INTEGRATING SOFTWARE ISSUE TRACKING AND TRACEABILITY MODELS

INTEGRATING SOFTWARE ISSUE TRACKING AND TRACEABILITY MODELS

BY

NAVEEN GANESH MURALIDHARAN, B.Tech

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

Master of Applied Science (2022) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Integrating Software Issue Tracking and Traceability Models
AUTHOR:	Naveen Ganesh Muralidharan B.Tech. (Electronics & Communication Engineering), SASTRA University, Thanjavur, India
SUPERVISOR:	Dr. Vera Pantelic & Dr. Richard Paige
NUMBER OF PAGES:	xii, 82

Abstract

Awareness of the importance of systems and software traceability and tool support for traceability have improved over the years. However, an effective traceability solution must align with an organization's engineering processes. Specifically, the phases of the traceability process model (traceability strategy, creation, use and maintenance of traceability) must be aligned with the organization's engineering processes. Previous research has discussed the benefits of integrating traceability into the configuration management process. This research proposes a process based on Model-Driven Engineering (MDE) where traceability is integrated into Change Request (CR) management. In this process, traceability is managed with the status of the CR; for example, traceability is used to report the progress of the implementation of a CR. The proof of concept of the integrated process is demonstrated with a tool, Traceability Centric Issue Tracking System (TraceITS), based on Eclipse Modelling Tools and Epsilon model management framework. A preliminary evaluation of TraceITS indicates several advantages over regular ITS.

Acknowledgements

I sincerely thank Dr. Vera Pantelic, Dr. Victor Bandur and Dr. Richard Paige for their valuable guidance and support in creating this thesis.

I thank the McMaster Centre for Software Certification (McSCert) for providing me with an opportunity to assist with their research that allowed for the inception of the idea presented in this thesis.

Last but not least, I thank all the faculty and staff in the department of Computing and Software at McMaster University for all their support and guidance, especially for students like me who began their Master's degrees during the COVID-19 pandemic.

Contents

List of Figures

List of Tables

Definitions, and Abbreviations

Definitions

Artifact Any item relevant to the project, such as template, document, output, result or project deliverable. (Project Management Institute, 2021)

Baseline "Approved version of a work product or plan" (Project Management Institute, 2021)

Configuration Item
Any item, atomic or aggregate (treated as a single entity) in the project subjected to configuration management process (IEE, 2017)

Change Request
"A formal proposal to modify a document, deliverable, or baseline" (Project Management Institute, 2021)

Traceability "The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to

one another; for example, the degree to which the requirements and design of a given software component match" (IEE, 1990).

Traceability Data

Refers to the traceability information of a project. It can be represented as graphs, hyperlinks and matrices.

Trace Granularity

The level of detail of the artifacts in traceability data. For example, traceability to a Java source could be to a package or a class itself. (Gotel *et al.*, 2012a)

Traceability Information Model

A Traceability Information Model (TIM) is a schema for the traceability data; it describes the structure of the project's traceability data (Mader *et al.*, 2009).

Traceability Process Model

A set of activities that govern the lifecycle of traceability data.

Traceability Recovery

The process of creating traceability links after the artifacts are created. (Gotel *et al.*, 2012a)

Abbreviations

ALM Application Lifecycle Management

CASE Computer Aided Software Engineering

CI	Configuration Item
CIA	Change Impact Analysis
CMMI	Capability Maturity Model Integration
CR	Change Request
DSL	Domain Specific Language
ECL	Epsilon Comparison Language
EGL	Epsilon Generator Language
EMC	Epsilon Model Connectivity
EMF	Eclipse Modeling Framework
EML	Epsilon Merging Language
EOL	Epsilon Object Language
ETL	Epsilon Transformation Language
GMF	Graphical Modeling Framework
GTM	Global Traceability Model
GUI	Graphical User Interface
IEEE	Institute of Electrical and Electronics Engineers
IR	Information Retrieval
ITS	Issue Tracking Systems

LTM	Local Traceability Model
LOE	Level Of Effort
MDE	Model-Driven Engineering
ML	Machine Learning
PMI	Project Management Institute
PoC	Proof of Concept
RTM	Requirement Traceability Matrix
RUP	Rational Unified Process
SCM	Software Configuration Management
TIM	Traceability Information Model
UID	Unique Identifier
VCS	Version Control System
XMI	XML Metadata Interchange

Chapter 1

Introduction

The IEEE Standard Glossary of Software Engineering Terminology defines traceability as "the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another; for example, the degree to which the requirements and design of a given software component match" (IEE, 1990). Traceability data can be represented in various ways, namely, matrices, graphs and hyperlinks (Li and Maalej, 2012). The Requirements Traceability Matrix (RTM) is the most common representation form. An example RTM is depicted in Figure 1.1. The rows contain the IDs of test cases, and the columns contain the requirement IDs. The X in cells denote that that test case verifies the corresponding requirement.

There are numerous benefits of implementing traceability. Studies have indicated that software projects with well-defined traceability are of better quality than projects with insufficient/poor quality traceability (Rempel and Mäder, 2017; Mäder and Egyed, 2015). Particularly, projects with well-defined traceability have lesser defects (Rempel and Mäder, 2017) and are easier to maintain (Mäder and Egyed,

Requirement identifiers	Reqs tested	REQ1 UC 1.1	REQ1 UC 1.2	REQ1 UC 1.3	REQ1 UC 2.1	REQ1 UC 2.2	REQ1 UC 2.3.1	REQ1 UC 2.3.2	REQ1 UC 2.3.3	REQ1 UC 2.4	REQ1 UC 3.1	REQ1 UC 3.2	REQ1 TECH 1.1	REQ1 TECH 1.2	REQ1 TECH 1.3
Test cases	321	3	2	3	1	1	1	1	1	1	2	3	1	1	1
Tested implicitly	77														
1.1.1	1	x													
1.1.2	2		x	x											
1.1.3	2	x											x		
1.1.4	1			x											
1.1.5	2	x												x	
1.1.6	1		x												
1.1.7	1			x											
1.2.1	2				x		x								
1.2.2	2					x		x							
1.2.3	2								x	x					
1.3.1	1										x				
1.3.2	1										x				
1.3.3	1											x			
1.3.4	1											x			
1.3.5	1											x			
etc....															
5.6.2	1														x

Figure 1.1: An example traceability matrix (Wikipedia contributors, 2022)

2015). The specific benefits of traceability that leads to high-quality software are summarized by Berczuk et al. (Berczuk *et al.*, 2005) (also see (Cleland-Huang, 2012)) as follows.

- *Change Impact Analysis:* Traceability is used to assess the impact of an incoming change to the software as the traceability establishes all the necessary relationships among artifacts, such as the dependencies of an artifact.

- *Product Conformance:* From traceability, it can be confirmed that all the customer requirements are implemented in the software.

- *Process Compliance:* From traceability, it can be confirmed that all the processes specified by the organization or an appropriate safety-critical standard are followed.
- *Project accountability:* From traceability, it can be ensured that only the requested functionality has been implemented, and there are no "extra features" or "gold plating".
- *Baseline reproducibility:* Traceability helps with recreating a previous version (baseline) of a software.
- *Organizational learning:* Traceability can be used as a means of knowledge transfer to new engineers.

Consequently, software engineering standards recognize and, in some instances, mandate traceability. For instance, the Capability Maturity Model Integration (CMMI) (CMMI Institute, 2022) uses a process model to recognize an organization's engineering process on a scale of *Initial* (Level 1) to *Optimizing* (Level 5). In the CMMI process model, bi-directional traceability (from all the requirements) is one of the requirements for a Level 2 (*Managed*) recognition of the process. Likewise, in safety-critical standards such as the DO-178C (RTCA, 2011), traceability data is one of the output deliverables, and therefore there are specific requirements for traceability in DO-178C.

Despite the benefits and the recognition of traceability, there are several challenges in implementing traceability. Studies (Mäder *et al.*, 2013; Cleland-Huang *et al.*, 2014; Maro *et al.*, 2020) have indicated a wide variety of challenges ranging from organizations not recognizing the benefits of traceability (Cleland-Huang *et al.*, 2014; Mäder

et al., 2013) to the nuanced complexities faced by organizations implementing traceability (Maro *et al.*, 2020). An example of the latter is the interoperability of the traceability data among subteams in an organization working on different functionalities for the same product.

Among the challenges with traceability is an efficient implementation of the traceability process in an organization. For several reasons, such as a tight project schedule, traceability, including its use, is neglected by an organization's stakeholders. For example, Mäder et al. (Mäder *et al.*, 2013) studied ten deliverable artifacts submitted by various organizations to the US Food and Drug Administration for approval of their respective products. In their study, Mäder et al. found several cases where the traceability in deliverable artifacts appeared to have been created in an ad-hoc manner, likely right before submitting for approval. Likewise, in a recently published study on introducing a traceability process alongside an existing engineering process (Maro *et al.*, 2020), Maro et al. suggest that based on their observations, it could be challenging for organizations to adopt a new traceability process alongside their existing engineering process. Therefore, it is logical to conclude that the traceability process should co-exist with the organization's established engineering processes for an efficient adaptation. This conclusion fits well with the vision of "Ubiquitous Traceability" proposed by Gotel et al. (Gotel *et al.*, 2012b), where traceability is an outcome of an organization's engineering process. In other words, traceability should not be an additional activity in the software engineering process; instead, traceability should result from the process.

Traceability is already integral to the Software Configuration Management (SCM) process. For example, Section 9.2.5.1 in IEEE-828-2012 states that the traceability

data is also an artifact that should be verified as part of the software verification process. Therefore the question is, despite this, why is not traceability a mainstream SCM process like version control? One answer is that there is more to traceability than just creating and using the traceability data. A traceability process model (also known as methodology) is a set of activities that govern the lifecycle of the traceability data. For example, the generic traceability process model (Gotel *et al.*, 2012a) involves strategizing, creating, using and maintaining traceability data. All the activities in the traceability process model must be integrated with the software engineering process to implement traceability efficiently.

The COVID-19 pandemic is an additional motivation for this research. COVID-19 led to a change in the organizational work culture; organizations allow their personnel to work remotely. However, organizations are observing an increased amount of personnel leaving their positions (Microsoft, 2021). This phenomenon is popularly known as the "Great Resignation" or the "Big Quit" (Curtis, 2021). The adverse impact on the project cost and schedules due to the rapid change in personnel is evident.

One of the mitigating mechanisms in episodes such as the "Great Resignation" is traceability. Cleland-Huang et al. summarized a relevant use case in a previous study (Cleland-Huang *et al.*, 2014). They describe how preexisting traceability data would help a new developer in an Agile team implement a user story. The traceability data will help the developer with necessary information like the upstream/downstream artifacts of the user story and the information about a developer who has previously worked on the user story. Therefore, this research would provide a renewed interest for organizations to improve their traceability process.

Last but not least, portions of this thesis are verbatim from the paper, *Integrating Software Issue Tracking and Traceability Models* (Muralidharan *et al.*, 2022). The idea presented in this thesis was submitted to the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME)[1], and has been accepted for presentation and publication.

1.1 Approach

The goal of this research is to integrate all the activities of a traceability process model into the software engineering process so that traceability can no longer be seen as an additional task by an organization. This approach to attain this goal has three elements: the traceability process model, the engineering process and tool support & automation.

- Traceability process model: This research uses the generic traceability process model (Gotel *et al.*, 2012a) as it is simple yet comprehensive to integrate with an engineering process.

- Engineering process: With which engineering process the traceability process can be smoothly integrated? This research proposes a specific aspect of SCM: Change Request (CR) management. Specifically, this research proposes integrating the traceability process model with the lifecycle of CRs. The reason is that CRs are relevant to all the stakeholders in a project, and CRs are typically the starting point for any change (for example, new development or bug fix).

- Tool support and automation: What tool support should be provided to the

[1]https://cyprusconferences.org/icsme2022/new-ideas-and-emerging-results-accepted-papers/

organizations to successfully implement the integrated process? This research proposes a Model-Driven Engineering (MDE) tool suite. The reason is the availability of integrated visualization and automation in MDE. It was previously mentioned that traceability data could be represented as matrices, graphs or hyperlinks. Each representation has its benefits (Li and Maalej, 2012). An added benefit of representing traceability as models is that visualization, storage and management are augmented by the appropriate MDE tool suite. For example, when Eclipse modelling tools such as Eclipse Modelling Framework (EMF) (Eclipse Foundation, 2022a) and Graphical Modelling Framework (GMF) (Eclipse Foundation, 2022b) are chosen to represent traceability, traceability can be created and visualized using the graphical syntax editor provided by GMF and the traceability models are stored in XML Metadata Interchange (XMI) standard files. The model management (therefore traceability management) can be automated with a model management framework such as Epsilon (Epsilon Development Team, 2022b). Augmentation with an MDE suite would significantly reduce the organization's traceability development and maintenance efforts.

1.2 Contributions

The following are the contributions of this research.

1. A process based on Model-Driven Engineering (MDE) is proposed where traceability strategizing, creation, use and maintenance are integrated with the lifecycle of CRs. Developers on a project are encouraged to use and update traceability data, but rigid guidelines usually don't exist on when to use or update them and how. As a result, the developers may neglect traceability until the last moment when they are ready to close their CRs. Integrating traceability management with the state of a CR positions traceability at the forefront of the change management process. Augmented with MDE, the proposed process ensures the appropriate visualization and automation to engage the developers throughout the process. Currently, no process combines CRs and MDE for traceability management.

