia library system description

The UML class diagram is shown in Figure 4-25. There exists an inheritance relationship in Image, Video and Audio classes with the class MultimediaFile which is the parent class of them. Further, an aggregation and a composition relationship exist for MultimediaFile class with Folder and Metadata classes respectively. The corresponding Java source code artefacts, unit test script artefact in JUnit test file format and the Maven build script associated with the case study building are listed in the SAT-Analyser tool web site (“SAT-Analyser,” 2018).

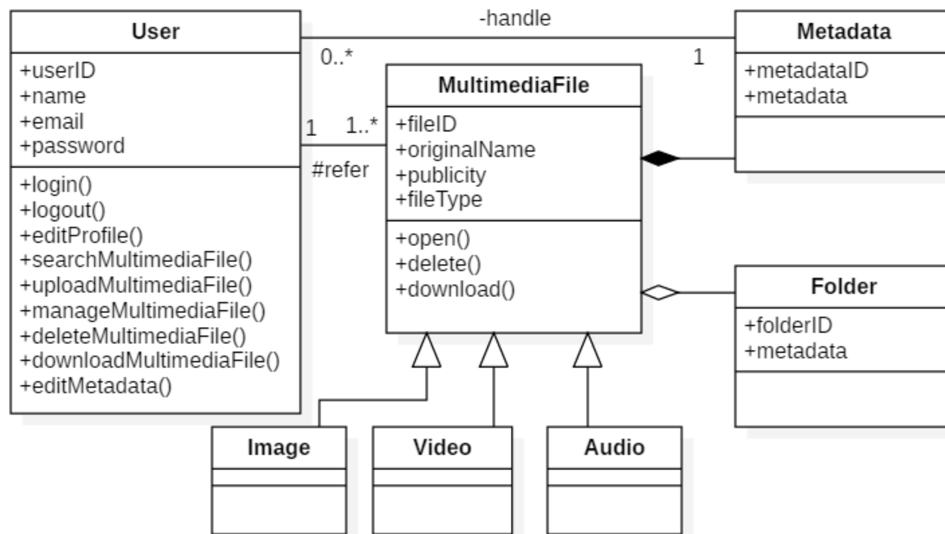


Figure 4-25 : MyDrive multimedia library system design diagram

Artefacts Information

Requirement	Design	Source Code	Unit Test	Build Script
RQ1: file	D1: User	SC1: AdminController	UT1: FileTest	BS1: multimedialibrary
RQ2: folder	D2: MultimediaFile	SC2: AdminServiceImpl	UT2: folderControllerTest	BS1_P1: [spring-boot-maven-plugin
RQ3: user	D3: Metadata	SC3: CustomSolrConfiguration	UT3: loginControllerTest	BS1_P2: maven-compiler-plugin
RQ1_F1: image	D4: Folder	SC4: DemoApplication	UT4: metadataControllerTest	
RQ1_F2: multimedia	D5: Image	SC5: File	UT5: SolrControllerTest	
RQ1_F3: kind	D6: Video	SC6: FileServiceImpl	UT6: SolrTest	
RQ1_F4: file ID	D7: Audio	SC7: FileSystemStorageService	UT7: UserImplTest	
RQ1_F5: publicity	D1_F1: userID	SC8: FileUploadController	UT8: UserTest	
RQ1_F6: video	D1_F2: name	SC9: Folder	UT1_M1: getFile	
RQ1_F7: audio	D1_F3: email	SC10: FolderController	UT5_M1: getSolrSearch	
RQ1_F8: metadatum	D1_F4: password	SC11: Folderinfo	UT5_M2: getFile	
RQ1_F9: original name	D1_M1: login	SC12: FolderinfoServiceImpl	UT6_M1: getSolrObject	
RQ1_M1: associate	D1_M2: logout	SC13: FolderServiceImpl	UT8_M1: getUser	
RQ2_F1: folder ID	D1_M3: editProfile	SC14: LoginController		

Figure 4-26 : SAT-Analyser artefact summary for MyDrive multimedia library system

The identified main artefact elements by the tool SAT-Analyser are listed in Figure 4-26 followed by the tool generated unique identifier of each artefact. Further, there exist artefact sub-elements for methods, attributes (fields) and plugins as partially shown in Figure 4-26 with _F, _P and _M notations. Table 4.4 summarises the manual artefact identification and categorization of MyDrive Multimedia Library system based on expert knowledge.

Table 4.4 : Artefact categorization: MyDrive multimedia library system

Artefact type	Low	Medium	High
Requirement	-	RQ2	RQ1, RQ3
Design	D3, D2	D4	D1, D2, D5, D6, D7
Source code	S4, S3, S7, S11, S12, S13	S2, S6, S9, S10	S1, S3, S5, S8, S14
Test script	UT4, UT5, UT6	UT2, UT7	UT1, UT3, UT8
Configuration files	-	-	BS1

4.4.1 Evaluation of traceability establishment component

According to Figure 4-27, the established trace relationships are written down into the pre-defined XML format. For instance, it shows two intra-relationships among SC11: FolderInfo with its source methods SC11_M8 and SC11_M9. Figure 4-28 provides a section of the full traceability graph for the Multimedia Library system.

```xml
<Relation id="112">
    <SourceNode>SC11</SourceNode>
    <RelationPath>SourceClassToSourceMethod</RelationPath>
    <TargetNode>SC11_M8</TargetNode>
</Relation>
<Relation id="113">
    <SourceNode>SC11</SourceNode>
    <RelationPath> SourceClassToSourceMethod </RelationPath>
    <TargetNode>SC11_M9</TargetNode>
</Relation>
```

Figure 4-27 : Multimedia library system Relations.xml instance

Figure 4-28 : Traceability visualization - multimedia library system

SAT-Analyser: Traceability Validation

Network Analysis

Traceability Network Graph & Traceability Interactive Graph | Info

Degree Centrality | Visualize Degree Centrality

Closeness Centrality | Visualize Closeness Centrality

Betweenness Centrality | Visualize Betweenness Centrality

EigenVector Centrality | Visualize EigenVector Centrality

Centrality Measures Textual Summary | Visualize Centrality Measures Summary

Traceability Graph Info: #Nodes= 153 #Edges= 194
Max degree centrality: SC5 = 0.236842105263
Min degree centrality: UT2 = 0.0
Max in-degree centrality: BS1 = 0.0921052631579
Min in-degree centrality: RQ2 = 0.0
Max out-degree centrality: SC5 = 0.230263157895
Min out-degree centrality: SC3_F1 = 0.0
Max closeness centrality: BS1 = 0.0950657894737
Min closeness centrality: RQ2 = 0.0
Max betweenness centrality node: SC5 = 0.00151766004415
Min betweenness centrality node: RQ2 = 0.0
Max betweenness centrality edge: ('RQ1', 'SC5') = 0.00154082100677
Min betweenness centrality edge: ('SC11', 'SC11_F5') = 4.29996560028e-05
Max eigenvector centrality: SC9_M6 = 0.612443916265
Min eigenvector centrality: SC5_M6 = -0.00053448819473

Figure 4-29 : Network analysis summary - multimedia library system

Figure 4-29 states the tool computed centrality measures summary for the network analysis based traceability validation. The Maven build script artefact (BS1) shows the maximum closeness and the highest in-degree centrality measures. As this case study also has only one Maven build script which is linked with all Java source class artefacts, the validation result is verifiable. One of the maximum EVC among all types of artefacts is taken by SC9_M6 that represents the Java method setFolderName () in the Folder Java class as one of the highly influenced artefacts.

4.4.2 Evaluation of continuous integration process

We have considered three change types as follows.

- **C7:** Modify a design component
- **C11:** Delete a source code artefact
- **C14:** Delete a unit-test artefact

The change detection results are shown in Figure 4-30 and the corresponding CIA results are given in Figure 4-31. The modification of D4_F2 artefact sub-element has impacted on itself and associated source code artefacts. Also, the deletion of SC5, UT8 has impacted many artefact items, but with a lower influential factor.

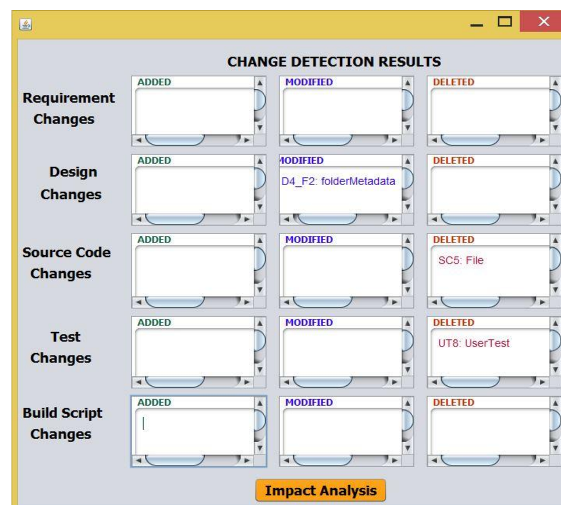


Figure 4-30 : Multimedia library system change detection window

Figure 4-32 provides a section of the change propagated traceability graph visualization. The modified design artefact sub-element D4_F2 can be seen in a larger size signifying the modification while SC5 has been removed. Regarding the D4_F2 design sub-element's impacted artefact set, there exist two highly

influential artefact items as SC9_M6 and SC9_M5. But they have not been propagated with the impact since the modified node D4_F2 itself contains a *low* influential value. Thus, the outgoing traces of a *low* influential impact node are discarded without further change propagation according to the defined CIA model.

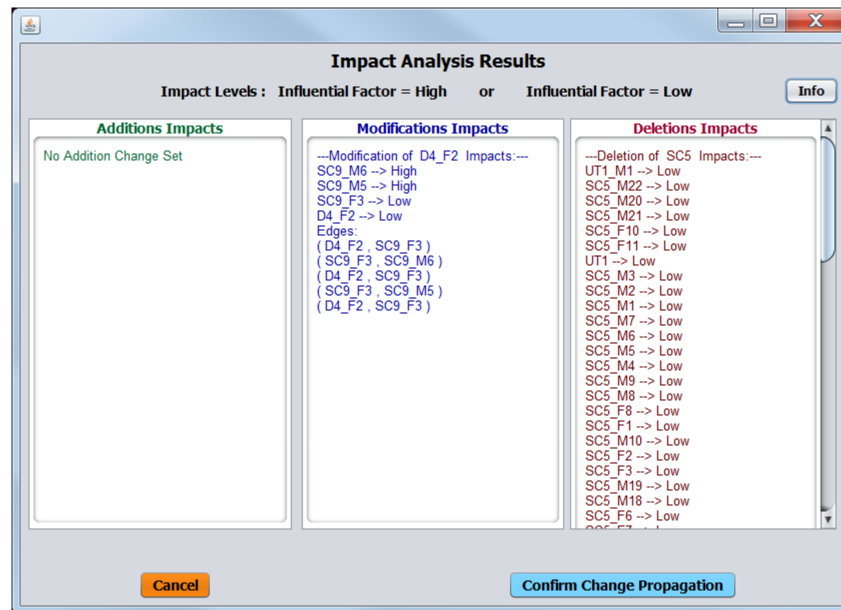


Figure 4-31 : Multimedia library system impact analysis window

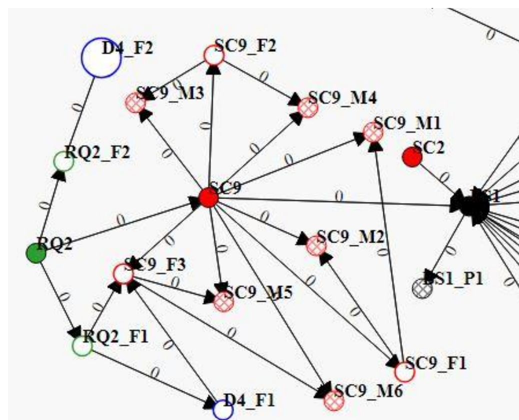


Figure 4-32 : Multimedia library system change propagation instance

4.4.3 Performance analysis

The corresponding accuracy results of the CIA process of this Multimedia Library system case study is shown in Figure 4-33. The impact set of *deletion* changes are completely identified by the SAT-Analyser tool. There is one missing item as a DIS element in the *modification* impacts as the alteration on D4_F2 may impact on a UT item. Accordingly, the final precision has been 1.0, recall is 0.98 and the F-measure is 0.99.

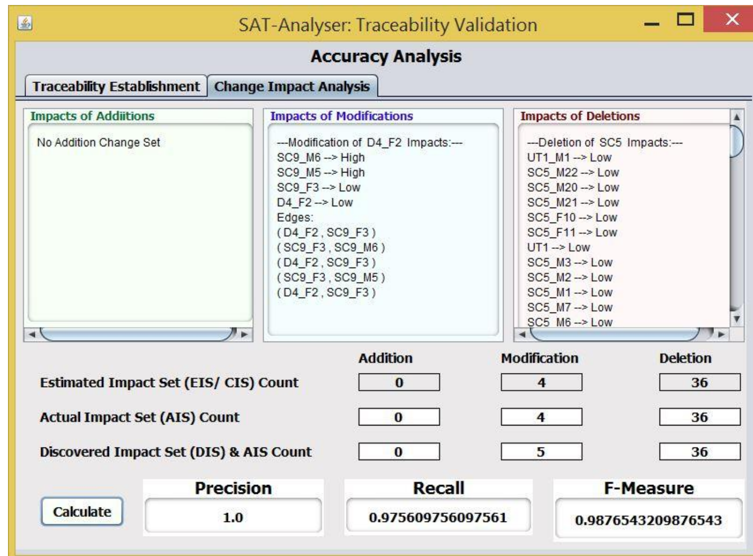


Figure 4-33 : CIA statistical analysis results: multimedia library system

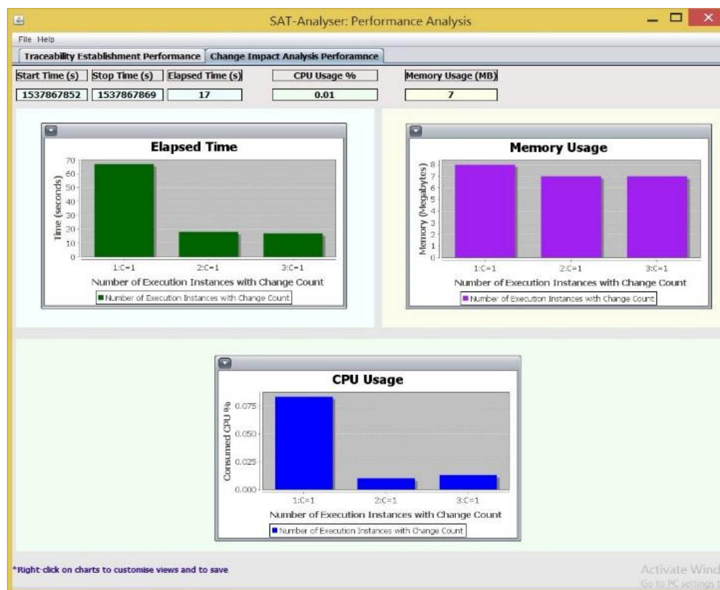


Figure 4-34 : CIA performance analysis results: multimedia library system

The performance analysis results of the CIA process for change types C7, C11 and C14 are given in Figure 4-34. The maximum time, memory and CPU usage has been reported for the C7, which is a *modification* change type on a design artefact such that D4_F2: metadata modified into D4_F2: folder metadata. The remaining C11 and C14 have consumed a lesser amount of resources since they both are later stage artefact *deletions* on source code and unit test artefact items, respectively, where a minimum number of trace links are associated. Although the deletion of SC5 in C11 change type has impacted on many items, the number of paths to be checked in graph traversal is lesser, compared to C7 where a design artefact is

modified. According to the defined rule-based graph traversal for CIA, the design level artefacts check design level, source code level and the unit test level, whereas source code artefacts are only subjected to check source code level and unit test level paths. Hence, it is justifiable that the artefact modifications tend to consume more resources, when the artefact belongs to an earlier stage.

4.5 Experimental results: case study 4 (Disease management system)

This section presents an overview of the software artefacts in case study 4 with the obtained evaluation results by SAT-Analyser. The selected case study, S16: *C-CARE* is a digital health system for chronic disease management and prevention. It helps health centres to manage admitted patients remotely by use of bulk SMS and also other citizens can subscribe to get health tips and monitor their patients without the patients availing themselves physically. The doctors can send SMS notifications to their patients either giving them appointments, advice the drugs they should take, food and exercises. The patients can attend to their daily jobs at the same time receiving treatment. Hence, this contributes to the economic growth of the country as opposed to when they are hospitalized. The provided text-based requirements are shown in Figure 4-35.

> The chronic disease management system has three types of users such as Admin, Doctor and Patient. Admin, each doctor and each patient has an associated login to system. There should be a username and a password to login. Every doctor who sign up with system must provide first name and last name. Each doctor has doctor records in a doc table. Every patient who sign up with system must provide full name and system generates an ID for each patient. Each patient has patient records in a table inside the system. Doctor and admin can access patient records and admin can access doctor records too. There is a SMS facility named healthSMS. Every patient can send SMS to doctors via the system's healthSMS. Doctors can view patient messages via the system's healthSMS. The healthSMS needs receiver, phone number and message to process. Each patient can be in two types such as a selected patient or an admitted patient. Admitted patients can prefill doctors. Also, admitted patients have admission details separately. Admin can access healthSMS.

