Cleland-Huang, Jane; Rayadurgam, Sanjai; Mäder, Patrick; Schäfer, Wilhelm:

Software and systems traceability for safety-critical projects : report from Dagstuhl Seminar 15162

Original published in:	Dagstuhl Reports / Schloss Dagstuhl, Leibniz-Zentrum für Informatik. - Wadern : Schloss Dagstuhl. - 5 (2015), 4, p. 76-97.
Original published:	2015-12-21
ISSN:	2192-5283
DOI:	10.4230/DagRep.5.4.76
[*Visited:*	2020-03-12]

Report from Dagstuhl Seminar 15162

Software and Systems Traceability for Safety-Critical Projects

Edited by

Jane Cleland-Huang[1], Sanjai Rayadurgam[2], Patrick Mäder[3], and Wilhelm Schäfer[4]

1 **DePaul University – Chicago, US,** `jhuang@cs.depaul.edu`
2 **University of Minnesota – Minneapolis, US,** `rsanjai@cs.umn.edu`
3 **TU Ilmenau, DE,** `patrick.maeder@tu-ilmenau.de`
4 **Universität Paderborn, DE,** `wilhelm@uni-paderborn.de`

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 15162 on "Software and Systems Traceability for Safety-Critical Projects". The event brought together researchers and industrial practitioners working in the field of safety critical software to explore the needs, challenges, and solutions for Software and Systems Traceability in this domain. The goal was to explore the gap between the traceability prescribed by guidelines and that delivered by manufacturers, and starting from a clean slate, to clearly articulate traceability needs for safety-critical software systems, to identify challenges, explore solutions, and to propose a set of principles and domain-specific exemplars for achieving traceability in safety critical systems.

Seminar April 12–17, 2015 – http://www.dagstuhl.de/15162
1998 ACM Subject Classification D.2.4 Software/Program Verification
Keywords and phrases safety-critical software development, assurance cases, software and systems traceability
Digital Object Identifier 10.4230/DagRep.5.4.76
Edited in cooperation with Patrick Rempel

1 Executive Summary

Jane Cleland-Huang
Sanjai Rayadurgam
Patrick Mäder
Wilhelm Schäfer

Safety-critical systems, defined as systems whose "failure could result in loss of life, significant property damage, or damage to the environment"[1], pervade our society. Developing software is a challenging process. Not only must the software deliver the required features, but it must do so in a way that ensures that the system is safe and secure for its intended use. To this end safety-critical systems must meet stringent guidelines before they can be approved or certified for use. For example, software developed for the aerospace industry must comply to the

[1] *Failure Analysis and the Safety-Case Lifecycle*, W. S. Greenwell, E. A. Strunk, and J. C. Knight in Human Error, Safety and Systems Development, 2004, http://dx.doi.org/10.1007/1-4020-8153-7_11.

ISO12207 and/or the DO-178B/C guidelines, while software developed for European railway communication, signaling, and processing systems, must comply to EN50128. Most guidelines prescribe a set of steps and deliverable documents that focus around planning, analysis and design, implementation, verification and validation, configuration management, and quality assurance activities. In addition they often provide specific guidelines for the creation and use of traceability in the project. For example, depending upon the criticality level of a requirement, the US Federal Aviation Authority guideline DO-178B requires traceability from requirements to design, and from requirements to source code and executable object code.

In practice, traceability is achieved through the creation and use of *trace links*, defined by the Center of Excellence for Software and Systems Traceability[2] as "specified associations between pair of artifacts, one comprising the source artifact and one comprising the target artifact". Software traceability serves an important role in demonstrating that a delivered software system satisfies its software design constraints and mitigates all identified hazards. When correct, traceability demonstrates that a rigorous software development process has been established and systematically followed. Current guidelines, in many safety-critical industries, prescribe traceability for two reasons. First, as an indirect measure that good practice has been followed, the general idea being that traceability information serves as an indicator that design and production practices were conducted in a sound fashion; and second, as a more direct measure, to show that specific hazards have been explored, potential failure modes identified, and that the system is designed and implemented in a "demonstrably rational way".

Unfortunately, there is a significant gap between prescribed and actual traceability. An analysis of the traceability information submitted by various organizations to the US Food and Drug Administration (FDA) as part of the medical device approval process [3], showed a significant *traceability gap* between the traceability expectations as laid out in the FDA's "Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices", and the traceability data documented in the submissions. While all of the submissions made some attempt to satisfy the FDA's traceability guidelines, serious deficiencies were found in almost all the submissions in terms of missing traceability paths, missing and redundant links, and problems in trace granularity. These deficiencies made it very difficult to understand the rationale for individual links. A more recent systematic analysis of seven software projects that originated from four different domains (automotive, aviation, medical, and space) revealed similar problems [4]. The provided software development artifacts were analyzed with respect to four technical guideline documents (ISO 26262-6, DO-178B, FDA Guide for Submissions, ECSS-E-40), where each document is a representative guideline of one of the four domains.

Problems are exacerbated in the systems engineering domain in which core concepts and designs are often documented across multiple models, each of which might depict a single viewpoint or perspective of the system. For example, the system might include separate models for functional and behavioral requirements, software components, electrical components, thermodynamics, and mechanical components. Furthermore, although each

[2] Center of Excellence for Software and Systems Traceability (http://www.CoEST.org)

[3] *Strategic traceability for safety-critical projects*, P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, IEEE Software, 30(3):58–66, http://dx.doi.org/10.1109/MS.2013.60.

[4] *Mind the gap: Assessing the conformance of software traceability to relevant guidelines*, P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, Proc. of the 36th Int'l Conf. on Software Engineering (ICSE'14), http://dx.doi.org/10.1145/2568225.2568290.

of these perspectives is modeled separately in isolation from one another, they interact to produce the final behavior of the system. Traceability solutions must extend across these heterogeneous models. Deficiencies in traceability are certainly not new. As far back as 1995, Gotel et al. identified several different traceability problems and attributed them to poor coordination, lack of perceived benefits, time to market pressures, and lack of sufficient tooling. These problems observed almost 20 years ago, continue to plague the traceability landscape today, meaning that the *traceability gap* between what is prescribed and what is practiced is still very real.

