2022 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 978-1-6654-7956-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSME55016.2022.00053

# Integrating Software Issue Tracking and Traceability Models

Naveen Ganesh Muralidharan, Vera Pantelic, Victor Bandur, Richard Paige
*Department of Computing and Software*
*McMaster University*
Hamilton, ON, Canada
{muralidn, pantelv, bandurvp, paigeri}@mcmaster.ca

***Abstract*—Awareness of the importance of systems and software traceability, as well as tool support for traceability, have improved over the years. But an effective solution for traceability must align and integrate 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. In this paper, we propose Change Request management using traceability data. In our approach, new Change Requests (CRs) are created from the traceability model of the corresponding project. The created CRs contain a portion of the project's overall traceability model that is relevant to that change. A proof-of-concept issue tracking system is proposed that uses a traceability model at its core.**

***Index Terms*—Traceability, Issue Tracking Systems, Change Request, Traceability Information Model**

## I. Introduction and Background

Traceability is required in formal development settings and encouraged in others. Implementing traceability however offers many benefits to engineering organizations, such as efficient change impact analysis, organizational learning, and process conformance [1]. For instance, well-chosen and well-defined traceability data can help a new developer on a project to effectively implement a change by providing necessary information, such as the affected artifacts, at the developer's fingertips [2].

However, implementing an effective traceability solution comes with some practical challenges. Among the challenges is locating traceability's place in the organization. Mäder *et al.* observe that in several cases traceability was created in an ad-hoc manner toward the end of projects. They refer to the phenomenon as "Traceability as an afterthought" [3]. Likewise, in a recently published study on introducing a traceability process alongside an existing engineering process [4], Maro *et al.* observe that it is practically challenging for organizations to adopt a new traceability process. This is not surprising, but it underlines the need for the traceability process to co-exist with the company's established engineering processes. Therefore, in addition to appropriate tool support for traceability, the traceability process must align with the existing engineering processes of the organization. Indeed, the phases of the traceability processes model (namely traceability strategy, traceability creation, traceability use and traceability maintenance [5]) must be integrated with the engineering process. This respects the vision of "Ubiquitous Traceability" from [6], where traceability is a side-effect of an organization's engineering process.

Our proposal in support of this aligns with the IEEE-828-2012 standard [7] for configuration management. One of the requirements of the standard is to define the Change Request (CR) forms relevant to the project. CRs are commonly known as "tickets". CR tickets are among the most important artifacts used in a software engineering project, because all project stakeholders interact with CRs.

Consider what typically happens when a customer creates a CR. The CR is forwarded to the appropriate lead (Product Owner/Scrum Master) for a change impact analysis (CIA). After the impact analysis is performed, the CR may be approved by the Change Control Board (CCB). Approved CRs are assigned to developers for implementation. The status of the CRs is then tracked and monitored by all the stakeholders. In this way, a CR ticket can be seen as a "cheat sheet" for stakeholders to understand a system's evolution. IEEE-828-2012 specifies a minimal set of information that should be present in a CR. The information includes the description of the change, state of the change, affected baseline (software version/build), impact on the project and originator of the change. However, additional fields can be added to the CR based on the project requirements. For example, before closing a CR, a developer could include pointers to useful artifacts, such as specific document sections, that can help others (including new engineers) in understanding the system and changes.

We propose integrating the process of CR management with the phases of the traceability process model. Specifically, our contribution is a process based on Model Driven Software Engineering (MDSE), where CR tickets and traceability data are coherently managed. We represent the traceability data as models, and use model creation, model validation and model merging to extend CR tickets into a graphical syntax editor that holds portions of the entire project's traceability model. We then propose a process where traceability is managed with the CR's lifecycle. We envision that this process will then be adopted into Issue Tracking Systems (ITS) to make traceability a regular part of the software engineering process.