Figure 4-35 : Disease management system description

The design level class diagram in UML notation for the Disease Management system is illustrated in Figure 4-36. Two composition relationships among classes Doctor and DoctorRecords and between Patient and PatientRecords can be seen in the design while other relationships being associations and inheritance.

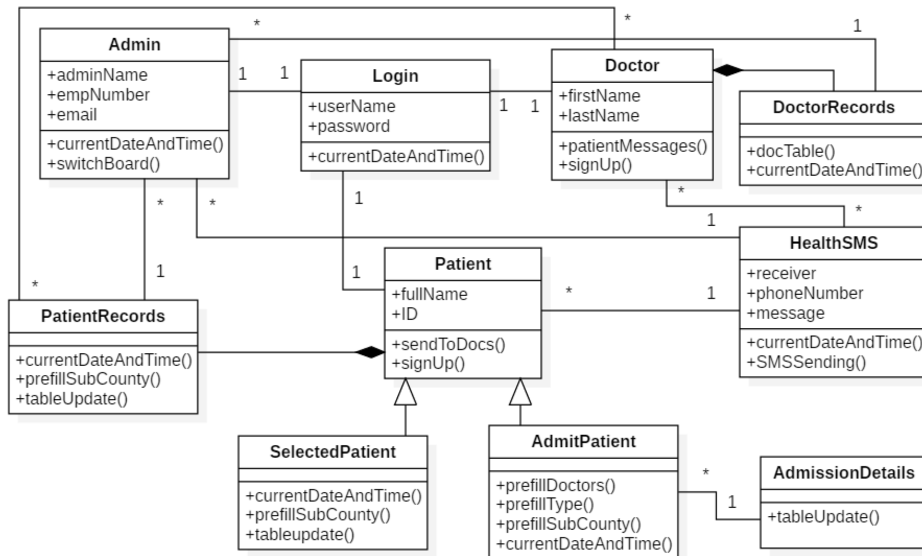


Figure 4-36 : Disease management system design diagram

Moreover, the associated source code, unit test and Maven build script artefacts of the case study are provided in the SAT-Analyser tool web portal ("SAT-Analyser," 2018). The tool identified artefact elements are listed in Figure 4-37 with the unique identifier of each artefact. Further, there exists artefact sub-elements for methods, attributes and plugins as partially shown in Figure 4-37 with _F, _P and _M notations. Table 4.5 summarises the manual artefact identification and categorization of Disease Management system based on expert knowledge.

Artefacts Information

Requirement	Design	Source Code	Unit Test	Build Script
RQ1: doctor	D1: Admin	SC1: Admin	UT1: AdminTest	BS1: DiseaseManagementSystem-parent
RQ2: patient	D2: Doctor	SC2: AdmissionDetails	UT2: AdmissionDetailsTest	BS1_P1: [nbm-maven-plugin
RQ3: name	D3: Patient	SC3: AdmitPatient	UT3: AdmitPatientTest	BS1_P2: maven-compiler-plugin
RQ4: admin	D4: SelectedPatient	SC4: DoctorRecords	UT4: DoctorRecordsTest	BS1_P3: maven-jar-plugin
RQ5: healthsm	D5: AdmitPatient	SC5: HealthSMS	UT5: HealthSMSTest	BS1_P4: junit
RQ1_F1: last name	D6: HealthSMS	SC6: Login	UT6: LoginTest	BS1_P5: maven-source-plugin
RQ1_F2: first name	D7: AdmissionDetails	SC7: PatientMessages	UT7: PatientMessagesTest	BS1_P6: maven-remote-resources-plugin
RQ1_F3: doc table	D8: PatientRecords	SC8: PatientRecords	UT8: PatientRecordsTest	
RQ1_M1: store	D9: DoctorRecords	SC9: SelectedPatient	UT9: SelectedPatientTest	
RQ2_F1: doctor	D10: Login	SC10: SendToDocs	UT10: SendToDocsTest	
RQ2_F2: sm	D1_F1: adminName	SC11: UserNames	UT11: UserNamesTest	

Figure 4-37 : SAT-Analyser main artefact summary for disease management system

Table 4.5 : Artefact categorization: disease management system

Artefact type	Low	Medium	High
Requirement	RQ3	RQ5	RQ1, RQ2, RQ4
Design	-	D6, D10	D1, D, D3, D4, D5, D7, D8, D9
Source code	SC11	SC5, SC6, SC7, SC10	SC1, SC2, SC3, SC4, SC8, SC9
Test script	UT11	UT5, UT6, UT7, UT10	UT1, UT2, UT3, UT4, UT8, UT9
Configuration	-	-	BS1

4.5.1 Evaluation of traceability establishment component

```xml
<Relation id="77">
	<SourceNode>SC6</SourceNode>
	<RelationPath>SourceClassToUnitTestClass</RelationPath>
	<TargetNode>UT6</TargetNode>
</Relation>
<Relation id="78">
	<SourceNode>SC9</SourceNode>
	<RelationPath>SourceClassToUnitTestClass</RelationPath>
	<TargetNode>UT9</TargetNode>
</Relation>
<Relation id="79">
	<SourceNode>SC8</SourceNode>
	<RelationPath>SourceClassToUnitTestClass</RelationPath>
	<TargetNode>UT8</TargetNode>
</Relation>
```

Figure 4-38 : Disease management system Relations.xml instance

Figure 4-38 shows a part of the established relations in XML predefined format in the source to the target structure. Three of the source classes to unit test class relations are shown therebetween Login-LoginTest, Selectedpatient-SelectedPatientTest and PatientRecords-PatientRecordsTest.

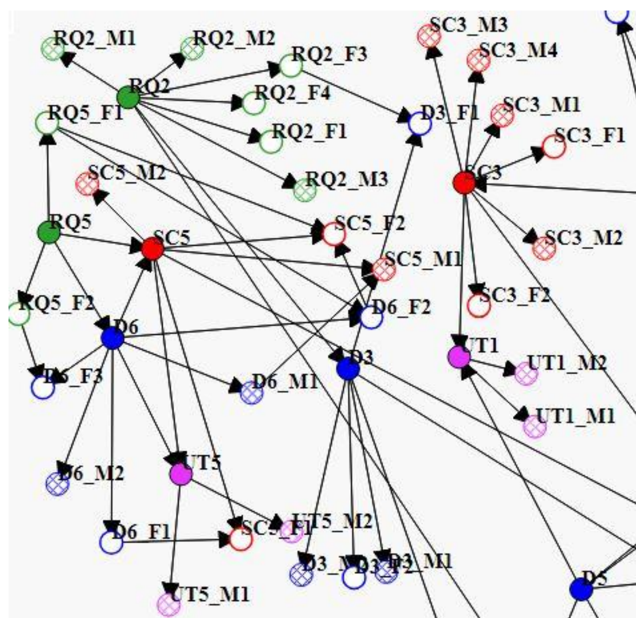


Figure 4-39 : Traceability visualization - disease management system

Figure 4-39 shows a part of the traceability graph for the Disease Management system. The relationship between RQ5, SC5 and D6 can be clearly seen that denotes the HealthSMS feature's requirement, source class and design class.

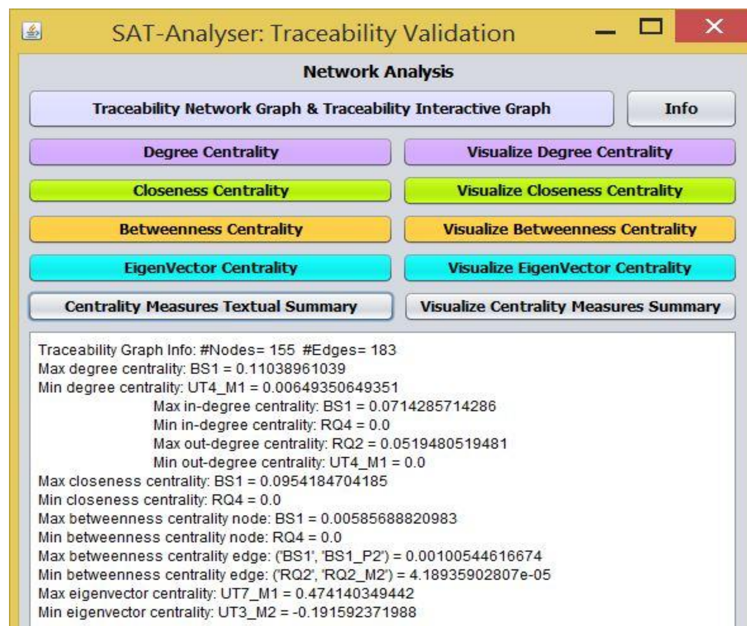


Figure 4-40 : Network analysis summary - disease management system

Figure 4-40 states the SAT-Analyser tool computed centrality measures summary for the network analysis based traceability validation. One of the minimum betweenness and closeness centrality are obtained by the RQ4: Admin as its having a lack of attributes and methods in each artefact category that results in having a lesser number of relationships comparatively.

4.5.2 Evaluation of continuous integration process

Following two change types are applied to this Disease Management case study.

- **C1:** Add a main requirement
- **C3:** Add a low importance requirement

A newer main requirement is added as requiring a Nurse and a lower importance requirement is specified as having a Timetable for a Nurse. The other remaining artefact types such that design, source code, test files and build script artefacts are supposed to be modified accordingly when an artefact addition is integrated. Figure 4-41 shows the detection of added main and lower requirement elements. Accordingly, the calculated change impact analysis results are shown in Figure 4-42. Due to not modifying the other artefact types along with the requirement artefact addition, it accurately shows that currently no impact of the newly added two requirements on others.

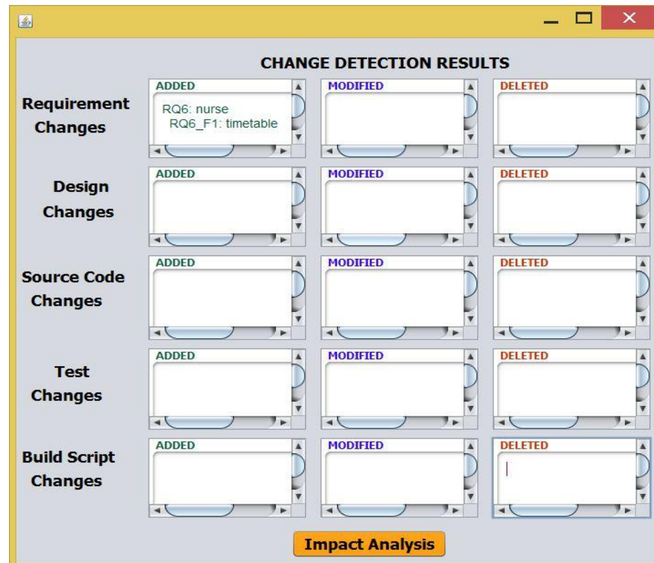


Figure 4-41 : Disease management system change detection window

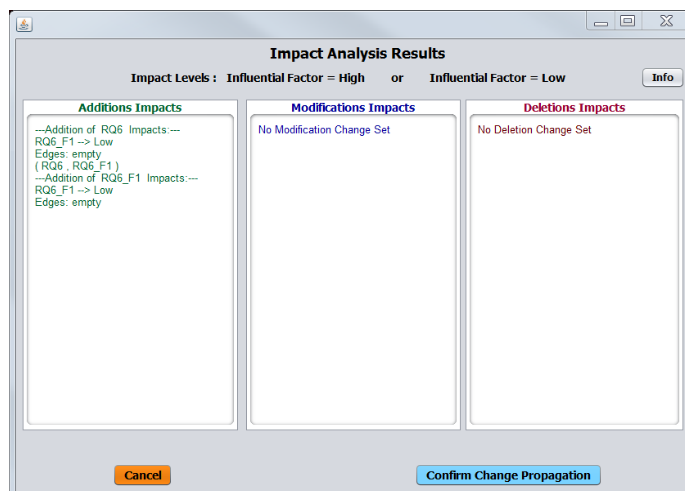


Figure 4-42 : Disease management system impact analysis window

The change propagation results of this scenario are shown in Figure 4-43. It can be clearly seen that RQ6: Nurse and RQ6_F1: timetable have intra-relationships. Also, there are no inter-relationships since no modifications are submitted on other artefact types along with these two requirement additions.

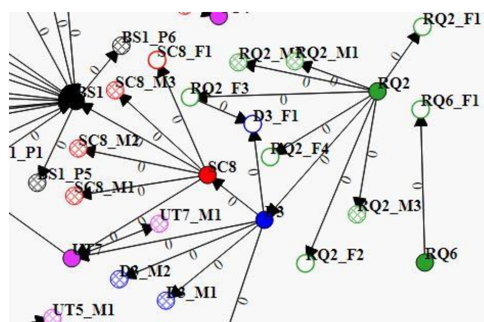


Figure 4-43 : Disease management system change propagation instance

4.5.3 Performance analysis

The accuracy of the CIA process in the Disease Management system case study is shown in Figure 4-44. As the modifications on other artefact types are not incorporated during the CI activity of the added changes, there exist missing impact identifications. Hence, the addition of RQ6 and RQ6_F1 must impact on a design, source code and a unit test item. The final precision has been 1.0 since the identified EIS is accurate though the recall and F-measure are lower due to missing DIS items which would be higher when adding all the artefacts during an *addition* change type following the SAT-Analyser's CI constraint.

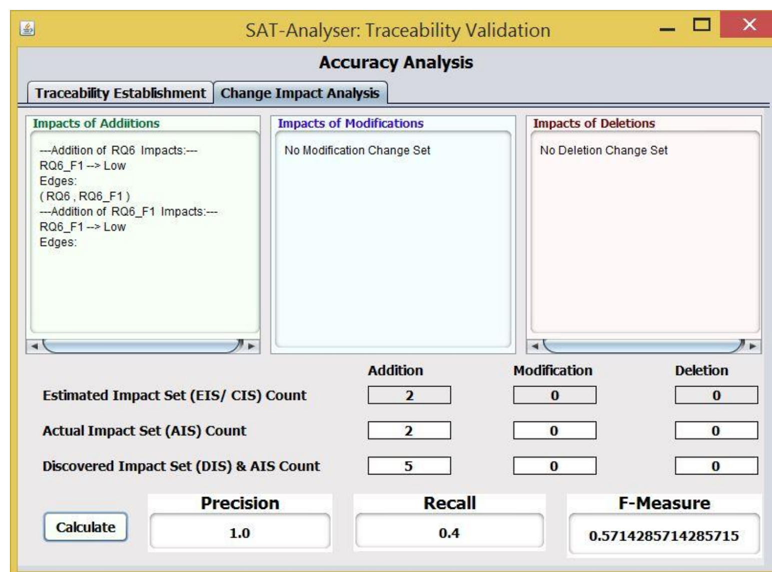


Figure 4-44 : CIA statistical analysis results - disease management system

The performance analysis of the applied change types C1 and C3 on Disease Management system are provided in Figure 4-45. Since C1 is about adding a main requirement artefact item such as RQ6: nurse it has shown a higher resource consumption than the other C3: lower importance requirement item addition such as RQ6_F1: timetable. The reason for the significant difference in these two changes though both are the same change type is in accordance to the CIA algorithm 3:11. Accordingly, if the changed artefact item is a sub-element it is supposed to check a lesser number of paths than a main artefact element. Therefore, the RQ6 addition is consuming a considerable resource amount while RQ6_F1 is lesser as an artefact sub-element.

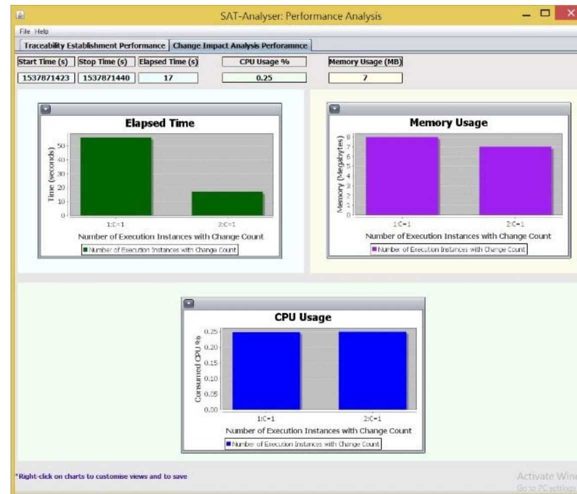


Figure 4-45 : CIA performance analysis results: disease management system

4.6 Experimental results: case study 5 (E-School management system)

As the fifth case study, we have selected, S8: E-School Management system that handles the student, teacher and subject courses workload within a domain of a school. It helps for data management in the school's management level activities and student activities by allowing to store student, teacher, course details and letting students enrol in courses. The requirements are shown in Figure 4-46.