Given that the software and systems engineering communities have been unable to solve this problem in over 20 years, it seems prudent to reexamine traceability needs and their prescribed solutions. Within this Dagstuhl seminar, we engaged software and systems engineering researchers and practitioners from the safety-critical domain alongside traceability experts, in highly focused discussions. The aim was to gain a deeper understanding of exactly what traceability is needed for safety-critical systems, and to identify practical and achievable solutions. To the best of our knowledge this was the first time researchers from the safety-critical and traceability domains came together in a dedicated forum to tackle this problem.

We started the week with a number of more general presentations and discussions from experts in the respective areas to form a common understanding for later discussions. Subsequently, the seminar continued with shorter talks focusing on a variety of specific aspects of open challenges and potential solutions accompanied by intensive and highly interactive discussions. In parallel, we parted for about one third of the time into four focus groups working on what had been identified as the most relevant and urgent challenges for closing the traceability gap. The four areas of focus were: tracing qualities, traceability in the context of models and tools, cost-benefit and stakeholder perspectives, and traceability in the context of evolution and change. In result, we intend to publish a white-paper that systematically analyzes the existing traceability gap based on the outcome of the four focus groups. Furthermore, the workshop has initiated collaborations and potential research projects between previously separate areas with the potential of significant impact.

2 Table of Contents

3 Overview of Talks

3.1 Reusing Traceability for Change Impact Analysis – A Case Study in a Safety Context

Markus Borg (Lund University, SE)

Change Impact Analysis (CIA) during software evolution of safety-critical systems is a fundamental task closely related to traceability. However, CIA is difficult and labor-intensive for complex systems, and several authors have proposed tool support. Unfortunately, very few have been evaluated in industrial settings. In this talk, I will introduce our tool ImpRec, a Recommendation System for Software Engineering (RSSE), tailored for CIA at an automation company. Building on research from assisted tracing using information retrieval solutions, as well as mining software repositories, ImpRec recommends development artifacts potentially impacted when resolving incoming issue reports. In contrast to previous work on automated CIA, our approach explicitly targets development artifacts that are not source code. I will present results from the evaluation of ImpRec, designed as a two-phase industrial case study. In the first part, we measured the correctness of ImpRec's recommendations by simulating the historical inflow of 12 years' worth of issue reports in the company. In the second part, we assessed the utility of working with ImpRec by deploying the RSSE in two development teams. Our results suggest that ImpRec presents about 40% of the true impact among the top-10 recommendations. Furthermore, user log analysis indicates that ImpRec can support CIA in industry, and the developers in our study also acknowledged the value of ImpRec in interviews. In conclusion, our findings show the potential of reusing traceability associated with developers' past activities in an RSSE. However, more research is needed on how to retrain the tool once deployed, and how to adapt processes when new tools are introduced in safety-critical contexts.

3.2 Questioning the Traceability Requirements of Certifying Bodies

Jane Cleland-Huang (DePaul University – Chicago, US)

Software traceability is a sought-after, yet often elusive quality. It is required in safety-critical systems by many certifying and/or approving bodies, such as the USA Federal Aviation Authority (FAA) or the USA Food and Drug Administration (FDA). However, our previous study of medical device submissions to the FDA [1] highlighted the fact that adequate traceability was rarely achieved in practice. We identified numerous traceability problems including missing traceability paths, missing individual trace links, redundant paths, and inconsistencies. Furthermore, conversations with practitioners revealed that traceability is frequently built into a system as an afterthought, primarily for compliance purposes. In short, practitioners often perceive traceability effort as a burden, and rarely realize its benefits for supporting a broad range of software engineering activities for querying project data.

A traceability gap exists between what is prescribed by certifiers and what is delivered by product manufacturers [2]. This gap has several root causes. The traceability prescribed by certifiers tends to be overly extensive, requiring traceability paths between almost every pair of artifact types but providing only weak rationales for each path. Practitioners often fail to understand the need for such extensive traceability and therefore deliver only a subset of the requested trace links. This tends to be accepted by certifiers thereby reinforcing the practice of delivering less than the prescribed traceability. Further, current traceability tools fall short of providing adequate support for trace link construction and maintenance. Trace links are created using drag-and-drop mechanisms while potentially outdated links are as 'suspect' whenever the source or target artifacts are modified. Significant manual effort is therefore needed to establish and maintain traceability. More promising state-of-the-art approaches capture trace links as a natural byproduct of the workflow, for example by requiring developers to tag work requests with the associated code. The high cost of traceability can also potentially be reduced using automated trace retrieval, or by utilizing more intelligent traceability solutions capable of integrating natural language processing techniques and domain knowledge in order to reason about the presence of links.

Traceability serves two primary purposes in safety-critical systems, to demonstrate that due process has been followed, and to assess whether specific hazards, regulatory codes, and/or mitigating requirements have been implemented in the design. However, it is not clear whether current certification guidelines capture ideal traceability requirements. Recent interest in building safety- and assurance-cases suggests that a better approach might focus traceability efforts on connecting hazards, claims, and evidence for those claims in order to demonstrate product safety while simultaneously showing that good process was followed.

Solving the traceability gap is going to require a multi-pronged effort. Tracing practices will need to become a natural byproduct of the software engineering process. State of the art solutions for retrieving and/or reconstructing missing trace links will need to improve so that the links they generate can be trusted by human users. Finally, certifying bodies will need to rethink their prescribed traceability requirements – so that any cost and effort involved in the traceability process returns clear benefits to both developers and certifiers.

References

1 J. Cleland-Huang, O. Gotel, J.H. Hayes, P. Mäder, and Zisman, A (2014). "Software Traceability: Trends and Future Directions". In: *Proc. of FOSE'14/ICSE'14*, pp. 55–69.

2 P. Mäder, P.L. Jones, Y. Zhang, and J. Cleland-Huang (2013). "Strategic Traceability for Safety-Critical Projects", In: *IEEE Software*, 30(3), pp. 58–66.