2576-3148/22/$31.00 ©2022 IEEE
DOI 10.1109/ICSME55016.2022.00053

## II. Related Work

Mohan *et al.* [8] [9] integrated the Software Configuration Management process specified by the Rational Unified Process (several commonalities with IEEE-828-2005) with traceability and demonstrated that it yields a combination of product and process knowledge. Since then, there have been a few technological changes: IEEE-828-2005 was superseded by IEEE-828-2012, the adoption of Agile software development has increased across the industry, and consequently, ITSs support more features, such as integrated project management and visualization of CRs. Our work is more granular. Specifically, our work focuses on the low-level change control requirements specified by IEEE-828-2012.

Similar to our proposal, TracIMO [4] proposes a process for integrating traceability with an existing engineering process. However, their proposal addresses engineering processes in general, whereas our proposal is specific to integration with configuration management processes that are driven by CRs. It is unclear at this point whether their proposal would resemble ours if it were applied to a configuration management process. It is likely that the same level of traceability can be achieved in both proposals, but the ergonomics of using the traceability information would be different. The authors demonstrate their approach on a case study in which software development activities are completely driven by CRs.

Appleton *et. al.* [10] (also see [11]) propose the idea of "Lean Traceability". They propose achieving automated traceability (Event-Based Traceability [12]) driven by the change management system, where fine-grained/atomic units of work, such as specific requirements, modules of code and tests, are linked to a single CR (Task-Based Development/Test-Driven Development). However, they do not specify the process of creating and managing the CRs.

Automated traceability recovery [13] [14] focuses on establishing traceability "on the spot" among the artifacts using Information Retrieval (IR) or Machine Learning (ML). In particular, Canfora *et al.* [15] and Shahid *et al.* [16] establish traceability from CRs using IR techniques. 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. Our work builds a process that includes automated traceability recovery, supported only by manual validation by developers.

Similarly, 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 can be visualized on demand. A few ALM suites support phases of the traceability processes model as well. From Steghofer's survey of traceability tools [17], System Weaver [18], Polarion ALM [19] and PTC Integrity/Windchill [20] offer limited support for planning traceability through Traceability Information Models (TIMs) [21]. However, from the publicly available documentation of these suites, it appears that the change management processes of the ITSs found in these suites do not directly depend on traceability data. Moreover, our work focuses on the change management process in an ITS, standalone or ALM.

## III. Integrated Change Request Management and Traceability Process

This section proposes the integrated CR management and traceability process. Section-III-A describes a few key elements of our process, while Sections III-B, III-C, III-D, III-E and III-F describe the integrated CR-traceability strategy, creation, maintenance, use and visualization process, respectively. Finally, Section-III-G describes a use case of the integrated process.

### A. Elements of the integrated process model

Before describing the steps of our proposed integrated process, we describe fundamental building elements required for the process.

- The Trace Information Model (TIM) – The metamodel represents the traceability requirements for the project [21]. There is no standard structure for the TIM; it varies according to the traceability requirements of the project.
- Global Traceability Model (GTM) – An instance of the TIM. This traceability model represents the complete traceability data for the project.
- Local Traceability Model (LTM) – LTMs are created for a CR and the artifacts relevant to that CR. An LTM is a copy of a portion of the GTM that is affected by the CR. Every CR will have an LTM. The purpose of the LTM is to aid the developer in understanding the scope of the CR, provide a platform to validate the traceability, and help the developer report the status of the CR implementation.
- The Change Request (CR) tickets – We consider the content of a CR to be the vital information for the project.

### B. Strategizing (Planning CRs and Traceability for New Software Baseline)

In a software engineering project, the CRs that are intended to be implemented for a new baseline are typically planned out. IEEE-828-2012's minimum requirements for the fields in a CR are the description of the change, status of the change, affected baseline, impact to the project, resolution and approval. We propose adding the following fields to a CR:

- Feedback about the TIM and the other fields in the CR tickets, *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 III-D).

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.

We also propose 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. To decide whether the TIM needs to be modified, the feedback field for the TIM in the CR can be used. 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. Model propagation operations can help with automating the GTM update.

