The Journal of Systems & Software 192 (2022) 111389

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

In Practice

When traceability goes awry: An industrial experience report☆

Davide Fucci [a,*], Emil Alégroth [a], Thomas Axelsson [b]

[a] SERL, Blekinge Institute of Technology, Valhallavagen 1, Karlskrona, 37141, Sweden
[b] COMPANY, Karlskrona, Sweden

ARTICLE INFO

Article history:
Received 13 October 2021
Received in revised form 28 February 2022
Accepted 31 May 2022
Available online 13 June 2022

MSC:
68N01

Keywords:
Industry-academia collaboration
Traceability
Software quality

ABSTRACT

The concept of traceability between artifacts is considered an enabler for software project success. This concept has received plenty of attention from the research community and is by many perceived to always be available in an industrial setting.

In this industry-academia collaborative project, a team of researchers, supported by testing practitioners from a large telecommunication company, sought to investigate the partner company's issues related to software quality. However, it was soon identified that the fundamental traceability links between requirements and test cases were missing. This lack of traceability impeded the implementation of a solution to help the company deal with its quality issues.

In this experience report, we discuss lessons learned about the practical value of creating and maintaining traceability links in complex industrial settings and provide a cautionary tale for researchers.

1. Introduction

The concept of traceability refers to links between different software artifacts, such as requirements, source code, and test cases. Establishing traceability within requirements helps the organization manage dependencies. In contrast, traceability between requirements and test cases, also referred to as alignment (Unterkalmsteiner et al., 2014), is necessary to measure coverage and ensure that the product fulfills customers' needs with a given degree of quality. Moreover, the traceability between source code and test cases enables the analysis of the impact when changes occur.

In this study, Blekinge Institute of Technology (BTH) collaborated with a large telecommunication company (hereafter, Company[1]), with the original intent to study Company's software quality. Specifically, the researchers collaborated with the organization within Company in charge of the verification before release (i.e., TestOrg). The initial aim of this collaboration was to investigate the challenges related to the state-of-practice of the Company quality assurance process. TestOrg was a suitable partner in this project since they are responsible for the quality assurance of all Company products and can be considered the first user of the product once it leaves the development phase. BTH and TestOrg organized a series of workshops to elicit pain points regarding software quality. One outcome of such workshops was the perceived mismatch between the organization level of quality assurance activities and the actual observed quality. We termed this difference *quality inflation*. The study of the quality inflation phenomenon at Company became the refined objective of our industrial collaboration.

The researchers analyzed the artifacts TestOrg uses to perform their activities, such as requirement specifications, issue reports, and test cases. To observe the symptoms and connect them to possible root causes of quality inflation, the researchers needed to connect the requirements to other artifacts in the development process. Therefore, the existing traceability links and the possibility of establishing new ones play a significant role for detecting quality inflation. However, we found that such traces were generally lacking and that reverse engineering these traces from existing artifacts was not only time-consuming but also, in most cases, not possible.

In this paper, we report our experience in detecting quality inflation at Company and how the investigation of this phenomenon failed due to the lack of traceability between artifacts. This experience serves as a case for academia to consider for future research and provide an example for practitioners of the challenges that can arise once traceability between software development artifacts is lacking.

The rest of the paper is organized as follows: Section 2 presents an overview of the existing literature on traceability in software projects and its application to different artifacts in the software development lifecycle. Section 3 presents the industrial context of our study, and Section 4 shows our experience in trying to

☆ Editor: Marcos Kalinowski.

* Corresponding author.
E-mail addresses: davide.fucci@bth.se (D. Fucci), emil.alegroth@bth.se (E. Alégroth), thomas.axelsson@telia.com (T. Axelsson).

[1] We omitted the Company name due to their wish to stay anonymous.

https://doi.org/10.1016/j.jss.2022.111389

establish traceability links in a large-scale organization to investigate a particular aspect of software quality—i.e., quality inflation. Section 5 reports a discussion of our experience and takeaways for researchers and practitioners. Finally, Section 6 concludes the paper.

2. Software artifacts traceability

The body of research in traceability is vast, with early works published in the 70s—e.g., Randell (1968). Since then, the concept has been explored in both industry and academia through empirical studies and summarized in systematic literature reviews (Charalampidou et al., 2021a; Tian et al., 2021; Nair et al., 2013). Traceability is essential to govern the software development process, to manage complexity, and mitigate costs (Watkins and Neal, 1994; Kukkanen et al., 2009). However, research has also established that maintaining traceability links of high quality over time can be a costly process (Cleland-Huang et al., 2003). Therefore, research has started focusing on traceability automation (Hayes et al., 2007).

Among the secondary studies, Mustafa and Labiche performed a literature review that identified a lack of tools for tracing heterogeneous artifacts (Mustafa and Labiche, 2017). Conversely, a review by Tufail et al. (2017), identified seven models, ten challenges, and 14 tools for traceability. The review by Javed and Zdun examined the connections between traceability and software architecture (Javed and Zdun, 2014). In contrast, Santiago et al. looked at managing traceability in the context of model-driven engineering and the associated complexities (Santiago et al., 2012). According to a recent mapping study surveying 63 papers between 2000 and 2020, traceability involving testing artifacts and related activities is the least investigated (Tian et al., 2021). Moreover, according to the authors, few tools support traceability in software testing activity (Tian et al., 2021). This conclusion is drawn, despite the numerous research on the tools that either explicitly focus on, or support, traceability in software development. Examples of such tools include PLM/ALM which, according to a study by Ebert (2013), was beneficial to define traceability for testing purposes. The project management tool `DOORS` is commonly used in the industry and is used as a driver for research into, for example, automated traceability links (Lin et al., 2006) and rich traceability (Dick, 2002). Another example is `Enterprise architect`—a large-scale modeling framework that can be used to model traceability between several aspects of development, including assets, processes, and the organization (Tang et al., 2007). Although used successfully in practice, these tools share a human component. Thereby, their success is tied to how rigorously they are used.

Tian et al. (2021) show that traceability in different software development activities has rarely been evaluated in industrial settings (i.e., 16% of the reviewed primary studies). However, despite several studies commending traceability as a prerequisite for a successful software project (and conversely pointing to a lack of traceability as a factor leading to failure (Fernández et al., 2017)), there is no empirical evidence, to the best of our knowledge, that supports these claims in the industrial context.

Traceability is often discussed as a sequential trace from requirements to code, and from code to test cases. Another critical dimension is alignment—i.e., the traceability between requirements and tests. Unterkalmsteiner et al. (2014) proposes a taxonomy for requirements and test alignment (REST). They show a method for designing contextual taxonomies of alignment, including concepts for how to reason about the establishment of traceability links from requirements specifications to design, development, and testing.

