Data-driven and Tool-supported Elicitation of Quality Requirements in Agile Companies

Marc Oriol[1] ✉, Silverio Martínez-Fernández[1, 2], Woubshet Behutiye[3], Carles Farré[1], Rafal Kozik[4,5], Pertti Seppänen[3], Anna Maria Vollmer[2], Pilar Rodríguez[3], Xavier Franch[1], Sanja Aaramaa[6], Antonin Abhervé[7], Michal Choras[4,5], Jari Partanen[8]

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
{moriol, smartinez, farre, franch}@essi.upc.edu
[2] Fraunhofer IESE, Kaiserslautern, Germany
{anna-maria.vollmer}@iese.fraunhofer.de
[3] University of Oulu, Oulu, Finland
{woubshet.behutiye, pertti.seppanen, pilar.rodriguez}@oulu.fi
[4] ITTI Sp. z o.o., Poznań, Poland
{rkozik, mchoras}@itti.com.pl
[5] University of Science and Technology, UTP, Bydgoszcz, Poland
[6] NOKIA, Oulu, Finland
sanja.aaramaa@nokia.com
[7] Softeam, Paris, France
antonin.abherve@softeam.fr
[8] Bittium Wireless Ltd., Oulu, Finland
jari.partanen@bittium.com

Abstract

Quality Requirements (QRs) are a key artifact needed to ensure the quality and success of a software system. Despite their importance, QRs rarely get the same degree of attention as their functional counterpart in Agile Software Development (ASD) projects. Moreover, crucial information that can be obtained from software development repositories (e.g., JIRA, GitHub,…) is not fully exploited, or is even neglected, in QR elicitation activities. In this work, we present a data-driven tooled approach for the semi-automatic generation and documentation of QRs in the context of ASD. The approach is based on the declaration of thresholds over quality-related issues, whose violation triggers user-defined alerts. These alerts are used to browse a catalog of QR patterns that are presented to the ASD team by means of a dashboard that implements several analysis techniques. Once selected, the patterns generate the QRs, which are documented and stored in the product backlog. The full approach is implemented via a configurable platform. Over the course of one year, four companies differing in size and profile followed this approach and deployed the platform in their premises to semi-automatically generate QRs in several projects. We used standardized measurement instruments to elicit the perception of 22 practitioners regarding their use of the tool. The quantitative and qualitative analyses yielded positive results; i.e., the practitioners' perception with regard to the tool's understandability, reliability, usefulness, and relevance was positive. We conclude that the results show potential for future adoption of data-driven elicitation of QRs in agile companies and encourage other practitioners to use the presented tool and adopt it in their companies.

Keywords. Requirements engineering, data-driven software engineering, software quality, quality requirements, non-functional requirements, quality attributes, agile software development.

✉ Marc Oriol
moriol@essi.upc.edu

1. Introduction

Quality management is known to be one of the critical success factors for software projects (Abbas et al. 2010). There are many examples of software with poor quality (e.g., software with critical bugs, security vulnerabilities, technical debt, low quality of service, poor code quality, etc.) that have caused millions of euros of losses (Krasner 2018). A report conducted by the software testing company Tricentis revealed that software failures caused more than \$1.7 trillion in financial losses in 2017 (Tricentis 2018).

Therefore, to be successful, software development companies must understand and manage software quality. Indeed, they should not only strive to avoid the aforementioned software failures but even aim at progressive improvement (Behnamghader et al. 2017). For this reason, many approaches have emerged aiming to improve quality in different phases of the software development lifecycle (Groen et al. 2017; Lu and Liang 2017). In this regard, market studies show a steady increase in the proportion of software development companies' budgets being spent on dealing with software quality (Capgemini 2015).

It is argued that an optimal approach to improving software quality should consider and address such quality early from the requirements (Franch 2018). Quality Requirements (QRs) – also known as non-functional requirements[1] – are those artifacts that requirements engineers use to state conditions on, and analyze the compliance of, software quality. A QR is defined as "*a requirement that pertains to a quality concern that is not covered by functional requirements*" (Pohl and Rupp 2015). QRs play an essential role in the success of software systems, and neglecting or failing to satisfy QRs can lead to critical consequences (Franch 2018; Spinellis 2006).

Despite their importance, QRs have traditionally not received the same degree of attention in the industry than their functional requirements counterpart (Wagner 2015). This is also true in trending software development methodologies like Agile Software Development (ASD), a software development approach that has been widely adopted in the software industry (Rodríguez et al. 2012). ASD advocates frequent releases and short development cycles to deliver partial (but fully operational) software. However, such rapid delivery should not compromise software quality.

To address this problem, in previous work we presented an explorative position paper where we envisaged a conceptual framework named Q-Rapids to generate and document QRs using a data-driven approach in the context of ASD (Franch et al. 2018a). The approach is based on obtaining data from heterogeneous data sources (e.g., static code analysis tools as Sonarqube and issue tracking systems as JIRA) (Martínez-Fernández et al. 2018b). On top of this, the approach generates QRs if an issue is detected after computing the values of quality metrics and project indicators. We then defined a software architecture, which was validated in four use cases. The results of this validation were used to iterate on the conceptual framework and architecture (Oriol et al. 2019a). In this article, we present an extension of this work with the following new contributions:

1. An extended description of the implementation of the solution, including complete implementation of all envisaged features (e.g., alert generation, QR patterns identification) and technical improvements (e.g., integration with a dashboard). Hereafter, we will refer to the implementation as tool-supported generation and documentation of QRs.