2. A tool, Traceability-centric Issue Tracking System (TraceITS), is developed to demonstrate the integrated traceability and CR process. TraceITS demonstrates that it is practical to implement the integrated process in practice and that it can be scaled and customized to an organization's needs. With TraceITS, the benefits of an MDE-based ITS over a regular ITS are discernible and pave the way for discussions on future ITS.

1.3 Thesis Outline

Chapter 2 of this thesis outlines the necessary background and related work; Chapter 3 explains the integrated process; Chapter 4 explains a Proof of Concept (POC) tool, TraceITS; Chapter 5 presents a discussion of the process and the tool, and finally Chapter 6 concludes the thesis.

Chapter 2

Background and Literature Review

This chapter explains the necessary background and related work regarding the integrated traceability and CR management process presented in this thesis. The integrated CR and traceability management process proposed in this thesis will be referred to as the *integrated process* throughout this chapter.

2.1 Background

This section provides the necessary background for the integrated process. Subsection 2.1.1 explains the generic traceability process model, Subsection 2.1.2 explains the CR management and lifecycle, and Subsection 2.1.3 explains Traceability Information Model (TIM).

2.1.1 Generic Traceability Process Model

The generic traceability process model proposed by Gotel et al. (Gotel *et al.*, 2012a) is depicted in Figure 2.1.

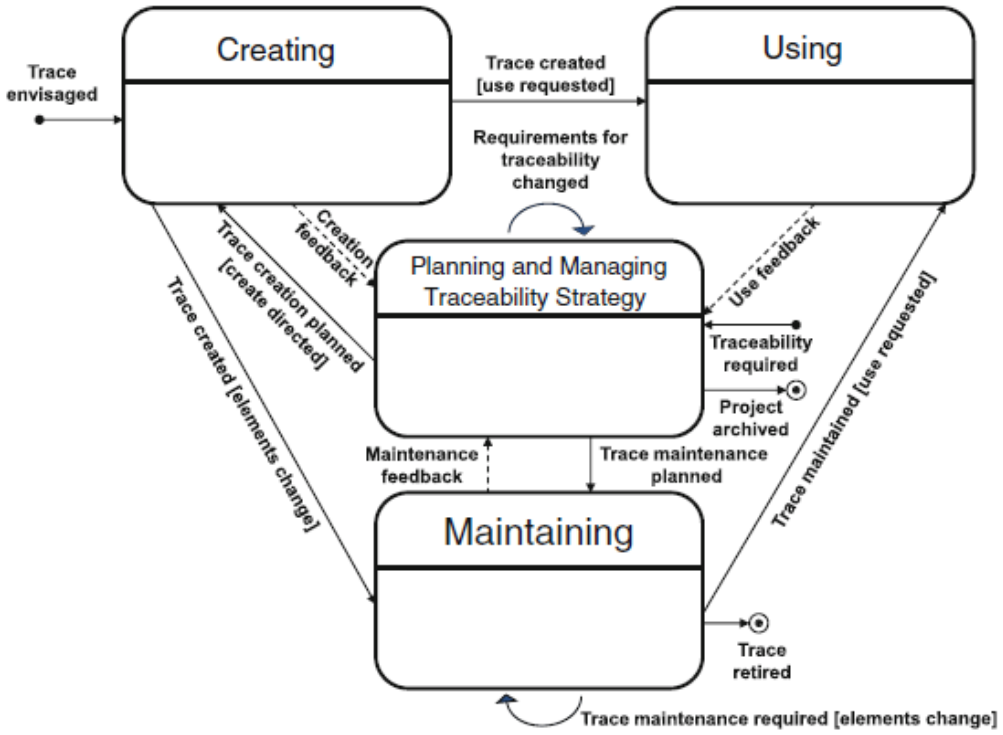

Figure 2.1: Generic Traceability Process Model (Gotel *et al.*, 2012a)

The first phase in Gotel et al.'s traceability process model is *Traceability Strategy*. *Traceability Strategy* involves the following steps. The first step is establishing the need for traceability that caters to the needs of all project stakeholders. For instance, a project manager or other lead would use traceability for resource allocation, whereas the developer would primarily use traceability to understand the technical scope of the proposed changes. In addition, there may be pre-requisite traceability requirements for safety-critical software prescribed by the standards such as DO-178C (RTCA, 2011). Accordingly, the relevant artifacts to be traced and the appropriate level of trace granularity must be chosen to satisfy the stakeholders' needs. The second step is project management, budgeting the costs for traceability, allocating resources, etc.

The third step is planning the traceability data, which involves designing the traceability using a Traceability Information Model (TIM) (Mader *et al.*, 2009), establishing a traceability process, and determining the appropriate tool support to augment the traceability process. The last steps in traceability strategy are implementing and continuously monitoring and evaluating traceability through user feedback.

Traceability Strategy is followed by *Traceability Creation*. As the name suggests, traceability creation involves creating and validating traceability data. There are two methods of creating traceability. Traceability can be created during the development of the artifacts or after development. The latter is called Traceability recovery (Section 2.2.3).

The next phase is *Traceability Maintenance*. Traceability Maintenance is the process of maintaining the established traceability data: specifically, managing changes to traceability during the system's evolution. For instance, if a new artifact is added to the TIM, the traceability data should be updated to conform to the new TIM.

The final phase in the generic traceability process model is Traceability Use. As mentioned previously, the usage of traceability data differs among stakeholders. Therefore, factors to consider during Traceability use are the presentation of the traceability data to users and the traceability queries posed by users. As mentioned in the Traceability Strategy, the usefulness of the traceability data should be continually evaluated and improved based on user feedback. Although not part of the process model, another step to consider is Traceability Visualization: presenting the traceability data to all the stakeholders in a dashboard. Studies (Mäder *et al.*, 2013; Cleland-Huang *et al.*, 2014) have indicated that traceability dashboards aid with traceability creation, maintenance and planning by eliciting user engagement

through use and feedback. Also, the visualization instills trust in the traceability data.

2.1.2 Change Request Forms

Change Request (CR) forms are essential to the Software Configuration Management (SCM) process. IEEE 828-2012 (IEEE, 2012) configuration management standard contains specific requirements for CRs (section 9.2.3) in its low-level change control process (section 9.2). IEEE-828-2012 states that a CR form, at a minimum, contains the following information:

- Description of the change, including the rationale for the request
- Status of the CR (for example, Open, Approved, Rejected, Closed)
- Affected software version (baseline)
- Impact on the product
- Resolution notes
- CCB (Change Control Board) approval notes

Among all the artifacts in a project, one salient characteristic of CRs is that all the stakeholders access them. For example, take the case where a customer creates a CR for a new feature. First, the appropriate lead (*e.g.* a product owner) performs a Change Impact Analysis on the CR. The CR is then forwarded to the Change Control Board (CCB) for approval. The CR is then assigned to a developer for implementation and verification upon approval. During the implementation, all the stakeholders track

the progress of the CR. Finally, the closed CR may also be subjected to audit by the quality assurance team per the low-level configuration audit process specified by IEEE-828-2012, Section 11. During the CR's lifecycle, the status field in the CR is updated to indicate the activity on the CR, for example, Open or Closed.

Although traceability is an essential part of IEEE-828-2012, the status of CRs is not directly related to traceability. Summarily, there are two types of traceability requirements in IEEE-828-2012. The first is to inspect the traceability among the Configuration Items (CI), for example, traceability from requirements to design and requirements to tests (IEEE-828-2012 Section 11.2.1.1). The second is to inspect the traceability between any change to a baseline and the driving CR (IEEE-828-2012 Section 8.2.5.7). Note that the word *inspect* is taken verbatim from IEEE-828-2012; however, *inspect* does not imply verification of requirements through inspection, but to inspect if a requirement is appropriately verified by examining the traceability data.

As a reminder to the reader, the integrated process integrates the generic traceability process model with the lifecycle of a CR. However, a few noteworthy points on CRs need to be clarified before explaining the integrated process. First, the type of CR that stakeholders have access to varies. For instance, the customer may use an IT service desk management suite to report a bug or request a new feature. Internal to an organization may be Issue Tracking Systems (ITS) to track the bug's status. Additionally, ITS may support different types of CRs, for instance, a bug versus a new feature (Atlassian Inc., 2022c). The scope of the integrated process is limited to ITS and CR types that are internal to the organization.

Next, the low-level change control requirements specified in IEEE-828-2012 apply

to CIs that are already baselined. IEEE-828-2012 explicitly states that a low-level change control requirement "*does not apply to items which have not yet been submitted to a CM repository for the first time and are still under development.*" (IEEE, 2012). Therefore, the process for new development may vary across organizations; that is, organizations may or may not create tickets for creating new artifacts (i.e. not baselined), typically at the beginning of their software development process.

2.1.3 Traceability Information Model (TIM)

A Traceability Information Model (TIM) is a schema for the traceability data; it describes the structure of the project's traceability data (Mader *et al.*, 2009). A TIM contains artifacts and traceability links between artifacts. An example TIM by Mader et al. is depicted in Figure 2.2.

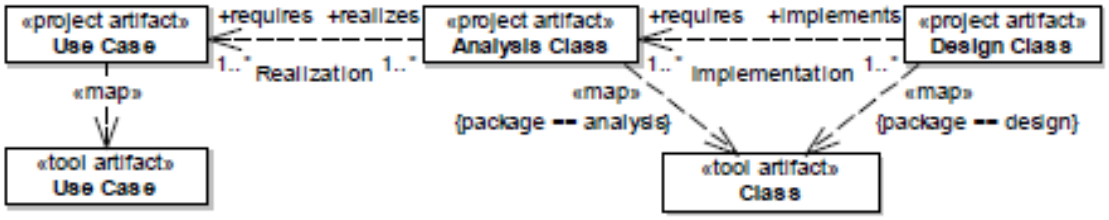

Figure 2.2: Example TIM (Mader *et al.*, 2009)

When the traceability data is represented as graph models, the TIM is the corresponding metamodel.

2.2 Literature Review

The outline of this section is as follows. First, Subsection 2.2.1 discusses related work on integrated configuration management and traceability processes. Next, Subsection 2.2.2 discusses related work traceability methodologies/strategies. Third, Subsection 2.2.3 discusses related work on automated traceability recovery techniques. Finally, Subsection 2.2.4 discusses related work on the traceability processes in specific ALM suites.

2.2.1 Traceability Integrated with SCM Process

Mohan et al. (Mohan *et al.*, 2008b,a) integrated Traceability management into the Software Configuration Management (SCM) process specified in the Rational Unified Process (RUP). The integrated process from their work is depicted in detail in Figure 2.3.

The summary of the process is as follows,

- Integrate SCM and traceability plans
- Integrate SCM and traceability tools
- Integrate SCM versioning (artifact Version Control and CR) with traceability data to help manage product baseline and obtain information specific to CRs, like the rationale behind the change and the approvals.
- Monitor and report status corresponding to specific product versions; design decisions and other rationales can now be discerned to specific versions of artifacts.

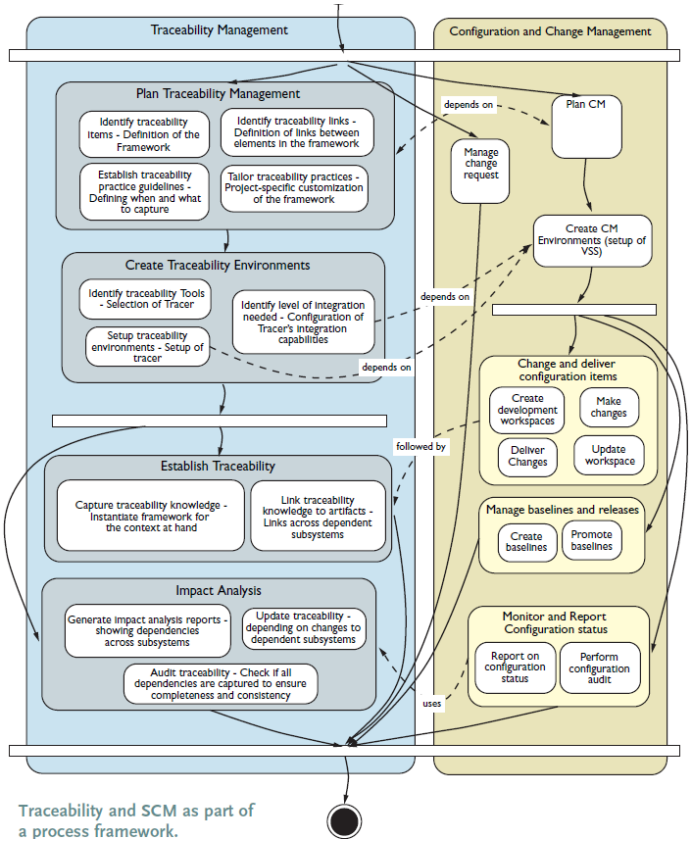

Figure 2.3: SCM process outlined by RUP integrated with traceability (Mohan *et al.*, 2008b)

There are a few similarities between the work of Mohan et al. and the integrated process proposed in this thesis. The similarities are linking CRs with artifacts and integrating the SCM and traceability work environments. However, the main difference between the two processes is the usage of CRs. The specific role of CRs in Mohan et al.'s work appears to be for baseline control and change rationale; neither CRs' state nor CRs' fields relate directly to the traceability data.

In this proposed process, the traceability data is managed with the CR's lifecycle; the state of the CR is tied to strategizing, creating, maintaining and using traceability. Specifically, the CR ticket is extended to hold a portion of traceability specific to that

change and is managed with the lifecycle of the CR.