### C. Creating (Creating and Validating Traceability During CR Implementation)

A change requested by a CR could result in new artifacts and/or modification of existing artifacts. If the CR is for new development, then the LTM in the CR should be created from scratch. The creation of the LTM must be automated or semi-automated to avoid burdening the developers. Techniques that can help with automated traceability creation are automated traceability recovery [13], traceability (links) mining, or Event-Based Traceability [12]. If the CR is for the modification of artifacts, then we propose 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 the 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-III-F) and re-validates the traceability links while implementing the changes.

Last but not least, per IEEE-828-2012's low-level change control requirements, CRs are for Configuration Items (CIs) that are already baselined. However, in practice, there may be several activities such as bug investigation where traceability would be helpful. These will be discussed in detail in the Master's thesis of this paper's first author [22].

### D. 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.* [23]) can be used for merging the LTMs back to the GTM; however, we propose 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.

### E. Using (Using Traceability Models for Change Impact Analysis and Status Reporting)

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 aggregated fields of the closed CRs of the previous baseline and selecting the relevant artifacts. Alternatively, the user can select a list of the known artifact affected by the CR in the GTM, and 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 LTM. The CR is then forwarded to the 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. Finally, the additional fields in a CR proposed in Section III-A 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."

The LTM in a CR can be used for several purposes. First, the LTM helps a developer understand the scope of the CR. Second, we propose using the LTM to also report the implementation status in a Scrum or other team meetings. Reporting the status of the implementation 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.

### F. Visualizing (Traceability Visualization During the CR Lifecycle)

While traceability visualization is not part of the process model, visualizing the traceability models in dashboards aids with traceability use and maintenance [3]. We begin with a description of the syntax of our traceability models.

**Abstract Syntax:** 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, we propose an additional attribute, *progress*, to indicate the progress of the implementation of the artifact for the given CR. As mentioned, a CR's LTM is a copy of part of GTM relevant to the CR. Therefore, there is one copy of the *progress* attribute for every artifact in every LTM. Moreover, the *progress* need not be numerical; it could also be an enumeration.

Additionally, we propose that the Change Request Ticket is not an artifact and therefore does not have traceability in this model. In simple terms, CRs are analogous to "flash cards" for understanding an artifact. However, the traceability among

CRs is implemented in the ITS, and we propose separating the two types of traceability to simplify the TIM.

**Concrete Syntax:** The traceability model's goals are to clearly and concisely represent traceability as well as to effectively use the model to report on the status of a ticket. We propose that the artifact be coloured to denote the status of the ticket: 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 Fig. 1. Also, in Fig. 1 the artifacts are represented as solid circles with the artifact ID and the artifact progress as the labels. The traceability is indicated with a green solid line (valid link) or a red dotted line (invalid link). The link label indicates the type of the traceability link [5]. For example, when a test case verifies a requirement, "verifies" could be the type of traceability link between the requirement and the test case.


Fig. 1. An example of the graphical syntax of the traceability models

We now describe traceability dashboards for visualizing the GTM and LTMs. In a project, the change request tickets are typically reviewed in Scrum/status meetings. Since the CR also holds the LTM, the CR dashboards can also be used to visualize traceability.

The integrated CR-traceability dashboard has two types of views: one to aid with change impact analysis, 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 dashboards is depicted in Figs. 2 and 3, 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.

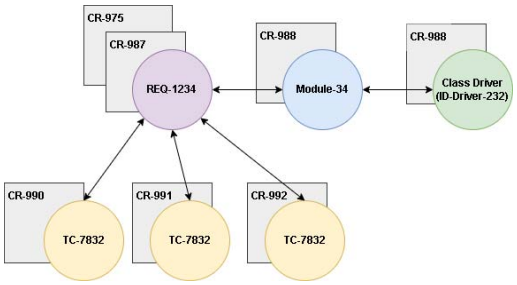


Fig. 2. A mock-up of the Change Impact Analysis View

The CIA view helps with the activities described in section-III-E. 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. This dashboard used in Scrum/status meetings would not only help with status updates but could also be a chance for other developers to provide ad-hoc feedback on traceability data. For example, during status