[1] There are some disagreement and discussion about the terminology for these type of requirements (Glinz 2007). For simplicity, in this paper we consider both terms as synonymous.

2. An empirical study for evaluating the functionalities for generating and documenting software QRs in four companies. We focus on two points of view:
    a. the extent to which product owners, managers, and developers perceive the provided tool-supported generation and documentation of QRs as understandable, complete, useful, reliable, easy to use, and efficient.
    b. the characteristics of the tool-supported generation and documentation of QRs that are perceived as needed by practitioners.
3. An open data package (Oriol et al. 2019b) making the assets produced in this study accessible, including the source code and documentation of the tool's components, different models and artifacts containing the explicit knowledge required for the tool (e.g., a QR pattern catalog), and the evaluation instruments (e.g., questionnaire).

The research was conducted in the context of the Q-Rapids H2020 project (Franch et al. 2019), which enabled us to elicit real scenarios based on practitioners' needs, use the tool-supported generation and documentation of QRs in different company-provided scenarios, and evaluate it based on users' experiences and perceptions.

The remainder of this paper is organized as follows. Section 2 presents the background and related work. Section 3 describes the research methodology we followed in our proposed approach. Section 4 describes the QR generation and documentation process, whose validation is presented in Section 5. Section 6 presents the implementation of the tool-supported generation and documentation of QRs, and Section 7 an empirical evaluation of the implementation. Section 8 presents the threats to validity of both the validation of the QR generation and documentation process and the empirical evaluation of the implementation. Finally, Section 9 provides the conclusions and an outlook on future work.

2. Background and related work

The ASD process is mostly driven by functional requirements (Schön et al. 2017). For example, in Scrum (Schwaber, 2004), requirements are specified as user stories in a product backlog and prioritized based on a customer perspective. This way of eliciting and managing requirements tends to favor functional requirements over QRs (Schön et al. 2017; Rodríguez et al. 2017). As a result, QRs are not properly documented and only managed tacitly (Bartsch 2011). Moreover, despite the numerous sources of information related to product quality that ASD provides (e.g., continuous integration systems and user feedback) (Martínez-Fernández 2018b), there is a lack of methods to support continuous elicitation and management of QRs throughout the whole software development lifecycle (Rodríguez et al. 2017).

On the other hand, traditional approaches for eliciting and managing QRs are usually inadequate in the highly dynamic scenarios in which ASD is suitable. Traditional techniques for eliciting QRs include structured and unstructured interviews, quality models, checklists, prioritization questionnaires, and the like (Zowghi and Coulin 2005). In this context, data-driven requirements engineering (Maalej et al. 2016) is advocated as the proper way for eliciting QRs.

Some recent proposals in this direction aim at exploiting explicit end-user feedback data (Groen et al. 2017; Kurtanovic and W. Maalej 2017; Lu and Liang 2017; Guzmán et al. 2016). However, explicit feedback requires user commitment and may be incomplete and/or biased. Implicit feedback can be considered as an alternative/complementary data source for requirements elicitation (Maalej et al. 2016). For instance, Liu et al. (2017) exploit implicit feedback but do not aim at generating QRs; rather, their aim is to discover user preferences and usage patterns. Brill and Knauss (2011) propose an approach for getting new requirements based on implicit feedback from the users' behavior and context, but their approach focuses on functional requirements for context-adaptive systems. The SUPERSEDE data-driven approach (Franch et al. 2018b) combines both explicit and implicit end-user feedback with other sources such as runtime monitors to detect and address different kinds of issues (bugs, new features, QoS violations).

However, none of the aforementioned approaches exploits data gathered from software repositories, project management tools, or code inspectors. Without these other relevant sources, QRs related more directly to "internal" aspects like code quality or the software development process itself can hardly be elicited.

To sum up, the novelty of this work is that it offers tool support for: (a) generating QRs from a configurable and expandable set of heterogeneous data sources (runtime monitors, code inspectors,...) so that both external quality properties (e.g., availability, response time), and internal ones (e.g., code maintainability, testability, ...) are fully addressed; and, (b) documenting the QRs in the backlogs of Agile teams. In addition, this work provides experiences made with the use of this tool support in four Agile companies.

3. Research methodology

To conduct our research, we followed the Design Science Research Methodology (DSRM) (Peffers et al. 2007, Cronholm and Göbel 2015). DSRM is a methodology that provides guidelines for researchers to conduct research in information systems based on the principles, practices, and procedures required in design science. DSRM defines an iterative process model that incorporates the following list of activities: *identification of problem & motivation*, *definition of the objectives of a solution, design & development, demonstration and evaluation*, and *communication* (see Figure 1).

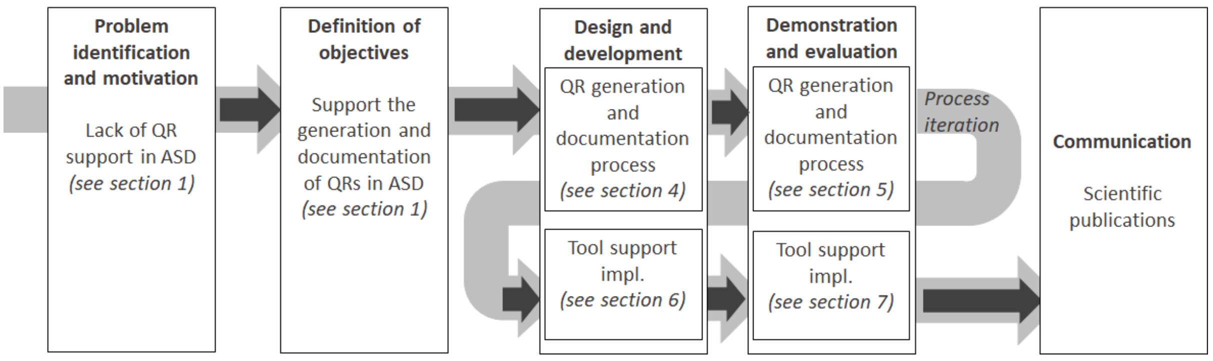


Fig 1. DSRM Process Model followed in this study

Below, we describe each of those activities and report how we addressed them in the context of our research work.