Another concept related to traceability is change management (Borg et al., 2017)—i.e., the idea that when a software artifact is changed, all associated artifacts need to be identified and updated accordingly. The responsibility for this task is often delegated to the development team, and its complexity is affected by the already available traceability links. In their case study, Borg et al. found that developers prefer flexible forms of information rather than formal information, including traceability information, when dealing with changes (Borg et al., 2017).

Both the research on requirements-tests alignment and change management demonstrate an academic idea of the *importance* of traceability in software development. This suggestion, although reasonable, does not account for the costs of keeping traceability and alignment up-to-date in complex industrial settings.

3. Investigation of software quality within COMPANY

BTH and COMPANY are collaborating on a research project in the area of software quality, with a focus on automated software testing. This project is enabled by BTH's approach to Industry-Academia collaboration and technology transfer based on the model proposed by Gorschek et al. (2006). Since the project is performed in co-production, BTH began by identifying the demands and constraints imposed on TestOrg by the process used to develop and deliver software at COMPANY. TestOrg is in charge of verifying digital business solutions (DBS)—e.g., online charging systems, mobile financial services, and service catalog manager—for telecommunication operators worldwide.

3.1. Preliminary workshops and research objectives

In the Fall of 2019, BTH organized two explorative workshops at COMPANY to understand the organizational challenges in the area of testing and quality assurance related to automation. The first workshop (Workshop 1) involved members of the quality assurance (QA) team and product managers, whereas the second (Workshop 2) involved developers, testers, and operations personnel. We assumed that the challenges highlighted within the two groups would differ based on norms and values (Lenberg et al., 2016; Lenberg and Feldt, 2018).

The workshops were organized into three parts. First, the participants answered the question *"What are the challenges your team and organization are currently facing with test automation?"* in brief statements on post-it notes (several answers are possible) in a time-boxed exercise (15 min). In the second part, each participant pitches their answers to the entire group and engages in a discussion. As the participants read and connect their answers, the researchers cluster the notes according to themes emerging during the discussion. This part of the workshop is not time-boxed to give everyone the possibility to voice their ideas. In the third part, the researchers present the themes that emerged during the discussions (i.e., clusters) and, in real time, validate with the participants the correct interpretation of their answers. We identified eight themes associated with QA and Test Automation challenges in Workshop 1 and six in Workshop 2.

The researchers prioritized the themes that emerged during the workshop in collaboration with the TestOrg leaders and test managers, taking the results from the developers as supporting input. The rationale for this decision was that the emerging challenges on the management level were of more general and larger complexity; thereby, they incorporated several of the challenges expressed by the developers. BTH presented the results to the TestOrg team, and during a discussion structured around the identified challenges, TestOrg provided further input to prioritize them.

Finally, BTH and TestOrg set out to study *quality inflation*—a mismatch between the perceived effort in quality assurance activities performed by TestOrg and the quality observed in use.

We use the term inflation to indicate that the effort in quality assurance activities is artificially increased; hence, creating such mismatch. TestOrg perceived that the motivations for quality inflation were related to both technical and human factors. For example, during Workshop 1, participants pointed out the misalignment between the metrics used by management and the ones used by developers to evaluate quality. TestOrg managers felt that the team would intentionally write "happy" test cases and defer more thorough testing activities down the development pipeline. During Workshop 2, developers felt that quality goals are not explicit and communicated with poor rationale.

Focusing on the technical causes first, BTH researchers collaborating with TestOrg focused on the following objectives.

O1. Evaluate the quality of the test suites associated with different types of Business Requirements (BR). A BR is a high-level requirement for the product, which is later broken down into smaller requirements. During Workshop 1, it emerged that some BRs within the DBS system are perceived to work well since they have few issue reports (IR) associated with them. We aim to locate such BRs, characterize them based on aspects such as their history and the history of the associated IR and test cases, and finally compare them to those BRs that are considered *troublesome* by the development and QA teams. This analysis aimed to show TestOrg possible root causes for quality inflation.

O2. Correlate the quality of the automated test cases with the quality of the test specification (e.g., test smells—suboptimal design of test code (Van Deursen et al., 2001)). This objective assumes that better-specified test scenarios lead to better test cases.

O3. Evaluate the effects of the triaging process on perceived quality. During Workshop 1, it appeared that the number of open IRs plays a crucial role in establishing the managers' perception of quality (e.g., when a BR receives many IRs, or IRs for a BR are often reopened, that BR may be perceived as low-quality). However, IRs go through a triaging process to decide if (and to what extent) they will be tested (either with additional tests or during regression testing). We wanted to study such triaging process as it directly impacts the number of open/closed IRs and, in turn, the perceived quality of a BR.

O4. Evaluate quality with respect to functional vs. non-functional BR features, such as performance. During Workshop 1, participants agreed that an area in which TestOrg should improve is testing of non-functional requirements. On the other hand, in Workshop 2, developers pointed out that non-functional testing is difficult. Our objective is to understand to what extent non-functional aspects contribute to quality inflation.

When conducting empirical investigations to address O1 and O2, it became apparent that although traceability links between artifacts (i.e., BR, IR, and test cases) were assumed to exist, this was not necessarily the case.

3.2. Development lifecycle at COMPANY

A simplified version of COMPANY's development workflow is reported in Fig. 1. The workflow is based on decisions taken together by development and product management and provides a transparent, common understanding of the development status.

COMPANY initiates the *Study* phase to fulfill new market needs, meet their R&D goals, or respond to a customer request. By the end of this phase, the stakeholders agree on the scope and resources available for development. In the *Specification* phase, the development organizations within COMPANY and the organization impacted by the development,[2] work on a document specifying a BR, and divide it into Business Sub-Requirements (BSR). The end of this phase is a synchronization point for the interested organizations, including TestOrg, to align with the BR scope and plan. Once the *Development* phase starts, there is an agreement on the development plans, dependencies, and test plans to be carried out. Inputs to this phase are the BR, the architecture model, general improvements for the area (e.g., mobile payments) and several guidelines (e.g., coding conventions, UX guidelines). *Development* ends when TestOrg completes internal verification consisting of unit testing and integration testing. TestOrg follows checklists containing the activities necessary to get the BR and BSR to a *done* state. The outputs of this phase are a package file containing the implemented solution (e.g., a Jar in case of a Java project), the source code, a release note document, and the updated architecture and risk management models. This phase is iterative and feedback-driven from internal channels—which perform continuous integration, simulation, and laboratory evaluation—and external ones, including customer laboratory evaluations and restricted launches. Each iteration usually lasts two weeks. The end of the *Release* signals that a feature is ready and can be commercially released.

3.3. Main development artifacts at COMPANY

Organizations within COMPANY handle requirements at different levels of granularity, from Business Opportunities (a high-level customer-centric definition of a solution to a problem or need) to User Stories, derived from BSR, implemented by development teams in Scrum sprints. TestOrg interacts mainly with BRs—i.e., requirements at intermediate granularity—which can be divided into BSR due to size and complexity. BR and BSR are specified following a template, and their contents vary in length (on average between 20 and 40 pages). These specifications include:

(i) General Information (e.g., scope, terminology).
(ii) Output from the *Study* phase (including recommendation for further studies).
(iii) Technical solution description which contains the requirements for the technical use case implementation.
(iv) A Glossary of terms used in the document.
(v) References used in the documents.
(vi) Changelog and revision information.