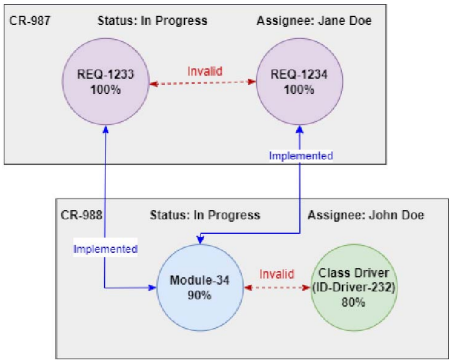


Fig. 3. A mock-up of the Build Tracking View

meetings, senior developers have a chance to review the CR and the LTM assigned to a junior developer. This could be an opportunity to point out any inconsistencies in traceability data.

### G. Use Case

Figure 4 The TIM and CR fields can be reviewed during sprint planning based on the aggregated feedback from the previous sprint. During the sprint, the developers could use the LTMs to understand the CR's scope and report the CR's status during the Scrum meetings. The developers may use the sprint retrospective meetings to capture feedback about the CR, which could be helpful in the subsequent sprints. Finally, in parallel, the relevant lead (*e.g.* the Product Owner) can use the CIA view (Fig. 2) to perform a CIA for new tickets.

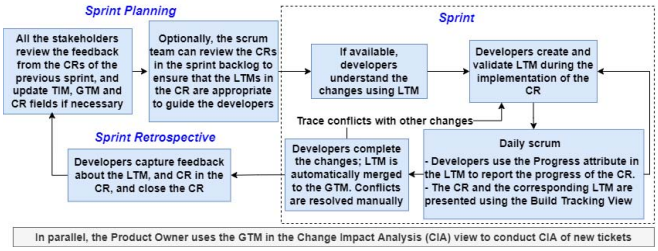


Fig. 4. Example of the process when applied to Agile (Scrum) software development

## IV. Implementing the Approach

To implement and validate this approach, we propose extending an existing ITS as explained next. The GTM is presented to the user in a graphical syntax editor (for simplicity of implementation), as read-only. Likewise, a graphical syntax editor is used for CR tickets; this editor is also used to build and validate an LTM. Model management operations, such as model querying, model validation and model merging, can be performed on the two traceability models.

Our plan is to start from an existing ITS such as Trac [24] and extend it with frameworks that are based on Eclipse

[25], such as the Eclipse Modeling Framework (EMF) for metamodel definition, the Graphical Modeling Framework (GMF) for the model editor, and Epsilon [26] for model management operations, especially querying. These elements are sufficient for building our proposed integrated process into the existing ITS.

## V. Conclusions, Challenges and Future Work

We have presented an early proposal of an integrated traceability-change management process. More precisely, we integrated the phases of the generic traceability process model with the lifecycle of a CR. Our integrated process can help the stakeholders seamlessly obtain the full benefits of traceability within an engineering process.

However, we face several challenges in this approach. The first is tool integration and trace granularity. The creation of LTMs rely on automated traceability techniques such as automated traceability recovery, mining pre-existing traceability links or EBTs. In the absence of the means to mine traceability links, the developer may have to resort to manual creation of traceability links, which may be ineffecient. Further, even if plugins exist, it is unclear whether the granularity desired in the TIM is obtainable through these plugins without modification.

The second challenge is developer engagement and the usability of the process. There are several dimensions to this. The first is the developers' proactive use of the LTMs to report status. If there are too many artifacts to report, it may be an additional burden on the developers to fill the progress attribute for every artifact. Therefore, simplification mechanisms must be put in place. Likewise, with the fields in CR tickets: if there are too many fields to be filled by a developer in a CR before closure, they may not fill them appropriately. Therefore, the fields in the CRs must not only pose the correct questions, but must not become a burden to developers.

The final challenge is the scalability of the process. For large organizations, whether the process caters to the needs of all the teams, and produces uniform traceability across the projects, remains an unanswered question.