Appleton et al. (Appleton *et al.*, 2007) (also see (Cleland-Huang, 2012)) propose the idea of "Lean Traceability". They claim that integration of software development processes, namely, Task-Based Development (TBD), Behavior-Driven Development (BDD), Test-Driven Development (TDD), along with an integration of the ITS, VCS, and the Wikis, would result in Event-Based Traceability (EBT) (Cleland-Huang *et al.*, 2003). In EBT, traceability is automatically captured from event notifications. Example event notifications include commits to a file or changes to requirements. In the context of Agile software development, a user story, including its development and verification, is assigned to one ticket in an ITS. Traceability for that user story is then automatically captured from the event notifications. The similarity between Appleton et al.'s process and the proposed integrated process is the relation between the CR and traceability; the traceability data specific to a CR is captured during the progression of the CR using EBT. However, Appleton et al. do not discuss the traceability process model and do not customize CRs for their process.

The advantage of the approach proposed in this thesis over the two aforementioned approaches is in adapting all the phases of the traceability process model into change request management so that traceability is managed along the status of the CR and using CRs for feedback.

2.2.2 Traceability Processes

Traceability process models are step-by-step guidelines for organizations to implement traceability in their projects. The generic traceability process model proposed by Gotel et al. (Gotel *et al.*, 2012a) is already discussed in Section 2.1.1. Elaborate

process models have been developed for a fine-grained implementation of traceability. The steps in a recently published process, TracIMO (Maro *et al.*, 2022), are depicted in Figure 2.4. TracIMO is more detailed than the Generic Process model and contains additional steps for traceability tool support and traceability evaluation.

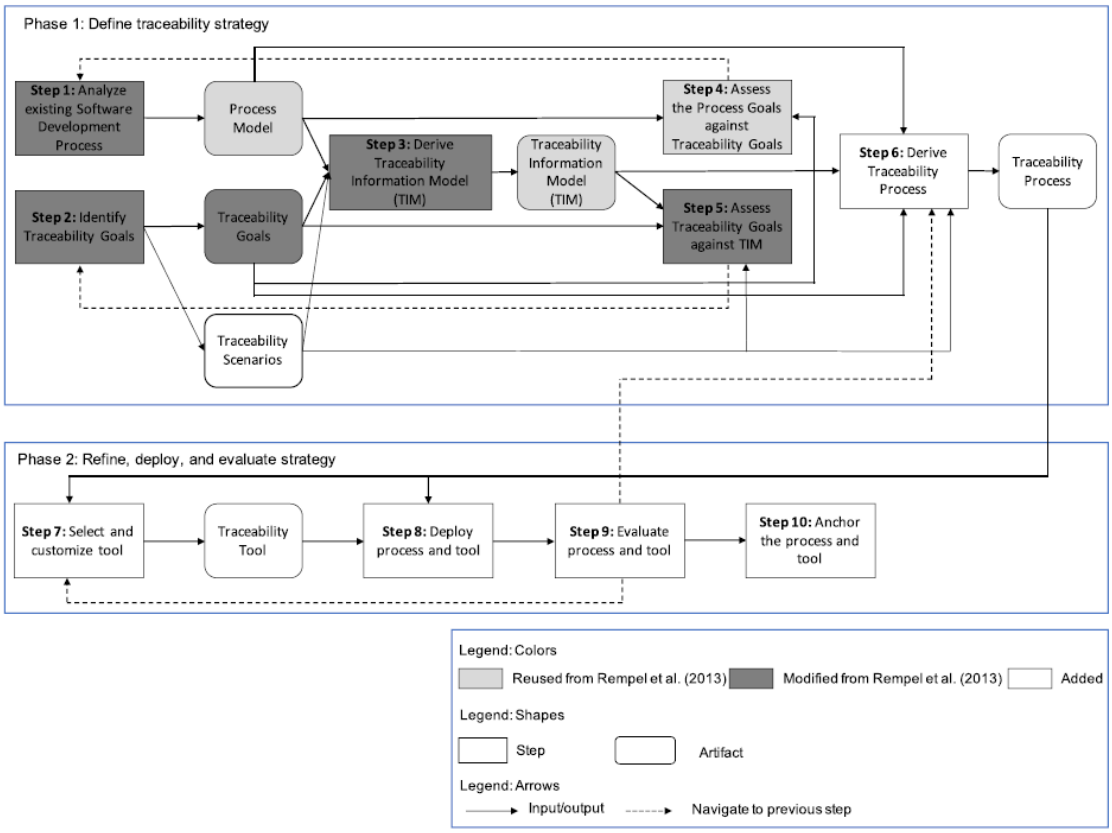

Figure 2.4: Traceability process model proposed by TracIMO (Maro *et al.*, 2022)

The difference between the integrated process and the traceability process models is that the integrated process is specific to a part of the SCM process, whereas the traceability process models do not advocate integrating traceability into any particular part of the software engineering process. The evaluation of the software engineering process to derive the traceability process and choosing the appropriate tool

support come at a later stage in the traceability process models. However, when the process model is applied to an existing engineering process, the outcome may be similar to the integrated process. For instance, in the case study where TracIMO was applied, the resultant traceability process and the Traceability Information Model (TIM) are CR-centric. Other than the similarity that the traceability is initiated from a CR, the traceability process in the case study and the integrated process is different. The most notable difference is the creation of traceability data. In TracIMO's case study, the traceability links are created by the lead developer in a third-party tool which is then converted into a traceability graph diagram and uploaded to the CR for the developer to understand the change. The integrated process proposes the extension of a CR into a graphical syntax editor to hold the traceability called the Local Traceability Model (LTM). The LTMs traceability models are not only constructed by the developer implementing the CR but are also used to report the status of the CR.

In a way, the research in this thesis proposes reverse engineering the traceability process model from the traceability process.

2.2.3 Automated Traceability Recovery Techniques

Automated Traceability Recovery (Lucia *et al.*, 2012; Aung *et al.*, 2020) is the process of creating traceability from existing artifacts through Information Retrieval (IR) or Machine Learning (ML). Automated Traceability Recovery is expected to be among the major research topic of traceability in the next ten years (Antoniol *et al.*, 2017).

Automated Traceability Recovery targetted for Change Impact Analysis attempts

to mine the traceability based on the incoming Change Request. Canfora and Cerulo (Canfora and Cerulo, 2005) use Information Retrieval (IR) techniques to mine artifacts affected by a new CR from the commits made for the previously closed CRs. Likewise, Shahid and Ibrahim's (Shahid and Ibrahim, 2016) approach seem to start from a CR, although it is unclear if their artifact set also includes CR tickets. This poses the question of whether automated traceability recovery from ITSs can be considered as traceability well integrated into the configuration management process. The answer is affirmative, but automated traceability recovery is still a work in progress and may require a manual process to plan and validate the recovered traceability data. The integrated process includes automated traceability recovery, supported only by manual validation by developers.

2.2.4 Traceability Process in ALM Tools

Application Lifecycle Management (ALM) suites are typically developed with an integrated issue tracker. The tickets in the issue trackers of these suites can be linked to artifacts in the development system and visualized on demand. A few ALM suites support phases of the traceability processes model as well. From Steghofer's survey of traceability tools (Steghöfer, 2017), System Weaver (System Weaver Inc., 2022a), Polarion ALM (Siemens Inc, 2022) and PTC Integrity/Windchill (PTC Inc., 2022) offer partial to full support for planning traceability through Traceability Information Models (TIMs).

System Weaver (System Weaver Inc., 2022a,b) is an ALM suite that offers a variety of features, including support for requirements management, system architecture, change management, project management and integration with external tools. In

particular to Change Management, Systems Weaver offers a built-in issue tracking mechanism; however, there is also a provision for integration with external suites like Atlassian JIRA and Git. An issue in System Weaver can be linked to any *items,* including other issues, requirements, or test cases. System Weaver also supports Model-Driven Software Engineering by allowing users to define metamodels for requirements and processes (workflow). Traceability is established in the model instances by linking the *System Weaver ID* of the metamodel elements. Issues do not seem to be directly part of the metamodels or the models. However, issues can be linked to one of the model's items (such as requirements).

Tuleap (Enlean Inc., 2022b,a), Siemens Polarion ALM (Enlean Inc., 2022b,a) and Atlassian JIRA (Atlassian Inc., 2022a,b) support full traceability among artifacts but offer none to limited built-in support for Model-Driven Engineering (MDE). Tuleap does not seem to provide any provision for MDE, while Polarion allows product line management by enabling the definition of feature models. JIRA does not directly support MDSE, but there are extensions (excentia, 2022; Optimizory Technologies Pvt. Ltd., 2022; Köstebek Teknoloji, 2022) in the Atlassian Marketplace that allow the visualization of traceability as model blocks.

In conclusion, it appears that the change management processes of the ITSs found in these suites do not directly depend on traceability data. Additionally, the goal integrated process proposed in this thesis is that it be applicable to all types of ITSs, standalone or ALM.

Chapter 3

Integrated Change Request Management and Traceability Process

This chapter explains the integrated change-request management and traceability process. Section 3.1 describes key elements of the process, while Section 3.2 describes the integrated CR-traceability strategy, creation, maintenance, use and visualization processes. Finally, Section 3.3 summarizes the integrated process with an example.

3.1 Process Fundamentals

This section explains the fundamental elements of the integrated process. Subsection 3.1.1 explains the specialized use of CRs and Subsection 3.1.2 explains the traceability models in the process.

3.1.1 Change Request Ticket

The background about CRs and the IEEE-828-2012's recommendation for the fields in a CR was discussed in Section 2.1.2. This research proposes adding the following fields to a CR:

- Feedback about the TIM and the other fields in the CR itself, *e.g.*, a simple Yes/No question such as "Was the traceability model helpful in making this change? Y/N", or "Do you find this CR too tedious to fill? Y/N".
- General Retrospectives: Lessons learned from implementing the ticket.
- Documentation that is not a deliverable artifact but is useful to understand the system. Examples include internal training slides or specific sections in a large document.
- Any miscellaneous information that helps with Change Impact Analysis: for example, any risks and opportunities when modifying an artifact affected by that change.
- Rationale in the case of merge conflicts between LTMs and GTMs (explained in Section 3.2.3).

These additional fields in a CR can be mined, aggregated and presented to the user to help them with change impact analysis, among other use cases.

This proposal can be accomplished using Model-Driven Engineering by linking every artifact in the traceability model with the relevant CRs, and, using model querying to mine and summarize the added fields in CRs. The process is depicted in

Figure 3.1, where it can be observed that the CRs are analogous to "flash cards" for an artifact.

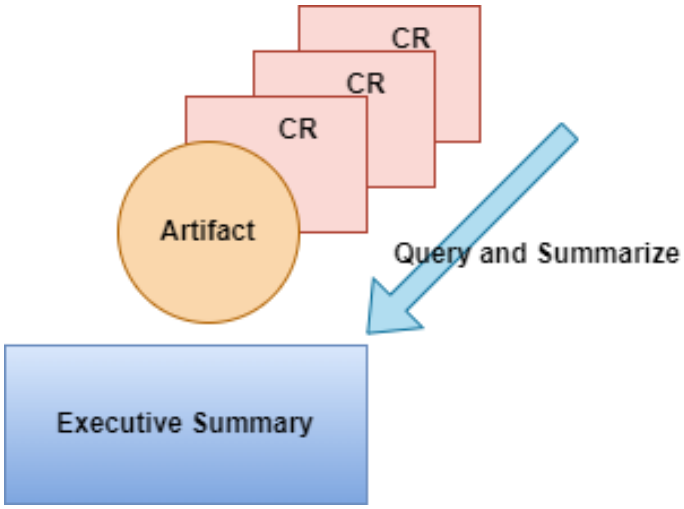

Figure 3.1: Information in CRs is queried and summarized for a Stakeholder

3.1.2 Traceability Models

This research proposes two types of traceability models: *Global Traceability Model (GTM)* and *Local Traceability Model (LTM)*.

The GTM is a megamodel (Bézivin *et al.*, 2004) representing the traceability of the entire project. GTM can be used for Change Impact Analysis (CIA), storing traceability, detecting changes to the traceability, and auditing the traceability.

The LTM is a copy of a subgraph of the GTM, representing the traceability data specific to a CR. The purpose of the LTM is for a developer to understand the scope of the change using traceability, validate the traceability, and report progress on the change using traceability.

3.2 Integrated Process

This section proceeds to explain the integrated process. Each subheading represents an amalgamation of CR and traceability processes.

3.2.1 Strategizing (Planning the CRs and Traceability for the New Software Baseline)

The CRs intended to be implemented for a new software baseline are typically planned in a software engineering project. For example, during sprint planning in Agile software development. Section 3.1.1 discussed adding additional fields to a CR. This research proposes that the TIM and the fields in a CR ticket be reviewed before each baseline to best suit the needs and characteristics of an organization and a baseline.

One of the additional fields in Section 3.1.1 is the *Feedback about the TIM and the CR*; This field can be used to decide if the TIM needs to be modified. This field can be aggregated from the closed CRs of the latest delivered software baseline and reviewed by the stakeholders. If the structure of the TIM is updated, then the GTM should also be updated for the current and subsequent software baselines. Model propagation operations (for example, using Epsilon Folk (Epsilon Development Team, 2022b)) can help automate the GTM update.

3.2.2 Creating (Creating and Validating Traceability During CR Implementation)

A change requested by a CR could result in new artifacts and modification of existing artifacts. If the CR is for new development, then LTM in the CR should be created

from scratch. LTM creation can be automated in several ways. For instance, automated traceability recovery (Lucia *et al.*, 2012; Aung *et al.*, 2020), or if traceability links already exist in other tools, they can be mined to the LTM (for example, using drivers provided by frameworks such as Epsilon (Epsilon Development Team, 2022b)), or EBT (Cleland-Huang *et al.*, 2003) (already explained in Section 2.2.1). If the CR is for the modification of artifacts, then this research proposes creating the LTM for that CR from the GTM. Portions of the GTM corresponding to that change can be selected and copied to the CR as LTM.