- *Problem identification and motivation*. As reported in the introduction, quality management is a critical success factor for software projects and QRs are an optimal artifact for ensuring good software quality. However, they have not received the same degree of attention as their functional requirements counterparts in the context of ASD.
- *Definition of objectives*. Our research goal is to provide a framework to support the continuous generation and documentation of QRs using a data-driven approach.
- *Design and development*. Following an iterative process, we conducted this activity in two iterations. In the first iteration, we designed an architecture of the QR generation and documentation process (see Section 4). In the second iteration, we implemented these components and provided a tool-supported solution (see Section 6).

- *Demonstration and evaluation*. These activities demonstrate the use of the artifacts and measure how well these artifacts solve the stated problem. For this activity we follow existing guidelines in empirical software engineering (Wohlin et al. 2012). In the first iteration, we demonstrated and evaluated the QR generation and documentation architecture (see Section 5), whereas in the second iteration, we demonstrated and evaluated the tool support implementation (see Section 7). The evaluation was conducted in the four companies of the Q-Rapids consortium (Franch et al. 2019), which have different profiles (one large corporation, two large/medium companies, and one SME) and produce different types of systems (e.g., from modeling tools to telecommunications software).
- *Communication*: This activity refers to communicating the research in scientific publications.

Industrial context. This research methodology was applied in the context of pilot projects from the four industry partners of the Q-Rapids project. These companies develop products using agile software development (Scrum-like) in diverse application domains such as telecommunications, security, military, transport, health, and public administration. All four participated in the aforementioned activities of our research methodology.

Company 1's products are used by its customers to develop critical systems in the military and transportation domains. In this context, Company 1 has committed to providing a quality guarantee for the software it releases. To achieve this goal, Company 1 expressed the need to evolve the quality management processes applied to its software development cycle by getting new insights to support the decision-making process in the context of rapid software development and to automate management of QRs across the organization.

Company 2 is striving to enhance its software development by focusing on data analysis and fast feedback methods for functional requirements and QRs. Such efforts aim at improving both the product readiness and process performance, which are a very attractive application from the company's point of view.

Company 3 aims to make the quality of both their product development processes and the products themselves more visible with the assistance of data-driven tool support for the generation and documentation of QRs. This is a key factor, considering the extensive portfolio of products developed in multiple distributed project(s).

Company 4 aims at improving their agile software development process. The studied product is an enterprise-class integrated software system for warehouse and manufacturing management. It is a web application providing up-to-date information about clients, storage, shipment, picking, production, and other business-related processes. It allows for monitoring and data analysis related to running processes, evaluation of their effectiveness, and related transactions.

4. QR generation and documentation process

Q-Rapids is a quality-aware ASD framework in which QRs are elicited using a data-driven approach (Franch et al. 2018a). Data from multiple data sources is gathered and evaluated by means of a quality model. Then, when an issue is identified in the quality model, it should be represented as a QR and considered during software development.

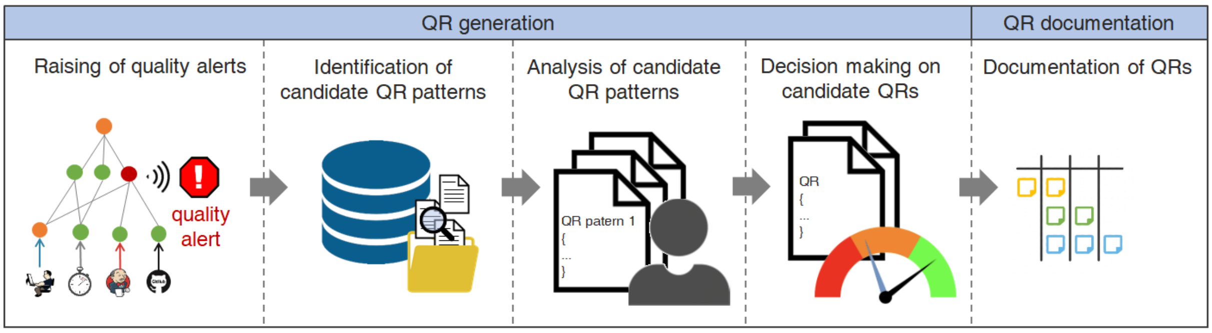

QR generation
QR documentation
Raising of quality alerts
Identification of candidate QR patterns
Analysis of candidate QR patterns
Decision making on candidate QRs
Documentation of QRs
quality alert
QR patern 1
QR

Fig. 2. Main logical process of QR generation and documentation: from quality alerts to elicited QRs in the Agile backlog.

Figure 2 depicts the overall process with its different phases for generating and documenting QRs in Q-Rapids. The QR generation process refers to the activities conducted from the time an issue is identified until the QR is generated, whereas the QR documentation process represents the activity where the generated QR is added to the organization's backlog.