We plan to address these challenges in the next steps of our research as follows. The first step is to demonstrate a proof of concept (PoC) with the extended ITS specified in Section IV. After the PoC is constructed, we will seek industry feedback on their current traceability practices, software development processes and the ITS used in their projects, and tailor our proposed process and PoC accordingly. Finally, we plan to validate and improve our process with experiments in an industrial setting.

## References


[1] S. Berczuk, B. Appleton, and R. Cowham, "The Trouble with Tracing: Traceability Dissected," https://www.cmcrossroads.com/article/trouble-tracing-traceability-dissected, Nov. 2005, accessed: 2022-05-06.

[2] J. Cleland-Huang, O. C. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Future of software engineering proceedings*, 2014, pp. 55–69.

[3] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE software*, vol. 30, no. 3, pp. 58–66, 2013.

[4] S. Maro, J.-P. Steghöfer, P. Bozzelli, and H. Muccini, "TracIMo: a traceability introduction methodology and its evaluation in an Agile development team," *Requirements Engineering*, vol. 27, no. 1, pp. 53–81, 2022.

[5] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. Maletic, and P. Mäder, *Software and Systems Traceability*. Springer, 2012, ch. Traceability Fundamentals, pp. 3–22.

[6] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, and J. Maletic, *Software and Systems Traceability*. Springer, 2012, ch. The Grand Challenge of Traceability v1.0, pp. 343–429.

[7] IEEE, "IEEE Standard for Configuration Management in Systems and Software Engineering," Institute of Electrical and Electronics Engineers, Standard, Mar. 2012, iEEE std 828-2012.

[8] K. Mohan, P. Xu, and B. Ramesh, "Improving the change-management process," *Communications of the ACM*, vol. 51, no. 5, pp. 59–64, 2008.

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

[10] B. Appleton, S. Berczuk, and R. Cowham, "Lean-Agile Traceability: Strategies and Solutions," https://www.cmcrossroads.com/article/lean-agile-traceability-strategies-and-solutions, Sep. 2007, online, accessed - 2022-06-15.

[11] J. Cleland-Huang, "Traceability in agile projects," in *Software and Systems Traceability*. Springer, 2012, pp. 265–275.

[12] J. Cleland-Huang, C. Chang, and M. Christensen, "Event-based traceability for managing evolutionary change," *IEEE Transactions on Software Engineering*, vol. 29, no. 9, pp. 796–810, 2003.

[13] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," in *Software and systems traceability*. Springer, 2012, pp. 71–98.

[14] T. W. W. Aung, H. Huo, and Y. Sui, "A literature review of automatic traceability links recovery for software change impact analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 14–24.

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

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

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

[18] SystemWeaver, "SystemWeaver," https://www.systemweaver.se/, May 2022.

[19] Siemens, "Polarion ALM," https://polarion.plm.automation.siemens.com/products/polarion-alm, May 2022.

[20] PTC Inc., "Windchill PLM Software," https://www.ptc.com/en/products/windchill, Jun. 2022, online. Accessed: 2022-06-15.

[21] P. Mader, O. Gotel, and I. Philippow, "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*. IEEE, 2009, pp. 21–25.

[22] Naveen Ganesh Muralidharan, "Integration Software Issue Tracking and Traceability Models," Master's thesis, McMaster University, Hamilton, ON, Canada, 2022.

[23] P. Mäder, O. Gotel, and I. Philippow, "Enabling automated traceability maintenance through the upkeep of traceability relations," in *European conference on model driven architecture-foundations and applications*. Springer, 2009, pp. 174–189.

[24] Egdewall Inc., "Trac," https://trac.edgewall.org/, Jun. 2022, online Accessed: 2022-06-20.

[25] Eclipse Foundation, "Eclipse Modeling Framework (EMF)," https://www.eclipse.org/modeling/emf/, Jun. 2022, accessed: 2022-06-20.

[26] Epsilon Development Team, "Eclipse Epsilon™," https://www.eclipse.org/epsilon/, Jun. 2022, online Accessed: 2022-06-20.