> eSchool management system handles a person list and a course list. There are two main types of person in the system as teacher and student. System checks whether a person is logged. Admin can handle person list, delete users and search users. Each person must provide name, dob and address. Then, a person can login, see profile, set password, and get new course details. A teacher has a staff main page to view and a student has a student main page to view. Student can enroll to courses. System checks the eligibility of a student and grant permission to enroll to requested course. The course list contains courses. Admin can update courses, delete courses and check course fees. A course contains a course name, course Id, beginning date, finishing date and a fee. Teacher can enroll to subjects to teach.

Figure 4-46 : E-School management system description

The corresponding UML class diagram is shown in Figure 4-47. It contains six classes with inheritance for Person categorized as Student and Teacher. Two composition relationships exist between PersonList, Person and CourseList, Course. The remaining artefacts involved in the case study such that source code, unit test script and build script are provided via the SAT-Analyser web site ("SAT-Analyser," 2018). The identified main artefact elements by the tool SAT-Analyser are listed in Figure 4-48 followed by the tool generated unique identifier of each artefact. Further, there exist artefact sub-elements for methods, attributes (fields) and plugins as partially shown in Figure 4-48 with _F, _P and _M notations.

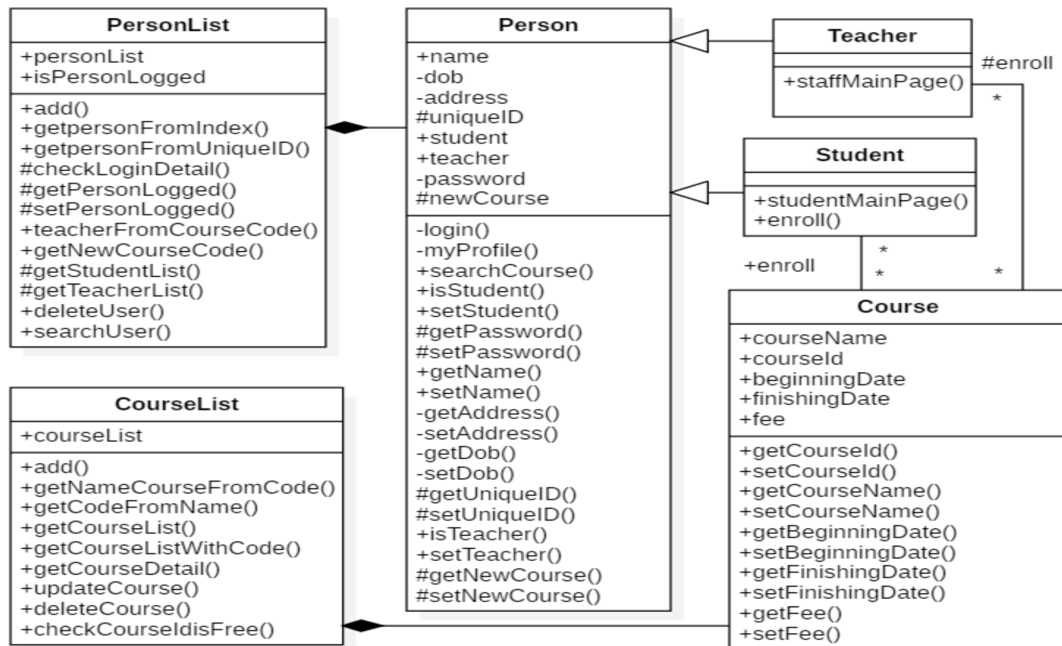


Figure 4-47 : E-School management system design diagram

Artefacts Information

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

Figure 4-48 : SAT-Analyser main artefact summary for E-School management system

Table 4.6 summarises the manual artefact identification and categorization of E-School Management system based on expert knowledge such as by a requirement engineer/ software engineer.

Table 4.6 : Artefact categorization: E-School management system

Artefact type	Low	Medium	High
Requirement	RQ1	RQ7, RQ8	RQ2, RQ3, RQ4, RQ5, RQ6
Design	-	D5, D6	D1, D2, D3, D4
Source code	SC10	SC1, SC3, SC4, SC5, SC9	SC2, SC6, SC7, SC8, SC11
Test script	UT10	UT1, UT3, UT4, UT5, UT9	UT2, UT6, UT7, UT8, UT11
Configuration files	-	-	BS1

4.6.1 Evaluation of traceability establishment component

The established traces are first written into a Relations.xml file as represented in Figure 4-49 having a source-target tag structure. For example, it shows two relations from SC8: Enroll and SC10: MyProfile source classes to the BS1: project's Maven build script artefact. Figure 4-50 provides a section of the traceability graph for the E-School Management system.


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


Figure 4-49 : Relations XML format of traceability establishment - E-School system

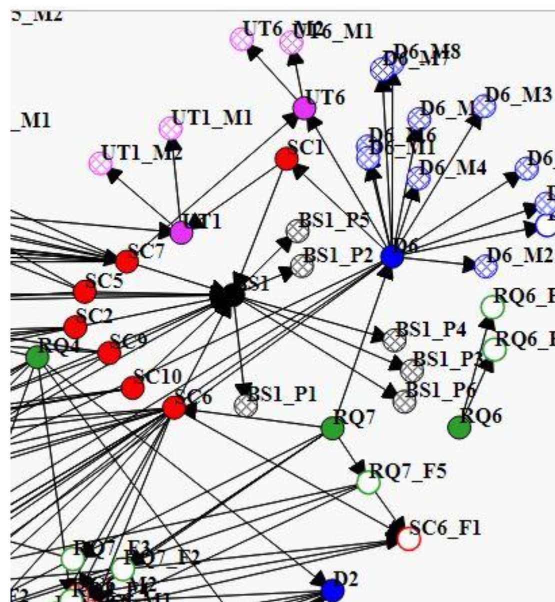


Figure 4-50 : Traceability visualization – E-School management system

Figure 4-51 shows the SAT-Analyser tool computed centrality measures summary for the network analysis based traceability validation. The node RQ6 (admin) has got the minimum betweenness centrality which is acceptable since it is isolated without relationships with other heterogeneous artefacts; design and source code as in Figure 4-50 evidently.

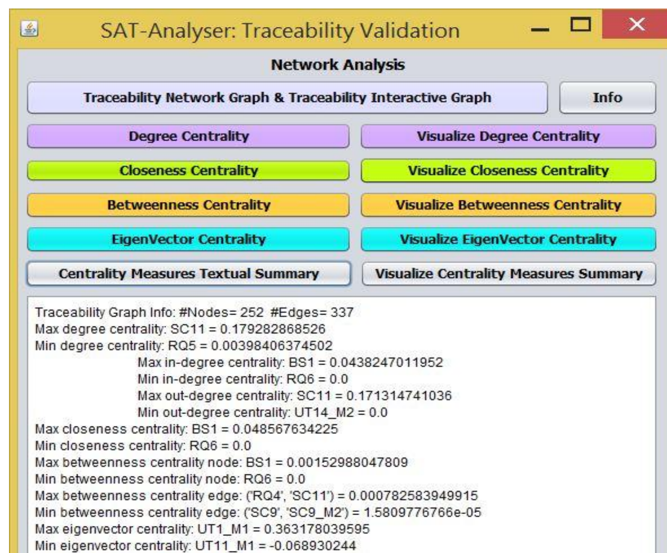


Figure 4-51 : Network analysis summary – E-School management system

4.6.2 Evaluation of continuous integration process

We have considered two change to analyse the impact in the E-School Management case study; **C8:** Delete a design component and **C16:** Modify a configuration artefact. The change detection results are shown in Figure 4-52 while the corresponding CIA results are provided in Figure 4-53. The deletion of D4: Teacher design artefact has impacted on its existed own method (D4_M1). However, the modification performed on BS1_P6 plugin name has not impacted on any other as it's a leaf node having no outgoing edges to propagate even though its own influential factor is *high*.

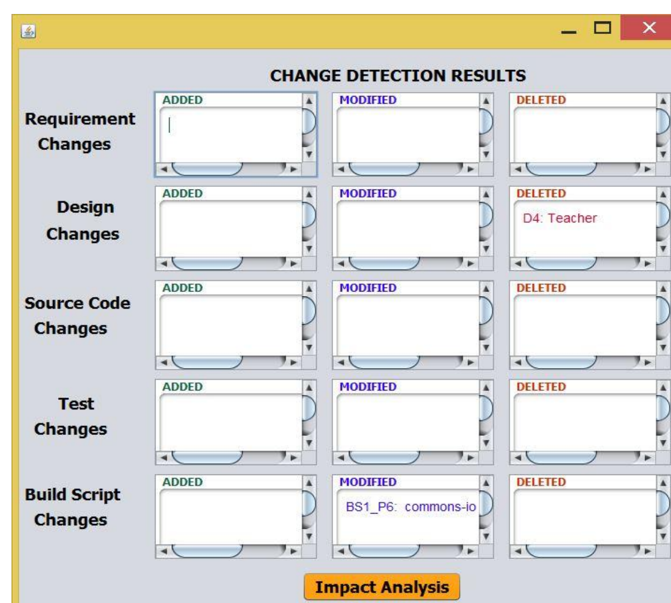


Figure 4-52 : E-School management system change detection window

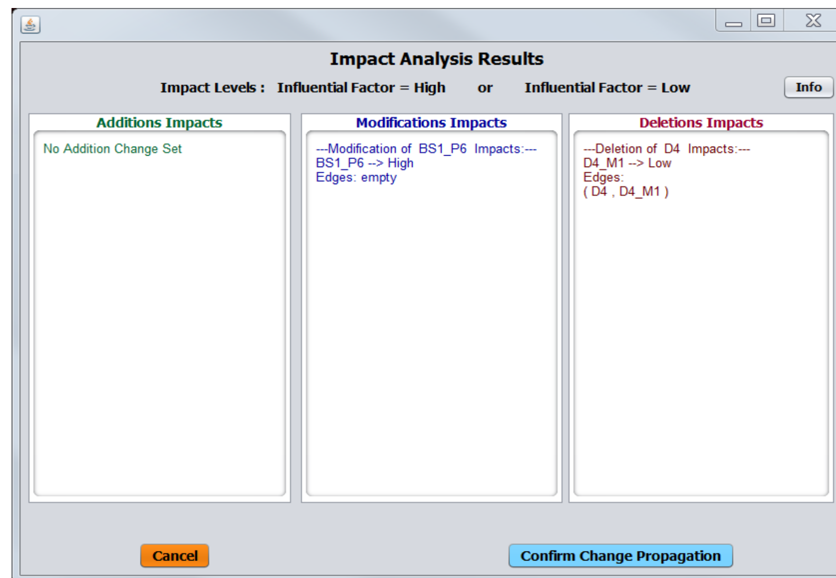


Figure 4-53 : E-School management system impact analysis window

The part of the change propagated traceability graph is shown in Figure 4-54. The nodes D4 and its impacted D4_M1 has been removed while BS1_P6 is modified.

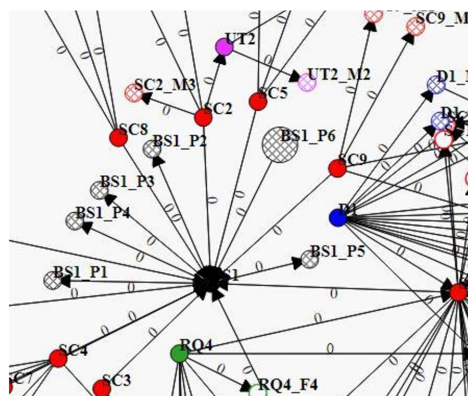


Figure 4-54 : E-School management system change propagation instance

4.6.3 Performance analysis

The impact sets accuracy of the E-school Management case study are measured using the statistical metrics as shown in Figure 4-55. The precision has been obtained as 1.0 since the identified all impact items are accurate. However, there is a lower recall and F-measure due to lacking two impact items with respect to the *deletion* impact items. The deletion of D4 must impact on an SC and a UT item though they have not been identified by the SAT-Analyser tool due to missing traceability establishments which can be improved with more rigorous NLP and name entity recognition techniques.

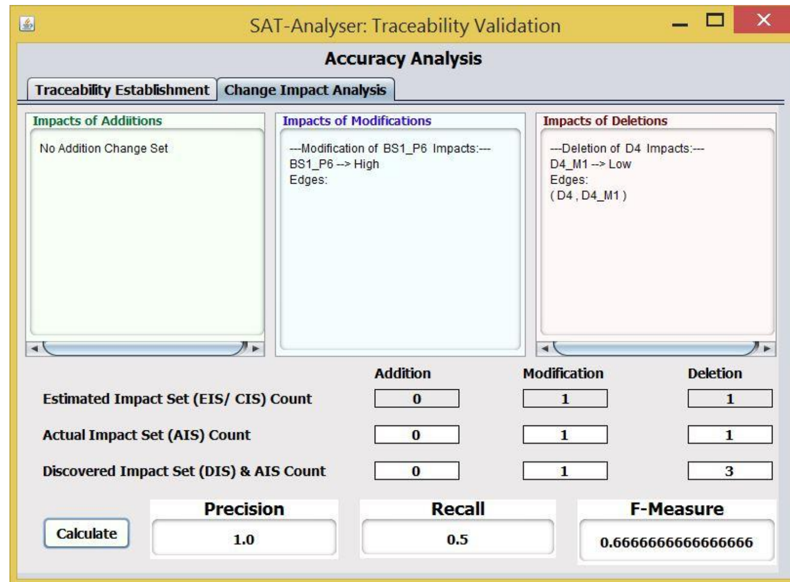


Figure 4-55 : CIA statistical analysis results: E-School management system

Figure 4-56 shows the performance analysis results with respect to time, memory and CPU consumption occupied by the change types C8 and C16 applied on E-School Management system. The C16 *modification* change type performed on a build script artefact item has occupied a higher elapsed time while the memory and CPU consumption remains equivalent for both C8 *deletion* and C16. It is again noticeable that artefact *modification* is requiring a higher resource allocation comparing to *deletion* and *addition* as further justified in previous case studies performance results.

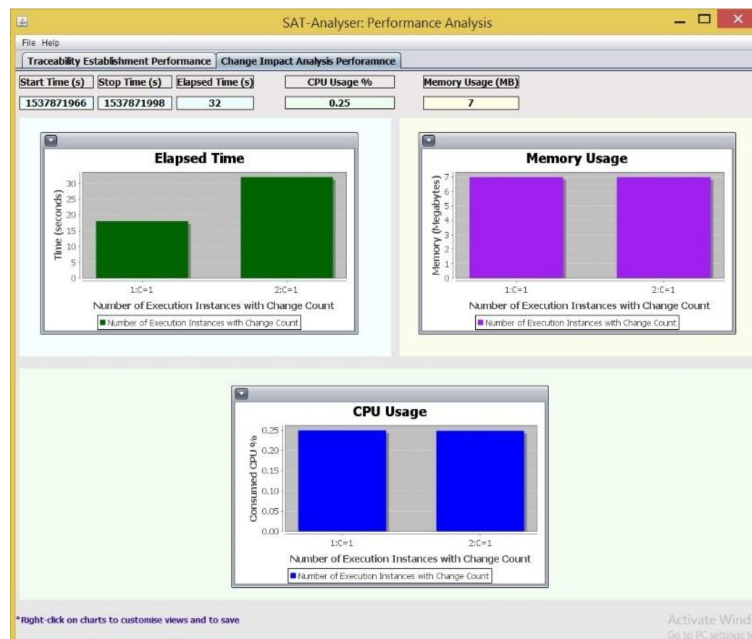


Figure 4-56 : CIA performance analysis results: E-School management system

4.7 SAT-Analyser performance analysis

This section provides an overall evaluation of the SAT-Analyser tool considering all five case studies' results in summary.

4.7.1 Traceability establishment performance

The overall traceability generation performance for the five case studies is shown in Figure 4-57 in terms of elapsed time, memory and CPU consumption. The total artefact count of each project is shown beneath each bar in the graphs.

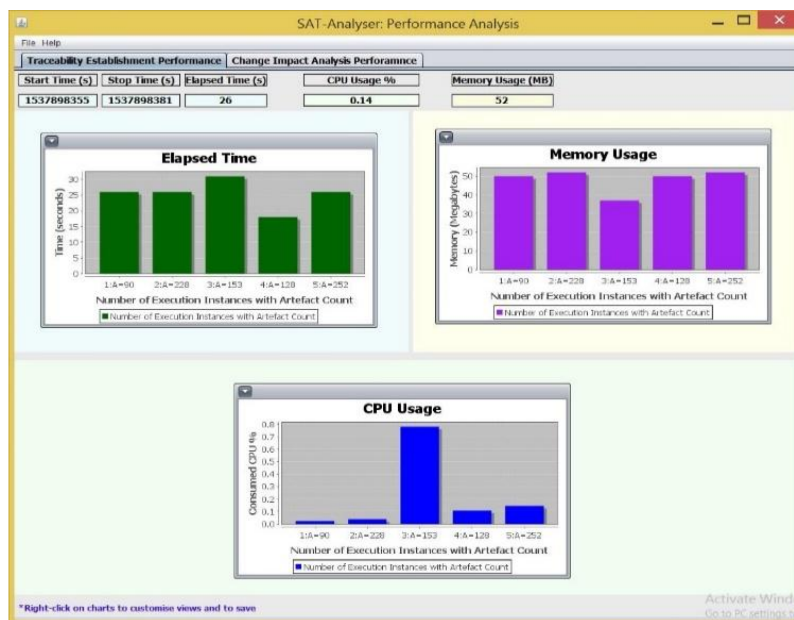


Figure 4-57 : SAT-Analyser traceability establishment performance

The third case study Multimedia Library system has occupied the maximum time and CPU consumption since it lacks many possible traces due to naming differences in artefact types. For instance, there exists a design class called User but lacks a source class with any similar name. Thus, the traceability establishment of the tool which relies on the string comparison consumes more time and CPU by performing string matching rigorously. However, it consumes less memory since a smaller amount of matching traces is resulting due to the same reason of naming differences. The least number of artefacts included two projects POS: 90 artefacts and Disease Management: 128 total artefacts are consuming the least amount of resources. Further, the second highest resource consumption is occupied by the maximum number of total artefacts holder, E-School Management: 252 project.