QR generation phases:

- *Raising of quality alerts:* As a first step, data from multiple and heterogeneous *data sources* is gathered (e.g., from Jira, SonarQube, github, runtime monitors, etc.). The collected data feeds a *quality model* that computes the quality of the software. The different elements of the *quality model* represent characteristics of the software quality at different abstraction levels (Martínez-Fernández et al. 2019). These elements of the *quality model* have customizable thresholds which, if violated, automatically raise a *quality alert.*
- *Identification of candidate QR patterns:* When a *quality alert* is raised, *candidate QR patterns* are identified. Those *candidate QR patterns*, after being instantiated to *QRs* and implemented, will restore the value(s) of the element(s) of the *quality model* that raised the alert. A key component used to identify such *candidate QRs* is the *QR pattern catalog* (Renault et al. 2009). The *QR pattern catalog* consists of a set of *QR patterns,* which are defined in terms of natural language sentences that include formal parameters (i.e., free variables). The *QR patterns* are bound to *quality model* elements in the schema of the *QR patterns catalog*. This binding is fundamental for matching the appropriate *candidate QR patterns* with the raised *quality alert*.
- *Analysis of candidate QR patterns:* The *candidate QR patterns* are presented to the decision makers —Product Owners, project managers, or other members of the development team—on a *strategic dashboard*. The decision makers evaluate the *candidate QR patterns* and instantiate them into particular *QRs* by setting the values of the formal parameters of the *QR pattern*.
- *Decision making on candidate QRs:* The strategic dashboard includes simulation techniques for predicting the impact the QRs would have on the values of the different elements of the *quality model* if such *QRs* were implemented. This way, decision makers can define the appropriate values for the candidate QRs and finally decide on their acceptance. It is worth to mention that, since QRs are generated semiautomatically, the understandability and other features perceived by decision makers on these QRs could differ from the features computed automatically (Caivano et al. 2018). For this reason, decision makers are responsible to decide upon the candidate QRs.

QR documentation phase:

- *Documentation of QRs:* In case a QR is accepted by the decision maker, it is forwarded to the backlog. The strategic dashboard provides the user with a link through which the accepted QR is automatically transferred into the organization's requirements backlog. The strategic dashboard itself does not depend on any fixed backlog management technology (e.g., Jira, Taiga, …) but rather utilizes the link mechanism to transfer the data content of the accepted QR to the backlog. Establishing the link between the strategic dashboard and the backlog is a task done in the Q-Rapids set-up actions.

We formalized the different steps and elements required by this requirements generation and documentation process by means of a Business Process Model and Notation (BPMN) process model (see Figure 3). The process starts when a quality alert is triggered. The quality alert is then notified to the decision makers by sending the *«artifact» quality alert*. The decision makers evaluate the quality alert and request the QR patterns to resolve it. Q-Rapids obtains from the *«repository» QR patterns catalog* those *«artifact» QR patterns* that are able to resolve the quality alert. The decision makers select and instantiate the QR pattern, generating the *«artifact» QR*, which is finally stored in the *«repository» Backlog*.

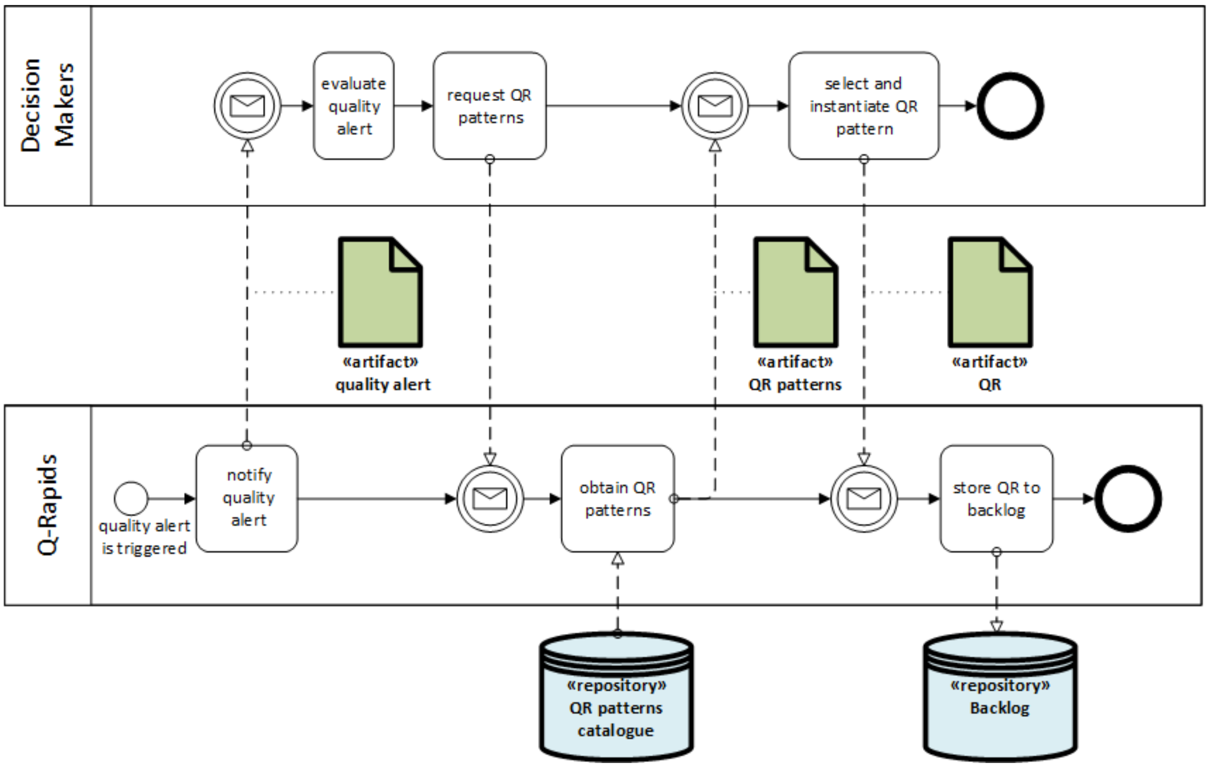


Fig. 3. Main tasks and artifacts of QR generation and documentation: BPMN process model.