3.3 Towards a Categorical Foundation of Model Synchronization

Krzysztof Czarnecki (University of Waterloo, CA)

Main reference Z. Diskin, A. Wider, H. Gholizadeh, K. Czarnecki, "Towards a Rational Taxonomy for Increasingly Symmetric Model Synchronization," in Proc. of the 7th Int'l Conf. on Theory and Practice of Model Transformations (ICMT'14), pp. 57–73, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08789-4_5

Model-driven engineering usually requires many overlapping models of a system, each supporting a particular kind of stakeholder or task. The consistency among these models needs to be managed during system development. Consistency management unfolds in the space of multiple model replicas, versions over time, different modeling languages, and complex relations among the models, which make the process complex and challenging.

This talk reports on our ongoing work to develop the theoretical foundation for model synchronization based concepts from category theory and illustrated its application to practical model synchronization problems. This theory views model synchronization as an algebra, abstracting from any concrete model structures and synchronization function implementations and with laws regulating the properties of the functions.

We have delivered several pieces of such foundation, including a precise notion of model overlap for two or more models expressed in one or more languages [4], a general notion of model mapping based on queries [7], and a general framework of delta lenses [5, 6, 8]. A lens, originally proposed by Benjamin Pierce et al., is a coordinated pair of functions: get to extract an abstract view from a source artifact and put to update the source to make it consistent with an updated view. In contrast to the original state-based lenses, delta lenses operate on model updates (represented as vertical model mappings) and traces relating the overlapping models (represented as horizontal model mappings) rather than model states only. Basing lenses on model mappings addresses limitations of the state-based setting, such as composition anomalies and inflexible signatures of the propagation functions [5]. We have also addressed the symmetric case of delta lenses [6], solving a long-standing problem of bidirectional transformations: too strong PUTPUT/undoability laws. We have also shown that our delta lens framework can be instantiated in the Triple-Graph-Grammar (TGG) setting, giving necessary and sufficient correctness conditions, checkable by tools [8]. Based on these concepts, we have recently constructed a design space of model synchronizers, capturing fundamental design choices, such as incrementality, and informational and organizational symmetry [10].

Practical application of delta lenses include bi-directional synchronization between models and code [1, 2] and between models specifying business processes and executable models implementing the specifications [11].

References

1 M. Antkiewicz and K. Czarnecki (2006). "Framework-Specific Modeling Languages with Round-Trip Engineering". In: *Proc. of MODELS'06*, 693–706.

2 M. Antkiewicz, T. Bartolomei, and K. Czarnecki (2009). "Fast extraction of high-quality framework-specific models from application code". In: *ASE*, 16(1), 101–144.

3 M. Antkiewicz and K. Czarnecki (2008). "Design Space of Heterogeneous Synchronization". In: *GTTSE'07, LNCS 5235*, 10, 3–46

4 Z. Diskin, Y. Xiong, and K. Czarnecki (2010). "Specifying Overlaps of Heterogeneous Models for Global Consistency Checking". In: *Proc. of MDI Workshop*, 42–51

5 Z. Diskin, Y. Xiong, and K. Czarnecki (2011). "From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case". In: *JOT*, 10(6), 1–25

6 Z. Diskin, Y. Xiong, K. Czarnecki, H. Ehrig, F. Hermann, and F. Orejas (2011). "From State- to Delta-based Bidirectional Model Transformations: the Symmetric Case". In *Proc. of MODELS'11*, 304–318

7 Z. Diskin, T. Maibaum, and K. Czarnecki (2012). "Intermodeling, queries, and Kleisli categories". In: *Proc. of FASE'12*, 163–177

8 F. Hermann, H. Ehrig, F. Orejas, F. Czarnecki, Z. Diskin, Y. Xiong, S. Gottmann, and T. Engel (2013). "Model Synchronization Based on Triple Graph Grammars: Correctness, Completeness and Invertibility". In: *SOSYM*, 14(1), 241–269

9 M. Branco, Y. Xiong, K. Czarnecki, J. Kuester, H. Voelzer (2014). "A case study on consistency management of business and IT process models in banking". In: *SOSYM*, 13(3), 913–940

10 Z. Diskin, A. Wider, H. Gholizadeh, K. Czarnecki (2014). "Towards a Rational Taxonomy for Increasingly Symmetric Model Synchronization". In: *Proc. of ICMT'14*, 57–73

11 J. Küster, H. Völzer, C. Favre, M. Branco, K. Czarnecki (2015). "Supporting Different Process Views through a Shared Process Model". In: *SOSYM*, 25 pages

3.4 Model-to-Model Traceability as a Key Enabler for Domain-Specific Safety Analysis

Christopher Gerking (Universität Paderborn, DE)

Safety-critical systems raise the need for formal verification at an early stage of the design process. Model checking is a verification technique that provides counterexamples in case of violated safety properties. Domain-specific model checking (DSMC) [1] hides the complexity of model checking by translating from a domain-specific language (DSL) to the input of a given model checker, and using traceability information to translate counterexamples back to the DSL. Our approach addresses the problem that existing settings assume only minor differences between DSL and model checking language, which allows for a single-step translation. This talk demonstrates how model-to-model traceability enables a back-translation of counterexamples even in case of major differences between DSL and model checking language. Our case study describes a successful application of DSMC to a multi-step translation scenario from the domain of interconnected cyber-physical systems.

References

1 W. Visser, M. B. Dwyer, and M. W. Whalen (2012). "The hidden models of model checking". In: *Software & Systems Modeling*, 11(4), 541–555

3.5 Runtime Traceability Challenges in Systems of Systems

Paul Gruenbacher (Universität Linz, AT)

This talk addresses challenges of using traceability links at runtime to diagnose problems in systems of systems (SoS). Specifically, it addresses traceability needs in setting up a requirements monitoring infrastructure for a system-of-systems architecture. The challenges are illustrated with examples from an industrial system of systems in the domain of metallurgical plants. Specifically, automated traceability techniques can support engineers defining requirements monitoring models. Better traceability between requirements and the SoS runtime architecture can further improve problem diagnoses after detecting violations of requirements.