4.7.2 Accuracy evaluation of change detection component

Table 4.7 provides a summary for the accuracy of XML comparison based change detection process in SAT-Analyser, where the applied change types versus tool identified change types are listed. All the applied change types have been accurately detected by the tool ranging from 5 to 2 change counts at a time.

Table 4.7 : Change detection component accuracy evaluation

	Project title	**The actual number of occurrences of artefact- changes/ commits within the CI process**	**Number of changes detected by the tool (automated by SAT-Analyser)**
1	POS system	C4, C9, C10, C12, C15	C4, C9, C10, C12, C15
2	Tour management system	C2, C5, C6, C13, C17	C2, C5, C6, C13, C17
3	Multimedia library system	C7, C11, C14	C7, C11, C14
4	Disease management system	C1, C3	C1, C3
5	E-school management system	C8, C16	C8, C16

4.7.3 Accuracy evaluation of impact analysis component

The statistical analysis results of CIA process in each case study project are summarized in Table 4.8.

Table 4.8 : Change impact analysis component accuracy evaluation

Project title	**Change type**	**Identified impact set by the tool (EIS)**	**Actual impact set (AIS)**	**Non identified correct impact set (DIS)**	**Statistical analysis for the accuracy of impact analysis**		
					Precision	***Recall***	***F-Measure***
POS system	Addition	3	3	1	1.0	0.95	0.97
	Modification	34	34	1			
	Deletion	0	0	0			
Tour management system	Addition	3	3	5	1.0	0.86	0.93
	Modification	2	2	0			
	Deletion	27	27	0			
Multimedia library system	Addition	0	0	0	1.0	0.98	0.99
	Modification	4	4	1			
	Deletion	36	36	0			
Disease management system	Addition	2	2	3	1.0	0.4	0.57
	Modification	0	0	0			
	Deletion	0	0	0			
E-school management system	Addition	0	0	0	1.0	0.5	0.67
	Modification	1	1	0			
	Deletion	1	1	2			

The POS, Tour Management and Multimedia Library system ranging from Small to Medium scale have shown a higher accuracy in terms of precision, recall and F-measure. The Disease Management and E-School Management system have shown a reduction in recall and F-measure due to not following the SAT-Analyser's defined CI constraint and lack of applied NLP capabilities respectively which can be improved in future. The advanced NLP features, information retrieval techniques and deep learning capabilities can be applied to handle the situations with meaningless artefact names and inconsistent naming conventions which currently affect the results according to String comparison approach in traceability. Nevertheless, the precision has been 1.0 in all five case studies successfully.

4.8 Usability of the extended SAT-Analyser tool

The usability of a tool has to be assessed with practitioners in the considered domain (Bangor, Kortum, & Miller, 2009). We have used the System Usability Scale (Brooke, 2013) that is a researched usability assessment Likert scale, to evaluate the usability of prototype tool SAT-Analyser involving DevOps practitioners in the industry from various software companies as listed in Appendix E.

SUS is a standard reliable tool to measure a system usability with a pre-defined set of questions along with a provided set of answers for each. Participants' every single response is quantified based upon the selected answer option following a pre-defined fixed scale and output a final average score in the range of 0-100. Scores above 68 are considered as having an above average usability level while below 68 as an average usability level.

SAT-Analyser prototype tool in this research work is highly focused on the CIA in software artefact traceability for DevOps based on a novel theoretical model that supports CICD pipeline. Thus, SAT-Analyser can be categorized as a prototype-level support tool for DevOps tool stack having unique features that the existing tools in DevOps tool stack do not facilitate. The tool is live demonstrated interactively with Q/A sessions and given the standard SUS questionnaire among a

total of 20 DevOps practitioners. The questionnaire consists of 10 items consisting of 5 positive statements and 5 negatives. Each has 5 level options to respond ranging from strongly disagree to strongly agree as provided in appendix B with all received individual responses.

Figure 4-58 shows a summary of the responses for 5 positive SUS statements and Figure 4-59 depicts the analysis of the responses obtained for 5 negative statements. Consequently, a final overall SUS score of 62.5 is obtained signifying the tool SAT-Analyser usability level as Average in terms of user experience at front-end which is mainly based on Human-Computer Interaction (HCI) aspects.

Majority participants in the usability study experienced the notion of traceability and CIA for the first time. They had a new experience of the features including traceability creation, visualization, validation, change detection, CIA, change propagation and PM all in a single tool that supports continuous integration in DevOps practice. Figure 4-58 indicates the highest value for *Well integration* and least for *Confidence in using* the tool accordingly. Moreover, their lack of awareness in the traceability aspects depicts in Figure 4-59 as the highest agreed percentage is for the statement of *Need prior learning*.

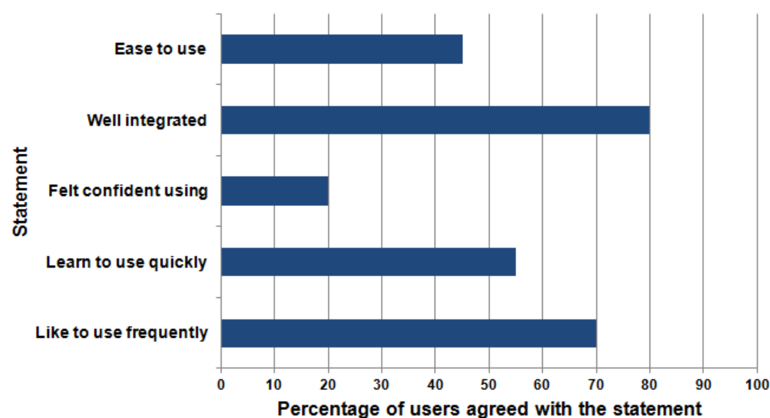


Figure 4-58 : SUS positive responses analysis

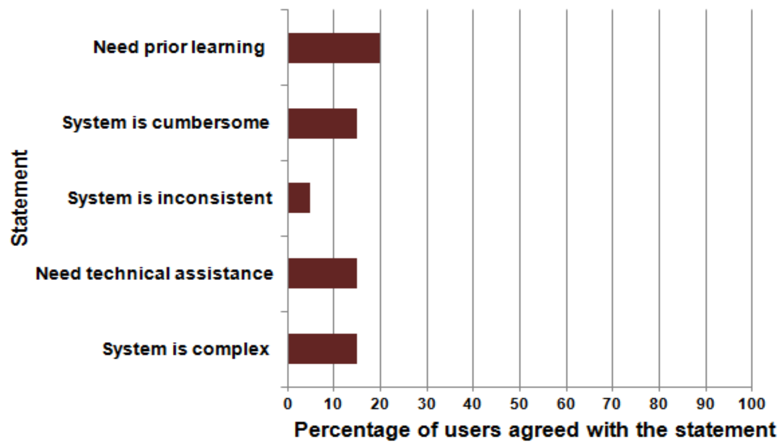


Figure 4-59 : SUS negative responses analysis

In addition, an extra option is provided in the same survey for the participants to respond as a selection of three most relevant words that best describe their own perspective about SAT-Analyser among 20 tool related term choices as in appendix B. A tag cloud which stands as a methodology to visualize user feedback attractively is generated based on those user selections. As shown in Figure 4-60, the most emphasized terms about SAT-Analyser tool by participants are Traceability, Supportive and Improvable which derive a positive level of user satisfaction about the tool SAT-Analyser denoting a future direction to enhance the usability beyond a prototype-level with more HCI aspects.

Figure 4-60 : SAT-Analyser usability tag cloud

Section 5
Discussion

5.1 Feature analysis of SAT-Analyser in practice

The industry level software development environments highly embrace Agile principles and transform into DevOps practices. The nature of CICD in DevOps practices drives software projects in any domain and any scale towards successful ROI benefits. However, adapting to operational level in DevOps practice is challenging due to the lack of formalism compared to traditional software models. Therefore, the requirement of having traceability support in a DevOps environment is significant than in a traditional software development environment which is addressed in this research work further with CIA.

The feature selection of the proposed SAT-Analyser tool is based on an initial survey conducted among DevOps practitioners. We can consider the requirements written in natural language as it is the industry practice. UML class diagrams were selected as the code base is dependent on that. Unit testing is used, as it tests the individual functionalities for errors. Considering the continuous integration tasks, we have set up the scheduler with different options: automated fixed intervals, dynamically and manually to detect the changes as necessary, to avoid overhead and reduce the cost. In current practice, change detection is defined mainly for source code changes and no proper tools to automated detectors for other artefact changes. Source code change detection tools that industry is aware of are Jenkins Cron Job, ServiceNow, JIRA Service Desk, ServiceNow. SAT-Analyser detects changes of all artefacts not being limited to the source code. We also gave prominence to source code changes, since it is the most affected artefact. Github repository paths, Jenkins are configured with the tool for that. Also, we have performed change detection for rest of the artefacts as any artefact can be changed in DevOps such as a requirement and design change. Thus, our tool supports traceability management for all the major artefact types and can be extended for remaining sub artefacts as well.

According to the survey, 66.7% accepts that traceability handling might be useful while rest is unaware of the concept of traceability. Thus, we have represented traceability with graph-based interactive and analytical visualization. Traceability graph is the most used trace representation mechanism according to the literature.

Many analytical approaches can be conducted on graphs using graph theories and mathematical models which we have used for CIA. 33.3 % reported that they do not use an impact analysis for changes and rest is having a vague idea about CIA methods due to lack of knowledge. CIA limitations that lead to not practising in the industry are suggested as being time consuming and hardship in calculating the exact impact. We have used a dependency-based CIA with a mathematical weighting scheme model using EVC based on the influential factor of an artefact. Dependency-based methods are discussed in the literature and ideal for graph calculations. Network analysis centrality measures are selected due to the significant performance and variety of metrics. We have calculated the CIA with a minimal cost and complexity with a rule-based algorithm. CIA rules are defined considering the practical dependency scenarios by avoiding calculation overhead and only proceeding with higher impact artefact items to increase performance.

In practice, change propagation methods are automatically deployed to the Jenkins server and used pre-defined protocols and policies specific to the company. Change propagation is crucial for decision making and hence traceability graphs are re-visualized for every change propagation. Continuous integration is performed frequently using tools such as Jenkins, CodeDeploy, CodePipeline, Puppet, Jira, TravisCI, and TeamCity. In order to support it, SAT-Analyser is integrated with Jenkins and Docker for deployment activities due to wide usage of them and with the GitHub repository with opensource facilities. Additionally, our tool is integrated with the popular project management tool Trello, as it provides free Agile Kanban boards, thus support CICD pipeline. Finally, we have shown the applicability of the tool for different project scales and domains, using case study evaluation.

Since the software industry mainly uses Jenkins to support continuous integration, Table 5.1 gives a comparison of Jenkins with the proposed tool SAT-Analyser.

Table 5.1 : Comparison between Jenkins and SAT-Analyser for CI

Jenkins		**SAT-Analyser Tool**	
Pros	***Cons***	***Pros***	***cons***
Source code change detection.	Consider only source code.	Source code, requirement, design, unit test, build script change detection.	Pre-processing time complexity.
Build automation with scheduling.	No traceability support.	Traceability support for all artefacts.	-
Open source.	No impact analysis.	CIA for every change detection.	Prototype level tool.
Support any scale projects.	-	Better performance with small to medium scale projects.	Scalability issues in artefact pre-processing.
Project deployment with many integrated plugins support.	Lack traces, change propagation visualization.	Traces, changes with impact are visualized in graph format.	-

The existing industry perspectives in DevOps practice for traceability, change detection, change impact analysis, change propagation and CI over the SAT-Analyser solutions are summarized in Table 5.2.

Table 5.2 : Industry level traceability management vs. SAT-Analyser tool

Feature	**Industry practice**	**SAT-Analyser tool**
Traceability establishment and visualization.	No proper tools.	String similarity based traceability establishment with graph-based visualization. Network analysis and statistical analysis based traceability validation.
Change detection.	No proper tools to auto-detect changes of every artefact. Use monitoring tools to detect failures in Jenkins for source code building.	Detect changes for every artefact for every integration using XML based comparison.
Impact analysis.	Manually decide the range of affected artefacts in code level, Lacks CIA. Time-consuming and hard to calculate the exact impact.	Calculate the level of impact as *high* or *low* for every change detection using Eigenvector centrality.
Change propagation.	Automatically deploy to the server with Jenkins. Use pre-defined protocols to manage code.	Propagate changes according to impacts and re-visualize in a traceability graph.
Continuous integration.	Jenkins as an ideal solution for source code integration with build automation.	Integrated with Jenkins and source code management repository (GitHub) integrated CI along with PM facility using Trello.

According to Table 5.2, change detection and CI are addressed in tool level in the industry, mainly with Jenkins and source code management without considering other artefact types. The industry perspective of traceability management is less and they believe impact analysis as a challenge. In contrast, SAT-Analyser tool provides the ability in each activity integrated with DevOps tools stack. Table 5.3 validates the SAT-Analyser capabilities over mostly cited traceability management tools encountered in literature as discussed in Chapter 2. Thus, the limitations in those tools such as lack of heterogeneous artefact support, change detection, CIA for heterogeneous artefacts, change propagation, IDE independency are successfully solved in the SAT-Analyser prototype tool.

Table 5.3 : Existing traceability management tools vs. SAT-Analyser

Tools	**TraceME**	**IBM DOORS**	**TraceAnalyzer**	**LDRA-TBmanager**	**ArchEvol**	**SAT-Analyser**
Features						
Requirement traceability	√	√		√		√
Design level traceability	√		√		√	√
Heterogeneous artefact traceability	√		√	√		√
Traceability visualization		√	√		√	√
Traceability validation						√
CI/ scheduling/ versioning		√		√	√	√
Change detection						√
Change impact analysis	√	√				√
Change impact analysis validation						√
Change propagation visualization						√
Consistency management, Project management		√				√
DevOps tools stack supportability						√
IDE independence						√
Tool performance analysis						√

The approach we designed and developed as SAT-Analyser tool, supports traceability management of software projects in both traditional and Agile based process. It is specifically, designed to facilitate requirements in DevOps environment with CICD concepts. The identified major differences in traditional software re-development versus DevOps environments are acceptance of artefact changes and collaborative behaviour. In traditional software development, the frequency of artefact changes is minimal due to the sequential nature, where the artefact changes are not accepted at a later stage of SDLC. Thus, in general software development, the traceability and impact analysis process are required

only at the beginning and end of the process. Hence, the requirement of incorporating the CI features with scheduling and versioning included in this research work would be lesser significant in a traditional software development process, while the traceability model would be equally important as for DevOps. Hence, the frequency of change detection, their impact analysis and change propagation, visualization, team collaboration and validation features included in this research work are uniquely useful and supports continuous integration in DevOps practice. Thus, these features of the research work SAT-Analyser are actively useful for the daily usage of CICD pipeline in a DevOps environment since any artefacts change is always welcomed at any stage of SDLC.

In contrast, the features of CI, change detection, CIA, change propagation, PM notifications are uniquely useful for DevOps environments as the artefacts management is having the utmost importance due to collaborative team-based nature. Thus, these features of the research work SAT-Analyser are actively useful for the daily usage of CICD pipeline in a DevOps environment since any artefacts change is always welcomed at any stage of SDLC.

5.2 Analysis of the usability study based evaluation

The theoretical traceability and CIA model are mainly focused in this research work while usability aspects are scoped into the prototype SAT-Analyser tool. The usability of the research outcome SAT-Analyser prototype is evaluated based on standard SUS score methodology with the involvement of DevOps practitioners in the software industry.

An overall **62.5** SUS value is obtained from 100.0 representing the SAT-Analyser prototype tool usability level as **Average**. Usability is important in deploying the SAT-Analyser tool as an industry tool since it contributes to the DevOps tools stack consists of a large number of dynamic tools evolving rapidly. Thus, the usability features of SAT-Analyser can be refined more with better performance parameters such as speed, memory consumption, as well as by integrating HCI principles in front-end for both stand-alone desktop version and web-based version in transforming it from prototype-level to a standard industry-level tool.