In the following subsections, we will describe in detail how we designed these artifacts (i.e., *quality alert, QR patterns,* and *QR*) and repositories (i.e., *QR patterns catalog* and *Backlog*).

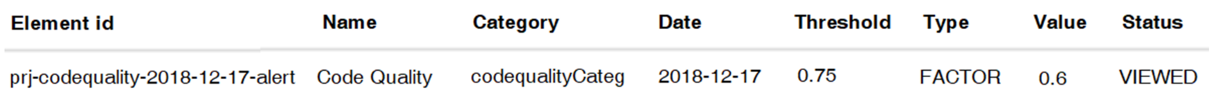
4.1 Quality alert artifact

In a previous work, we defined a quality model based on expert knowledge stemming from the companies participating in the Q-Rapids consortium (Martínez-Fernández et al. 2018a). The nodes of the quality model are of different types depending on the abstraction level: At the highest level, there are project indicators (e.g., product quality), which are decomposed into quality factors (e.g., code quality), which are in turn decomposed into quality metrics (e.g., duplicated lines of code).

Starting from this quality model, we defined customizable thresholds on each of the different nodes in order to raise a quality alert if such a threshold is violated. We defined the quality alert artifact in JSON with the following metadata:

- *Element id***:** a unique identifier for the alert.
- *Name*: description of the alert.
- *Type*: identifies whether the alert is at the quality metric, quality factor, or project indicator level.
- *Category*: used to bind the alert with the QRs that can solve it. This information is obtained from the node of the quality model that raised the alert and is defined at design time.
- *Date*: date on which the alert was raised.
- *Status*: identifies the state of the alert: new, viewed, or processed.

The process of raising a quality alert can be illustrated with the following example: A company is using a quality model that includes several quality factors and metrics. One of these quality factors is *Code quality*, which has gone down until reaching the value 0.6 (all values in the quality model are normalized from 0 to 1, where 0 is the worst case scenario and 1 is the best case). In this case, the monitored value is below the threshold defined for this quality factor, which was set to 0.75 (this threshold was defined by the company based on historical data from similar projects and their experience with it). Due to this situation, a quality alert is raised for *Code quality* (see Figure 4).



Element id	Name	Category	Date	Threshold	Type	Value	Status
prj-codequality-2018-12-17-alert	Code Quality	codequalityCateg	2018-12-17	0.75	FACTOR	0.6	VIEWED

Fig. 4. Example of quality alert in Q-Rapids

4.2 QR patterns catalog repository

To design and instantiate the QR patterns catalog, we used the PABRE framework (Renault et al. 2009) and extended it to support the QR generation process. PABRE is a tooled framework that facilitates reuse in requirements engineering by using requirement patterns. PABRE provides the capability to define a repository with a list of QR patterns that, among other features, can be classified according to a schema following the same tree-structured form as the quality model.

This allows for a clear mapping between the categories of the quality alerts generated and the QRs that can solve them. A generic catalog of QR patterns is available as part of the supporting material of this paper (Oriol et al. 2019b).

It is worth mentioning that such an approach makes it possible to have multiple QR patterns for a given quality alert, or for a quality alert at the quality factor level to be resolved by the QR patterns bounded to the quality metrics that decompose this quality factor.

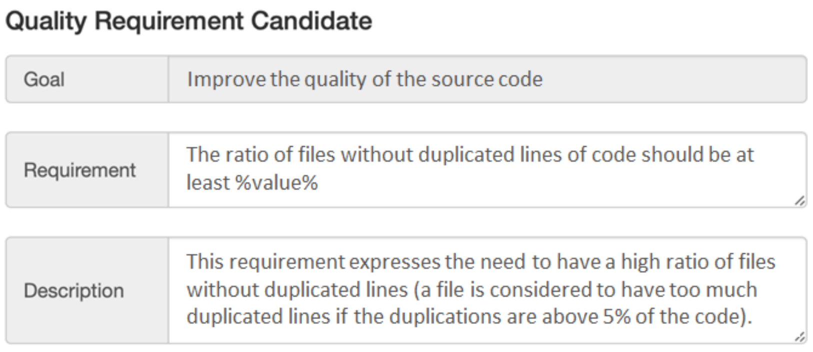
For instance, referring to the previous example, PABRE can retrieve the QRs that are able to resolve the alert of the *Code quality* factor. In this case, the QR patterns are: *ComplexFilesReq* (which aims to reduce the proportion of files with high cyclomatic complexity), *CommentedFilesReq* (which aims to reduce the proportion of files with a high number of commented lines of code), and *DuplicationsReq* (which aims to reduce the proportion of files with a high number of duplicated lines of code). All these QR patterns are bound to the quality metrics *Complex files*, *Commented files,* and *Duplications*, respectively, which are quality metrics of the quality model that decomposes *Code quality*.