3.6 Tracebility and the CoWolf framework

Lars Grunske (Universität Stuttgart, DE)

Agile and iterative development with changing requirements leads to continuously changing models. In particular, the researchers are faced with the problem of consistently co-evolving different views of a model-based system. Whenever one model undergoes changes, corresponding models should co-evolve with respect to this change. On the other hand, domain engineers are faced with the huge challenge to find proper co-evolution rules which can be finally used to assist developers in the co-evolution process. In the presentation, the CoWolf framework is introduced that enables co-evolution actions between related models and provides a tooling environment. Furthermore, the results of a case study for the co-evolution of architecture and fault tree models [1] are presented.

References

1 S. Getir, A. Van Hoorn, L. Grunske, M. Tichy (2013). "Co-evolution of software architecture and fault tree models: An explorative case study on a pick and place factory automation system". In: *Proc. of NiM-ALP@MoDELS'13*, 32–40

3.7 Model-based Reliability and Safety Engineering

Kai Hoefig (Siemens – München, DE)

Main reference K. Höfig, M. Zeller, L. Grunske, "metaFMEA-A Framework for Reusable FMEAs", in Proc. of the 4th Int'l Symp. on Model-Based Safety and Assessment (IMBSA'14), pp. 110–122, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-12214-4_9

Model driven development is currently one of the key approaches to cope with increasing development complexity, in general. Applying model-based approaches during the development of complex products aims at a systematic reuse of models or model elements and thus aims at a reuse of effort that already has been accomplished. A shorter time to marked and decreased development costs are strong drivers from industry. Domain specific languages or model elements come into play to handle complexity and ease the development of systems. Domain specific and universal modeling languages provide purpose-oriented views on a system model. The ability to include variation points is used if product lines are being developed. The overall strategy of divide and conquer brakes complexity down into manageable parts.

Applying similar concepts to safety engineering is a promising approach to extend the advantages of model driven development to safety engineering activities aiming at a reduction of development costs, a higher product quality and a shorter time-to-market. First, it makes safety engineering as a standalone subtask of system development more efficient. Second, and even more important, this is an essential step towards a holistic development approach closing the gap between functional development and safety engineering.

3.8 The Benefits of Traceability During Software Implementation

Patrick Maeder (TU Ilmenau, DE)

Joint work of Maeder, Patrick; Egyed, Alexander
Main reference P. Mäder, A. Egyed, "Do developers benefit from requirements traceability when evolving and maintaining a software system?", Empirical Software Engineering, 20(2):413–441, 2015.
URL http://dx.doi.org/10.1007/s10664-014-9314-z

Software traceability is a required component of many software development processes. Advocates of software traceability cite advantages like easier program comprehension and support for software maintenance (i.e., software change). However, despite its growing popularity, for a long time there existed no published evaluation about the usefulness of requirements traceability. It is important, if not crucial, to investigate whether the use of requirements traceability can significantly support development tasks to eventually justify its costs [3, 1, 4, 5]. We thus conducted a controlled experiment with 71 subjects re-performing real implementation tasks on two third-party development projects: half of the tasks with and the other half without traceability. Our findings show that subjects with traceability performed on average 24% faster on a given task and created on average 50% more correct solutions [2, 6] – suggesting that traceability not only saves effort but can profoundly improve software implementation quality. For a follow-up study [7], we selected medium to large-scale open-source projects and focused especially on the discovered effect for implementation quality. We quantified for each developed component of each software project, the degree to which a studied development activity was enabled by existing traceability and set this metric in relation to the number of defects that occurred in a component. We found that traceability significantly affects the defect rate in a component. Overall, our results provide for the first time empirical evidence that traceability significantly improves implementation speed as well as implementation quality during software development.

References

1 P. Mäder, O. Gotel, and I. Philippow (2009). "Motivation matters in the traceability trenches", In: *Proc. of RE'09*, pp. 143–148.

2 P. Mäder and A. Egyed (2012). "Assessing the effect of requirements traceability for software maintenance", In: *Proc. of ICSM'12*, pp. 171–180.

3 E. Bouillon, P. Mäder, I. Philippow (2013). "A Survey on Usage Scenarios for Requirements Traceability in Practice", In: *Proc. of REFSQ'13*, pp. 158–173.

4 P. Rempel, P. Mäder, and T. Kuschke (2013). "An empirical study on project-specific traceability strategies", In: *Proc. of RE'13*, 195–204.

5 P. Rempel, P. Mäder, T. Kuschke, and I. Philippow (2013). "Requirements Traceability across Organizational Boundaries – A Survey and Taxonomy", In: *Proc. of REFSQ'13*, pp. 125–140.

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

7 P. Rempel and P. Mäder (2015). "Estimating the Implementation Risk of Requirements in Agile Software Development Projects with Traceability Metrics", In: *Proc. of REFSQ'15*, pp. 81–97.

3.9 Model-based design inspection based on traceability information models and design slicing

Shiva Nejati (University of Luxembourg, LU)

Joint work of Nejati, Shiva; Sabetzadeh, Mehrdad; Briand, Lionel; Falessi, Davide
Main reference S. Nejati, M. Sabetzadeh, D. Falessi, L. C. Briand, and T. Coq (2012). "A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies", Information & Software Technology, 54(6): 569–590, 2012.
URL http://dx.doi.org/10.1016/j.infsof.2012.01.005

Traceability is one of the basic tenets of all software safety standards and a key prerequisite for certification of software. Despite this, the safety-critical software industry is still suffering from a chronic lack of guidelines on traceability. An acute traceability problem that we have identified through observing the software safety certification process has to do with the link between safety requirements and software design. In the current state of practice, this link often lacks sufficient detail to support the systematic inspections conducted by the certifiers of the software safety documentation. As a result, the suppliers often have to remedy the traceability gaps after the fact which can be very expensive and the outcome might be far from satisfactory.