The traceability links are invalidated when the CR is assigned to a developer for implementation. The developer reports the artifact's progress (see Subsection *Visualizing*) and re-validates the traceability links while implementing the changes.

3.2.3 Maintaining (Traceability Maintenance as CR is Closed)

After the change is implemented and the CR is ready to be closed, the developer will complete the fields in the CR and close the CR. When the CR is closed, the LTM in the CR is merged back to the GTM. Automated traceability maintenance algorithms (e.g. (Mäder *et al.*, 2009)) can be used to merge the LTMs back to the GTM; however, this research proposes an automated merging only if there is no inconsistency between LTM and GTM. Any inconsistency during the merge should be resolved manually, accompanied by a rationale. The reason for the manual resolution is that the inconsistency could be a traceability error; for instance, a developer removing a traceability link in the LTM. Overlapping changes can also be identified during this process, that is, multiple CRs affecting the same artifact resulting in conflicting traceability links.

3.2.4 Using (Using Traceability models for Change Impact Analysis and Status Reporting)

This subsection describes the various uses of the GTM and the LTM.

Since the GTM represents the traceability of the project, the Change Impact Analysis for a new CR can be performed by the relevant stakeholder using the GTM. The CIA process can be semi-automated by comparing the fields in a new CR against the fields of the closed CRs of the previous baseline. Then the artifacts affected by the relevant CRs can be automatically selected. Alternatively, the user can select a list of the known artifact affected by the CR in the GTM. The artifacts connected through traceability links are then chosen automatically. The artifacts (and the corresponding traceability links) affected by the new CR in the GTM are now copied to the CR as the CR's Local Traceability Models (LTM). The additional fields in a CR proposed in Subsection *Change Request Forms* may be helpful for the stakeholder during the impact analysis. For instance, the actual work hours in a CR can be aggregated and summarized from the previously closed CR and displayed to the user as "On average, the last three changes involving this artifact took 300 work hours."

After completing the CIA, the CR is forwarded to the Change Control Board (CCB) for approval. If the CCB rejects the CR, the artifact-CR link in the GTM is preserved for subsequent CIAs. Otherwise, the CR is assigned to a developer for implementation.

The LTM in a CR can be used for a couple of purposes. First, the LTM helps a developer understand the scope of the CR. Second, this research proposes using the LTM to report the implementation status in a scrum or other team meetings. Reporting the status of the implementation of a CR using the LTM not only enables

a precise and "quantified" status update but is also another opportunity for the developer to engage with traceability models. The specifics of status reporting are explained in the following section.

3.2.5 Visualizing (Traceability Visualization During CR Lifecycle)

This subsection describes the presentation of the traceability models to the user. The subsection first describes the syntax of the traceability models and then discusses the presentation of the traceability models in a dashboard.

Syntax of the Traceability Models

Abstract Syntax of the Traceability Models

TIM (see Section 2.1.3) is the metamodel for the GTM and LTMs. Although the TIM varies according to the traceability requirements of an organization, at the core of a TIM is an artifact and a traceability link. In addition to standard attributes like the artifact ID, this research proposes the following to the *Artifact* class.

- Section 3.2.4 described using the LTM to report the progress of implementation. Therefore a new attribute, *progress*, should be added to indicate the progress of the implementation of the artifact by the developer. This attribute could be an enum or a numerical value. Every artifact in the LTM would have a *progress* attribute, and the progress of the CR is a function of the progress of all the artifacts in the LTM. For instance, the progress of the CR could be the weighted average of the progress of all the artifacts, where the weight of the artifact could

be proportional to the granularity of the artifact. To explain the attribute in detail, the LTM in Figure 3.2 is taken as an example. In this figure, *REQ-1234* denotes the requirement ID, *Module-34* denotes the ID of the design module, and *Class Driver-1* denotes the code. The progress of the requirement and the design is *100%*, and the progress of the code is *0%*.

This means that the developer has completed writing the requirement and the UML design of the requirement; however, they have not yet begun writing the code for the requirement. Assuming every artifact is weighed equally, the progress of the CR, in this case, would be,

$$Progress_{CR} = \frac{\sum_{artifact} Weight_{Artifact} \cdot Progress_{Artifact}}{\sum_{artifact} Weight_{Artifact}}$$

$$\implies \frac{(Weight_{requirement} \cdot Progress_{requirement}) + (Weight_{design} \cdot Progress_{design}) + (Weight_{code} \cdot Progress_{code})}{Weight_{requirement} + Weight_{design} + Weight_{code}}$$

$$\implies \frac{(1 \cdot 100) + (1 \cdot 100) + (1 \cdot 0)}{1+1+1}$$

$$\implies 66.6\%$$

- Since the LTMs are local to a CR, every artifact in the LTM must reference the CR. In the LTM and GTMs, the CRs are not artifacts and therefore do not have traceability links to other CRs. The traceability among the CRs is implemented in the Issue Tracking Systems (ITS), and this research proposes separating the two types of traceability to simplify the TIM.

Concrete Syntax of Traceability Models

The concrete syntax of GTMs and LTMs is depicted in Figure 3.2. The traceability model's goals are to clearly and concisely represent traceability and to effectively use the model to report on the status of a ticket. This research proposes that the artifact be coloured to denote the ticket's status. For example, 0% progress on the

ticket would be indicated by a white-coloured artifact and 100% by a dark shade of a colour, as in Figure 3.2. Also, in Figure 3.2, the artifacts are represented as solid circles with the artifact ID and the artifact progress as the labels. The traceability is indicated with a solid green line (valid link) or a red dotted line (invalid link). The link label indicates the type of traceability link. For example, when a test case verifies a requirement, "verifies" could be the type of traceability link between the requirement and the test case.

Figure 3.2: An example of the graphical syntax of the traceability models

Traceability Dashboard

In a software engineering project, the CRs planned for the current software delivery are typically reviewed in scrum/status meetings with the help of dashboards. Since the CR also holds the LTM, the CR dashboard can also be used to visualize traceability.

The integrated CR-traceability dashboard has two types of views: one to aid with CIA, called the *Change Impact Analysis (CIA) view*, and another to track the progression of the software delivery called the *Build Tracking view*. A mock-up of the two dashboard views is depicted in Figure 3.3 and Figure 3.4, respectively. In these figures, REQ-1234 denotes the requirement ID, Modules-34 denotes the ID of the design module, and TC-7832 denotes the test case ID.

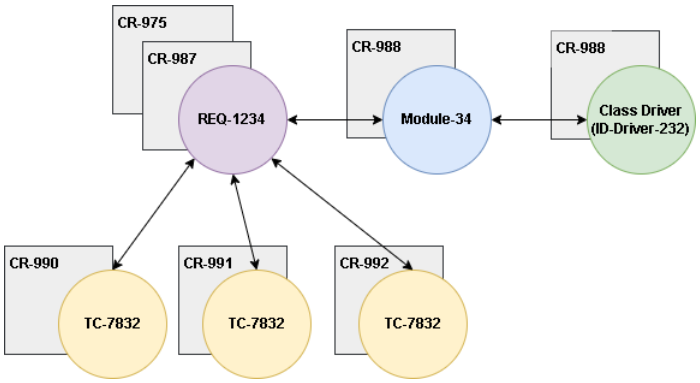

Figure 3.3: A mock-up of the Change Impact Analysis View

The CIA view helps with the activities described in Subsection *Using*. The Build Tracking view helps with a CR's status update and ad-hoc review of the LTM. This view displays all the CRs assigned for a software delivery version and the corresponding LTM in every CR. Reviewing the LTMs could allow other developers to provide ad-hoc feedback on traceability data. For example, during status meetings, senior developers reviewing the LTM assigned to a junior developer can use the opportunity to point out any inconsistencies in traceability.

3.3 Process Summary

Figure 3.5 depicts this process's summary when applied to Agile (Scrum) software development.

3.3.1 Sprint Planning

First, in the sprint planning meeting, all the stakeholders review the aggregated traceability feedback from the CRs of the previous sprint. Based on the feedback, the TIM can be revised, and the GTM updates can be automated using model propagation

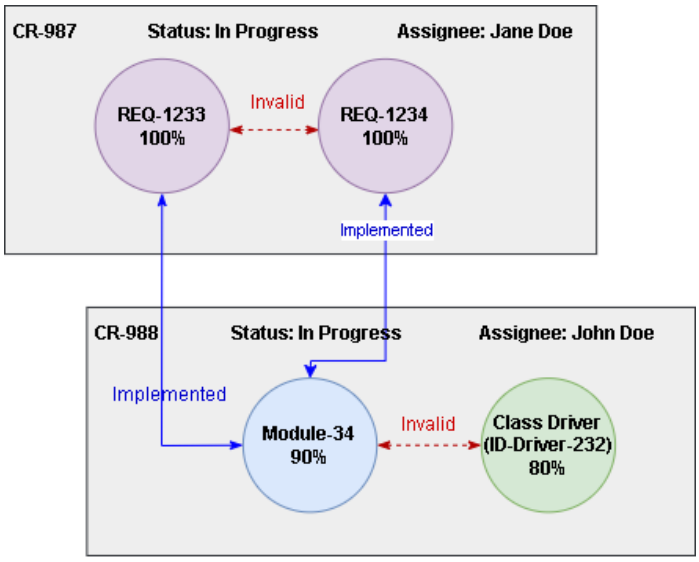

Figure 3.4: A mock-up of the Build Tracking View

operations.

Also, during the sprint planning, the scrum masters and the team can do a cursory review of the LTM and CR feedback (of other fields) of every CR in the sprint backlog and ensure that they aid with implementing the change.

3.3.2 The Sprint

During the sprint, the developer reviews the LTM and the feedback aggregates in the CRs to understand the change and proceeds to implement the change. During the implementation, the developer also validates the traceability links in the CR and updates the progress attribute for every artifact.

The Build Tracking View can be used in the scrum meetings to review the progress of the CRs. After the developer has completed the changes, they change the state of the CR to trigger the automated LTM to GTM merge. If there are any merge conflicts, the developer reviews the reason for the conflict and proceeds to address or

force the merge with a merge rationale.

3.3.3 Sprint Retrospectives

During the sprint retrospective meetings, among other activities, the developers fill the feedback fields of the CRs and close them. Note that the process of closing the tickets varies among organizations. For instance, a CR could be closed during the sprint's progression or moved to an intermediate state, such as "Dispositioned," and closed later.

Finally, in parallel, the Product Owner or the appropriate stakeholder uses the Change Impact Analysis (CIA) view to perform an impact analysis of new tickets.

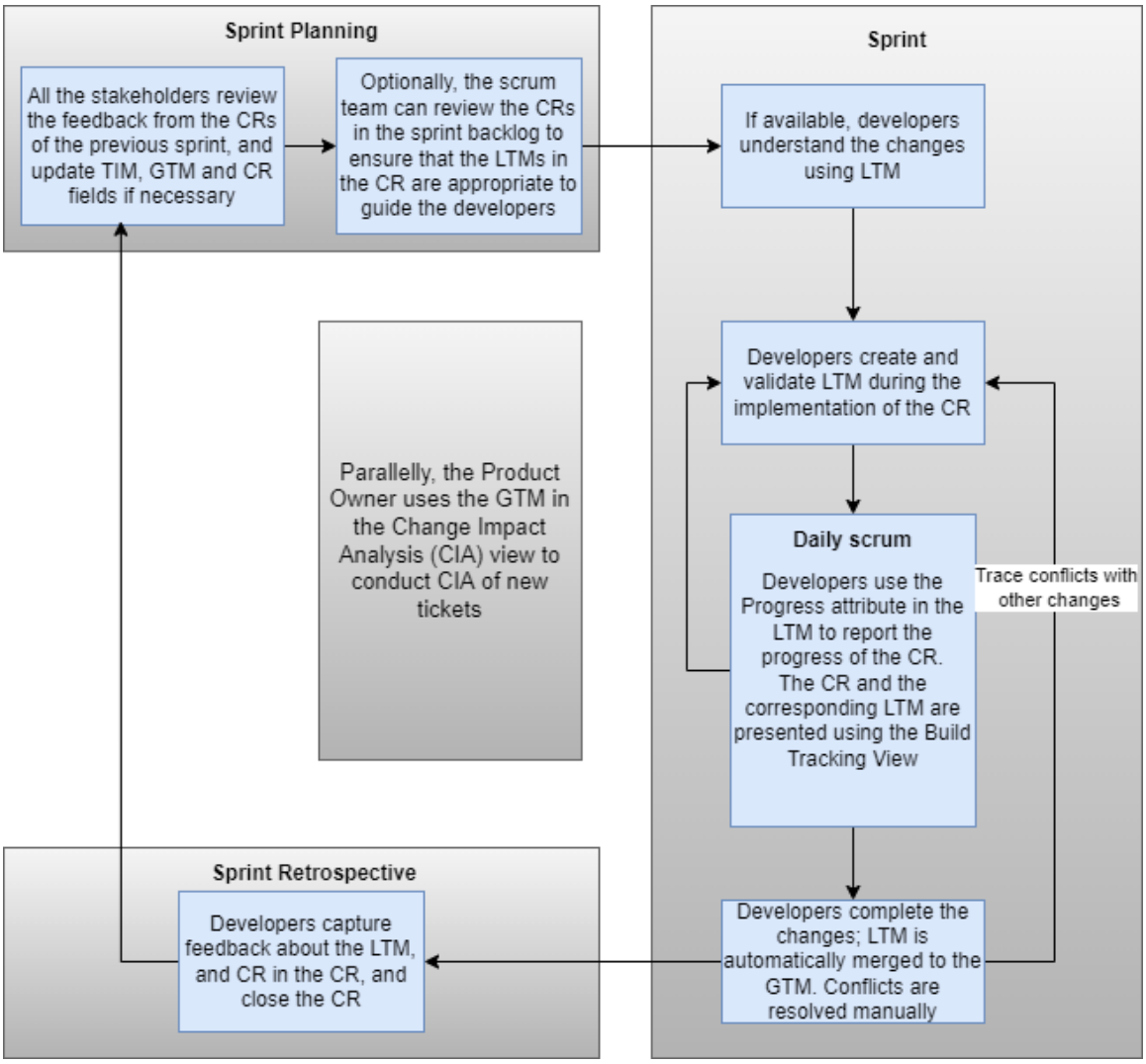

Figure 3.5: Summary of the integrated process when applied to Agile software development

Chapter 4

Traceability-centric Issue Tracking System (TraceITS)

This chapter presents a tool, *Traceability-centric Issue Tracking System (TraceITS)*[1], that demostrates the integrated process from Chapter 3. Section 4.1 explains the architecture of TraceITS, Section 4.2 explains the conformance of TraceITS to the integrated process, and Section 4.3 explains the limitations of TraceITS.