4.3 QR patterns artifact

The internal structure of a QR pattern is also based on PABRE (Palomares et al. 2013) and has been tailored to the specific needs of Q-Rapids. A requirement pattern includes several metadata as described and specified in (Franch et al. 2010). However, from the point of view of the decision makers, just the following information is visible:

- **Goal**: describes the objective or problem that the QR pattern aims to solve.
- **Requirement form**: the textual form of the QR pattern. In this textual form, one or more formal parameters can be defined. The formal parameters are free variables that need to be instantiated by the decision makers to produce the QR.
- **Description**: a detailed description of the QR pattern.

An example of the QR pattern *DuplicationsReq* is depicted in Figure 5.



Quality Requirement Candidate

Goal	Improve the quality of the source code
Requirement	The ratio of files without duplicated lines of code should be at least %value%
Description	This requirement expresses the need to have a high ratio of files without duplicated lines (a file is considered to have too much duplicated lines if the duplications are above 5% of the code).

Fig. 5. QR pattern – DuplicationsReq

4.4 QR artifact

A candidate QR is an instantiation of the QR pattern. In particular, it is the result produced by the decision maker after instantiating the formal parameter(s) of the QR pattern with actual values that resolve the quality alert. To assist the decision maker in this task, the Q-Rapids platform provides simulation techniques that show the impact of the instantiated QR on the elements of the quality model. Such simulation techniques are based on Bayesian networks as proposed in the VALUE framework (Mendes et al. 2018).

Continuing with the example described above, the decision maker could instantiate the QR patterns provided by Q-Rapids (i.e., *ComplexFilesReq, CommentedFilesReq, DuplicationsReq*) with different values for their parameters and evaluate the impact that these instantiations have on the quality model. During this simulation process, the decision maker can experiment with different alternatives and combinations in order to determine which QRs to add. For instance, after checking different combinations, the decision maker might choose to instantiate *DuplicationsReq*, setting its value to 85% (leading to the QR "The proportion of files without duplicated lines of code should be at least 85%"), and *ComplexFilesReq*, setting its value to 70% (leading to the QR “The proportion of files with low cyclomatic complexity should be at least 70%”) since, according to the results of the simulation, these two QRs combined would improve the value of *Code quality* to 0.7, which is above the defined threshold, hence resolving the quality alert.

4.5 Backlog repository

In a previous study (W. Behutiye et al. 2017), we found that companies adopt different practices and tools for documenting QRs. Hence, the Q-Rapids approach for integrating generated QRs in a project’s requirements backlog considers various documentation practices (e.g., hierarchy level, description, decisions on who documents the generated QR, etc.), as well as multiple requirements management tools (e.g., JIRA, openProject, etc.). To address this heterogeneity, Q-Rapids uses a generic service interface to link the QRs generated in Q-Rapids to a project’s backlog. Such a service interface can have multiple implementations to meet the needs of each requirements management tool and can be tailored to the specific needs of a company. Hence the generated QRs can be added to the project’s requirements backlog following the specific practices adopted by a company.

5. Evaluation of QR generation and documentation process

In order to evaluate the artifacts and repositories of the above-defined process, we designed an evaluation that involved the participation of the four Q-Rapids’ companies following a structured workshop format.

5.1 Goal and design

The goal of the workshop was twofold: on the one hand, to validate the QR generation (i.e., the process and its artifacts) and, on the other hand, to conduct an exploratory study of the QR documentation process (i.e., the step that documents the QRs into the backlog). Thus, we defined two research questions:
RQ1. Are the proposed artifacts *Quality alert*, *QR pattern*, *QR,* and *QR patterns catalog* adequate, complete, and not overwhelming for decision makers to generate and document QRs?
RQ2. What important characteristics should the documentation of QRs have in order to enable effective deployment of QRs into the organization’s backlog?

The workshop was structured into two parts according to the two research questions defined above.
The first part addressed RQ1 and started with a short presentation by the researcher describing the workflow and the structure and content of each of the artifacts and the repository. Also, an illustrative example akin to the one presented in Section 4 was presented by means of mock-ups. After the description of each artifact, the researcher asked the following questions:

- Is the amount of information provided adequate?
- Is there any information missing?
- Is the amount of information provided overwhelming?

Questions were asked orally to motivate discussion among the companies’ representative participants. For each question, the researcher codified the responses in a boolean form (yes/no), and in case the answer was negative (i.e., the amount of information was not adequate, there was information missing or the amount of information was overwhelming), the researcher made sure that participants described the particular issues they found. The participants were also invited to provide feedback or comments at any time.

The second part of the workshop focused on RQ2 to explore the QR documentation practices of the companies and identify important aspects for documenting the generated QRs in the projects' requirements backlogs. In contrast to RQ1, RQ2 did not aim at measuring any quantitative metric but rather obtaining open feedback from the participants. The researchers used findings from an earlier study with the companies regarding requirements documentation (W. Behutiye et al. 2017) to initiate the discussion. We used requirements documentation templates based on the requirements management tool applied in the projects (e.g., JIRA) to guide the discussion and asked the participants to identify aspects they considered important while documenting QRs, with the aim of obtaining lightweight and informative QR documentation.

5.2 Execution

As mentioned above, the workshops were conducted at the four companies of the consortium. The members of the companies who participated in the workshops were involved in the development process or the management of requirements for the software project used as a pilot test and acted as representatives of their respective development teams. Each workshop had between one and three members representing the company. Due to both the early stage of the models and the limited number of participants, the analysis was qualitative in nature. Three of the four workshops were conducted on the premises of the company, whereas the fourth one was conducted online. The workshops were conducted between June 12, 2018, and September 7, 2018. Their duration ranged from 124 minutes to 202 minutes. Details are summarized in Table 1.