The specification documents are stored in a repository that allows tracking of changes. A web-based project management tool tracks BR and BSR status, responsible team, and other metadata. Table 1 shows the main BR attributes tracked using the tool.

Another type of artifact involved in the development process at COMPANY is Issue Report (IR). IRs are defects reported before release with varying granularity and can impact several BRs and BSRs. COMPANY uses a taxonomy of eight software characteristics associated with an IR. These range from functional suitability (e.g., functional completeness or correctness) to usability and maintainability. The IR lifecycle is handled using a web-based issue-tracking tool. Table 2 show the main IR attributes tracked using the tool.

Testing activities occur at different levels of abstraction. Manual or semi-automated testing is usually performed for BRs that impact several organizations responsible for different DBS solutions within COMPANY. Manual tests are managed and reported

[2] The list of organizations impacted by the development of a new requirement is one of the outcome of the study phase.

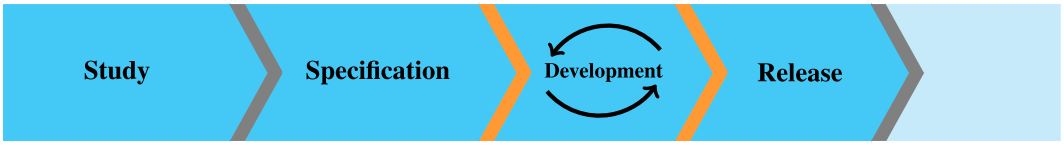

Fig. 1. Simplified version of the SDLC in use at Company. Orange marks indicate TestOrg main decision points in the process. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
BR attributes used in this study extracted from Company's requirements management system.

Attribute	Type	Description
Name	string	Name of a requirement
Description	text	Description of the requirements. Contains purpose, DoD and references to sub-requirements
Type	emun	Indicates the granularity of a requirement—i.e., task, user story, epic
Project	enum	The project the requirement belongs to
Version	integer	Existing revisions of the requirement
Created	date	Date when the requirement is created
Modified	date	Date when the requirement is modified
Creator	string	Name of the employee who created the requirement
Status	enum	Current status of the requirement (released, tested, etc.)
Release	enum	The targeted release in which the requirement should be included
Business goal	enum	The targeted high level business requirement
Validation	enum	Context in which the requirement is validated
Planned phase X	date	When the requirement is intended to be moved to the next phase X in the development X ∈ [study, specification, development, release]
Moved to phase X	date	When the requirement is actually moved to the next phase X in the development X ∈ [study, specification, development, release]
Issue reports	list	Issue reports currently associated with the requirement
Reference	link	Link to the complete document specification stored in another system)

Table 2
IR attributes used in this study extracted from Company's issue tracker system.

Attribute	Type	Description
Name	string	Name given to the IR
Content	string	Content of the IR
Priority	enum	Priority of the IR according to the submitter
Registered	date	When the IR is created
Assigned	date	When the IR is assigned to a team
Answered	date	When a fix for the IR is proposed
Completed	date	When the work to address the IR is completed
Product	string	Product experiencing the fault
Market	string	Market reference in which the product is experiencing the fault
Issuing BR	string	Name of the BR related to this IR
Characteristic	enum	Quality characteristic describing the IR (e.g., functional, reliability)
Hot	bool	Whether the IR requires immediate attention
Duplicate	bool	Whether the IR is a duplicate of an existing one
Is child	bool	Whether the IR is in a is-a relationship with another IR
Parent	string	Reference to the parent IR (iff is child is True)
Rejected	bool	Whether the IR will be addressed or not
Observation	text	Free text (e.g., stack traces, steps to reproduce)

in a separate web-based application. Automated tests—written in different programming languages and version-controlled in a repository—are also present and implemented to verify (parts of) BRs and BSRs.

4. The role of traceability in investigating software quality

BTH had access to BRs, source code, test results, and IRs. Given such a rich set of data, we set out to map artifacts (and their quality) developed early in the Development phase to later ones. In this section, we present the approach that was initially taken to identify such traces and the challenges that arose during this work.

4.1. Research methodology

During this research collaboration with Company, we followed the design science paradigm (Runeson et al., 2020) as presented in Fig. 2. After establishing the research objectives and familiarizing with the existing literature, the researchers at BTH started the solution design activities.

Initially, the researchers needed to familiarize themselves with the artifacts, domain-specific language, information systems, and processes at Company. To that end, BTH obtained access to the infrastructure and worked in collaboration with TestOrg[3] to clarify uncertainties and be onboarded on the internal processes and systems in use within the organization. Moreover, BTH researchers and TestOrg members had more structured meetings during which the former presented their current understanding of the problem, based on the analysis of the artifact, and proposed alternative ways forward and possible solutions. TestOrg gave feedback on the results pointing out, for example, wrong assumptions the researchers made. The researchers followed up with ad

[3] One of the researchers spent approximately 20 hours/month at the company office in early 2020 before work-from-home was established due to the pandemic.

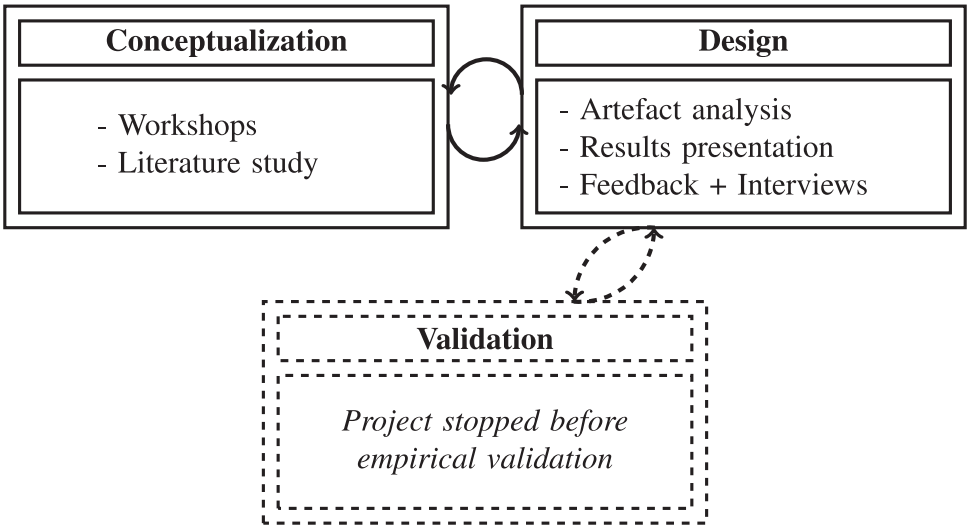

Fig. 2. Overview of design science approach and activities for each phase.