The goal of our work is to provide a traceability methodology and a design slicing algorithm for software safety certification by applying and specializing the Systems Modeling Language (SysML). Our methodology enables the establishment of traceability links prescribed by a traceability information model as well as a mechanism for extracting a minimized and relevant slice of the design with respect to the specified traceability links. The certifiers can then utilize the links and the design slices to effectively investigate safety claims. To validate our approach, we report on an industrial case study applying the approach to a safety IO software module used on ships and offshore facilities.

In this talk, I describe the context in which the above work was carried out, explain our proposed solutions, and discuss how our solutions have been applied to case studies from the Maritime and Energy domain.

3.10 Traceability Through Precise Process Definitions

Leon J. Osterweil (University of Massachusetts – Amherst, US)

Traceability should be viewed as a property needed support tracing, whose purpose should be viewed as the gathering and maintenance of key knowledge and understandings about software products. Precise and detailed definitions of the processes by which these products are developed, tested, and evolved are excellent vehicles for continuous creation and maintenance of the inter- and intra- artifact links that provide the desired traceability. This talk describes the Little-JIL process definition language and how it can be used to create processes that can be used to create these links and support this kind of traceability.

3.11 Evolving Trace Links across Versions of a Software System in Safety-Critical Domain

Mona Rahimi (DePaul University – Chicago, US)

Trace links provide critical support for numerous software engineering activities including safety analysis, compliance verification, test-case selection, and impact prediction in safety-critical systems. However, as the system evolves over time, there is a tendency for the quality of trace links to degrade into a tangle of inaccurate and untrusted links. This is especially true with the links between source-code and upstream artifacts such as requirements-because developers frequently refactor and change code without updating links. We present TLE(Trace Link Evolver), a solution for automating the evolution of trace links as the change is introduced to source code. We use a set of heuristics, open source tools and information retrieval methods to detect common change scenarios in different versions of code. Each change scenario is the associated with a set of link evolution heuristics which are used to evolve trace links. We evaluated our approach through a controlled experiment and also through applying it to a selection of classes taken from Cassandra Database System. Results show that the trace links produced by our approach is significantly more accurate than those produced using information retrieval alone.

3.12 Medical Device Verification and Validation: Experiences and Perspectives

Sanjai Rayadurgam (University of Minnesota – Minneapolis, US)

Medical devices such as infusion pumps and pacemakers are safety-critical systems that are strictly regulated by government agencies. The safety of these devices must be demonstrably established prior to gaining approval for sale. Assurance cases, which are structured arguments that use evidence gathered through the course of development to establish desirable claims, are being used, and in some cases mandated, to establish safety of these devices. The feasibility and the merits of such arguments are critically dependent on traceability across all development process artifacts, which provide the evidence for the assurance case. Maintaining and evolving traceability information throughout the development process is challenging. In particular, showing that requirements are realized in specific design elements and that realization has been verified is necessary. Often, requirements and architecture co-evolve [4] and so attempting to specify one without the other leads to inconsistent specifications.

When the architecture is formally modeled and the requirements are decomposed along architectural lines, compositional verification techniques can be used to prove that the components satisfying their requirements and interacting as specified by the architecture, are sufficient to ensure that the system meets its requirements [3]. However, this is typically insufficient to make a complete argument for claiming verification. Components at the lowest levels of the architectural decomposition, have to be elaborated below into realizable implementations and their behavior verified or tested to check conformance to their respective requirements. Above the architectural model, there may be models of usage scenarios for the system in its context, which may then be checked to validate that the system as specified would meet its intended needs. To tie these together into a complete satisfaction argument for safety claims [2], trace links that span multiple models and relate elements of that model at finer levels of granularity are needed. Further, the assurance arguments and consequently the trace links that enables argumentation must evolve along with the corresponding artifacts throughout development. In general, the models developed during the course of system building support several activities such as simulations, analysis, verification and code-generation. The relationships between the models, and the relationships established by these activities are essential to the assurance arguments. When architectural designs are annotated with logical rules of argumentation, generation of assurance cases can be automated [1].

A few observations related to traceability in this regard merit consideration. First, requirements placed on traceability solutions for safety-critical medical devices include both a stable core (that is mandated by regulations) as well as an evolving frontier (that is driven by changes in development methods and tools employed). Second, while automation can speed up several traceability related tasks, it is not helpful especially when the output produced has to be manually analyzed. Strategic thinking is needed to strike a good balance between automation and manual analysis. Third, questions of provenance tend to be difficult to answer, but those are the most useful for constructing good assurance arguments. Fourth, supporting multiple viewpoints of various stakeholders during development requires incremental evolution of trace semantics because the stakeholder requirements also evolve throughout development.

References

1 A. Gacek, J. Backes, D. Cofer, K. Slind, and M. Whalen (2014). "Resolute: An assurance case language for architecture models", In: *Proc. of HILT '14*, pp. 19–28.

2 A. Murugesan, O. Sokolsky, S. Rayadurgam, M. Whalen, M. Heimdahl, and I. Lee (2014). "Linking abstract analysis to concrete design: A hierarchical approach to verify medical CPS safety", In: *Proc. of ICCPS'14*, pp. 139–150.

3 A. Murugesan, M. W. Whalen, S. Rayadurgam, and M. Heimdahl (2013). "Compositional verification of a medical device system", In *Ada Lett.*, 33(3), pp. 51–64.

4 M.W. Whalen, A. Gacek, D. Cofer, A. Murugesan, M. Heimdahl, and S. Rayadurgam (2013). "Your 'What' Is My 'How': Iteration and Hierarchy in System Design", In: *IEEE Software*, 30(2), pp. 54–60.

3.13 Traceability Asessment and Roadmap for Medical Device Domain

Gilbert Regan (Dundalk Institute of Technology, IE)

Within the medical device domain, as in other safety critical domains, software must provide reliability, safety and security because failure to do so can lead to injury or death. Software is a complex element of a medical device; it's role, functionality and importance continually increases. Additionally due to changes in the 2007 medical device directive (MDD 2007/47/EC), standalone software can now be classed as an active medical device in its own right. Developing medical device software-based systems in a disciplined and cost-effective way poses major challenges (esp. with the move towards mobile devices, patient-driven applications, wireless devices and cloud-based solutions). Therefore highly effective software practices are required. Additionally, regulation normally requires safety critical systems are certified before entering service. This involves submission of a safety case (a reasoned argument that the system is acceptably safe) to the regulator. A safety case should include evidence that the organization has established effective software development processes that are based on recognized engineering principles appropriate for safety critical systems. At the heart of such processes, they must incorporate traceability.