5.3 Mapping of the objectives and the methodology

The research problem addressed in this research project is to obtain software artefact change impact analysis in traceability especially for DevOps environments having frequent continuous integrations, where Agile principles are practically applied deviating from traditional sequential software development patterns. The research statement was defined consisting of several research questions;

- how to enable software artefact synchronization since multiple artefacts are highly affected all the time in a DevOps environment and
- how to maintain consistency among all artefacts with CIs.

Therefore, the research objectives are defined into each unique milestone in the research, each having a detailed technical methodology for, traceability management, artefact change detection in CIs, CIA and change propagation in DevOps as illustrated in Figure 5-1. Thus, the research milestones are achieved in the form of a prototype traceability tool SAT-Analyser as the proof-of-work.

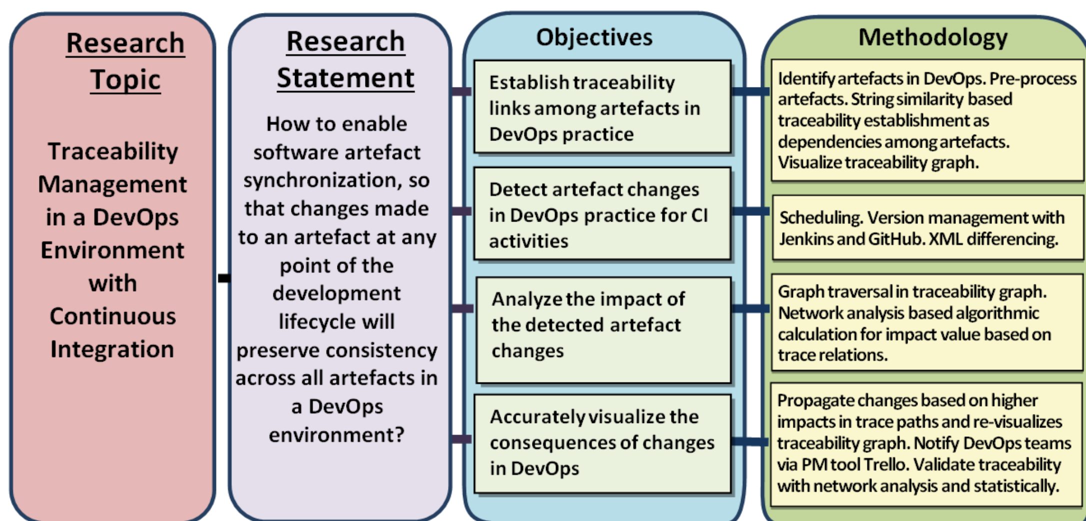


Figure 5-1 : Research objectives-methodology-results mapping

5.4 Limitations

The traceability establishment and traceability visualization remain challenging in large scale software projects, where a large number of artefacts and relationships are expectable. The JavaScript D3.js based interactive visualization shows a lighter weight compared to the other two visualizations used in this research namely Neo4j, Gephi based visualization and Python network analysis based

visualization. The traceability establishment in this research work faces the main limitation due to restrictions in NLP over practical issues such as meaningless artefact naming, different naming conventions. That limitation exists even in the use of NLP with dictionary ontologies and *WordNet* databases.

Moreover, the computation of CIA is limited to Eigenvector centrality in this research work that is subjected to be contradictory with expert-based impact values. As one of the solutions, we have provided the user alteration capability in CIA results. Furthermore, the performance of the SAT-Analyser traceability tool of this research work is limited due to the intermediate use of XML for the artefact data extraction process that also affects the CI scheduling capabilities of the tool. Hence, more dynamic CI scheduling features can be supported to improve the artefact pre-processing and traceability establishment performance.

5.5 Future work

This study can be extended in many directions. Performance and accuracy of the traceability establishment can be enriched with advanced NLP features, information retrieval techniques and deep learning capabilities. This would be a significant future improvement to facilitate traceability support regardless of project scale. Traceability visualization with better scalability is another promising future work. Also, integrating the three visualization variations provided in SAT-Analyser together would be useful. Moreover, the supported artefact types can be extended with more sub artefact categories such as support for other programming languages other than Java as the tools stack in DevOps environments are more dynamic with latest technologies. Another major future research direction would be improving the CIA model, which is based on eigenvector centrality in network analysis that shows the influential value of a node or a link. Further, SAT-Analyser can be extended for a function such as a software quality assessment tool that assesses the quality of the design and code. In addition, the usability aspects of the tool can be improved into an industry-level DevOps supportive tool by integrating HCI concepts along with refined performance parameters.

5.6 Conclusion

This research has addressed traceability management of heterogeneous software artefacts covering all the stages of SDLC, with change impact analysis to cope with continuous integrations in DevOps practices. Initially, the raw artefacts were processed using string comparison and NLP, and traceability links were established between the extracted artefacts. The traceability visualization is developed in three views, Gephi-based informative, Python-based analytical and JavaScript-based interactive. The traceability validation process is based on the network analysis centrality measures and statistical accuracy measures. The continuous integration tasks are combined with supporting processes including collaboration with DevOps tool stack, scheduling algorithms, versioning, XML-based artefact change detection, weighting scheme based CIA model for artefact impact computation, graph-based change propagation and project management to maintain the artefact consistency.

The DevOps support is ensured in traceability establishment by providing heterogeneous artefact support for each major stage in SDLC such that requirements artefact, design, code, unit test and build script artefacts. Traceability visualization is enhanced in three variations to overcome scalability issues and to fasten decision making since time is critical in a more collaborative DevOps environment. Due to the extra cost of traceability management, the validation of traceability results is identified to be important in DevOps, where a higher number of tools stack is always actively in use.

Change impact analysis, which is a result of artefact changes accepted during software development in a DevOps environment in following continuous integration is a core part of this research work. Hence, it supports the CICD pipeline concept following change detection, change impact analysis, change propagation and consistency management with project management features. The weighting scheme based on a mathematical model is used for CIA. It has used eigenvector centrality measure that captures the level of importance in each artefact with respect to all artefacts. A rule-based scenario is adapted for graph

traversal paths and further user alteration is used to improve the accuracy. Traceability is re-visualized based on impact values according to the CIA.

The research work is evaluated using real software projects based case studies in different scales and user acceptance interview and survey among industry DevOps practitioners. The results have shown the usefulness of the research outcome for the software engineering domain as a migration from theoretical principles to practical use since there is a hindrance of the awareness about CIA and traceability in the current industry. Further, the SAT-Analyser tool is featured with web-based with multi-user accessibility to allow DevOps teams to use the tool actively along with DevOps tools stack.

References

3SL. (2018). Retrieved July 6, 2017, from https://www.threesl.com/cradle/

Acharya, M., & Robinson, B. (2011). Practical change impact analysis based on static program slicing for industrial software systems. In *33rd International Conference on Software Engineering - ICSE '11* (pp. 746–755). New York, USA: ACM. https://doi.org/10.1145/1985793.1985898

AjaxSwing. (2018). Retrieved October 19, 2018, from http://creamtec.com/products/ajaxswing/index.html

Alves-Foss, J., Conte de Leon, D., & Oman, P. (2002). Experiments in the use of XML to enhance traceability between object-oriented design specifications and source code. In *35th Annual Hawaii International Conference on System Sciences* (pp. 3959–3966). IEEE. https://doi.org/10.1109/HICSS.2002.994466

Apiwattanapong, T., Orso, A., & Harrold, M. J. (2004). A differencing algorithm for object-oriented programs. In *19th International Conference on Automated Software Engineering, 2004.* (pp. 2–13). IEEE. https://doi.org/10.1109/ASE.2004.1342719

Apiwattanapong, T., Orso, A., & Harrold, M. J. (2005). Efficient and precise dynamic impact analysis using execute-after sequences. In *27th International Conference on Software Engineering - ICSE '05* (pp. 432–441). New York, USA: ACM. https://doi.org/10.1145/1062455.1062534

ArchStudio. (2018). Retrieved July 5, 2017, from http://isr.uci.edu/projects/archstudio/setup-easy.html

Arunthavanathan, A., Shanmugathasan, S., Ratnavel, S., Thiyagarajah, V., Perera, I., Meedeniya, D., & Balasubramaniam, D. (2016). Support for traceability management of software artefacts using Natural Language Processing. In *Moratuwa Engineering Research Conference (MERCon)* (pp. 18–23). IEEE. https://doi.org/10.1109/MERCon.2016.7480109

Athira, B., & Samuel, P. (2011). Traceability Matrix for Regression Testing in Distributed Software Development. In *Abraham A., Lloret M. J., Buford J. F., Suzuki J., Thampi S. M. (eds) Advances in Computing and Communications. ACC 2011.* Communications in Computer and Information Science, 191, (pp. 80–87). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22714-1_9

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: adding an adjective rating scale. *Journal of Usability Studies*, *4*(3), 114–123.

Bass, L. J., Weber, I. M., & Zhu, L. (2015). *DevOps : a software architect's perspective* (1st ed.). Addison-Wesley Professional.

Bavota, G., Colangelo, L., De Lucia, A., Fusco, S., Oliveto, R., & Panichella, A. (2012). TraceME: Traceability Management in Eclipse. In *28th IEEE International Conference on Software Maintenance (ICSM)* (pp. 642–645). IEEE. https://doi.org/10.1109/ICSM.2012.6405343

Berg, A. M. (2015). *Jenkins Continuous Integration Cookbook* (2nd ed.). Packt Publishing.

Bitrix24. (2018). Retrieved October 2, 2018, from https://www.bitrix24.com/

Borg, M., Wnuk, K., Regnell, B., & Runeson, P. (2017). Supporting Change Impact Analysis Using a Recommendation System: An Industrial Case Study in a Safety-Critical Context. *IEEE Transactions on Software Engineering*, *43*(7), 675–700. https://doi.org/10.1109/TSE.2016.2620458

Borgatti, S. P. (2005). Centrality and network flow. *Social Networks*, *27*(1), 55–71. https://doi.org/10.1016/j.socnet.2004.11.008

Borland. (2006). *Borland ® CaliberRM ™*.

Brooke, J. (2013). SUS: A Retrospective. *Journal of Usability Studies*, *8*(2), 29–40.

Burgaud, L. (2006). A Novel development framework combining requirement driven and model based engineering processes. In *4ème Conférence Annuelle d'Ingénierie Système* (pp. 1–7).

Chang, S. K. (2005). *Handbook of Software Engineering And Knowledge Engineering: Recent Advances*. *World Scientific Publishing*. World Scientific Publishing. https://doi.org/10.1142/5785

Chawathe, S. S., Rajaraman, A., Garcia-Molina, H., & Widom, J. (1996). Change detection in hierarchically structured information. *ACM SIGMOD Record*, *25*(2), 493–504. https://doi.org/10.1145/235968.233366

Chen, X., Hosking, J., & Grundy, J. (2012). Visualizing traceability links between source code and documentation. In *IEEE Symposium on Visual Languages and Human-Centric Computing*

(VL/HCC) (pp. 119–126). IEEE. https://doi.org/10.1109/VLHCC.2012.6344496
Cleland-Huang, J., Chang, C. K., & Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, *29*(9), 796–810. https://doi.org/10.1109/TSE.2003.1232285
Cleland-Huang, J., Gotel, O. C. Z., Hayes, J. H., Mäder, P., & Zisman, A. (2014). Software traceability: trends and future directions. In *Future of Software Engineering - FOSE 2014* (pp. 55–69). New York, USA: ACM. https://doi.org/10.1145/2593882.2593891
Cleland-Huang, J., Zisman, A., & Gotel, O. (2012). *Software and Systems Traceability*. *Software and Systems Traceability* (1st ed.). London: Springer-Verlag London. https://doi.org/10.1007/978-1-4471-2239-5
Cobena, G., Abiteboul, S., & Marian, A. (2002). Detecting changes in XML documents. In *18th International Conference on Data Engineering* (pp. 41–52). IEEE. https://doi.org/10.1109/ICDE.2002.994696
Czibula, I. G., Czibula, G., Miholca, D. L., & Marian, Z. (2017). Identifying Hidden Dependencies in Software Systems. *Studia Universitatis Babeș-Bolyai Informatica*, *62*(1), 90–106. https://doi.org/10.24193/subbi.2017.1.07
D3.js. (2018). Retrieved May 21, 2018, from http://d3js.org/
Dantas, C., Murta, L., & Werner, C. (2007). Mining Change Traces from Versioned UML Repositories. In *Brazilian Symposium on Software Engineering (SBES'07)* (pp. 236–252).
De Lucia, A., Fasano, F., & Oliveto, R. (2008). Traceability management for impact analysis. In *Frontiers of Software Maintenance* (pp. 21–30). IEEE. https://doi.org/10.1109/FOSM.2008.4659245
De Lucia, A., Oliveto, R., & Tortora, G. (2008). Adams re-trace: traceability link recovery via latent semantic indexing. In *13th International Conference on Software Engineering - ICSE '08* (pp. 839–842). New York, USA: ACM. https://doi.org/10.1145/1368088.1368216
Déhoulé, F., Badri, L., & Badri, M. (2017). A Change Impact Analysis Model for Aspect Oriented Programs. In *12th International Conference on Evaluation of Novel Approaches to Software Engineering* (pp. 144–157). SCITEPRESS Publications. https://doi.org/10.5220/0006350701440157
diffmk. (2018). Retrieved June 27, 2018, from https://docs.oracle.com/cd/E36784_01/html/E36870/diffmk-1.html#scrolltoc
Docker. (2018). Retrieved August 28, 2018, from https://docs.docker.com
Duarte, A. M. D., Duarte, D., & Thiry, M. (2016). TraceBoK: Toward a Software Requirements Traceability Body of Knowledge. In *24th International Requirements Engineering Conference (RE)* (pp. 236–245). IEEE. https://doi.org/10.1109/RE.2016.32
Duvall, P. M., Matyas, S., & Glover, A. (2007). *Continuous Integration: Improving Software Quality and Reducing Risk*. *Addison-Wesley Professional* (1st ed.). Addison-Wesley Professional.
DZone DevOps. (2018). Retrieved August 28, 2018, from https://dzone.com/articles/what-is-cicd
Eck, A., Uebernickel, F., & Brenner, W. (2014). Fit for continuous integration: how organizations assimilate an agile practice. In *20th Americas Conference on Information Systems (AMCIS '14)* (pp. 1–11). GA, USA: AIS. https://doi.org/10.1109/ACCESS.2017.2685629
Egyed, A. (2001). A scenario-driven approach to traceability. In *23rd International Conference on Software Engineering. ICSE 2001* (pp. 123–132). IEEE. https://doi.org/10.1109/ICSE.2001.919087
Farcic, V. (2016). *The DevOps 2.0 Toolkit: Automating the Continuous Deployment Pipeline with Containerized Microservices* (1st ed.). CreateSpace Independent Publishing Platform.
Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, *35*(3), 379–383. https://doi.org/10.3758/BF03195514
Fernández, P. (2008). Book Review: Google's PageRank and Beyond: The Science of Search Engine Rankings. *The Mathematical Intelligencer,* 30(1), 68–69. https://doi.org/10.1007/BF02985759
Filho, G. A. de A. C., & Lencastre, M. (2012). Towards a Traceability Visualisation Tool. In *8th International Conference on the Quality of Information and Communications Technology* (pp. 221–223). IEEE. https://doi.org/10.1109/QUATIC.2012.60

Fockel, M., Holtmann, J., & Meyer, J. (2012). Semi-automatic establishment and maintenance of valid traceability in automotive development processes. In *2nd International Workshop on Software Engineering for Embedded Systems (SEES)* (pp. 37–43). IEEE. https://doi.org/10.1109/SEES.2012.6225489

Galbo, S. P. D. (2010). *A Survey of Impact Analysis Tools for Effective Code Evolution*. University of Central Florida.

Galvão, I., & Goknil, A. (2007). Survey of traceability approaches in model-driven engineering. In *IEEE International Enterprise Distributed Object Computing Workshop, EDOC* (pp. 313–324). IEEE. https://doi.org/10.1109/EDOC.2007.4384003

Gephi. (2017). Retrieved October 14, 2017, from https://gephi.org/

Ghantous, G. B., & Gill, A. (2017). DevOps: Concepts, Practices, Tools, Benefits and Challenges. In *21st Pacific Asia Conference on Information Systems* (pp. 1–13). Langkawi, Malaysia: AIS.

GNU "ed." (2018). Retrieved June 27, 2018, from http://www.gnu.org/software/ed/manual/ed_manual.html

Goknil, A., Kurtev, I., & Berg, K. van den. (2016). A Rule-Based Change Impact Analysis Approach in Software Architecture for Requirements Changes. *Eprint ArXiv:1608.02757*, 1–44.

Goknil, A., Kurtev, I., van den Berg, K., & Spijkerman, W. (2014). Change impact analysis for requirements: A metamodeling approach. *Information and Software Technology*, *56*(8), 950–972. https://doi.org/10.1016/j.infsof.2014.03.002

Graph Visualization-Neo4j. (2018). Retrieved July 21, 2017, from https://neo4j.com/developer/guide-data-visualization/

Hambling, B., & Goethem, P. van. (2013). *User Acceptance Testing: A Step-by-step Guide* (1st ed.). BCS, The Chartered Institute for IT.