Table 3
Metrics used to characterize *Troublesome BR*. In the table, *phase* ∈ [*study*, *specification*, *implementation*, *release*].

Name	Definition
Monthly IR for BR	The number of functional IR associated to a specific BR averaged over a monthly period.
Release IR for BR	The number of functional IR associated to a specific BR averaged over release.
Time BR spent in phase	Number of days a BR stayed in a specific phase as indicated by its status.

hoc interviews about specific topics when necessary. After several iterations between *Conceptualization* and *Design*, BTH and TestOrg could not produce a solution—i.e., an intervention to reveal where quality inflation was taking place—which could be *Validated* due to the lack of traceability between artifacts.

4.2. Provisional design solution

The initial design goal was a set of guidelines to support TestOrg in identifying issues with quality inflation, inform them about test cases that needed improvement, and suggest BRs that need to be better tested or refined (i.e., reworded, simplified, re-scoped). Furthermore, we aimed to create automated support through a recommender system, based on such guidelines, which could be integrated into the TestOrg continuous integration environment and reveal "quality inflated" BR and BSR.

Based on the initial conceptualization and discussion with TestOrg, we hypothesize that writing automated tests can be more challenging for some BRs than for others (Objective O1). In the first design iteration (left pane of Fig. 3), we characterize such *Troublesome BRs* using several metrics mined from the Company artifact repositories (see Table 3). In particular, we considered the number of IRs associated with a BR over time (e.g., in a release), the amount of time a BR spent in the different phases of the development (e.g., as reported in Fig. 4), and the difference between the planned and actual time for a BR to advance to the next phase. When defining the *Troublesome BR* metric, we used the existing traceability link between IR and BR available in the issue tracking tool (marked with ① in Fig. 3).

In informal interviews, the researchers presented and discussed their results together with TestOrg and got early feedback that helped better conceptualize the solution design.

For the second iteration (right pane of Fig. 3), we needed to define the quality of the test cases for a BR to obtain a metric (i.e., *Test suite quality*) which we could then correlate with *Troublesome BR*. We discussed several ways of establishing the necessary traceability links between BRs and test cases. The more immediate one—a direct bidirectional link between BR and test cases is unavailable (marked with ② in Fig. 3). In their development process, Company does not enforce traceability between high-level requirements, such as BR, and low-level code artifacts, such as test cases.

Next, since we already had a traceability link between IR and BR, we investigated downstream (i.e., IR-to-source code) and upstream (i.e., source code-to-IR) traceability links to connect IR and test cases via IR-fixing commits (marked with ③ in Fig. 3). From a commit, it is possible to establish a link to the test cases (marked with ④ in Fig. 3)—e.g., using the approach proposed in Zaidman et al. (2008). However, the field that explicitly connects IR and commit in the issue-tracking system is barely used (approximately 10% of IRs contains a reference to a commit).

We then looked at the upstream traceability between test cases and IRs, using information from the IR-fixing commit (the other direction of the arrow marked as ③ in Fig. 3). Development organizations within Company are required to use a structured commit template. The template has a field in which the developer can indicate, among other things, whether the commit is part of a fix. The developer can do this by including the *id* of the artifact describing the issue, such as an IR. From the source code version control system, we mined patches and associated discussions taking place during the same timeframe during which BRs were implemented and IRs were addressed (i.e., 2016–mid 2020). We parsed the commit messages looking for fixes mentioning IRs and their id. However, this automated approach returned a low number of hits. Upon manual inspection of 50 random commit messages, it appeared that the commit template is mostly filled in by automatic tools (e.g., static code analyzers) or used for

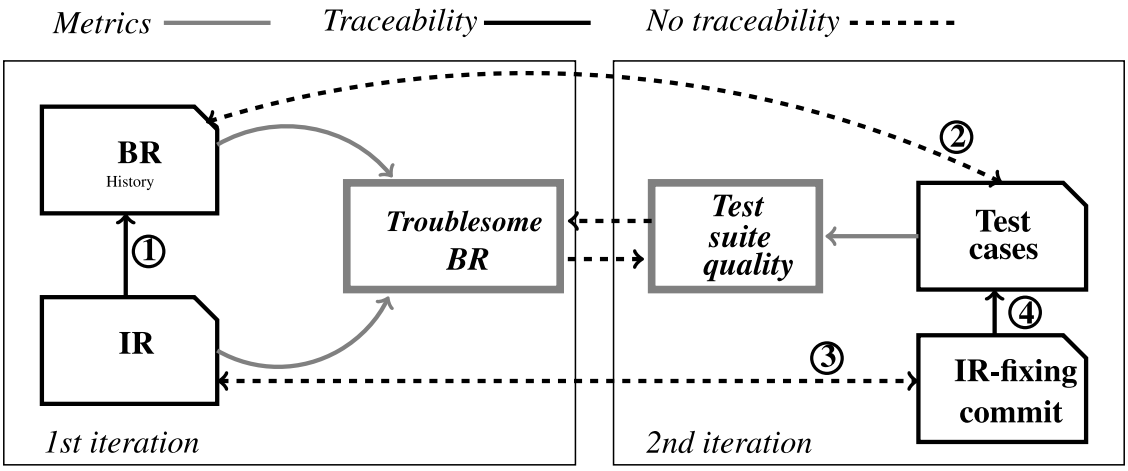

Fig. 3. Traceability (and lack thereof) between artefacts used for the proposed design.

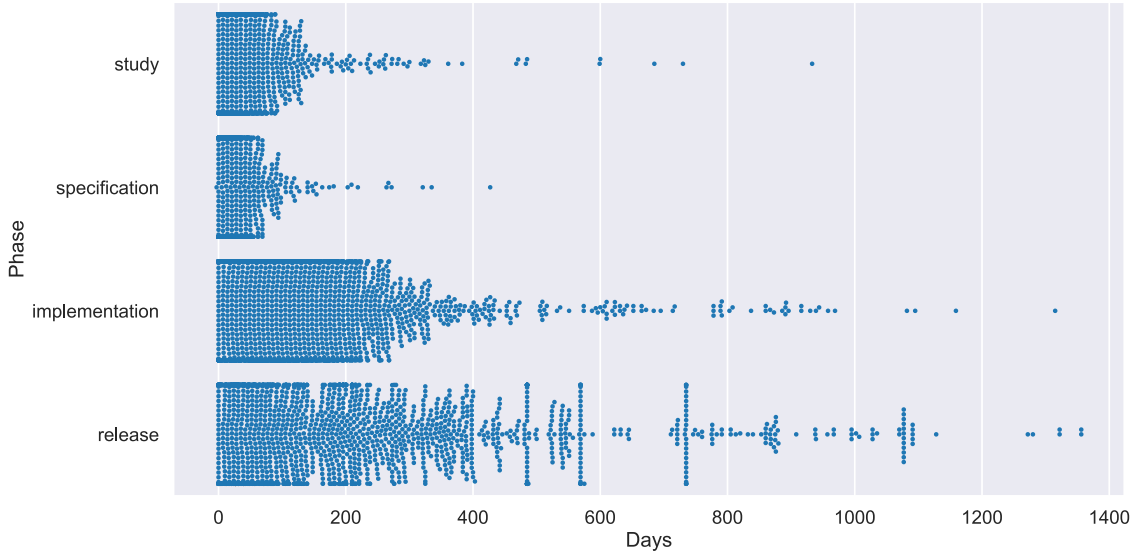

Fig. 4. BRs times in the different SDLC phases.