However, numerous barriers hamper the effective implementation of traceability such as cost, complexity of relationship between artifacts, calculating a return on investment, different stakeholder viewpoints, lack of awareness of traceability and a lack of guidance on what traceability to implement and how to implement it. There are a number of standards which medical device manufacturers must conform to, however these standards have different traceability requirements. This leads to confusion as to what traceability manufacturers should implement. Additionally medical device software manufacturers are often very small organisations with little experience in traceability, and, in addition to 'what traceability to implement', they are often unsure as to 'how to implement it'.

In Ireland the importance of the medical device Industry is obvious from its contribution of 8.5% of Ireland's total merchandise exports, and this sector has been identified as one of the key drivers of industrial growth for the future. Consequently the Irish government fund research into how medical device organizations can improve their software development process. The implementation of traceability through the software development lifecycle and supporting processes of risk management and change management has been identified as a weakness within these medical device organizations. To assist these organizations improve their implementation of traceability, a decision was taken to address the 'lack of guidance' on what traceability to implement and how to implement it. This decision lead to the development of the following research question: "To what extent can the development of a traceability assessment and implementation framework assist medical device software organizations improve their traceability practices and put them on the path to regulatory compliance?"

To answer this question a traceability process assessment model and a roadmap for the implementation of traceability have been developed. In this presentation the experience of developing and trialling a traceability assessment model in two medical device organizations is presented. We show that the assessment model was successful in identifying strengths and weaknesses in both organisations implementation of traceability. Additionally a roadmap to assist organizations implement traceability that is both efficient and compliant is presented. Finally, through the experience of trialling the assessment model in two medical device organizations, I think the traceability assessment model could be improved through automation and so propose an initial idea of using the Open Services for Lifecycle Collaboration (OSLC) initiative.

3.14 Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines

Patrick Rempel (TU Ilmenau, DE)

Joint work of Rempel, Patrick; Mäder, Patrick; Kuschke,Tobias; Cleland-Huang, Jane
Main reference P. Rempel, P. Mäder, T. Kuschke,J. Cleland-Huang, "Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines", in Proc. of the 36th Int'l Conf. on Software Engineering (ICSE'14), pp. 943–954, ACM, 2014.
URL http://dx.doi.org/10.1145/2568225.2568290

Many guidelines for safety-critical industries such as aeronautics, medical devices, and railway communications, specify that traceability must be used to demonstrate that a rigorous process has been followed and to provide evidence that the system is safe for use. However, practitioners rarely follow explicit traceability strategies [2, 1]. Organizations struggle to establish and maintain accurate and complete sets of traceability links [3, 4]. In practice, there is a gap between what is prescribed by guidelines and what is implemented in practice, making it difficult for organizations and certifiers to fully evaluate the safety of the software system [5]. We present an approach, which parses a guideline to extract a Traceability Model depicting software artifact types and their prescribed traces. It then analyzes the traceability data within a project to identify areas of traceability failure [7]. Missing traceability paths, redundant and/or inconsistent data, and other problems are highlighted. We used our approach to evaluate the traceability of seven safety-critical software systems and found that none of the evaluated projects contained traceability that fully conformed to its relevant guidelines [6].

References

1 P. Mäder, O. Gotel, and I. Philippow (2009). "Getting back to basics: Promoting the use of a traceability information model in practice", In: *Proc. of TEFSE@ICSE'09*, pp. 143–148.
2 P. Mäder, O. Gotel, and I. Philippow (2009). "Motivation matters in the traceability trenches", In: *Proc. of RE'09*, pp. 143–148.
3 P. Rempel, P. Mäder, and T. Kuschke (2013). "An empirical study on project-specific traceability strategies", In: *Proc. of RE'13*, pp. 195–204.
4 P. Rempel, P. Mäder, T. Kuschke, and I. Philippow (2013). "Requirements Traceability across Organizational Boundaries – A Survey and Taxonomy", In: *Proc. of REFSQ'13*, pp. 125–140.
5 P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang (2013). "Strategic Traceability for Safety-Critical Projects", In: *IEEE Software*, 30(3), pp. 58–66.
6 P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang (2014). "Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines", In: *Proc. of ICSE'14*, pp. 943–954.
7 P. Rempel and P. Mäder (2015). "A Quality Model for the Systematic Assessment of Requirements Traceability", In: *Proc. of RE'15*, 8 pages.

3.15 An Analysis of Challenges in Safety Certification and Implications for Traceability Research

Mehrdad Sabetzadeh (University of Luxembourg, LU)

Many safety-critical systems in domains such as healthcare, aviation, and railways are subject to safety certification. The goal of safety certification is to provide confidence that a system will function safely in the presence of known hazards. Safety certification can be associated with the assessment of products, processes, or personnel. For software-intensive safety-critical systems, certification of products and processes are the most challenging.

In my talk, I will discuss several challenges in the safety certification of software-intensive systems. These challenges were gleaned from a large systematic review of the academic literature on safety certification, a number of practitioner surveys, and my personal experience working with the safety-critical software industry. I will argue that safety certification is closely intertwined with traceability, and that many of the challenges faced in safety certification today are caused by traceability gaps.

At the end, I will briefly present some technical work from our research group lying at the intersection of safety certification and traceability research.