4.1 TraceITS Architecture

The high-level architecture of TraceITS is depicted in Figure 4.1. The following is a summary of the components of TraceITS.

- *Edgewall Trac (Egdewall Inc., 2022)* – Trac is used for CR management. Trac is configured according to the specifications of the integrated process and is

[1] https://github.com/muralidn/TraceITS

connected to Eclipse using a Java application. Section 4.1.1 explains Trac and its configuration in detail.

- *Eclipse Modeling Tools* – The *Eclipse Modeling Framework (EMF) (Eclipse Foundation, 2022a)* is used for metamodelling and *Eclipse Graphical Modeling Framework (GMF) (Eclipse Foundation, 2022b)* is used for defining the concrete syntax of the models and the graphical syntax editor for the models.

- *Epsilon Model Management Framework (Epsilon Development Team, 2022b)* – Epsilon is used for a variety of operations, such as automated generation of the graphical syntax editor, automated model querying, automated model validation, automated model-to-model transformation, and automated mining of traceability links from an example Computer Aided Software Engineering (CASE) tool (explained in Section 4.1.2).

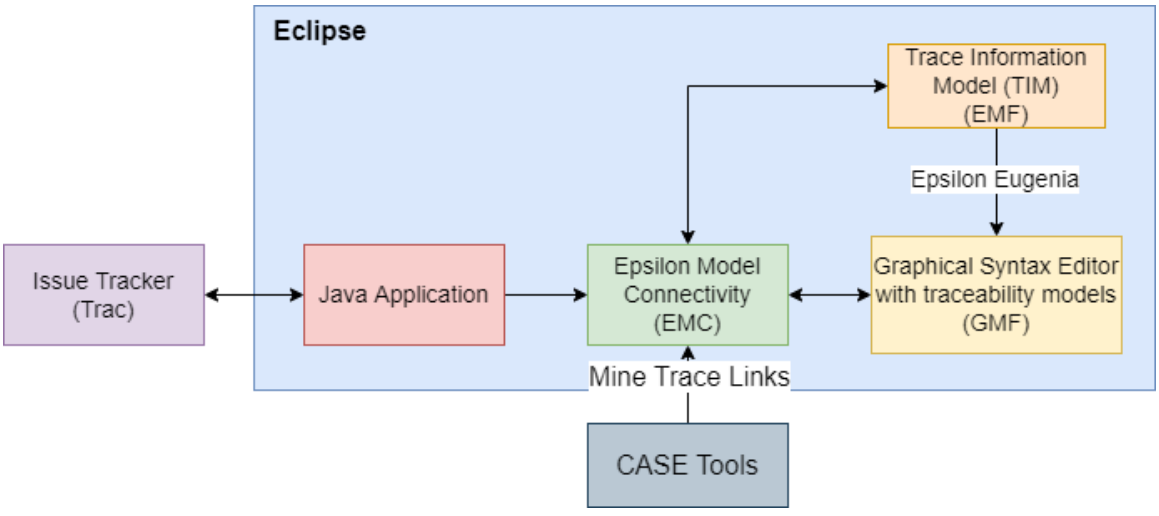

Figure 4.1: Architecture of TraceITS

The following subsections explain the components of TraceITS in detail. Subsection 4.1.1 explains the configuration of Trac, Subsection 4.1.2 explains an example

project using a CASE tool used to demonstrate TraceITS, and Subsection 4.1.3 explains the design of the traceability models and the model management operations.

4.1.1 Trac Configuration

Per the integrated process, additional fields are added to a CR in Trac. The additional fields are depicted in Figure 4.2, and explained in Table 4.1.

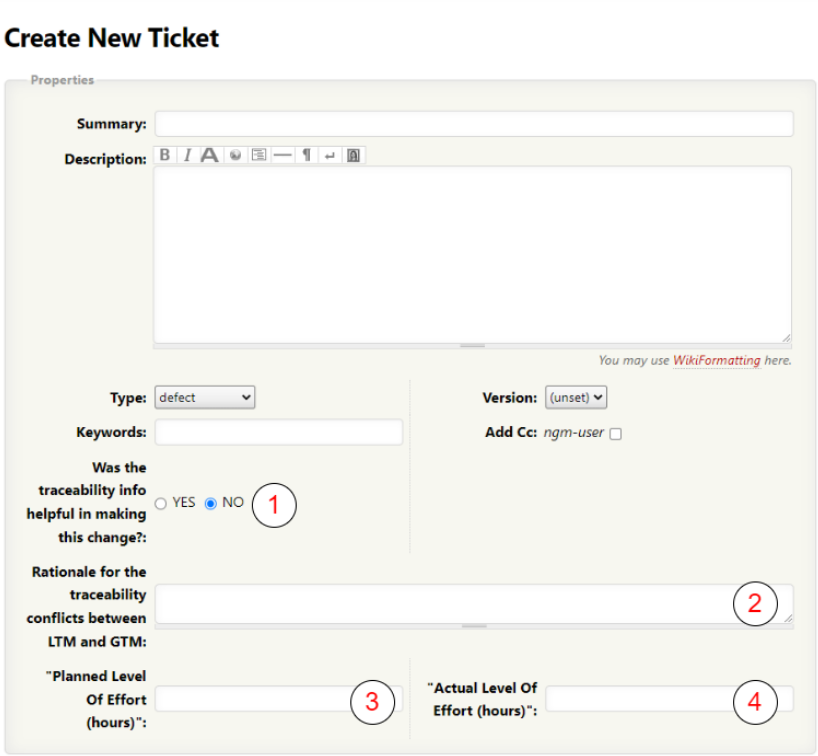

Figure 4.2: Customized CRs fields in Trac

Trac is executed in standalone mode (localhost), and Trac's SQLite database is the interface to Eclipse. A Java application in Eclipse is the bridge between Trac and Epsilon. The application queries from and stores the CRs into Trac's SQLite database and uses Epsilon's Java API to manage the traceability models.

Table 4.1: Customized CR fields in Trac

Field	**Type**	**Description**	**Marker in Fig. 4.2**
Feedback about Traceability	Radio Button (Yes/No)	Field for the developer to indicate if the traceability models and the queried data from CRs were useful	1
Merge Rationale	Multi line text	Field for the developer to explain the merge conflict (if any) when the LTM is merged to the GTM	2
Planned LOE	Single line text	An example field to aid with CIA. This field indicates the man-hours planned for the CR	3
Actual LOE	Single line text	An example field to aid with CIA. This field indicates the actual man-hours it took to implement the CR	4

4.1.2 CASE Tool Example

To demonstrate TraceITS, an example project from Mathworks Simulink-Requirements (Mathworks Inc., 2022b) is used. The project, *Requirements Definition for a Cruise Control Model* (Mathworks Inc., 2022a), demonstrates a requirements-driven Model-Based Design (MBD).

In this example, the systems and functional requirements are defined in Simulink Requirements; The models are in designed in Simulink, and the test cases are defined in Simulink-Test (Mathworks Inc., 2022c).

A simplified TIM of the example is depicted in Figure 4.3. A functional requirement could *derive(s)* from one or more system requirements, a model *implements*, and a test *verifies* one or more functional requirements. Another traceability link,

relates to is ommitted from the proof of concept for simplification.

Also, there are no unit tests in Simulink Test; therefore, there is no direct traceability link between the models and the test cases.

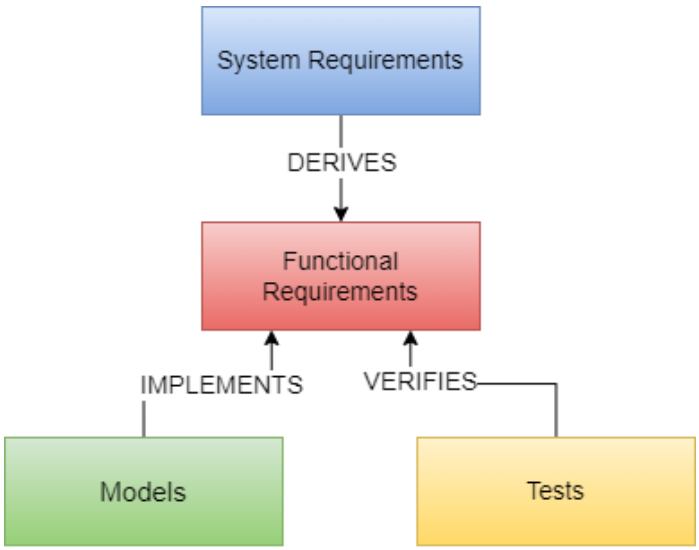

Figure 4.3: Simplified TIM of the Simulink Cruise Control example

4.1.3 Traceability Models

This section explains the syntax of TraceITS's traceability models' and the model management operations.

Abstract Syntax

The TIM of TraceITS is designed according to the TIM of Simulink Cruise Control project from Section 4.1.2 and the specifications in Section 3.1.2.

An abstract class, *Artifact* is defined and the classes *System Requirements*, *Functional Requirements*, *Models* and *Tests* are derived from *Artifact*. The traceability

links between these artifacts are strongly typed: *Derive*, *Implement* and *Verify*. *Derive* traces functional to system requirements, *Implement* traces models to functional requirements and *Verify* traces tests to functional requirements. Additionally, the CR ticket is modelled, and the attributes of the CR models match the fields of the CR tickets in Trac. The Java application described in Section 4.1.1 mines Trac and synchronizes the CR models in Eclipse with its equivalent in Trac. There are also bidirectional references between artifacts and CRs; every artifact references the corresponding CR driving the change, and every CR references a list of artifacts affected by the change. The complete metamodel in TraceITS is depicted in Figure 4.7. Table A.1 in Appendix A, explains the metamodel in detail.

Eclipse Modeling Framework (EMF), Ecore, is used to design the metamodel, and Emfatic (Epsilon Development Team, 2022a) (a meta-metamodel) is used to generate the ECore model.

Concrete Syntax

The Graphical Syntax and the editor are designed using Eclipse GMF. The GMF editor is autogenerated from the Emfatic source file using Epsilon Eugenia (Epsilon Development Team, 2022c). An example traceability model from the tool is depicted in Figure 4.4.

The artifacts are solid ellipses, and the traceability links are bi-directional arrows with the type of traceability link as the label. The artifacts have the artifact ID and the progress as the labels. The artifact ID is the name of the Simulink file, followed by the Unique ID (UID) of that element. The UID of the requirements is the corresponding requirement ID in Simulink Requirements. The CRs are white

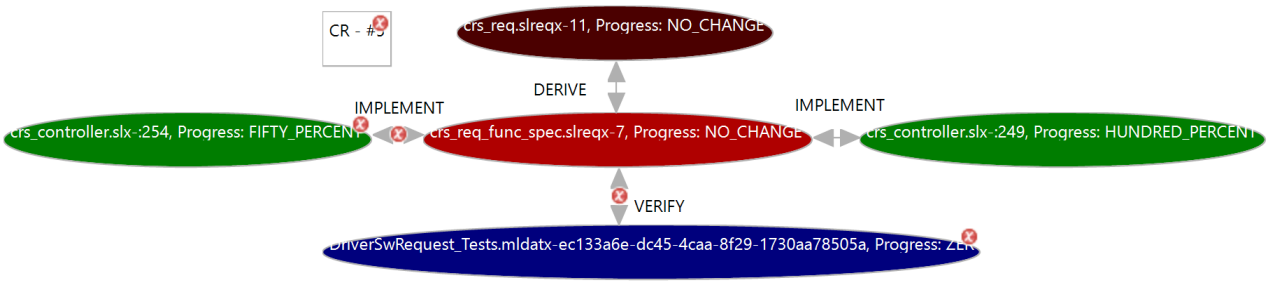

Figure 4.4: An example traceability model in TraceITS depicting the graphical syntax

rectangular boxes with the label in the format, **CR-#***ID*, where ID is the CR ID from Trac. Invalid traceability links and artifacts are annotated with a red cross, as shown in Figure 4.4.

However, there are deviations between the syntax of the models in TraceITS and the specifications in Section 3.1. These are dicussed in Section 4.3.