Table 4
Goal-Question-Metrics for the study of quality inflation at Company.

Goal (based on *O1* and *O2*)	Question	Metrics	Answered?
Purpose: Characterize the *Issue*: spread of *Object*: quality inflation *Viewpoint*: QA managers at *Context*: Company	How widespread is quality inflation?	$\frac{\text{Quality inflated TR}}{\text{All TR}}$	No
	When is there a mismatch between QA effort and quality observed for a BR?	$\frac{\text{QA effort for TR}}{\text{Observed Quality for TR}} >$ Threshold	No
	What is the QA effort dedicated to a BR?	Test smells Test suite effectiveness Test suite time to complete Defect density Defect age	**No, due to lack of traceability**
	Why are some BR perceived to be more troublesome than others?	Monthly IR for BR Release IR for BR Time BR spent in phase	Yes

tracing refactoring to code smells (e.g., from SonarQube) which are outside the scope of this study.

In summary, Table 4 show how the goal of the study relates to the metrics using the GQM framework (Caldiera and Rombach, 1994). In particular, we defined troublesome BRs but could not establish the QA effort associated with them due to lack of traceability. In turn, we could not answer the remaining questions to fulfill our original goal.

5. Discussion

In this section, we provide the lessons learned from our failed attempt at studying (and eventually addressing) quality inflation at Company due to lack of traceability and discuss the implications of these results for research in this field. We summarize our considerations about the causes and consequences of lack of traceability in Fig. 5.

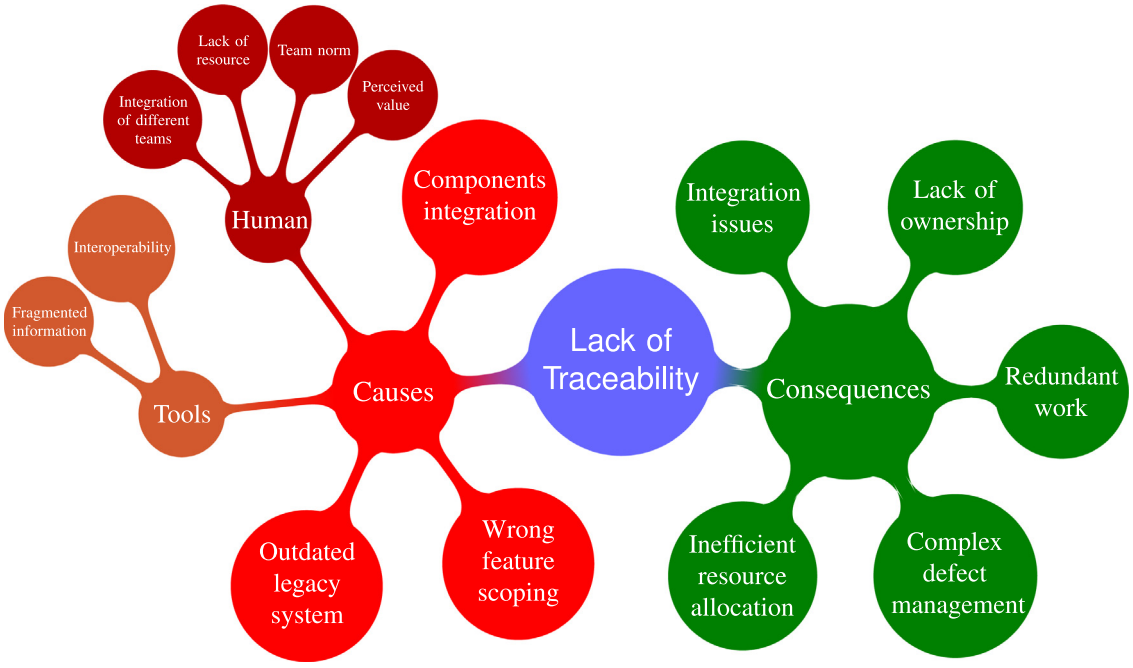

Fig. 5. Causes and consequences of lack of traceability.

The lessons learned and considerations presented in this section are derived from our own experience analyzing the artifacts used within COMPANY and our interactions with practitioners in TestOrg during workshops, feedback, and interview sessions. The takeaways are complemented with references to the literature that can provide further insights.

5.1. Lessons learned

We report the lessons learned from practitioner and researcher perspectives. For the latter, we include takeaways that should be considered when studying traceability in complex industry settings.

For practitioners. In the context of a company, the traceability links are maintained for a purpose that can be different from the one of a research project. We recommend that establishing and updating traceability links, at least the one that matters for the company, should be treated as a backlog item and tracked like any other items in the development process. Moreover, the explicit lack of traceability needs to be treated as a technical debt item and included in Sprints aimed at paying it back.

Toledo et al. (2021) show that the lack of traceability between artifacts leads to architectural technical debt. They show how, in the case of microservice architectures, maintaining data-to-data source traceability is often necessary to fulfill regulations and identify services that are not needed anymore. Whereas, also in the area of architecture and design, Charalampidou et al. (2018) show that it is useful to document traceability links to manage and estimate the cost of paying debt. Similarly, traceability supports managing documentation and requirements debt (Charalampidou et al., 2021b); conversely, lack thereof is detrimental to maintenance tasks.

Considering traceability is important as lack of links, or hard-to-establish links, can be *smells* for other problems, such as wrong scoping. For example, when a requirement is too large in scope, its implementation is expected to receive several change requests. However, understanding the scope requires a mechanism to trace change requests to requirements.

Traceability of artifacts is important from a management perspective. Without it, several overhead costs can be expected due to lack of implementation ownership—i.e., which team is responsible for implementing a functionality (for example, see Diaz et al. (2013), Ahlgren et al. (2020)). This lack of ownership can, in the worst case, lead to the same functionality being implemented several times or for the implementation to be disrupted during integration due to lacking knowledge of the code dependencies (Dasanayake et al., 2019).

Traceability also helps mitigate failure propagation—i.e., due to defects in the code that endure through the development cycle and potentially reach the customer (for example, see Mäder and Egyed (2015), Cornu et al. (2016)). Aligning tests with requirements to establish coverage metrics is vital, and without this information, it can be unclear if a requirement has been correctly tested or not. Despite defects reaching the customer or not, lingering faults cause additional overhead, delay releases, and result in longer implementation time.