3.16 Traceability in the Nuclear Energy Industry. Challenges and Lessons Learned from an Industrial Project

Nicolas Sannier (University of Luxembourg, LU)

Joint work of Sannier, Nicolas; Baudry, Benoit
Main reference N. Sannier, B. Baudry, "INCREMENT: A Mixed MDE-IR Approach for Regulatory Requirements Modeling and Analysis," in Proc. of the 20th Int'l Working Conf. on Requirements Engineering: Foundation for Software Quality (REFSQ'14), pp. 135–151, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-05843-6_11

The basic intuition behind any thermal power plant (independently of the primary resource they use: coal, gas, oil or uranium) is rather simple. Legitimate safety issues and safety measures to prevent catastrophic accidents or mitigate their consequences make these systems incredibly complex, and the more complex, the harder the safety qualification. Instrumentation and Control (I&C) systems allow to measure and control the plant's behavior. Since 1986, I&C Systems are now mostly made of software systems and represent the toughest part for safety qualification. Those important to safety must conform to safety and regulatory requirements. Regulatory requirements are written by national safety authorities and are completed using a set of national recommendation guides or national and international standards. All these documents are weakly interrelated. Due to the lack of international consensus on regulatory practices, building such systems in different countries requires facing practices of several safety authorities. In order to minimize design and qualification effort, traceability between regulatory requirements became suddenly important. These observations set three important challenges. First, the global domain knowledge is scattered, not formalized and hold by few experts. Second, traceability links and, said differently, the organization within the domain, is implicit. The third problem is the consequence of the two firsts. Bridges between different national practices are not developed, whereas the understanding of regulations and practices becomes a significant industrial issue. In our context, traceability comes with two facets. Traceability basically means linking one element to another. However, two elements are basically linked for a particular purpose. In this case, traceability also means organizing the domain. (1) We need to define different types of traceability links between elements of our domain. It is one thing to define traceability links. It is another one to concretely present the relationship(s) between two artifacts as traceability links. In the modeling community, people are used with diagrams, where traceability between elements can be visualized as arrows between two boxes. However, for this particular project, nuclear engineers are mainly working with a large amount (thousands) of textual fragments, using excel spreadsheets. This representation put the text as first class citizen, and traceability cannot be handled properly in a class diagram neither in a matrix for such a huge amount of data. (2) We need to provide a convenient way to present these traceability links. Another concern worth considering is how to implement these traceability links. Said differently, it will be more than helpful to automatically build these traceability links. In our context, building traceability links between requirements means investigating relationships between requirements from the same corpus, and from different corpora in order to find similarities and possible coverage between requirements. In practice, these traceability links rarely exist in the regulation (regulations may have been written before the standards and were not always updated, practices may have changed). Moreover, these texts are generic, do not target any particular system and are voluntary ambiguous in order to last along the years. Finally regulations, standards and their interpretation as well as practices widely differ from one country to another. Consequently, there is no straightforward mapping between safety requirements. (3) Before creating traceability links between elements, we must first reduce the amount of elements to compare and analyze. In this project, we investigated different areas of research. We used Metamodeling to model the domain and precisely define the traceability links we wanted to represent. Modeling is good for formally defining domain elements and their relationship, however, it is not useful for representing information that is mainly textual and "would never fit in a box", neither it is to analyze these textual fragments. For the latter, we investigated the use of different techniques such as overlapping clustering algorithms, machine learning and topic detection, and information retrieval. The objective was to be able to build topic clusters and reduce the size of the search space. We draw some observations from these experiments. In particular, machine learning and clustering approaches were considered suspicious by our industry partners, as defined topics could not be legitimately argued and validated. Results were "not good enough" to impose confidence. On the other hand, Information Retrieval performed reasonably well and received more positive feedbacks as the querying tool we proposed made more sense to them as it was in line with the support our industry partners were expecting. Among the many lessons that can be learned with respect to traceability, we can highlight the following ones:

- Traceability is not only a matter of linking objects together; it is also a matter of amount of objects to link together.
- Traceability techniques may also require trust and understandability to be accepted.

3.17 Systems Engineering and Traceability at the Model Level

Wilhelm Schäfer (Universität Paderborn, DE)

Todays embedded and often safety-critical systems require traceability from requirements down to the implementation in terms of software and hardware. The talk presents a systematic V-model based approach which includes a discipline spanning model in the requirements and early design phase. This model consists of seven views which cover all disciplines as for example a so-called active structure. The active structure includes the definition of system components and energy, material and information flow between the components. Other views define scenarios or use cases and the underlying abstract shape model which comes from CAD. Partially automatic transformations which are based on a formal , semantically well-defined mechanism, define how discipline-specific models are derived. These transformations form the basis for defining traces between all concerned models.

3.18 Gene-Auto & QGen: Experiences and ideas on ACG specification, qualification and verification

Andres Toom (IB Krates OÜ – Tallinn, EE)

Automatic Code Generators (ACG) or model transformers make Model Driven Engineering (MDE) really powerful and have a great potential for reducing human errors and assisting certification by having an ability to generate together with the expected output also complete and consistent trace data. However, ACGs are complex tools themselves and need to be also qualified/certified.

This presentation reports on the experiences of two consecutive collaborative initiatives Gene-Auto [1] and its continuation carried out in two affiliated projects Project P [2] and Hi-MoCo [3], aiming at developing open-source code generators for safety critical domains. These code generators are designed to transform high-level modelling languages such as Simulink, Scicos and Stateflow to low-level program code in languages such as C or Ada. Since the intended end-user domains include avionics, their qualification plans have been set up according to the DO-178 B/C software qualification guideline. As this guideline is also one of the most stringent and refined ones among the safety critical domains, it is expected to be relatively easy to apply the results also in other domains. The outcomes of these initiatives are due to the effort of several organisations and many individuals over several years. References of the contributors can be found from the provided websites.

Gene-Auto was the first project, with most of the development taking place during an ITEA project in 2006-2008. The goals of the project included the clarification of the requirements for such a tool across different domains, investigating the qualification of Java language based software, and usage of formal methodologies such as development with a formal proof assistant in a qualifiable tool development process. Below is a brief summary of all of these outcomes.