Hattori, L., Guerrero, D., Figueiredo, J., Brunet, J., & Dam, J. (2008). On the Precision and Accuracy of Impact Analysis Techniques. In *7th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2008)* (pp. 513–518). IEEE. https://doi.org/10.1109/ICIS.2008.104

Hayes, J. H., Dekhtyar, A., Sundaram, S. K., Holbrook, E. A., Vadlamudi, S., & April, A. (2007). REquirements TRacing On target (RETRO): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, *3*(3), 193–202. https://doi.org/10.1007/s11334-007-0024-1

Herman, I., Melancon, G., & Marshall, M. S. (2000). Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, *6*(1), 24–43. https://doi.org/10.1109/2945.841119

Holten, D. (2006). Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE Transactions on Visualization and Computer Graphics*, *12*(5), 741–748. https://doi.org/10.1109/TVCG.2006.147

IBM-Rational DOORS. (2017). Retrieved October 14, 2017, from https://www.ibm.com/us-en/marketplace/rational-doors

Ibrahim, S., Idris, N. B., Munro, M., & Deraman, A. (2005). Integrating Software Traceability for Change Impact Analysis. *Integrating Software Traceability for Change Impact Analysis*, *2*(4), 301–308.

Ibrahim, S., Munro, M., & Deraman, A. (2005). A Requirements Traceability To Support Change Impact Analysis. *Asian Journal of Information Technology*, *4*(4), 335–344.

Jaro Winkler Distance. (2017). Retrieved October 14, 2017, from http://alias-i.com/lingpipe/docs/api/com/aliasi/spell/JaroWinklerDistance.html

Jashki, M.-A., Zafarani, R., & Bagheri, E. (2008). 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* (pp. 84–90). New York, USA: ACM. https://doi.org/10.1145/1512475.1512493

Javed, M. A., & Zdun, U. (2014). A systematic literature review of traceability approaches between software architecture and source code. In *18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14* (pp. 1–10). New York, USA: ACM. https://doi.org/10.1145/2601248.2601278

JFreeChart. (2018). Retrieved August 14, 2018, from http://www.jfree.org/jfreechart/

JIRA Software. (2018). Retrieved October 2, 2018, from https://www.atlassian.com/software/jira

Kabeer, S. J., Nayebi, M., Ruhe, G., Carlson, C., & Chew, F. (2017). Predicting the Vector Impact of Change - An Industrial Case Study at Brightsquid. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 131–140). IEEE. https://doi.org/10.1109/ESEM.2017.20

Kama, N. (2013). Change impact analysis for the software development phase: State-of-the-art. *International Journal of Software Engineering & Its Applications*, *7*(2), 235–244.

Kamalabalan, K., Uruththirakodeeswaran, T., Thiyagalingam, G., Wijesinghe, D. B., Perera, I., Meedeniya, D., & Balasubramaniam, D. (2015). Tool support for traceability of software artefacts. In *Moratuwa Engineering Research Conference (MERCon)* (pp. 318–323). IEEE. https://doi.org/10.1109/MERCon.2015.7112366

Kchaou, D., Bouassida, N., & Ben-Abdallah, H. (2017). UML models change impact analysis using a text similarity technique. *IET Software*, *11*(1), 27–37. https://doi.org/10.1049/iet-sen.2015.0113

Keenan, E., Czauderna, A., Leach, G., Cleland-Huang, J., Shin, Y., Moritz, E., … Hearn, D. (2012). TraceLab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *34th International Conference on Software Engineering (ICSE)* (pp. 1375–1378). Zurich, Switzerland: IEEE. https://doi.org/10.1109/ICSE.2012.6227244

Keiningham, T. L., Aksoy, L., Cooil, B., Andreassen, T. W., & Williams, L. (2008). A holistic examination of Net Promoter. *Journal of Database Marketing & Customer Strategy Management*, *15*(2), 79–90. https://doi.org/10.1057/dbm.2008.4

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2016). *The DevOps Handbook* (1st ed.). IT Revolution Press.

Kitsu, E., Omori, T., & Maruyama, K. (2013). Detecting Program Changes from Edit History of Source Code. In *20th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 299–306). IEEE. https://doi.org/10.1109/APSEC.2013.48

Knoke, D., & Yang, S. (2008). *Social Network Analysis* (3rd ed.). London: SAGE Publications.

Kugele, S., & Antkowiak, D. (2016). Visualization of Trace Links and Change Impact Analysis. In *IEEE 24th International Requirements Engineering Conference Workshops (REW)* (pp. 165–169). Beijing, China: IEEE. https://doi.org/10.1109/REW.2016.039

LDRA. (2018). Retrieved July 5, 2017, from http://www.ldra.com/en/software-quality-test-tools/group/by-software-life-cycle/requirements-traceability

Lee, C., Guadagno, L., & Jia, X. (2003). An agile approach to capturing requirements and traceability. In *2nd International Workshop on Traceability in Emerging Forms of Software Engineering* (pp. 1–7). Citeseer.

Lee, J., Cho, B., Youn, H., & Lee, E. (2009). Reliability analysis method for supporting traceability using UML. In *Communications in Computer and Information Science* (pp. 94–101). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-10619-4_12

Lee, M., & Offutt, A. J. (2002). Algorithmic analysis of the impacts of changes to object-oriented software. In *Technology of Object-Oriented Languages and Systems* (pp. 61–70). CA, USA: IEEE. https://doi.org/10.1109/TOOLS.2000.868959

Lee, W.-T., Deng, W.-Y., Lee, J., & Lee, S.-J. (2010). Change impact analysis with a goal-driven traceability-based approach. *International Journal of Intelligent Systems*, *25*(8), 878–908. https://doi.org/10.1002/int.20443

Lehnert, S. (2011). 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* (pp. 41–50). New York, USA: ACM. https://doi.org/10.1145/2024445.2024454

Lehnert, S. (2015). *Multiperspective Change Impact Analysis to Support Software Maintenance and Reengineering*. University of Hamburg.

Lehnert, S., Farooq, Q. U. A., & Riebisch, M. (2013). Rule-based impact analysis for heterogeneous software artifacts. In *European Conference on Software Maintenance and Reengineering, CSMR* (pp. 209–218). Genova, Italy: IEEE. https://doi.org/10.1109/CSMR.2013.30

Levenshtein-Algorithm. (2017). Retrieved October 14, 2017, from http://www.levenshtein.net/

Li, B., Sun, X., Leung, H., & Zhang, S. (2013). A survey of code-based change impact analysis techniques. *Software Testing Verification and Reliability*, *23*(8), 613–646. https://doi.org/10.1002/stvr.1475

Lindholm, T., Kangasharju, J., & Tarkoma, S. (2006). Fast and simple XML tree differencing by sequence alignment. In *ACM Symposium on Document Engineering - DocEng '06* (pp. 75–84). New York, USA: ACM. https://doi.org/10.1145/1166160.1166183

Lucia, A. De, Fasano, F., Oliveto, R., & Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology*, *16*(4), 13:1-13:50. https://doi.org/10.1145/1276933.1276934

Mäder, P., & Gotel, O. (2012). Towards automated traceability maintenance. *Journal of Systems and Software*, *85*(10), 2205–2227. https://doi.org/10.1016/j.jss.2011.10.023

Mäder, P., Gotel, O., Kuschke, T., & Philippow, I. (2008). traceMaintainer - Automated Traceability Maintenance. In *16th IEEE International Requirements Engineering Conference* (pp. 329–330). Catalunya, Spain: IEEE. https://doi.org/10.1109/RE.2008.25

Marcus, A., Xie, X., & Poshyvanyk, D. (2005). When and how to visualize traceability links? In *3rd International Workshop on Traceability in Emerging Forms of Software Engineering - TEFSE '05* (pp. 56–61). New York, USA: ACM. https://doi.org/10.1145/1107656.1107669

Maro, S., Anjorin, A., Wohlrab, R., & Steghöfer, J.-P. (2016). Traceability maintenance: factors and guidelines. In *31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016* (pp. 414–425). New York, USA: ACM. https://doi.org/10.1145/2970276.2970314

Matplotlib. (2018). Retrieved August 14, 2018, from http://matplotlib.sourceforge.net

Maule, A., Emmerich, W., & Rosenblum, D. S. (2008). Impact analysis of database schema changes. In *13th International Conference on Software Engineering - ICSE '08* (pp. 451–460). New York, USA: ACM. https://doi.org/10.1145/1368088.1368150

Measuring Requirements. (2018). Retrieved August 31, 2018, from https://www.jamasoftware.com/blog/measuring-requirements-product-size-requirements-quality/

MeasuringU. (2018). Retrieved November 7, 2018, from https://measuringu.com/nps-sus/

Mens, T., Buckley, J., Zenger, M., & Rashid, A. (2005). Towards a Taxonomy of Software Evolution. *J. Softw. Maint. Evol.*, *17*(5), 309–332. https://doi.org/10.1002/smr.v17:5

Merten, T., Jüppner, D., & Delater, A. (2011). Improved representation of traceability links in requirements engineering knowledge using Sunburst and Netmap visualizations. In *4th International Workshop on Managing Requirements Knowledge, MaRK'11 - Part of the 19th IEEE International Requirements Engineering Conference, RE'11* (pp. 17–21). Trento, Italy: IEEE. https://doi.org/10.1109/MARK.2011.6046557

Meyer, M. (2014). Continuous integration and its tools. *IEEE Software*, *31*(3), 14–16. https://doi.org/10.1109/MS.2014.58

Mills, C. (2017). Towards the automatic classification of traceability links. In *32nd IEEE/ACM International Conference on Automated Software Engineering-ASE 2017* (pp. 1018–1021). IL, USA: IEEE. https://doi.org/10.1109/ASE.2017.8115723

Mischler, A., & Monperrus, M. (2014). An Approach for Discovering Traceability Links between Regulatory Documents and Source Code Through User-Interface Labels. *Eprint ArXiv:1403.2639*, 1–14.

Mohan, K., Xu, P., Cao, L., & Ramesh, B. (2008). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, *45*(4), 922–936. https://doi.org/10.1016/j.dss.2008.03.003

Molhado Project. (2017). Retrieved July 5, 2017, from http://www.ece.iastate.edu/~tien/molhado/index.html

NetworkX. (2018). Retrieved August 19, 2018, from https://networkx.github.io/

Newman, M. (2010). *Networks: An Introduction* (1st ed.). Oxford University Press. https://doi.org/10.1093/acprof:oso/9780199206650.001.0001

Nguyen, T. N., Munson, E. V., & Boyland, J. T. (2004). The molhado hypertext versioning system. In *15th ACM Conference on Hypertext & Hypermedia - HYPERTEXT '04* (pp. 185–194). New York, USA: ACM. https://doi.org/10.1145/1012807.1012859

Nistor, E. C., Erenkrantz, J. R., Hendrickson, S. A., & van der Hoek, A. (2005). ArchEvol: versioning architectural-implementation relationships. In *12th International Workshop on Software Configuration Management - SCM '05* (pp. 99–111). New York, USA: ACM. https://doi.org/10.1145/1109128.1109136

Oliva, G. A., Gerosa, M. A., Milojicic, D., & Smith, V. (2013). A change impact analysis approach for workflow repository management. In *IEEE 20th International Conference on Web Services, ICWS 2013* (pp. 308–315). CA, USA: IEEE. https://doi.org/10.1109/ICWS.2013.49

Olsson, M. (2015). Document Object Model. In *JavaScript Quick Syntax Reference* (pp. 39–44). Berkeley, CA: Springer. https://doi.org/10.1007/978-1-4302-6494-1_10

Omori, T., & Maruyama, K. (2008). A change-aware development environment by recording editing operations of source code. In *International Workshop on Mining Software Repositories - MSR '08* (pp. 31–34). New York, USA: ACM. https://doi.org/10.1145/1370750.1370758

Passos, L., Apel, S., Kästner, C., Czarnecki, K., Wasowski, A., & Guo, J. (2013). Feature Oriented Software Evolution. In *7th International Workshop on Variability Modelling of Software-intensive Systems-VaMoS '13* (pp. 17:1-17:8). Pisa, Italy: ACM. https://doi.org/10.1145/2430502.2430526

Perera, I., Miller, A., & Allison, C. (2017). A Case Study in User Support for Managing OpenSim Based Multi User Learning Environments. *IEEE Transactions on Learning Technologies*, *10*(3), 342–354. https://doi.org/10.1109/TLT.2016.2632126

Pete, I., & Balasubramaniam, D. (2015). Handling the differential evolution of software artefacts: A framework for consistency management. In *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* (pp. 599–600). QC, Canada: IEEE. https://doi.org/10.1109/SANER.2015.7081889

Phetmanee, S., & Suwannasart, T. (2014). A Tool for Impact Analysis of Test Cases Based on Changes of a Web Application. In *International MultiConference of Engineers and Computer Scientists-IMECS '14*, I, (pp. 497–500). Hong Kong: IAENG.

Priority Blocking Queue. (2018). Retrieved June 29, 2018, from http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/PriorityBlockingQueue.html

QASource DevOps Experts. (2018). Retrieved August 28, 2018, from https://www.qasource.com/devops#!devops-expertise

Rajlich, V. (2014). Software evolution and maintenance. In *Future of Software Engineering - FOSE 2014* (pp. 133–144). New York, USA: ACM. https://doi.org/10.1145/2593882.2593893

Rational RequisitePro. (2017). Retrieved July 5, 2017, from https://www.oit.va.gov/Services/TRM/ToolPage.aspx?tid=41

Redmiles, D., Van Der Hoek, A., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., … Trainer, E. (2007). Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects. *Wirtschaftsinformatik*, *49*, 28–38. https://doi.org/10.1038/leu.2008.246

Rempel, P., & Mader, P. (2017). Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, *43*(8), 777–797. https://doi.org/10.1109/TSE.2016.2622264

Ren, X., Ryder, B. G., Stoerzer, M., & Tip, F. (2005). Chianti: a change impact analysis tool for java programs. In *27th International Conference on Software Engineering - ICSE '05* (pp. 664–665). New York, USA: ACM. https://doi.org/10.1145/1062455.1062598

ReqView. (2017). Retrieved May 7, 2018, from https://www.reqview.com/

Riebisch, M., Bode, S., Farooq, Q. U. A., & Lehnert, S. (2011). Towards comprehensive modelling by inter-model links using an integrating repository. In *18th IEEE International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2011* (pp. 284–291). NV, USA: IEEE. https://doi.org/10.1109/ECBS.2011.32

Rodrigues, A., Lencastre, M., & Filho, G. A. de A. C. (2016). Multi-VisioTrace: Traceability Visualization Tool. In *10th International Conference on the Quality of Information and Communications Technology (QUATIC)* (pp. 61–66). Lisbon, Portugal: IEEE. https://doi.org/10.1109/QUATIC.2016.019

Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2017). Towards Traceability Management in Continuous Integration with SAT-analyzer. In *3rd International Conference on Communication*

and Information Processing (ICCIP 2017) (pp. 77–81). Tokyo, Japan: ACM. https://doi.org/10.1145/3162957.3162985

Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018a). Automated Inter-artefact Traceability Establishment for DevOps Practice. In *2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS 2018)* (pp. 211–216). Singapore: IEEE. https://doi.org/10.1109/ICIS.2018.8466414

Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018b). Software Artefact Traceability Analyser : A Case-Study on POS System. In *6th International Conference on Communications and Broadband Networking (ICCBN 2018)* (pp. 1–5). Singapore: ACM. https://doi.org/10.1145/3193092.3193094

Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018). Traceability Management with Impact Analysis in DevOps based Software Development. In *2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1956–1962). Bangalore, India: IEEE. https://doi.org/10.1109/ICACCI.2018.8554399

Sager, T., Bernstein, A., Pinzger, M., & Kiefer, C. (2006). Detecting similar Java classes using tree algorithms. In *International Workshop on Mining Software Repositories - MSR '06* (pp. 65–71). Shanghai, China: ACM. https://doi.org/10.1145/1137983.1138000

Santiago, I., Vara, J. M., De Castro, V., & Marcos, E. (2014). Visualizing Traceability Information with iTrace. In *9th International Conference on Evaluation of Novel Approaches to Software Engineering* (pp. 5–15). SCITEPRESS Publications. https://doi.org/10.5220/0004865400050015

Sarma, A., Redmiles, D. F., & Van Der Hoek, A. (2012). Palantír: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering*, *38*(4), 889–908. https://doi.org/10.1109/TSE.2011.64

SAT-Analyser. (2018). Retrieved November 10, 2018, from https://sites.google.com/cse.mrt.ac.lk/sat-analyser/case-studies

Selenium. (2018). Retrieved October 2, 2017, from http://www.seleniumhq.org

SERG :ReqAnalyst. (2017). Retrieved July 5, 2017, from http://swerl.tudelft.nl/bin/view/Main/ReqAnalyst

Shahid, M., & Ibrahim, S. (2016). Change impact analysis with a software traceability approach to support software maintenance. In *13th International Bhurban Conference on Applied Sciences and Technology (IBCAST)* (pp. 391–396). IEEE. https://doi.org/10.1109/IBCAST.2016.7429908

Sharafat, A. R., & Tahvildari, L. (2007). A Probabilistic Approach to Predict Changes in Object-Oriented Software Systems. In *11th European Conference on Software Maintenance and Reengineering (CSMR'07)* (pp. 27–38). Amsterdam, Netherlands: IEEE. https://doi.org/10.1109/CSMR.2007.9

Shneiderman, B. (1992). Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics (TOG)*, *11*(1), 92–99. https://doi.org/10.1145/102377.115768

Sinclair, J., & Cardew-Hall, M. (2008). The folksonomy tag cloud: when is it useful? *Journal of Information Science*, *34*(1), 15–29. https://doi.org/10.1177/0165551506078083

Slack. (2018). Retrieved October 2, 2018, from https://slack.com

Sommerville, I. (2010). *Software Engineering* (10th ed.). New York: Addison-Wesley Professional.