Model Management Operations

This section explains the automated model management operations in TraceITS. *Automated Traceability Mining* explains the automated creation of the LTM by mining traceability links from Simulink; *Model Validation* explains the automated validation rules for the LTM; *Other Model Management Operations* explains the model-to-model transformation and model merging operations, for the GTM to LTM transformation and vice versa respectively.

Automated Traceability Mining

In TraceITS, the traceability links are mined from Simulink Requirements using Epsilon. Epsilon offers integration with heterogeneous models, such as a Simulink model,

through the Epsilon Model Connectivity (EMC) layer. The EMC interfaces with MATLAB's Java API and provides an interface to mine various types of Simulink traceability links. Using the Epsilon Object Language (EOL), these links are queried, filtered, and modelled in EMF.

User input is used to filter the artifacts relevant to the LTM; The user is prompted with a dialogue to input the IDs of the functional requirements that they would like to mine to the LTM. After the input, the functional requirement and the artifacts traced to the functional requirement are mined to the LTM. For example, from Figure 4.4, if the user selects *crs_req_func_spec.slreqx-7* (functional requirement), *crs_req.slreqx-11* (system requirement), *crs_controller.slx-:254*, *crs_controller.slx-:249* (models) and *DriverSwRequest_Tests.mldatx-ec133a6e* (test case) are mined to the LTM. Additionally, if any of the mined artifacts trace to other artifacts, all the traced artifacts are mined as well.

Model Validation

Model validation is implemented using the Epsilon Validation Language (EVL). The validation applies only to the LTMs. The goal of the validation is to ensure that the LTM is ready to be merged into the GTM. Therefore the constraints shown in Table 4.2 are implemented.

Other Model Management Operations

LTMs can also be created by copying artifacts from the GTM. Model-to-Model transformation can be used for creating LTM from the GTM. For the GTM to LTM transformation, the user can use the *select* attribute to select one or more artifacts in the

Table 4.2: EVL Constraints and Critiques for the LTMs

Context	**Type**	**Name**	**Description**
Traceability	Constraint	Validity	Ensure that all the traceability links in the LTM are valid.
Artifact	Constraint	Progress	Ensure that the progress on every artifact in the LTM is a hundred percent or a no-change.
Change_Request_Ticket	Constraint	PlannedLOE	Ensure the CR field, *Planned LoE*, is filled by the corresponding stakeholder
	Constraint	ActualLOE	Ensure the CR field, *Actual LoE*, is filled by the corresponding stakeholder
	Critique	TIM	This Critique is meant to be a reminder to the stakeholder that they had indicated that traceability was not useful to them
	Critique	MergeRationale	This Critique is meant to be a reminder to the stakeholder that they have not filled the merge rationale for the CR

GTM. The selected artifact and all the traced artifacts are copied to the LTM. Post copy operations include invalidating the trace links in the LTM, resetting the progress of the artifacts to ZERO, and referencing the (LTM's) CR to every artifact.

Model-Merging is used for merging an LTM to the GTM after a change is complete. However, before merging, the LTM must be compared with the GTM; Model-Comparision is used for this purpose. In TraceITS, every artifact in the LTM is

compared with the corresponding copy (if present) in GTM, and if there are mismatches, the user is notified. The user can choose to overwrite the artifact in the GTM with the copy in the LTM.

EOL is used for all three model management operations. The reason is explained in Section 4.3.

4.2 Conformance to the Integrated Process

This section explains TraceITS's conformance to the integrated process.

Strategizing (Planning the CRs and Traceability for the New Software Baseline)

Traceability is strategized based on the user feedback from the *Feedback about Traceability* field in the CR (see Table 4.1). The Java application in Eclipse mines the closed CRs from Trac and aggregates the feedback field in every CR. The user is then recommended to either review or not review the traceability model or the queried data. A screenshot of the recommendation from TraceITS is shown in Figure 4.5. In this example, the value of the *Feedback* field in about $3/4^{\text{th}}$ of the closed CRs was *Yes*, signifying that about 75% of the developers found the traceability models and the support information helpful.

```
Based on the feedback from the previous sprint,  75.00% of the developers
indicated that traceability was helpful in making the changes
TIM & CR review IS NOT NECESSARY
```

Figure 4.5: An example of TraceITS strategy recommendation to the user

Automated Model-Migration for GTM update is not implemented in this version of TraceITS. The reason is discussed in Section 4.3.5.

Creating (Creating and Validating Traceability During CR Implementation)

In TraceITS, the LTMs can be created in one of the following ways,

- Mining traceability links from Simulink Requirements – Explained in Section 4.1.3, *Automated Traceability Mining*.
- Copy from GTM – Explained in Section 4.1.3, *Other Model Management Operations*.
- Manual – Although it is not recommended, the user may create the LTM manually since the CR has been extended into a graphical syntax editor.

In all of these cases, the traceability links are invalidated, and the progress on every artifact is reset to "ZERO".

Maintaining (Traceability Maintenance as CR is Closed)

Upon the closure of the CR, the artifacts in the LTM and GTM are compared. If the artifacts and the traceability match, the LTM's CR is copied to the GTM. The artifacts now reference the new CR. An example is shown in Figure 4.6. Assume CR-#7 modified *crs_req.slreqx-10* (and all its downstream artifacts), and CR-#5 (closed) initiated the developement of *crs_req.slreqx-10*. After CR-#7 is implemented and

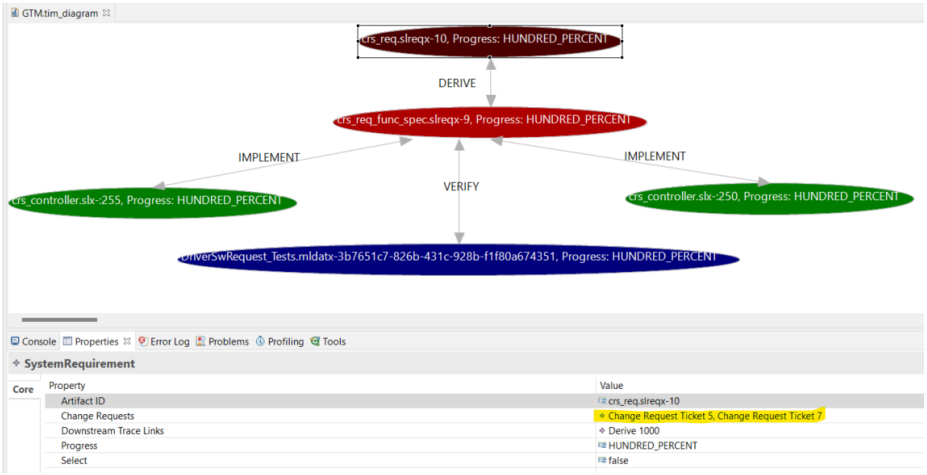

Figure 4.6: The artifacts reference additional CR after the merge

merged with the GTM, CR-#7 is added to the list of CRs referenced by *crs_req.slreqx-10*, and other artifacts modified with CR-#7.

If there are conflicts between the LTM and the GTM, the user is expected to update the merge rationale, forcing the merge or manually merging the LTM with the GTM. In the former case, the artifact LTM replaces the corresponding copy in the GTM, and the trace links are modified accordingly.

Using (Using Traceability models for Change Impact Analysis and Status Reporting)

The LTMs and GTMs in TraceITS can be used in the following ways,

- **Reviewing Traceability** - Developers can validate the mined traceability link by toggling the *validity* attribute in Traceability instances to True.
- **Reporting Progress** - Developers can update every artifact's *progress* attribute to precisely report the progress of the CR in meetings or otherwise.

- **Change Impact Analysis** - In TraceITS, the user can select one more artifact from the GTM if it is known that artifact is affected by the change. Then the chosen artifacts and the traced artifacts are selected automatically. The attributes *Planned LoE* and *Actual LoE* are mined from the referenced CRs. The probability of risk associated with the artifacts is presented to the user to aid with CIA.

4.3 Limitations

4.3.1 CIA and Build-Tracking Views

The CIA and the Build-Tracking views are not implemented in this version of TraceITS. The views aid stakeholders when the integrated process is deployed in a team setting and does not add value for a PoC. The graphical syntax editors for the GTM and LTM serve as the views for the PoC.

Moreover, implementing the views is a straightforward process. Creating the Build-Tracking view involves filtering the CRs using the *version* attribute and highlighting the progress and the assignee attributes in the LTM and CR respectively. This can be accomplished using the Epsilon Generator Language (EGL) and Picto (Kolovos *et al.*, 2020). Creating the CIA view is also straightforward; the GTM must be supplemented with the appropriate GUI to filter the CRs and the artifacts.

4.3.2 Limitations in GMF

Eugenia recommends using Eclipse Sirius (Eclipse Development Team, 2022) or Picto (read-only models) instead of GMF because GMF tooling is not actively maintained (Epsilon Development Team, 2022c). As a result, there are a few limitations in TraceITS due to GMF, as summarized in the following bullet points,

- **Artifacts are represented as ellipses instead of circles** - Section 3.1.2 specifies the artifacts are solid circles, but they are solid ellipses in TraceITS. The labels of the artifact models are the artifact ID and the progress, with progress in a new line. However, in GMF, the formatting characters in a string are ignored, resulting in the ID and progress in the same line. Therefore ellipses were chosen for readability.

- **Varying the graphical syntax of the models in the runtime** - This affects two aspects of the integrated process. The first is the inability to adjust the colour of the Artifact class when the *progress* attribute is varied. Second, varying the colour of the trace link based on the validity of the link.

- **The redundant *select* attribute** - Ideally, the editor should handle a model selection, where the user could select multiple artifacts by clicking on them. Implementing this requires creating an event handler on top of the Eugenia-generated Java code. Currently the selection is handled with the help of an attribute *select* in the *Artifact* class.

- **Updating the common attributes simultaneously** - Since the *progress* attribute is unique for every artifact, it may be tedious for the user to update it

individually for every model. It is unclear if this is a limitation of GMF itself, but it can be implemented using EOL.

4.3.3 Using Epsilon Object Language (EOL) for Everything

Although Epsilon provides specialized Domain Specific Languages (DSLs) such as ETL, EGL, and ECL, TraceITS uses only EOL for all the model management operations except validation. The reason is that EOL it was used when LTMs were created from mining Simulink; ETL did not work for mining. As a result, many functions (*operation* in EOL) written for mining were reused for other model management operations.

4.3.4 Software Version

The EMC uses MATLAB's Java API to mine the traceability links. However, MATLAB is compatible only with Java 8[2]. The current releases of Eclipse are compatible only with Java 11 and later [3]. Therefore in TraceITS, Eclipse 2020-06 and Epsilon 2.2 are used. The latest version of Eclipse and Epsilon are 2022-03 and 2.4, respectively. Using Epsilon 2.3 or 2.4 with Eclipse 2020-06 seemed to cause unexpected issues, such as an unusually long Eclipse Runtime application startup time.

4.3.5 Other Limitations

Other limitations of TraceITS are as follows,

1. Automated Model-Migration is not implemented in this version of TraceITS.

[2]https://www.mathworks.com/support/requirements/language-interfaces.html
[3]https://wiki.eclipse.org/Eclipse/Installation

The reason is that it does not add any value to a proof of concept. There are adequate instructions for Automated Model-Migration using Epsilon Folk, which can be implemented when TraceITS is deployed in a team setting.

2. In practice, the creation of the GTM should be an iterative process. That is, GTM should be created by merging the LTMs in subsequent software builds. In this version of TraceITS, since a preexisting project is taken as an example, the GTM is also automatically mined from Simulink Requirements.

3. The UIDs of the models and tests from Simulink and Simulink Test are long strings instead of numerical values. This makes the identification of the corresponding model or test blocks a difficult task. For the PoC, the Simulink Cruise-Control project is used as an example. Therefore, this limitation is not addressed. However, if Simulink is part of the CASE tools when TraceITS is deployed in practice, then there must be a lookup table that translates the UID into identifiable keys for the models.

4. The Trac-Eclipse connection is currently not event-driven but has to be manually triggered from Eclipse. This is due to the current architecture of TraceITS where the CRs are directly queried from Trac's database. When TraceITS is deployed in a team setting, direct access to Trac's database is no longer possible. Trac would be hosted on the server, and client Eclipse applications would be on the developer's PC. In this case, Trac should be extended with plugins to communicate changes to the Java application in Eclipse and can be completely event-driven.