Traceability also helps mitigate uncertainties regarding the allocation of resources (e.g., Wohlrab et al. (2021)). While changed or added requirements give input to allocate more resources, verification of said requirements provides grounds for their deallocation. However, without knowing if a development task has been properly addressed, such deallocation can be delayed or spent inefficiently (for example, see Cetin (2019), Çetin and Tüzün (2020), Sülün et al. (2019)).

For researchers. Researchers should validate their assumptions about what is available in terms of traceability when collaborating with a large company. Some activities that are taken for granted in some settings (e.g., open source) are hard to apply in a complex industrial organizational context, such as COMPANY. In such contexts, it is inherently difficult to have an overview of what is available in terms of information, data, and artifacts, and what is not. In the case of TestOrg, although the managers were aware of the traceability between BRs and IRs, the lack of traceability links between the source code and IRs or BRs was not considered since it is outside the scope of their activities. For the researchers, this became clear in discussions with people in more operational roles.

Takeaway 1

When assessing traceability links between different artifacts involve early personnel working day-to-day with such artifacts. For example, when traceability between source code and requirements is needed, developers and business analysts in the company should be involved.

Within the organization, we realized that horizontal traceability (i.e., traceability between artifacts at the same level of abstraction, such as requirement-to-requirement) has more value than vertical traceability (e.g., at different levels, such as test cases-to-requirements). This may be the case in large companies, where the different phases of the development cycle—and, therefore, their associated artifacts—are managed by different internal organizations. For the practitioners we interacted with, limited traceability (mostly among BRs, and between BRs and IRs) was enough to perform their daily tasks. To enact a different type of traceability, researchers need a strong use case for the company. The company will have to i) allocate resources to support the researchers in establishing extra traceability links, ii) maintain the traceability links (e.g., for further evaluation by the researchers).

Takeaway 2

We recommend gaining an early understanding of the organizational structure, and being aware that the amount of traceability information available may be influenced by such structure.

Take-away 1 and *Take-away 2* are related to research on Conway's law (e.g., Herbsleb and Grinter (1999a,b)) and the impact that traceability has on organizational vs. architectural structure. Moreover, research on *communication* of traceability highlights how organizational structure influences the way tasks are allocated and *who* within the company possesses the necessary knowledge about artifacts of interest for the researcher (Imtiaz and Ikram, 2011).

In the context of a large organization, different artifacts are tracked at different levels of detail. In our collaboration with Company, we realized that the system used to track BRs was not populated with much information (e.g., many fields were left blank or filled with boilerplate values). The organizational *norm* in case BR details are needed is to refer to the BR specification document (through a link in the tracking system) stored in a separate repository. This limited the automatic extraction of information from BRs, as the two systems are not designed to communicate autonomously.

Takeaway 3

When establishing traceability, consider that in complex settings different information about the same artifact are likely to be scattered across systems. Regardless of the approach, consolidating such information requires knowledge of the company's norms.

Several researchers tried to address the challenge of scattered information (Shen et al., 2021; Wang et al., 2021; Taromirad et al., 2013). Promising approaches have combined traditional information retrieval with model-driven engineering (Sannier and Baudry, 2012) and (semi)supervised machine learning (Bella et al., 2019).

TestOrg was aware of the problem with dispersed information. They led, within Company, an initiative to centralize test cases and BR tracking for several purposes, including improving their traceability. By the end of such initiative, Company will use a single tracking system for all these artifacts.

Takeaway 4

Depending on the structure and location of the information required to establish traceability, a fully-automated solution may not be feasible. A solution for creating traceability links should not start by considering full-fledged automation but by accommodating human intervention.

Tool support is fundamental when establishing, using, and maintaining traceability links. Organizations developing complex systems deal with artifacts at different levels of abstraction, details, and formats, which entail using different artifact-tracking tools. Moreover, practitioners choose, configure, and use tools according to the level of traceability necessary for *their* tasks—for them interoperability may not be a decisive criterion for selecting a tool. For example, Company uses an homebrew system for tracking requirements specification documents and IRs, a third-party commercial solution (which reached end-of-life in early 2019, but it is still maintained for legacy reasons) for tracking BRs, and different open-source systems for source code version control and code reviews. Some offer APIs, but none offered out-of-the-box integration towards any of the other systems.

Takeaway 5

Fragmentation in terms of tools within a company developing complex systems is to be expected. Therefore, effort is required to achieve interoperability between systems when establishing traceability links.

Addressing tools fragmentation is significant for safety-critical software development (Baumgart and Ellen, 2014) and could be mitigated, for instance, through modeling (Drivalos et al., 2008).

In the case of Company, fragmentation (and the consequent lack of traceability) derived from a tradeoff between other system properties deemed more desirable. During our feedback sessions with TestOrg, it became apparent that the homebrew solution was selected to have control over (i) the security of the specifications as they contain information pivotal for Company's competitive advantage, and (ii) the redundancy of the storage to avoid costly data loss.

5.2. Considerations on traceability in complex industrial settings

The results of this study provide insights into the state-of-practice, lessons learned, and considerations for industrial practitioners and academics to reflect upon regarding traceability. The experience reported in this paper highlights a dichotomous and puzzling situation that was surprising for the researchers involved in this study. The academic literature on traceability and alignment supports the idea that traceability among software artifacts is needed to manage complexity (Fernández et al.,

2017). Despite these claims, Company is producing systems in the order of millions of lines of code, with thousands of developers, yet traceability links from high-level requirements to source code and tests are not readily available. Paradoxically, traceability within Company may not be achieved because of the system's complexity, while traceability could mitigate said complexity.

What is going on in this case? The results may seem baffling, but in real-world settings—when software grows and the organization along with it—several factors influence the evolution of a product, its development process, and the environment. We did not study the root cause of the current situation, but several hypotheses can be formulated.

First, the system is considered a system of systems (SoS), developed in a heterogeneous environment of processes, tools, and third-party components. Without a strong culture to tie this development together with an emphasis on traceability, it is only natural that its amount and consistency will vary. Integration and SoS development are still considered possible, as such is managed at a higher level of abstraction, primarily considering the interfaces of the underlying components, despite the lack of end-to-end traceability.

Second, achieving traceability in a large system is resource-intensive, and human commitment also comes into play. In a study by Borg et al. about change requests and change management, it was observed that teams are hesitant to even touch upon other teams' code (Borg et al., 2017). In larger silo organizations this situation is exacerbated since silos may assume that other individuals or teams keep traceability up to date, especially when teams suffer from resource constraints.