Spijkerman, W. (2010). *Tool Support for Change Impact Analysis in Requirement Models*. University of Twente.

Sun, X., Li, B., Tao, C., Wen, W., & Zhang, S. (2010). Change impact analysis based on a taxonomy of change types. In *International Computer Software and Applications Conference* (pp. 373–382). IEEE. https://doi.org/10.1109/COMPSAC.2010.45

Sünnetcioglu, A., Brandenburg, E., Rothenburg, U., & Stark, R. (2016). *ModelTracer: User-friendly Traceability for the Development of Mechatronic Products*. *Procedia Technology,* 26, (pp. 365–373). Elsevier. https://doi.org/10.1016/j.protcy.2016.08.047

Suzuki, N. (2002). A Structural Merging Algorithm for Xml documents. In *International Conference on WWW/Internet* (pp. 699–703). IADIS.

Tang, A., Jin, Y., & Han, J. (2007). A rationale-based architecture model for design traceability and reasoning. *Journal of Systems and Software*, *80*(6), 918–934. https://doi.org/10.1016/j.jss.2006.08.040

Thommazo, A. Di, Malimpensa, G., De Oliveira, T. R., Olivatto, G., & Fabbri, S. C. P. F. (2012). Requirements Traceability Matrix: Automatic Generation and Visualization. In *26th Brazilian Symposium on Software Engineering, SBES 2012* (pp. 101–110). Natal, Brazil: IEEE. https://doi.org/10.1109/SBES.2012.29

Tóth, G., Hegedűs, P., Beszédes, Á., Gyimóthy, T., & Jász, J. (2010). 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* (pp. 109–118). Vienna, Austria: ACM. https://doi.org/10.1145/1852761.1852777

Travis CI. (2018). Retrieved July 5, 2017, from https://travis-ci.org/

Trello. (2018). Retrieved October 2, 2018, from https://trello.com/

Trung, P. T., & Thang, H. Q. (2009). Building the reliability prediction model of component-based software architectures. *World Academy of Science, Engineering and Technology*, *35*(11), 911–918.

Tyree, J., & Akerman, A. (2005). Architecture decisions: Demystifying architecture. *IEEE Software*, *22*(2), 19–27. https://doi.org/10.1109/MS.2005.27

Vector Space Model. (2017). Retrieved July 3, 2017, from http://cogsys.imm.dtu.dk/thor/projects/multimedia/textmining/node5.html

Vrignat, P., Avila, M., Duculty, F., & Kratz, F. (2015). Failure Event Prediction Using Hidden Markov Model Approaches. *IEEE Transactions on Reliability*, *64*(3), 1038–1048. https://doi.org/10.1109/TR.2015.2423191

Wang, W., He, Y., Li, T., Zhu, J., & Liu, J. (2018). An Integrated Model for Information Retrieval Based Change Impact Analysis. *Scientific Programming*, *2018*(Article ID 5913634), 1–13. https://doi.org/10.1155/2018/5913634

Wang, Y., DeWitt, D. J., & Cai, J. Y. (2003). X-Diff: An effective change detection algorithm for XML documents. In *International Conference on Data Engineering* (pp. 519–530). IEEE. https://doi.org/10.1109/ICDE.2003.1260818

Wang, Y., Zhang, J., & Fu, Y. (2017). Rule-Based Change Impact Analysis Method in Software Development. In *2nd International Conference on Computer Engineering, Information Science & Application Technology (ICCIA 2017)* 74, (pp. 396–403). Atlantis Press.

Wijesinghe, D. B., Kamalabalan, K., Uruththirakodeeswaran, T., Thiyagalingam, G., Perera, I., & Meedeniya, D. (2014). Establishing traceability links among software artefacts. In *14th International Conference on Advances in ICT for Emerging Regions (ICTer)* (pp. 55–62). IEEE. https://doi.org/10.1109/ICTER.2014.7083879

Winkler, S. (2008). On Usability in Requirements Trace Visualizations. In *Requirements Engineering Visualization* (pp. 56–60). Catalunya, Spain: IEEE. https://doi.org/10.1109/REV.2008.4

Wong, S., Cai, Y., & Dalton, M. (2011). *Change Impact Analysis with Stochastic Dependencies*. *Technical Report*. PA, USA.

XMLUnit. (2018). Retrieved October 20, 2018, from https://www.xmlunit.org/

YAKINDU Traceability. (2019). Retrieved January 25, 2019, from https://www.itemis.com/en/yakindu/traceability/

Zeugmann, T., Poupart, P., Kennedy, J., Jin, X., Han, J., Saitta, L., … Fürnkranz, J. (2011). Precision and Recall. In *Encyclopedia of Machine Learning* (pp. 781–781). Boston, MA: Springer US. https://doi.org/10.1007/978-0-387-30164-8_652

Zhang, S., Gu, Z., Lin, Y., & Zhao, J. (2008). Change impact analysis for AspectJ programs. In *IEEE International Conference on Software Maintenance* (pp. 87–96). Beijing, China: IEEE. https://doi.org/10.1109/ICSM.2008.4658057

Zhang, Y., Wan, C., & Jin, B. (2016). 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)* (pp. 121–126). Shanghai, China: IEEE. https://doi.org/10.1109/SNPD.2016.7515889

Zimmermann, T., Zeller, A., Weissgerber, P., & Diehl, S. (2005). Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, *31*(6), 429–445. https://doi.org/10.1109/TSE.2005.72

Zoho Sprints. (2018). Retrieved October 2, 2018, from https://www.zoho.com/sprints/

Questionnaire

1. Please indicate your gender
    - Female
    - Male
    - Prefer not to say
2. What is your age group?
    - 18 - 24 years
    - 25 - 29 years
    - 30 - 34 years
    - 35 - 39 years
    - 40+ years
3. Please indicate your highest educational level
    - Diploma
    - Bachelor Degree
    - Master Degree
    - Professional
    - Other
4. Which of the following best describes your role?
    - Programmer/ Junior level
    - Quality Assurance level
    - Deployment level
    - Operational level
    - Other:____________
5. How long you have been working with DevOps?
    - Less than 1 year
    - 1 - 3 years
    - More than 3 years
6. What are the involved software artefacts for stages in SDLC?
7. Please specify other artefacts you use if any:________________
8. What is/are the involved programming language(s)?
    - Java
    - Python
    - C/C++
    - Other:____________
9. What types of tests are conducted?
    - Unit tests
    - Integration tests
    - Functional tests
    - Regression tests
    - Other:____________

Change Management: Regarding the change management in your DevOps environment.

10. How often do you check for software artefact (source code/ design etc) changes?
11. What is/are the tool(s) used for change detection/ change management?
12. How do you handle the traceability? (Manually/ Tools used/ Do you visualize traceability?)
13. Is it helpful to have a traceability tool with visualization?
14. What are the limitations you experience in detecting changes?
15. How do you propagate changes and limitations (if any)?
16. How do you analyse the impact of changes?
    - Dependency-based calculation
    - Traceability-based calculation
    - Static analysis
    - Dynamic analysis
    - Manually
    - We don't do
    - Other:____________
17. What are the limitations you experience in analysing impacts?

Continuous Integration: Regarding Continuous Integration in your DevOps environment.

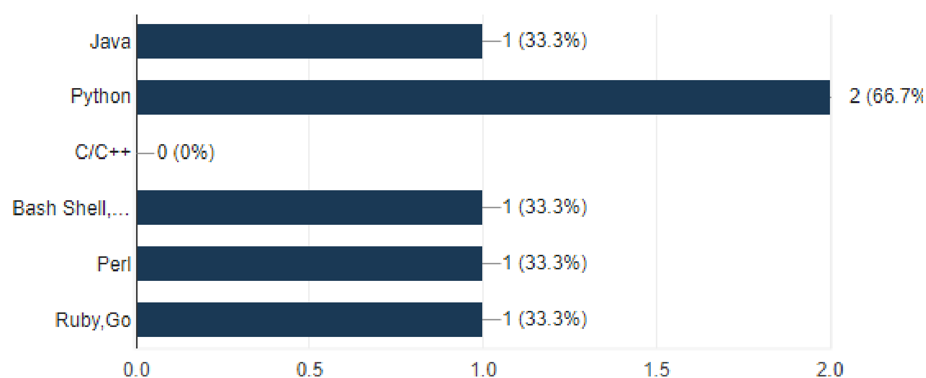
18. How often do you perform continuous integrations?
    - Very frequently-anytime
    - Hourly
    - Daily-Once in a day
    - Other:___________
19. How do you perform Continuous Integration process and limitations (if any)?
20. What are the Continuous Integration/ Continuous Delivery/ DevOps tools you use?
    - CVS: Github/ Bitbucket
    - Jenkins
    - Puppet
    - Jira
    - Travis CI
    - Docker
    - Other:___________
21. What types of projects are done with DevOps practices?
    - Small scale projects (i.e. less than 5 OOP classes)
    - Any scale project
    - Any domain project
    - Other:___________
22. What is the maximum number of classes involved in a project you have done using DevOps? (eg. number of classes in the class diagram/ number of classes in the source code)
23. What are the difficulties you face in working with DevOps?

Summary of responses for the initial survey

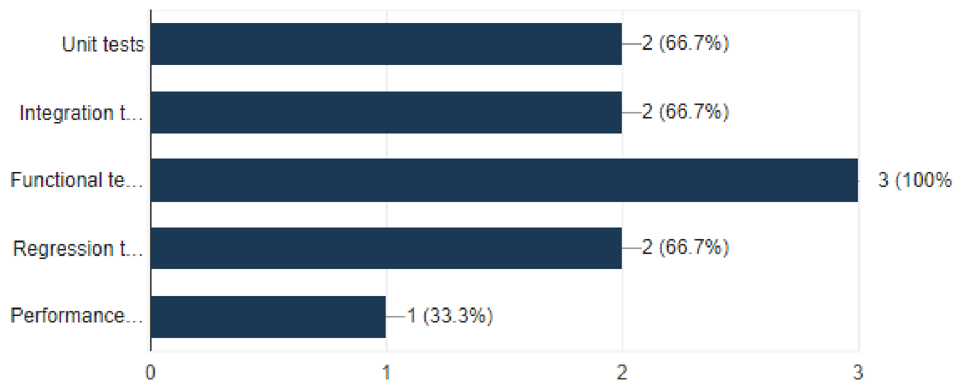
Involved artefacts:

Artefact	Requirements engineering	Design	Development	Testing	Configuration	Deployment	Operations
SRS document	X						
User stories	X	X					
Story cards	X	X					
Class diagram	X	X					
Use case diagram	X	X					
sequence diagram	X	X					
Other design diagrams	X	X	X				
Source code			X				
Build scripts			X		X	X	X
Test cases				X			
Test scripts				X			
Configuration/ dependency files					X		
Deployment scripts					X	X	X
Cloud integration scripts					X	X	X
User manuals					X	X	X
Containerized images						X	
KB articles							X
Monitoring/ synthetics							X

Programming languages:



Testing types:



Change detection frequency:

- Automatically Hourly
- Manually Weekly

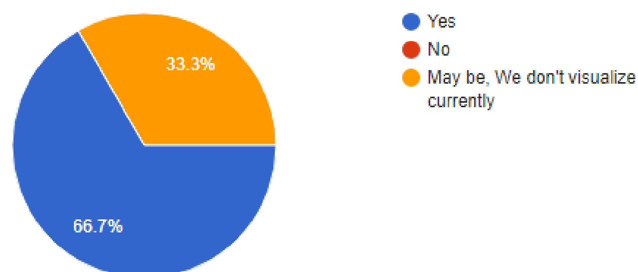
Change detection tools:

- Jenkins Cron Job
- ServiceNow
- JIRA Service Desk
- ServiceNow - to manage tickets

Traceability handling methods:

- Jenkins
- Custom audit tools/ CloudTrail

Usefulness of Traceability and Visualization:



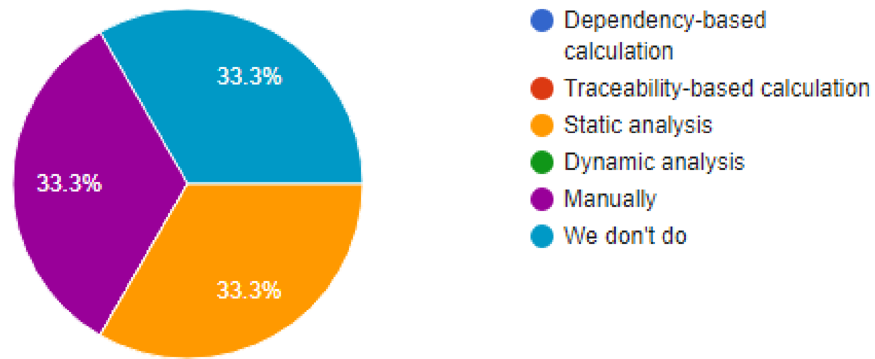
Change detection limitations:

- With Jenkins this is done automatically
- No proper tools to auto detect changes, Just monitoring tools to detect failures

Change propagation methods:

- Automatically deploy to the server with Jenkins
- Using pre-defined protocols and policies defined by the company

Impact analysis methods:



Impact analysis limitations:

- Time consuming
- It's hard to calculate the exact impact

CI frequency:

- Very frequently-anytime

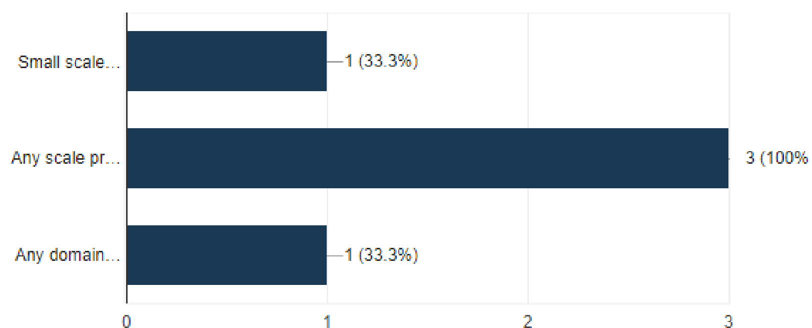
CI methods:

- Jenkins - No limitations encountered as we do CI/CD
- Tools: Jenkins, CodeDeploy, CodePipeline, TravisCI, TeamCity

CI / CD / DevOps tools chain:

- CVS: Github, BitBucket; Jenkins; Puppet; Jira; Travis CI; Docker; Code Pipeline; OpsWorks; Octopus; TeamCity

DevOps suitability:



DevOps limitations:

- Adapting to Operational role at times is difficult
- Broader domain. You need to have excellent knowledge in programming, networking, OS and storage domains to practice

Appendix B: User acceptance survey

Questionnaire for the interview (post-interview)

	Strongly Agree				Strongly Disagree
1. I think that I would like to use this system frequently	1	2	3	4	5
2. I found the system unnecessarily complex	1	2	3	4	5
3. I thought the system was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use this system	1	2	3	4	5
5. I found the various functions in this system were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this system	1	2	3	4	5
7. I would imagine that most people would learn to use this system very quickly	1	2	3	4	5
8. I found the system very cumbersome to use	1	2	3	4	5
9. I felt very confident using the system	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this system	1	2	3	4	5

11. Please underline 3 words that best describe your impression/ idea/ quality of the tool

Poor	Average	Usable	Efficient	Novelty
Prototype	Good	Adaptable	Accuracy	Originality
Slow	Excellent	Decision-making	Analysis	Simplicity
Improvable	Innovative	Supportive	Traceability	Collaborative

Summary of responses for the interview

The SUS calculation values (SUS score) for all the responses obtained from 20 participants (P) for each question (Q) averaged to a final SUS score of 62.5.