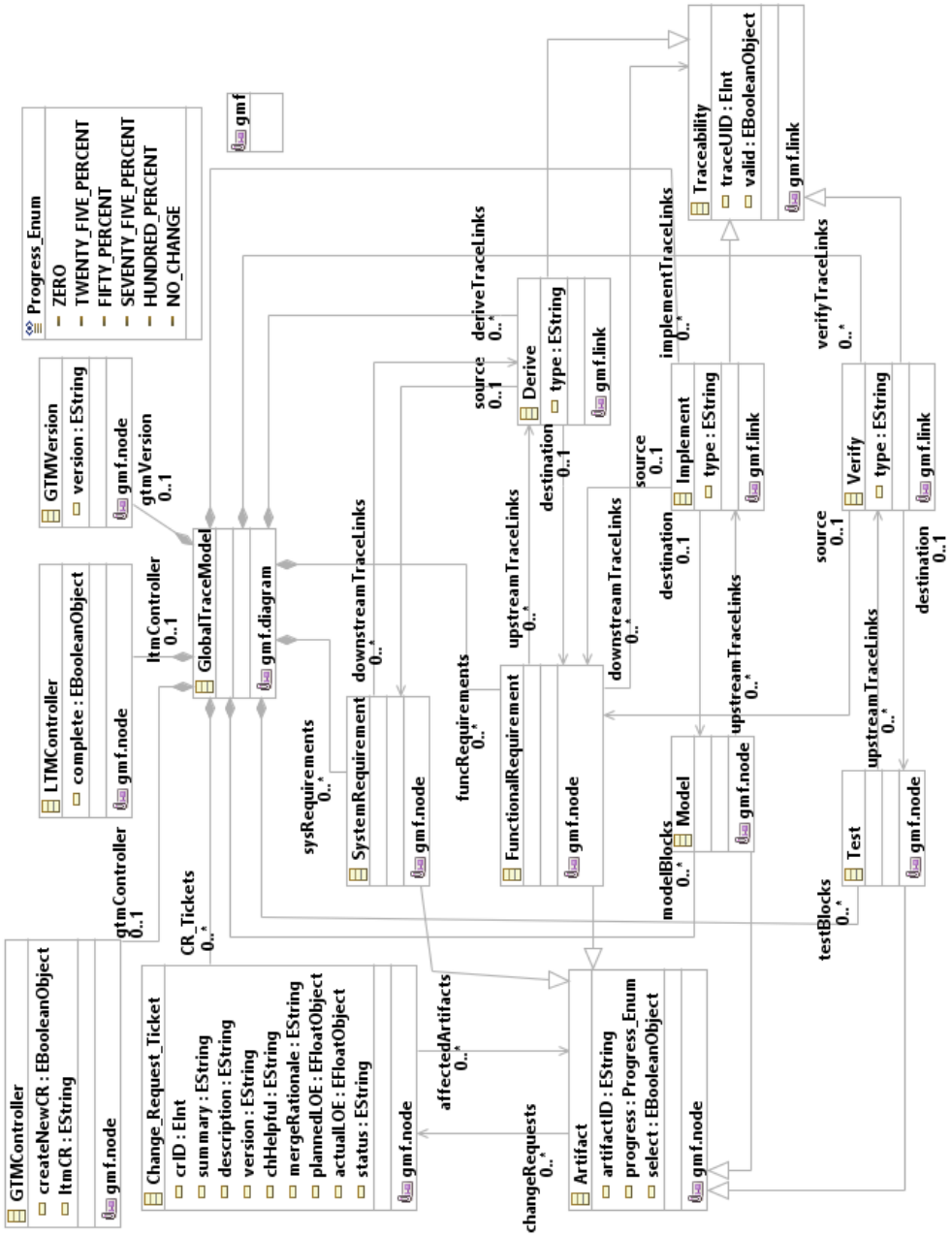

Figure 4.7: TIM for the Simulink Cruise Control example

Chapter 5

Evaluation

This chapter presents an evaluation of the integrated process and TraceITS. Section 5.1 presents a discussion about the benefits and limitations of the integrated process and TraceITS.

5.1 Discussion

This section discusses the benefits and limitations of the integrated process and TraceITS. Subsection 5.1.1 compares TraceITS with a regular ITS; Subsection 5.1.2 compares the traceability data in TraceITS with generic traceability data; Subsection 5.1.3 discusses the long-term benefits of the integrated process; Finally, Subsection 5.1.4 discusses the limitations and challenges of the integrated process.

5.1.1 Comparision with a regular ITS

There are several benefits with TraceITS when compared with a regular ITS such as Trac (without extensions). Table 5.1 explains these benefits in detail.

Table 5.1: Comparision of TraceITS with a regular ITS

Regular ITS	**TraceITS**
The lifecycle of the CR is not related to traceability data.	Traceability data is managed with the lifecycle of the CR.
CR tickets hold information about the change only; After the closure of a CR, they are "stowed away" for future audits.	In addition to information about the change, TraceITS also uses the CR as a mechanism for feedback. As a result, in TraceITS, the CR is often mined and reviewed by the user.
Unless part of an ALM suite, the user cannot directly see the artifact affected by the CR.	Every CR has LTM through which the user can review the affected artifacts.
The progress of the CR is deduced from the *state* of the CR and any comments added by the user.	The progress of the CR is a function of the *progress* attribute of all the artifacts in the LTM. This allows for an accurate status reporting of the CR.
May not directly aid with CIA; The user may have to manually look up previous CRs and the traceability data of the project.	Allows for a semi-automated CIA; The previously worked CR are referenced by every artifact in the GTM.

5.1.2 Comparison with Generic Traceability Data

The traceability models used in the integrated process provide certain benefits over generic traceability data (*e.g.* traceability matrix). Table 5.2 explains these benefits in detail.

5.1.3 Long-term Benefits of the Integrated Process

Implementing the integrated traceability-CR management process provides several long-term benefits to organizations. These benefits are summarized in the subsequent subsections.

Table 5.2: Comparision of the traceability data in integrated process with generic traceability data

Generic Traceability Data	**Traceability Data of TraceITS**
Represented as graphs/matrices/links. Possible overhead in developing the infrastructure for storage, access and visualization	Represented as graph models only. Storage, access and visualization are abstracted by the graphical syntax editor and model management framework. Transformation to other forms of representation using model to text transformation.
Provides information about the relationship between the various types of artifacts only.	Additional information, such as the names of the previous developers, is provided since every artifact is linked with CR.
Passive; Provides information only.	Active; Allows the developers to validate the trace links and update the progress.

Continuous Feedback and Improvement

The stakeholders' feedback on traceability would enable organizations to understand the stakeholder's needs clearly. For example, in the Cruise Control Example from Chapter 4, the TIM covers all the deliverable artifacts of the project. However, if the feedback from the CRs indicates that the traceability was not helpful, organizations could take measures to improve the traceability quality, such as:

- Supplementing the TIM or the fields of the CRs with other formal artifacts such as release documentation or informal artifacts such as an internal Wiki page.
- Improving the quality of existing artifacts to be useful for the stakeholders. For example, switching to formal specifications for requirements instead of natural language.
- Improving the engineering/management process; for example, reducing the scope

of a CR so that the developer is not overburdened.

Over time, the feedback may help the organization develop a mature and efficient engineering process.

Traceability Models for Communication among Multicultural Teams

Software developers would use the traceability models to understand the scope of the CR and report its progress. If organizations have a fully remote or hybrid work environment, or in the case of the Global Software Development (Herbsleb and Moitra, 2001), communication among teams could be a challenge. The traceability models could alleviate some of the communication gaps by acting as a DSL to communicate the scope of the CR, or report the progress of the CR. This can also be an added motivation for the developers to create and maintain traceability data.

Scope for Automated Root Cause Analysis

The root cause analysis can be automated using model management operations, optionally supplemented by Machine Learning (ML). For example, as with the case of the CIA, keywords from a bug report can be compared with previously closed CRs, and relevant artifacts can be mined and analyzed. The LTMs leave a trail for Quality Analysts or other Stakeholders to identify the root causes of product defects. For instance, if a defect was introduced in a specific product version, it is easy to query and analyze the corresponding portions of the traceability data. In other words, the LTMs corresponding to that software build version can be analyzed automatically. Automated analysis of the LTMs of a released software build can help identify root causes such as the following (but not limited to):

- Process Compliance – From the LTMs, it can be identified if a requirement was verified and validated in the first place.
- Inadequate tracing – From the LTMs, the dependencies across artifacts, i.e. missed dependencies across requirements, can be identified and addressed for the ongoing development.
- Improper planning – If a requirement was not adequately verified, the LTMs could help identify if the tester was given enough time for the verification. For instance, TraceITS could be extended to log the progress of an artifact which can then be graphed. A steep progress curve could indicate that the artifact was either easy to verify or that it was rushed.

Return on Investment

Initial investment and training are required to set up a system like TraceITS. However, in the long run, when a system like TraceITS is fully operational, the organization can realize the benefits of traceability. For example, realizing the full benefits of traceability would result in engineers understanding their work faster and better, which among other factors, contributes to fewer defects in the product. This would ensure a return on investment for the organization.

Organizational Collaboration

The ITS infrastructure is typically common to all the teams in an organization. Therefore, interfaces could be written so that the TIM or parts of the TIM are shared among teams in an organization. This ensures that the traceability data from one team is compatible with another.

5.1.4 Limitations of Integrated Process

There are several limitations and challenges to the integrated process. The following subsections summarize the limitations in detail.

Tool Integration and Trace Granularity

The creation of LTMs relies on automated traceability techniques such as automatic traceability recovery, mining pre-existing traceability links or EBTs. Without the means to mine traceability links, the developer may have to create traceability links manually, which may be inefficient. Further, even if plugins exist, it is unclear whether the granularity desired in the TIM is obtainable through these plugins without modification.

Automated traceability is a significant effort to set up and maintain and therefore is a separate research topic. For instance, if the development effort is spread across multiple CASE tools, plugins must be written and maintained to mine the traceability data from every device. However, since manual validation is involved in this process, the desired precision and recall metrics of IR or ML traceability recovery techniques are unknown. The precision and recall metrics will be determined after the experimental validation of the integrated process explained in Section 6.1.

Adopting the Process during Product Development

As mentioned in the benefits, implementing the process in an organization requires significant initial effort and investment. For instance, a new ITS such as TraceITS should be developed with appropriate plugins for automated traceability, an initial version of TIM should be planned, and the developers should be trained to use the

new process and the ITS. Querying feedback from CRs is also an incremental task; the previously closed CRs, without customized fields, may not be helpful. Therefore, though plausible, adapting this process in the middle of product development may be a significant effort.

Tickets for All Activities

Every task involving traceability must be driven by a ticket for the integrated process to succeed. This includes tickets for CIA, traceability audits etc. It is unclear if this practice in organizations may be met with resistance from stakeholders.

Scalability of the Process

Large organizations developing a complicated product would have multiple teams working on different parts of the product. It is not usual to find that the engineering process among the teams differs. However, the traceability data of multiple teams should be compatible with each other if they are working on the same (overall) product. Whether the process caters to the needs of all the teams and produces uniform traceability across the projects will be determined only during phase 3 of the validation explained in Subsection 6.1.

Chapter 6

Conclusions and Future Work

This thesis presented a process in which traceability management is integrated with the lifecycle of a Change Request (CR) ticket. The two main elements of this integrated process are CRs and traceability data, represented as models in the process.

In the integrated process, CRs, in addition to communicating a change, are also used as a medium of feedback. Additional fields are recommended to be added to CRs, which can be queried and summarized to users in the future. Examples of the additional fields include feedback on traceability or any information (such as sections of documentation) that could help subsequent developers.

The second element of the process, the traceability model, can be a Global Traceability Model (GTM) or a Local Traceability Model (LTM). The GTM holds all the traceability models of the project. The LTM of a CR is a copy of the subgraph of the GTM and holds the traceability models specific to the CR. The lifecycle of the LTM is tied to the lifecycle of the CR. When CRs are planned for a new software build, the structure of the LTM is reviewed. When the CR is in progress, the developers use the LTM to understand the change and report the progress of the implementation using

the LTM. Before the CR is closed, the developers fill the customized fields in the CR, and the LTM is automatically merged with the GTM. The fields in the closed CRs are then queried and aggregated in the GTM to help with activities such as traceability strategizing or Change Impact Analysis.

A proof-of-concept of the integrated process was demonstrated using the Traceability-centric Issue Tracking System (TraceITS) tool. TraceITS integrates the ITS Trac with Eclipse Modeling Framework (EMF) and Epsilon Model Management Framework. An open source Simulink project, *Requirements for Cruise Control Model*, is used as the test project for traceability management.

Compared to a regular ITS, TraceITS and the integrated process offer several short- and long-term benefits to organizations since traceability is at the forefront of change management. The benefits include faster learning for developers, the possibility of automated or semi-automated CIA support, or the possibility of automatic root cause identification for product defects.

6.1 Future Work

The next step of the integrated process is to validate the process. The validation of the process is a three-phase approach. First, use TraceITS to conduct a small-scale experiment in a university setting, for example, in an existing research project with student developers, over a span of at least two software deliveries. After this, tailor the process according to the lessons learned from this experiment.

The second phase is to experiment with this process in an industrial setting. Implement the process in an organization (preferably a startup) with no traceability requirements and implement the integrated process. Here the process should be

experimented with over a longer time, at least for 5-6 software deliveries. If the validation is successful, update the process with the lessons learned.

Finally, implement the (updated) process in a medium-large scale organization with established traceability requirements. If the validation is successful, update the process with the lessons learned, which will be the final version of the process.

Last but not least, if all three validation phases prove successful, calling for an update to IEEE-828-2012 with this process may be worthwhile.

Appendix A

Abstract Syntax of TraceITS Models

This appendix contains a detailed explanation of the TIM of TraceITS explained in Section 4.1.3, and Figure 4.7.

Table A.1: A detailed explanation of the abstract syntax of the models in TraceITS

Class	**Constraints**	**Description**	**Attributes**	**Type**	**Description**
GlobalTrace-Model	Singleton	The system boundary of the traceability models. This class contains instances of all other types in the metamodel.			
GTMController	Singleton	GTMController applies to the GTM only. It is used to create new CRs from the GTM or copy traceability models from GTM to LTM.	createNewCR	Boolean	The flag indicates that a new CR must be created from the selected traceability models.
			ltmCR	String	The ID of the CR if the traceability models are to be copied to a CR. This attribute is ignored if the createNewCR attribute is set.