The project went through several iterations and ended with a rather mature prototype ACG. The high-level user requirements and low-level software requirements (architecture and component requirements) were considerable refined, but the global functional requirements (resulting from the toolset integration) were not yet explicitly specified. Similarly, source code and test cases were not explicitly traced to the requirements. For this project, differently from some more typical critical software development tasks, this was a rather natural process. As the requirements were largely clarified during an iterative process, rigorous management of trace links at all artifact levels was considered not that useful at that stage. Also, since the toolset had a very clear architectural design the risk of software deviating from high-level requirements was minimal.

The part of requirements that handled language definitions were formalised as metamodels and grammars. Other parts were specified as tagged Open Office documents. These documents were parsed and analysed by the Tramway (Topcased-Requirements) tool. The requirement split-down and coverage analysis performed by Tramway was rather useful. However, document based requirement management became soon complicated in terms of version and change management.

As for the qualification of Java-based software, then the main open questions remained related to the qualification of the Java Virtual Machine (JVM) and libraries, both native and external (such as parser generator, XML library). The advantage of Java is that it has a large user community and rich functionality. However, the libraries have much more functionality than is needed for a tool with concise, but safety critical functionality. One would either have to qualify or remove this extra functionality. Both options have considerable cost.

A specific procedure in the qualification plan was also set up for component development with a proof assistant. In particular the Coq proof assistant was used. In this process the initial component requirements specification phase is followed by a formal specification phase and a rather minimal design phase. Software code (with the exception of some interface and glue code) is automatically generated from the proof of the properties expressed in the formal specification. The program extraction technology has been developed out earlier and has been used on some tools of considerable complexity (e.g. the CompCert compiler by X. Leroy et al.). The certification experts considered the process outlined above acceptable for the qualification of such a tool possible in the long term. However, all the components used in the process, including the Coq kernel and program extractor need to be qualified. On the other hand, usage of such deep formal methods and tools in the industry was not considered possible by the Gene-Auto industrial partners at the current time.

The Gene-Auto initiative was continued in the joint follow-up projects Project P and Hi-MoCo, which laid the foundations for the QGen [4] code generator. The requirements of Gene-Auto have been refined and extended and different technical platforms are used to ease the tool qualification process. The main implementation language of the tool is Ada, which has been developed specifically for safety critical domains. Only a minimal set of already qualified external Ada libraries are used. Currently, the complete tool-chain has been implemented in Ada. The toolset has a standard Ecore metamodel-based interface for exporting-importing models between the transformation steps. This enables substituting elementary transformation steps by components implemented using other technologies (e.g. formal methods-based), when they are at a sufficient maturity level for tool qualification.

All qualification artifacts for QGen are managed using the Qualifying Machine (QM) [5] tool. The main functionalities of this tool currently include importing artifacts from different formats, analysing and displaying them, and (to some extent) also modifying via a common user interface. The low level requirements are written as structured annotations or formal pre/post conditions to the Ada spec (.ads) files that specify the public interface of source code modules of QGen. This way it is easy to ensure that all the public functions have associated requirements, as well as update either the source code or low level requirement each time one of them changes. When requirements are expressed as formal invariants or pre-post conditions, mismatches between the requirement and implementation can be also automatically detected. Mapping between the test cases and requirements is achieved by explicit QM cross-links. Overall, at this stage the QGen tool maturation as well as qualification process and data refinement are still in progress. In addition we are performing complementary studies with more formal, but light-weight approaches that are close to the current state of the art and practice in the industry. These experiments include the formal specification of block libraries for dataflow languages [6] and transformation contracts for the specification of model transformations [7]. However, it is evident that most factors that complicated and impeded the qualification process of the Gene-Auto project have been refined and resolved in the QGen project.

References

1 Gene-Auto project (2006-2008). http://www.geneauto.org.
2 Project P (2011–2015). http://www.open-do.org/projects/p.
3 High-Integrity Model Compiler (Hi-MoCo) project (2011-2014). http://www.eurekanetwork.org/project/-/id/6037.
4 QGen tool (2015). http://www.adacore.com/qgen.
5 Qualifying Machine (QM) project (2015). http://www.open-do.org/projects/qualifying-machine.
6 A. Dieumegard, A. Toom, and M. Pantel (2014). "A software product line approach for semantic specification of block libraries in dataflow languages", In: *Proc. of SPLC'14*, pp. 217–226.
7 A. Toom, A. Dieumegard, M. Pantel (2014). "Specifying and verifying model transformations for certified systems using transformation models", In: *Proc. of ERTS2'14.*

3.19 Model-based safety engineering: Challenges and opportunities in practice

Marc Zeller (Siemens – München, DE)

Joint work of Zeller, Marc; Hoefig, Kai

The technology path MbRSE develops and integrates models and methods for the model-driven engineering of critical systems. MbRSE stands for Model-based Reliability and Safety engineering and is primarily motivated by the challenges of dynamic reconfigurable cyber-physical systems.

We provide Siemens business units with top-notch technologies to establish systematic reuse of critical development artifacts and security-aware runtime certification.

Our methods and models enable divide and conquer strategies for critical systems development to reduce effort and increase quality of Siemens products.

Participants

- Markus Borg
Lund University, SE
- Jane Cleland-Huang
DePaul University – Chicago, US
- Krzysztof Czarnecki
University of Waterloo, CA
- Christopher Gerking
Universität Paderborn, DE
- Paul Grünbacher
Universität Linz, AT
- Lars Grunske
Universität Stuttgart, DE
- Kai Höfig
Siemens – München, DE
- Patrick Mäder
TU Ilmenau, DE
- Shiva Nejati
University of Luxembourg, LU
- Leon J. Osterweil
University of Massachusetts – Amherst, US
- Mona Rahimi
DePaul University – Chicago, US
- Sanjai Rayadurgam
University of Minnesota – Minneapolis, US
- Gilbert Regan
Dundalk Institute of Technology, IE
- Patrick Rempel
TU Illmenau, DE
- Mehrdad Sabetzadeh
University of Luxembourg, LU
- Nicolas Sannier
University of Luxembourg, LU
- Wilhelm Schäfer
Universität Paderborn, DE
- Andres Toom
IB Krates OÜ – Tallinn, EE
- Marc Zeller
Siemens – München, DE