Participant	Q 1	Q 2	Q 3	Q 4	Q 5	Q 6	Q 7	Q 8	Q 9	Q 10	SUS score
P1	4	4	3	3	5	2	3	3	4	3	60.0
P2	4	1	3	1	3	4	2	3	3	2	60.0
P3	4	1	3	1	3	2	4	3	4	4	67.5
P4	4	2	4	3	4	2	4	1	3	2	72.5
P5	4	2	4	1	5	1	2	3	2	4	65.0
P6	4	3	4	2	4	1	4	3	3	5	62.5
P7	4	2	3	4	5	2	4	2	2	3	62.5
P8	3	1	4	3	3	2	4	2	3	3	65.0
P9	4	2	4	4	5	2	5	4	3	4	62.5
P10	4	2	3	1	4	2	4	3	3	3	67.5
P11	4	2	3	4	4	2	4	3	2	2	60.0
P12	3	2	4	3	4	3	4	3	4	3	62.5
P13	3	2	3	3	4	3	3	2	3	3	57.5
P14	3	4	3	3	3	3	2	3	3	2	47.5
P15	3	3	3	2	4	2	3	1	3	2	65.0
P16	4	3	4	2	4	2	4	2	3	3	67.5
P17	3	4	4	3	4	3	3	4	2	2	50.0
P18	4	3	3	3	4	3	3	3	3	2	57.5
P19	4	2	4	3	4	2	4	4	3	2	65.0
P20	4	2	3	2	4	1	3	1	4	3	72.5
										Average	**62.5**

Note: user.home means the PC's logged in account's user directory where you can find the Documents, Desktop etc directories listed. i.e C:\Users\User\ or C:\Users\cse

1. Make sure these software are installed;
   a. JDK 1.8 and JRE both with environment variables set to JDK bin
   b. WordNet 2.1 into PC's user home directory (user.home\WordNet\bin)
   c. Python 2.7 into PC's C:\ drive (C:\Python27\python.exe)
   d. Microsoft visual C++ 2010 redistribution x64 or x32 (https://www.microsoft.com/en-us/download/confirmation.aspx?id=15336)
   e. Wampserver x64 or x32 into C:\ drive (C:\wamp\www)
   f. D3.js (C:\wamp\www\d3\d3.js)
   g. Google chrome browser
2. Install following Python packages (using pip-Win tool: https://sites.google.com/site/pydatalog/python/pip-for-windows)

Package Name	pip-Win tool Command
networkx	pip install networkx
numpy	pip install numpy
matplotlib	pip install matplotlib
scipy	pip install scipy

3. Copy these directories and files into exact following local locations in PC;
   a. SATAnalyzer → user.home\SATAnalyzer
   b. Resources → user.home\Resources
   c. SAT_CONFIGS → user.home\SAT_CONFIGS
   d. Files within www directory → C:\wamp\www *(For any existing files don't copy or replace and skip)*
   e. SAT_Analyser_2_0 → D:\ SAT_Analyser_2_0

4. **Stand-Alone Desktop Access:-**

   A. Run the JAR file sat-0.2-jar-with-dependencies.jar in path D:\SAT_Analyser_2_0\ SAT_Analyser_2_0\target\
   a. Can double click on the file (Not-recommended as cannot track any exceptions since this is a prototype level tool)
   b. Open command prompt (Recommended method)
   i. Change the drive to D:\ by typing D: and hit enter
   ii. Type the following command and hit enter

   java -jar "D:\SAT_Analyser_2_0\SAT_Analyser_2_0\target\sat-0.2-jar-with-dependencies.jar"

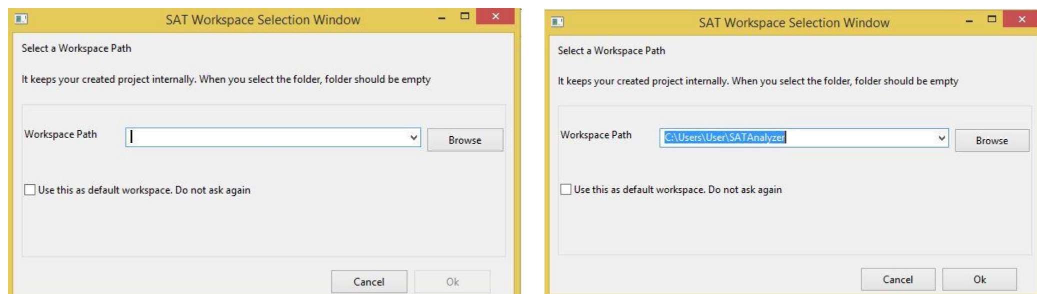
   Else double click on the exe file SAT_Analyser_2_0.exe in path D:\SAT_Analyser_2_0\ SAT_Analyser_2_0\target\

5. **Multi-User Web Access:-**

   A. Install AjaxSwing application with built-in Apache Tomcat server on one PC as Server_Machine (http://creamtec.com/products/ajaxswing/install/index.html).
   B. Copy file SAT2.properties file in path D:\SAT_Analyser_2_0\ SAT_Analyser_2_0\target\ to AjaxSwing installed path's conf directory (i.e. C:\AjaxSwing4.6.0\conf\SAT2.properties)
   a. Go to http://localhost:8040/ajaxswing/apps/SAT2 in browser (Server_Machine)
   b. Client machine(s) connected within same local area network; go to <Server_Machine's_IP_Address>:8040/ajaxswing/apps/SAT2 in a browser

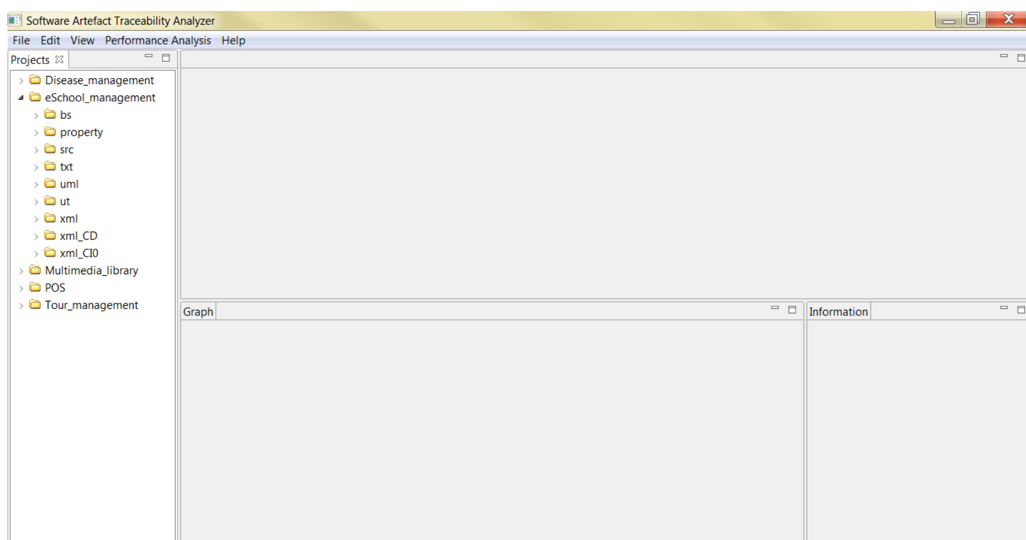
Initializing SAT-Analyser

Once the SAT-Analyser is executed for the first time, the workspace selection window will be prompted as below.

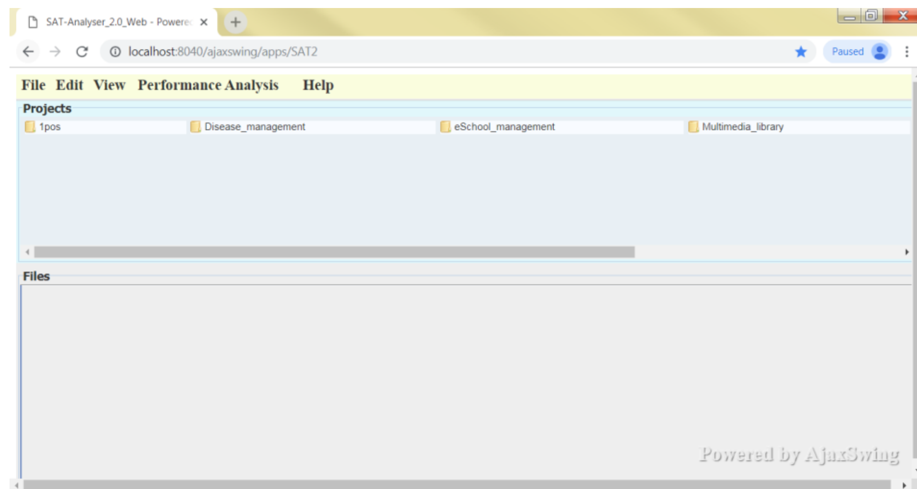


You can provide a location in your machine and click Ok. Then, SAT-Analyser **main window** will be loaded.

In the stand-alone desktop version; It consists of four main subsections for listing any existing traceability projects' directory structure vertically on the left-hand side corner, file opening section on the top, traceability results default visualization section on the bottom center and bottom right-hand side for listing the details of traceability results. Moreover, there is a top main menu for selecting further functionalities.

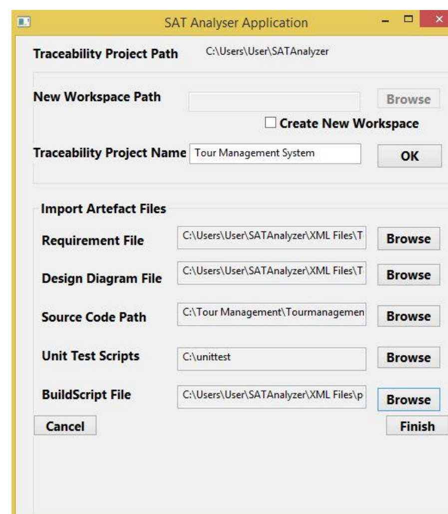


In the multi-user accessible web version; It consists of two main subsections for listing any existing traceability projects' directory structure on the top horizontally and file opening section on the bottom. The top main menu for selecting further functionalities is the same as in the desktop version.



Creating a software traceability project

In the top menu bar, select **File → New → Project** to start creating a software project for traceability generation. Then, the following **artefact input window** will be prompted to provide traceability project name and to insert the artefact inputs. First, you must give a project name which is not null and click Ok for the rest of the form items to be enabled. Once a valid project name is given and clicked Ok, the **Import Artefact Files** section will be activated to provide artefact input files. Provide each artefact separately by clicking on the **Browse** button and finally click **Finish** to create the project or click **Cancel** for the cancellation of the process.



Generating traceability outcomes

The requirement artefact element extraction process is set visible via a **Requirement Artefact Confirmation window** for the user to have a generic idea about the traceability items etc. Click on each of the elements listed to expand, edit, add or delete the element items as necessary. Then, click **Confirm** for the confirmation to start the traceability establishment process.

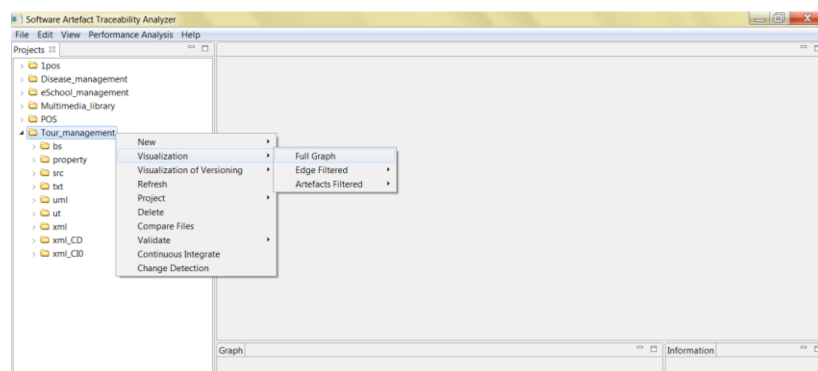
The **project main window** will be loaded with the newly created **project file tree** visible on the left-hand side corner section. The

- folder *bs* lists the Maven build script artefact file,
- *property* folder contains the intermediate traceability graph related files,
- folder *src* contains the source code Java file set,
- folder *txt* includes the SRS artefact text document file,
- folder *uml* consists of the UML class diagram artefact file,
- folder *ut* includes the set of JUnit unit test class files,
- folder *xml* holds all the SAT-Analyser tool generated intermediate XML format files of each artefact type such as;
    - Requirement Artefact File.xml,
    - UML Artefact File.xml,
    - Source Code Artefact File.xml,
    - Unit Test Artefact File.xml,
    - Build Script Artefact File.xml,
    - XML conversion of artefact traceability links: Relations.xml.

Select any of the files and double click on it to view the contents which will be opened in the top section.

Traceability visualization

Right click on the project name to view the traceability outcomes. Select **Visualization** and click a visualization type.

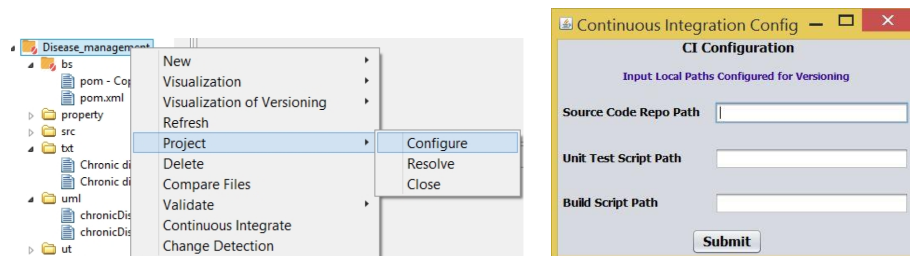


For instance, in the view option **Full Graph**: - overall traceability graph including all the types of artefacts as nodes and their relationships as links will be visible in the Graph section of the window. Zoom the view by scrolling the mouse pointer in and out. The naming conventions used in this traceability graph visualization is as follows; RQ - requirement, D – design, SC - source code, UT - unit test, BS - build script, _M - method/ function, _F - field/ attribute

Continuous integration

Frequent occurrences of integrations take place in a DevOps environment. It is featured with the traceability results in the SAT-Analyser tool. Whenever a continuous integration is to be submitted, the project source code path that is integrated with the build automation such as with Jenkins/ Github must be specified via the **configuration** as a prerequisite at the time of traceability project creation.

Right-click on the project and click on **Project** → select **Configure**. Insert the source path, unit test path and build script locations corresponding to the associated build automation repositories. Then, right-click on any project name and click on the option **Continuous Integrate**. Once the first integration task is triggered, an artefact input window pops up with the project path and an assigned integration ID number to submit each type of individual artefact inputs.



As a constraint of the SAT-Analyser tool if the integration contains additions of artefact elements/ sub-elements, then you must update all the artefact types to be tallied with the new additions and upload all types of artefact inputs. If your integration contains only modifications and/ or deletions of artefacts, you can specify that by clicking on a button named **Include Only Artefact Modifications and Deletions**. Then, you are allowed to upload only that particular type(s) of artefacts.

Change detection, change impact analysis and change propagation

There must be more than one successfully completed integration to proceed with the **Change Detection** option in the menu, by right-clicking the project name. Then it will be prompted with any changes artefact type and change type wise.

A click on the **Impact Analysis** button in the change detection window to proceed or else click **Cancel** to terminate the change analysis process. The impact analysis results window lists the impact of detected artefact changes on remaining artefact items with the manual editing feature. Then, click on the **Change Propagation** button at the bottom of the window to confirm the impact results. That will load the updated traceability graphs highlighting the changed/ modified artefact items.

For complete user guide: - https://sites.google.com/cse.mrt.ac.lk/sat-analyser/tool-support

Appendix E: List of companies involved in the surveys/ interviews

1.	Pearson Lanka
2.	Metatechno Lanka Company (Pvt) Ltd
3.	Typefi Colombo
4.	Apigate Sri Lanka Ltd
5.	Creative Software
6.	Sysco Labs
7.	HNB IT
8.	Epic Lanka (Pvt) Ltd
9.	John Keells Holdings
10.	Zone24x7 (Pvt) Ltd
11.	Tiqri (Pvt) Ltd

1. Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2017). Towards Traceability Management in Continuous Integration with SAT-analyzer. In *3rd International Conference on Communication and Information Processing (ICCIP 2017)* (pp. 77–81). Tokyo, Japan: ACM. https://doi.org/10.1145/3162957.3162985

2. Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018). Software Artefact Traceability Analyser : A Case-Study on POS System. In *6th International Conference on Communications and Broadband Networking (ICCBN 2018)* (pp. 1–5). Singapore: ACM. https://doi.org/10.1145/3193092.3193094

3. Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018). Automated Inter-artefact Traceability Establishment for DevOps Practice. In *IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS 2018)* (pp. 211–216). Singapore: IEEE. https://doi.org/10.1109/ICIS.2018.8466414

4. Rubasinghe, I. D., Meedeniya, D. A., & Perera, I. (2018). Traceability Management with Impact Analysis in DevOps based Software Development. In *International Conference on Advances in Computing, Communications and Informatics (ICACCI)* (pp. 1956–1962). Bangalore, India: IEEE. https://doi.org/10.1109/ICACCI.2018.8554399

5. Meedeniya, D. A., Rubasinghe, I. D., & Perera, I. (2019). Software Artefacts Consistency Management Towards CICD: A Roadmap. *International Journal of Advanced Computer Science and Applications (IJACSA)*, *10*(4).