Take, as an example, a scenario in which Team A is adapting a system core component to conform to a BR. As BRs are high level, they likely require changes to surrounding components. The developers in Team A make changes, but due to the lack of insight or knowledge about the Team B artifacts, not all artifacts associated with the modified components are updated. This causes a slight misalignment between the requirement traces and the system under development. After many such changes, traceability will naturally degrade over time, leading to a situation where traces will have to be maintained or reverse engineered. However, since reverse engineering is expensive, process solutions that enforce localized knowledge—i.e., silo organizations are instead encouraged.

Finally, there can be a legacy component to the challenge—i.e., that components within the SoS are decaying. These components may have had extensive traceability information, but as Agile practices are leaner in terms of documentation, such traces are no longer maintained. Hence, a situation that is perceived as a product of a new way of working with software in which artifacts of long-term value—associated with high cost—are de-prioritized in favor of short-term value gains and other forms of light-weight documentation that fulfill similar purposes.

Furthermore, we observed that traceability between high-level requirements, source code, and test suites is not maintained at Company. A question remains, *how can* Company *continue producing high-quality, large-scale, and complex software on time while keeping their customers satisfied?* As discussed in this section, we believe it is due to the evolution of the development organization and its ability to adapt to circumstances in which traceability is not always available. Hence, instead of relying on traceability information, workarounds and alternate processes, coupled with organizational structures and architectural design decisions, provide Company a cohesive understanding of the system.

Hence, although the academic literature highlights the need for traceability for understanding of how the system fits together (Kukkanen et al., 2009), the situation we observed at Company indicates that such requirements may have been overstated.

6. Conclusion

In this paper, we reported our experience in applied research with Company. In particular, our goal was to define and apply an intervention for identifying quality inflation using artifacts in different development phases. However, our attempt failed due to the lack of traceability between such artifacts.

We reflected on the outcomes of this industrial study, which led us to question the role and perceived value of traceability in industry, and its return-on-investment for large software projects. From our experience, we encourage practitioners to be selective about the traceability links necessary for their organization while highlighting the importance of requirement-to-test case trace links. Moreover, practitioners need to explicitly manage traceability links as they do for other artifacts (e.g., requirements, documents, test cases). We also suggest takeaways that can support researchers performing empirical studies that consider traceability a prerequisite. In particular, they need to be aware that the organizational structure and norms can impact which traceability links exist and how they are managed. For complex projects in large settings, scattered information among different tools and systems is to be expected.

In the future, we aim to systematically study the state-of-practice related to traceability among our industrial partners.

CRediT authorship contribution statement

Davide Fucci: Conceptualization, Methodology, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization. **Emil Alégroth:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Thomas Axelsson:** Resources, Project administration, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Davide Fucci and Emil Alégroth would like to acknowledge that this work was supported by the KKS foundation through the S.E.R.T. Research Profile project at Blekinge Institute of Technology, Sweden.

References

Ahlgren, J., Berezin, M.E., Bojarczuk, K., Dulskyte, E., Dvortsova, I., George, J., Gucevska, N., Harman, M., He, S., Lämmel, R., et al., 2020. Ownership at large: Open problems and challenges in ownership management. In: Proceedings of the 28th International Conference on Program Comprehension. pp. 406–410.

Baumgart, A., Ellen, C., 2014. A recipe for tool interoperability. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). IEEE, pp. 300–308.

Bella, E.E., Creff, S., Gervais, M.-P., Bendraou, R., 2019. Atlas: A framework for traceability links recovery combining information retrieval and semi-supervised techniques. In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC). IEEE, pp. 161–170.

Borg, M., Alégroth, E., Runeson, P., 2017. Software engineers' information seeking behavior in change impact analysis-an interview study. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC). IEEE, pp. 12–22.

Caldiera, V.R.B.G., Rombach, H.D., 1994. The goal question metric approach. Encycl. Softw. Eng. 528–532.

Çetin, H.A., Tüzün, E., 2020. Identifying key developers using artifact traceability graphs. In: Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE 2020. Association for Computing Machinery, New York, NY, USA, pp. 51–60. http://dx.doi.org/10.1145/3416508.3417116.

Cetin, H.A., 2019. Identifying the most valuable developers using artifact traceability graphs. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2019. Association for Computing Machinery, New York, NY, USA, pp. 1196–1198. http://dx.doi.org/10.1145/3338906.3342487.

Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Tsiridis, N., 2018. Integrating traceability within the IDE to prevent requirements documentation debt. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). pp. 421–428. http://dx.doi.org/10.1109/seaa.2018.00075.

Charalampidou, S., Ampatzoglou, A., Karountzos, E., Avgeriou, P., 2021a. Empirical studies on software traceability: A mapping study. J. Softw.: Evol. Process 33 (2), e2294.

Charalampidou, S., Ampatzoglou, A., Karountzos, E., Avgeriou, P., 2021b. Empirical studies on software traceability: A mapping study. J. Softw.: Evol. Process 33 (2), http://dx.doi.org/10.1002/smr.2294.

Cleland-Huang, J., Chang, C.K., Christensen, M., 2003. Event-based traceability for managing evolutionary change. IEEE Trans. Softw. Eng. 29 (9), 796–810.

Cornu, B., Barr, E.T., Seinturier, L., Monperrus, M., 2016. Casper: Automatic tracking of null dereferences to inception with causality traces. J. Syst. Softw. 122, 52–62. http://dx.doi.org/10.1016/j.jss.2016.08.062.

Dasanayake, S., Aaramaa, S., Markkula, J., Oivo, M., 2019. Impact of requirements volatility on software architecture: How do software teams keep up with ever-changing requirements? J. Softw.: Evol. Process 31 (6), e2160.

Diaz, D., Bavota, G., Marcus, A., Oliveto, R., Takahashi, S., De Lucia, A., 2013. Using code ownership to improve ir-based traceability link recovery. In: 2013 21st International Conference on Program Comprehension (ICPC). pp. 123–132. http://dx.doi.org/10.1109/ICPC.2013.6613840.

Dick, J., 2002. Rich traceability. In: Proceedings of the 1st International Workshop on Traceability in Emerging Forms of Software Engineering. Citeseer, pp. 18–23.

Drivalos, N., Kolovos, D.S., Paige, R.F., Fernandes, K.J., 2008. Engineering a dsl for software traceability. In: International Conference on Software Language Engineering. Springer, pp. 151–167.

Ebert, C., 2013. Improving engineering efficiency with plm/alm. Softw. Syst. Model. 12 (3), 443–449.

Fernández, D.M., Wagner, S., Kalinowski, M., Felderer, M., Mafra, P., Vetrò, A., Conte, T., Christiansson, M.-T., Greer, D., Lassenius, C., et al., 2017. Naming the pain in requirements engineering. Empir. Softw. Eng. 22 (5), 2298–2338.

Gorschek, T., Garre, P., Larsson, S., Wohlin, C., 2006. A model for technology transfer in practice. Ieee Softw. 23 (6), 88–95. http://dx.doi.org/10.1109/MS.2006.147.