Table 1. Summary of workshop execution.

Company	Company 1	Company 2	Company 3	Company 4
Country	France	Finland	Finland	Poland
Number of participants	2	2	3	1
Date of the workshop	June 19, 2018	June 12, 2018	June 13, 2018	September 7, 2018
On premise / On-line	On premise	On premise	On premise	Online
Duration of workshop	124 min	196 min	190 min	202 min

5.3 Data Analysis

The research data was gathered in the workshops by recording the discussions. The recordings were transcribed by a professional transcription company based in Finland to MS Word documents.

The research data was analyzed by using a combination of thematic synthesis and narrative synthesis (Cruzes and Dyba 2011; Cruzes et al. 2015). We opted for the combination of two synthesis practices because, at a detailed level, the practices of the case companies were highly company-specific.

The analysis was started by reading the Word documents and dividing the content into sections relevant for QR generation and QR documentation. This first-level division was necessary due to the fact that in the actual discussion, the interviewees sometimes commented on both viewpoints in parallel.

The documentation-specific sections of the Word documents were transferred into MS Excel tables, one for each case company; the sections were coded and the codes were grouped into higher-order themes according to the thematic synthesis principles (Cruzes and Dyba 2011). Excel was selected as the tool for the analysis because it is easy to share within an international network of researchers.

The themes identified in the four case companies were summarized and the consistency of the summarized themes was checked by using the principles of narrative synthesis (Cruzes et al. 2015).

5.4 Results

5.4.1 Results on QR generation (RQ1)

Quality alert artifact. All case companies answered that the amount of information provided in the alerts was complete and adequate (e.g., "*to me it looks like the most important information*"), as well as not overwhelming (e.g., "*it seems quite clear*"). Most companies' representatives also provided valuable feedback and ideas based on their needs in order to improve this quality alert mechanism. All companies pointed out the need for top-down traceability in order to have "*a direct way to access the raw data*" since "*this does not point to the, let's say, guilty part of the software*". Apart from top-down traceability, most participants also required bottom-up traceability. That is, given a quality alert at a lower level (e.g., at the quality metric level), they wanted to be able to visualize the values of the upper levels "*to know that these metrics go to this factor*", even though their values are not violated. Finally, one company pointed out the importance of having easy-to-understand naming on the elements to improve their learnability; otherwise, this "*might lead to confusions*".

QR patterns artifact. All companies answered that the amount of information provided in the QR patterns was "*not overwhelming*" and they did not report any information missing.

Regarding the adequacy of the information, some companies' representatives asked to make the terminology of what is commonly understood as QRs more explicit; e.g., one participant requested "*something like stability or security or maintainability*". As another participant pointed out, the presented QR patterns are "*very low level*". To address this issue, one participant suggested that "*non-functional requirement-related keywords could be somehow highlighted in the text. So, that would give a clearer understanding that this relates for example to performance issues*".

QR artifact. The QR artifact is an instantiation of the QR pattern, therefore the information provided was also perceived as not overwhelming. In terms of adequacy, one company pointed out the need to be able to customize the message when instantiating the QR pattern into a QR in order to provide specific details. "*I would like to be able to customise the message*". In terms of completeness, one participant pointed out the need to see right away the "*simulation, in order to [see] what will be the result of the execution of these quality requirements*".

QR patterns catalog repository. All companies considered the QR patterns catalog adequate, complete, and not overwhelming. As valuable feedback, they pointed out the need to easily "*have the ability to add a new QR pattern*" as they evolve the quality model. One company went one step further in this direction and suggested adding the ability to extend the QR patterns catalog on demand. That is, if there is no QR pattern that can resolve a particular quality alert, there should be the possibility to extend the QR patterns catalog dynamically ("*one action button I'm missing here is, when there is no QR [to solve an alert], so there should be a button add one [QR pattern] to the catalogue*").

5.4.2 Results on QR documentation (RQ2)

The participants of the workshops raised documentation-related topics important for effective deployment of the QRs generated by the Q-Rapids solution: 1) backwards traceability, 2) information content and end-user value, 3) understandability of QRs, and 4) interfacing to the processes and tools deployed in a company.

While the QRs presented were derived from quality issues aggregated from raw quality data by the Q-Rapids quality model, the users of all involved companies highlighted backwards traceability as a key aspect when planning corrective actions for an accepted and documented QR. As one practitioner stated: "*So basically if we violate in the development phase something, some quality requirements we already have, we should be able to trace back what requirements we are violating*".

The companies had established, well-implemented processes and practices for ASD and quality assurance, and several tools for gathering and reporting quality-related information were in use. This places requirements on the documentation of QRs – the information content of the QRs must be exact and must fit the processes and practices of the company. This topic was taken up by all companies and is well highlighted in a discussion between the researchers and a practitioner: "*But a comment cannot be a mandatory field or is it, will it be used by Q-Rapids?*" - "*It's not mandatory though, it's.*" - "*Yeah but okay, do you have a vision that how quality requirements on Q-Rapids could benefit from this comment field information?*".

The companies differed from each other in terms of the stability of the deployed processes and tools they used. One had fairly stable processes and requirements repository tools, one was in the middle of changing to a new tool, and one company was improving the processes and tools in a continuous manner, resulting in a situation where several requirements repository tools were in use in different parts of the organization. Such a situation generates challenges for the automatic link for QRs between the Q-Rapids strategic dashboard and the requirements repositories, meaning that there would not be any one-fits-all solution: *"But then the question is which backlog." - "So you have different backlogs following that?" - "Yes...Should we then cover all of, the basic question is that if we are thinking about this mapping and our next step in Q-Rapids, should we select one of those and omit others?"*

6. Tool-supported generation and documentation of QRs

Based on the results of the previous evaluation, we were able to refine the artifacts and start the implementation of the components that automate the QR generation and documentation process.