Continued on the next page

Continued from previous page

Class	**Constraints**	**Description**	**Attributes**	**Type**	**Description**
GTMVersion	Singleton	Indicates the software baseline version that this GTM is applicable for.	version	String	Example: v1.0.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
Change_Request_Ticket	0 or more	Model of the Change Request form or ticket. The attributes of this class should match all the fields in the CR.	crID	int	CR ticket ID.
			summary	String	Summary (title) of the CR.
			description	String	A detailed description of the CR.
			version	String	The software baseline version for which this CR is implemented.
			chHelpful	String	A Yes/No feedback that indicates if the CR and the traceability data were helpful for the developer to implement the CR.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
			mergeRationale	String	Field explains the conflict between the LTM and the GTM after implementing a CR.
			plannedLOE	Float	Field to hold the man-hours planned for the CR. This field is an example explicitly created for the PoC.
			actualLOE	Float	Field to hold the actual man-hours for implementing the CR. This field is an example explicitly created for the PoC.
			status	String	The state of the CR, *e.g.* Open, Closed etc.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
			affectedArtifacts	Reference to Artifact	The list of artifacts affected by this CR.
Progress_Enum	Enumeration	This enumeration indicates the percentage of the progress made on an artifact. The attributes are 0, 25, 50, 75 and 100 percent, respectively.			
Artifact	Abstract	Abstract type to hold attributes that are common to all the artifact types.	artifactID	String	Unique ID of the artifact.
			progress	Progress_Enum	The implementation progress of the artifact.
			changeRequests	Reference to *Change_Request_Ticket*	A list of previous and current CRs related to this artifact.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
			select	Boolean	A flag for the user to select this artifact.
Traceability	Abstract	Abstract type to hold attributes that are common to all the traceability links.	traceUID	int	A unique ID for the trace link.
			valid	Boolean	Flag to indicate the validity of the trace link.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
Derive	0 or more	The Derive traceability link between the system and functional requirements. This class is extended from the Traceability class.	type	String	Read-only attribute to indicate the type of trace link in the Graphical Syntax. In this case, the value of the attribute is "DERIVE."
			source	Reference to SystemRequirement	Singleton reference to the system requirement from which the functional requirement is derived.
			destination	Reference to FunctionalRequirement	Singleton reference to the derived functional requirement from the system requirement.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
Implement	0 or more	The Implement traceability link between the functional requirement and model block. This class is extended from the Traceability class.	type	String	Read-only attribute to indicate the type of trace link in the Graphical Syntax. In this case, the value of the attribute is "IMPLEMENT."
			source	Reference to Functional-Requirement	Singleton reference to the functional requirement that the model implements.
			destination	Reference to Model	Singleton reference to the model that implements a functional requirement.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
Verify	0 or more	The Verify traceability link between the functional requirement and test case. This class is extended from the Traceability class.	type	String	Read-only attribute to indicate the type of trace link in the Graphical Syntax. In this case, the value of the attribute is "VERIFY."
			source	Reference to Functional-Requirement	Singleton reference to the functional requirement corresponding to the test case.
			destination	Reference to Test	Singleton reference to the test case that tests the functional requirement.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
SystemRequirement	0 or more	Node for the system requirements. This class is extended from the Artifact class.	downstreamTraceLinks	Reference to Derive	All the Derive trace links that are connected to this system requirement.

Continued on the next page

Continued from previous page

Class	Constraints	Description	Attributes	Type	Description
FunctionalRequirement	0 or more	Node for the functional requirements. This class is extended from the Artifact class.	downstream-TraceLinks	Reference to Traceability	All the Implement or Verify trace links that are connected to this functional requirement.
			upstream-TraceLinks	Reference to Derive	All the Derive trace links that are connected to this functional requirement from the system requirements.
Model	0 or more	Node for the Simulink Models. This class is extended from the Artifact class.	upstream-TraceLinks	Reference to Implement	All the Implement trace links that are connected to this model from the functional requirements.

Continued on the next page

Continued from previous page

Class	**Constraints**	**Description**	**Attributes**	**Type**	**Description**
Test	0 or more	Node for the Simulink Tests. This class is extended from the Artifact class.	upstream-TraceLinks	Reference to Verify	All the Verify trace links that are connected to this test from the functional requirements.

Bibliography

(1990). IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84.

(2017). ISO/IEC/IEEE international standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541.

Antoniol, G., Cleland-Huang, J., Hayes, J. H., and Vierhauser, M. (2017). Grand challenges of traceability: The next ten years. *arXiv preprint arXiv:1710.03129*.

Appleton, B., Berczuk, S., and Cowham, R. (2007). Lean-Agile traceability: Strategies and solutions. `https://www.cmcrossroads.com/article/lean-agile-traceability-strategies-and-solutions`. Online, accessed - 2020-06-15.

Atlassian Inc. (2022a). Jira documentation. `https://support.atlassian.com/jira-software-cloud/`. Accessed: 2022-05-25.

Atlassian Inc. (2022b). Jira software. `https://www.atlassian.com/software/jira`. Accessed: 2022-05-25.

Atlassian Inc. (2022c). What are issue types? `https://support.atlassian.com/`

jira-cloud-administration/docs/what-are-issue-types/. Online. Accessed-2022-07-10.

Aung, T. W. W., Huo, H., and Sui, Y. (2020). A literature review of automatic traceability links recoveryfor software change impact analysis. In *Proceedings of the 28th International Conference on Program Comprehension*, pages 14–24.

Berczuk, S., Appleton, B., and Cowham, R. (2005). The trouble with tracing: Traceability dissected. https://www.cmcrossroads.com/article/trouble-tracing-traceability-dissected. Accessed: 2022-05-06.

Bézivin, J., Jouault, F., and Valduriez, P. (2004). On the need for megamodels. In *Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 1–9. Citeseer.

Canfora, G. and Cerulo, L. (2005). Impact analysis by mining software and change request repositories. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 9–pp. IEEE.

Cleland-Huang, J. (2012). Traceability in Agile projects. In *Software and Systems Traceability*, pages 265–275. Springer.

Cleland-Huang, J., Chang, C., and Christensen, M. (2003). Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, **29**(9), 796–810.

Cleland-Huang, J., Gotel, O. C., Huffman Hayes, J., Mäder, P., and Zisman, A.

(2014). Software traceability: trends and future directions. In *Future of software engineering proceedings*, pages 55–69.

CMMI Institute (2022). Capability Maturity Model Integration. https://cmmiinstitute.com/. Accessed:2022-06-15.

Curtis, L. (2021). Why the big quit is happening and why every boss should embrace it. https://www.forbes.com/sites/lisacurtis/2021/06/30/why-the-big-quit-is-happening-and-why-every-boss-should-embrace-it/?sh=4c1ae2db601c. Accessed: 2022-05-06.

Eclipse Development Team (2022). Sirius. https://www.eclipse.org/sirius/. Online. Accessed 2022-08-10.

Eclipse Foundation (2022a). Eclipse Modeling Framework EMF. https://www.eclipse.org/modeling/emf/. Accessed: 2022-06-20.

Eclipse Foundation (2022b). GMF tooling. https://www.eclipse.org/gmf-tooling/. Online, accessed: 2022-08-07.

Egdewall Inc. (2022). Trac. https://trac.edgewall.org/. Online Accessed: 2022-06-20.

Enlean Inc. (2022a). Tuleap. https://www.tuleap.org/. Accessed: 2022-05-23.

Enlean Inc. (2022b). Tuleap documentation. https://docs.tuleap.org/. Accessed: 2022-05-23.

Epsilon Development Team (2022a). Eclipse Emfatic. https://www.eclipse.org/emfatic/. Online, accessed - 2022-08-08.

Epsilon Development Team (2022b). Eclipse Epsilon™. https://www.eclipse.org/epsilon/. Online Accessed: 2022-06-20.

Epsilon Development Team (2022c). Eugenia. https://www.eclipse.org/epsilon/doc/eugenia/#eugenia-and-gmf-tooling. Online. Accessed-2022-08-10.

excentia (2022). TraceabilityX for Jira. https://marketplace.atlassian.com/apps/1211138/traceabilityx-for-jira?hosting=datacenter&tab=overview. Atlassian Marketplace, Accessed: 2022-05-25.

Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., Maletic, J., and Mäder, P. (2012a). *Software and Systems Traceability*, chapter Traceability Fundamentals, pages 3–22. Springer.

Gotel, O., Cleland-Huang, J., Hayes, J. H., Zisman, A., Egyed, A., Grünbacher, P., Dekhtyar, A., Antoniol, G., and Maletic, J. (2012b). *Software and Systems Traceability*, chapter The Grand Challenge of Traceability v1.0, pages 343–429. Springer, first edition.

Herbsleb, J. D. and Moitra, D. (2001). Global software development. *IEEE Software*, **18**(2), 16–20.

IEEE (2012). IEEE standard for configuration management in systems and software engineering. Standard. IEEE std 828-2012.

Kolovos, D., De La Vega, A., and Cooper, J. (2020). Efficient generation of graphical model views via lazy model-to-text transformation. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 12–23.

Köstebek Teknoloji (2022). Traceability matrix and link graph. `https://marketplace.atlassian.com/apps/1211580/traceability-matrix-and-link-graph?hosting=server&tab=overview`. Atlassian Marketplace, Accessed: 2022-05-25.

Li, Y. and Maalej, W. (2012). Which traceability visualization is suitable in this context? a comparative study. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 194–210. Springer.

Lucia, A. D., Marcus, A., Oliveto, R., and Poshyvanyk, D. (2012). Information retrieval methods for automated traceability recovery. In *Software and systems traceability*, pages 71–98. Springer.

Mäder, P. and Egyed, A. (2015). Do developers benefit from requirements traceability when evolving and maintaining a software system? *Empirical Software Engineering*, **20**(2), 413–441.

Mäder, P., Gotel, O., and Philippow, I. (2009). Enabling automated traceability maintenance through the upkeep of traceability relations. In *European conference on model driven architecture-foundations and applications*, pages 174–189. Springer.

Mader, P., Gotel, O., and Philippow, I. (2009). Getting back to basics: Promoting the use of a traceability information model in practice. In *2009 ICSE Workshop on Traceability in Emerging Forms of Software Engineering*, pages 21–25. IEEE.

Mäder, P., Jones, P. L., Zhang, Y., and Cleland-Huang, J. (2013). Strategic traceability for safety-critical projects. *IEEE software*, **30**(3), 58–66.

Maro, S., Steghofer, J.-P., Knauss, E., Horkoff, J., Kasauli, R., Wohlrab, R., Korsgaard, J. L., Wartenberg, F., Strøm, N. J., and Alexandersson, R. (2020). Managing traceability information models: Not such a simple task after all? *IEEE Software*, **38**(5), 101–109.

Maro, S., Steghöfer, J.-P., Bozzelli, P., and Muccini, H. (2022). TracIMo: a traceability introduction methodology and its evaluation in an Agile development team. *Requirements Engineering*, **27**(1), 53–81.

Mathworks Inc. (2022a). Requirements definition for a cruise control model. `https://www.mathworks.com/help/slrequirements/gs/requirements-definition-for-a-cruise-control-model.html`. Online, accessed - 2022-08-07.

Mathworks Inc. (2022b). Requirements toolbox. `https://www.mathworks.com/products/requirements-toolbox.html`. Online, accessed 2022-08-07.

Mathworks Inc. (2022c). Simulink Test. `https://www.mathworks.com/products/simulink-test.html`. Online. Accessed: 2022-08-15.

Microsoft (2021). The next great disruption is hybrid work—are we ready? `https://www.microsoft.com/en-us/worklab/work-trend-index/hybrid-work`. Accessed: 2022-05-06.

Mohan, K., Xu, P., Cao, L., and Ramesh, B. (2008a). Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, **45**(4), 922–936.

Mohan, K., Xu, P., and Ramesh, B. (2008b). Improving the change-management process. *Communications of the ACM*, **51**(5), 59–64.

Muralidharan, N. G., Pantelic, V., Bandur, V., and Paige, R. (2022). Integrating software issue tracking and traceability models. Accepted for presentation and publication in the proceedings of the 38th IEEE International Conference on Software Maintenance and Evolution (ICSME).

Optimizory Technologies Pvt. Ltd. (2022). Links Explorer Traceability & Hierarchy. `https://marketplace.atlassian.com/apps/1211120/links-explorer-traceability-hierarchy?hosting=cloud&tab=overview`. Atlassian Marketplace, Accessed: 2022-05-25.

Project Management Institute (2021). *The standard for project management and a guide to the project management body of knowledge (PMBOK guide)*, chapter II.4 Models, Methods, and Artifacts. Seventh edition.

PTC Inc. (2022). Windchill PLM software. `https://www.ptc.com/en/products/windchill`. Online. Accessed: 2022-06-15.

Rempel, P. and Mäder, P. (2017). Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, **43**, 777–797.

RTCA (2011). Software considerations in airborne systems and equipment certification. Standard. DO-178C.

Shahid, M. and Ibrahim, S. (2016). Change impact analysis with a software traceability approach to support software maintenance. In *2016 13th International Bhurban conference on applied sciences and technology (IBCAST)*, pages 391–396. IEEE.

Siemens Inc (2022). Polarion ALM. `https://polarion.plm.automation.siemens.com/products/polarion-alm`. Accessed: 2022-05-24.

Steghöfer, J.-P. (2017). Software traceability tools: Overview and categorisation. *Report of the GI working group "traceability/evolution". German Informatics Society (GI)*, pages 2–7.

System Weaver Inc. (2022a). System Weaver. `https://systemweaver.com/`. Accessed: 2022-05-23.

System Weaver Inc. (2022b). System Weaver knowledge base. `https://support.systemweaver.se/en/support/solutions`. Accessed: 2022-05-23.

Wikipedia contributors (2022). Traceability matrix. `https://en.wikipedia.org/wiki/Traceability_matrix`. Online; accessed: 2022-06-23.