Hayes, J.H., Dekhtyar, A., Sundaram, S.K., Holbrook, E.A., Vadlamudi, S., April, A., 2007. Requirements tracing on target (retro): improving software maintenance through traceability recovery. Innov. Syst. Softw. Eng. 3 (3), 193–202.

Herbsleb, J.D., Grinter, R.E., 1999a. Architectures, coordination, and distance: Conway's law and beyond. Ieee Softw. 16 (5), 63–70.

Herbsleb, J.D., Grinter, R.E., 1999b. Splitting the organization and integrating the code: Conway's law revisited. In: Proceedings of the 21st International Conference on Software Engineering. pp. 85–95.

Imtiaz, S., Ikram, N., 2011. Effective task allocation in distributed environments: A traceability perspective. In: International Conference on Software Engineering Advances. pp. 563–569.

Javed, M.A., Zdun, U., 2014. A systematic literature review of traceability approaches between software architecture and source code. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. pp. 1–10.

Kukkanen, J., Väkeväinen, K., Kauppinen, M., Uusitalo, E., 2009. Applying a systematic approach to link requirements and testing: A case study. In: 2009 16th Asia-Pacific Software Engineering Conference. IEEE, pp. 482–488.

Lenberg, P., Alégroth, E., Feldt, R., Tengberg, L.G.W., 2016. An initial analysis of differences in software engineers' attitudes towards organizational change. In: Proceedings of the 9th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 1–7.

Lenberg, P., Feldt, R., 2018. Psychological safety and norm clarity in software engineering teams. In: Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 79–86.

Lin, J., Lin, C.C., Cleland-Huang, J., Settimi, R., Amaya, J., Bedford, G., Berenbach, B., Khadra, O.B., Duan, C., Zou, X., 2006. Poirot: A distributed tool supporting enterprise-wide automated traceability. In: 14th IEEE International Requirements Engineering Conference (RE'06). IEEE, pp. 363–364.

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

Mustafa, N., Labiche, Y., 2017. The need for traceability in heterogeneous systems: a systematic literature review. In: 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC), Vol. 1. IEEE, pp. 305–310.

Nair, S., De La Vara, J.L., Sen, S., 2013. A review of traceability research at the requirements engineering conference re@ 21. In: 2013 21st IEEE International Requirements Engineering Conference (RE). IEEE, pp. 222–229.

Randell, B., 1968. Towards a methodology of computing system design. In: NATO Working Conference on Software Engineering. pp. 204–208.

Runeson, P., Engström, E., Storey, M.-A., 2020. The Design Science Paradigm As a Frame for Empirical Software Engineering. Springer International Publishing.

Sannier, N., Baudry, B., 2012. Toward multilevel textual requirements traceability using model-driven engineering and information retrieval. In: 2012 Second IEEE International Workshop on Model-Driven Requirements Engineering (MoDRE). IEEE, pp. 29–38.

Santiago, I., Jiménez, A., Vara, J.M., De Castro, V., Bollati, V.A., Marcos, E., 2012. Model-driven engineering as a new landscape for traceability management: A systematic literature review. Inf. Softw. Technol. 54 (12), 1340–1356.

Shen, G., Wang, H., Huang, Z., Yu, Y., Chen, K., 2021. Supporting requirements to code traceability creation by code comments. Int. J. Softw. Eng. Knowl. Eng. 31 (08), 1099–1118.

Sülün, E., Tüzün, E., Doğrusöz, U., 2019. Reviewer recommendation using software artifact traceability graphs. In: Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE'19. Association for Computing Machinery, New York, NY, USA, pp. 66–75. http://dx.doi.org/10.1145/3345629.3345637.

Tang, A., Jin, Y., Han, J., 2007. A rationale-based architecture model for design traceability and reasoning. J. Syst. Softw. 80 (6), 918–934.

Taromirad, M., Matragkas, N.D., Paige, R.F., 2013. Towards a multi-domain model-driven traceability approach. In: MPM@ MoDELS. pp. 27–36.

Tian, F., Wang, T., Liang, P., Wang, C., Khan, A.A., Babar, M.A., 2021. The impact of traceability on software maintenance and evolution: A mapping study. J. Softw.: Evol. Process 33 (10), e2374.

Toledo, S.S.d., Martini, A., Sjøberg, D.I., 2021. Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. J. Syst. Softw. 177, 110968. http://dx.doi.org/10.1016/j.jss.2021.110968.

Tufail, H., Masood, M.F., Zeb, B., Azam, F., Anwar, M.W., 2017. A systematic review of requirement traceability techniques and tools. In: 2017 2nd International Conference on System Reliability and Safety (ICSRS). IEEE, pp. 450–454.

Unterkalmsteiner, M., Feldt, R., Gorschek, T., 2014. A taxonomy for requirements engineering and software test alignment. ACM Trans. Softw. Eng. Methodol. (TOSEM) 23 (2), 1–38.

Van Deursen, A., Moonen, L., Van Den Bergh, A., Kok, G., 2001. Refactoring test code. In: Proceedings of the 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2001). Citeseer, pp. 92–95.

Wang, H., Shen, G., Huang, Z., Yu, Y., Chen, K., 2021. Analyzing close relations between target artifacts for improving ir-based requirement traceability recovery. Front. Inf. Technol. Electron. Eng. 22 (7), 957–968.

Watkins, R., Neal, M., 1994. Why and how of requirements tracing. Ieee Softw. 11 (4), 104–106.

Wohlrab, R., Pelliccione, P., Shahrokni, A., Knauss, E., 2021. Why and how your traceability should evolve: Insights from an automotive supplier. Ieee Softw. 38 (4), 62–70. http://dx.doi.org/10.1109/MS.2020.2996369.

Zaidman, A., Van Rompaey, B., Demeyer, S., Van Deursen, A., 2008. Mining software repositories to study co-evolution of production & test code. In: 2008 1st International Conference on Software Testing, Verification, and Validation. IEEE, pp. 220–229.

Davide Fucci is an Assistant Professor at Blekinge Institute of Technology (Sweden). He received his Ph.D. from the University of Oulu (Finland) in 2016. His research interests lie in data-drive requirements engineering, test automation, security testing, and human aspects of software development. Dr. Fucci has published over 50 articles in international journals and conferences and received several best paper awards. He has served on the program committees of over 20 academic conferences, on the editorial or review boards of several top-tier software engineering journals. He started the AffectRE workshop series on emotional awareness in requirements engineering. He is a member of ACM, ACM SIGSOFT, IEEE and IEEE Computer Society. For more information please visit: http://dfucci.co.

Emil Alégroth is a software engineering researcher at Blekinge Institute of Technology and affiliated with Chalmers University of Technology. He has for several years worked with industrially close research while operating as the Chief Executive Officer (CEO) of a spin-off company from his Ph.D. studies. His research interests include decision-making, human factors, and in particular automated software testing.

Thomas Axelsson is a practitioner in the area of testing, testing management, and testing automation. He has more that 35 years of experience in technical and managerial roles.