The implemented components have been integrated with the Q-Rapids strategic dashboard, which orchestrates the execution flow, offers decision makers an easy-to-use user interface, and provides additional functionalities, such as quality assessment or what-if analysis (López et al. 2018). Figure 6 depicts the architecture of the implementation.

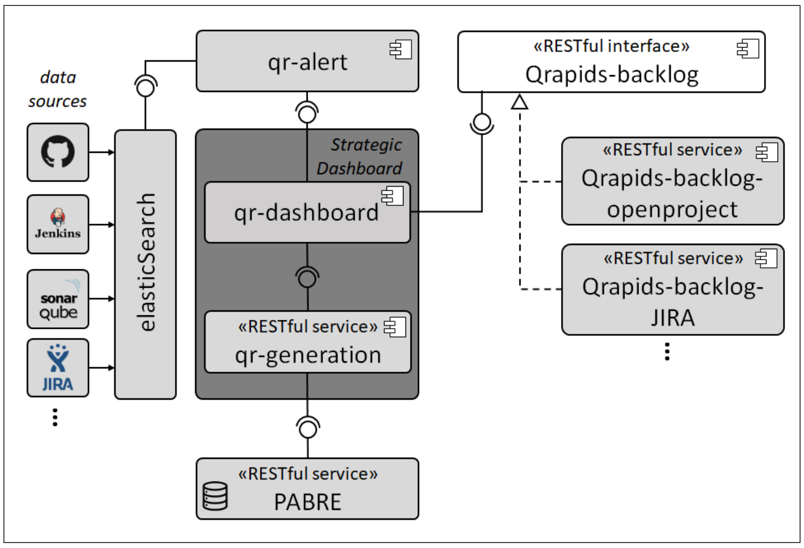


Fig. 6. Architecture of the tool-supported QR generation and documentation

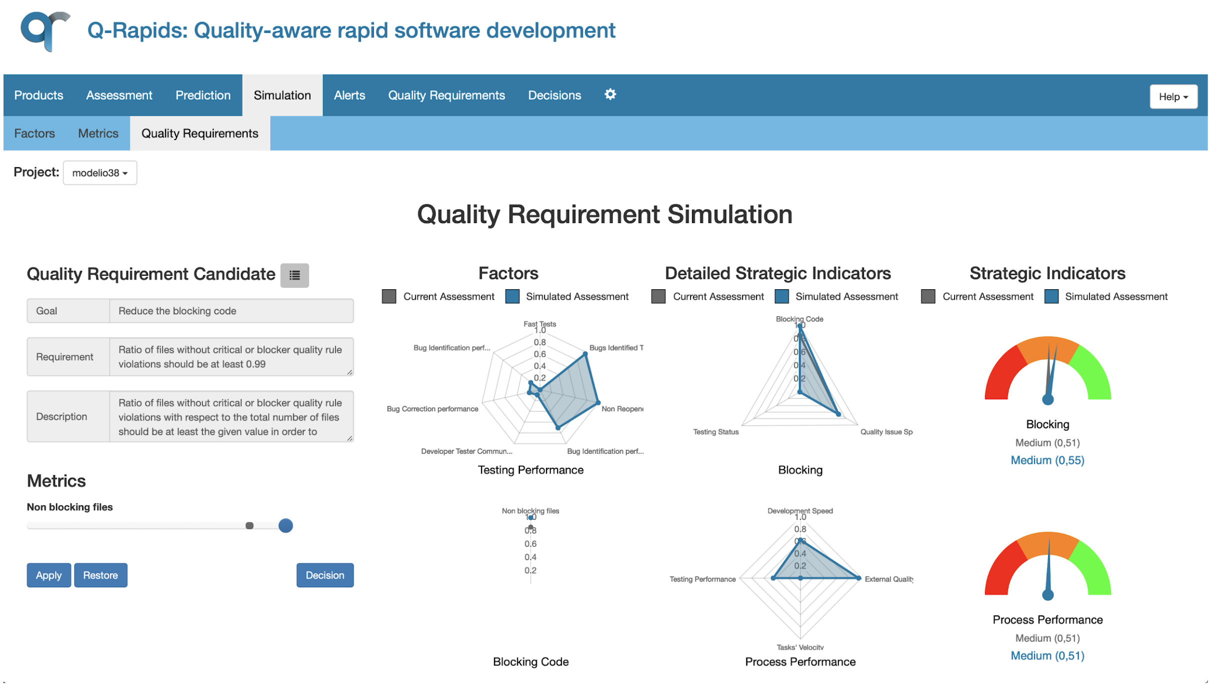
The modules implemented during this phase were:

- *qr-alert*: This module automates the process of evaluating the elements of the quality model and raises an alert if a threshold is violated. To collect the data, *qr-alert* is connected to elasticSearch, which is a data sink containing all the data coming from multiple data sources (e.g., GitHub, Jenkins, SonarQube, Jira,...). ElasticSearch stores not only the raw data, but also the aggregated metrics and factors of the defined quality model. Decision makers can specify the threshold for each of the nodes of the quality model and receive a notification when a violation is triggered. The module can be triggered in a time-based manner and can be configured in terms of running intervals during the day. Moreover, the module allows the user to specify more complex activation rules (i.e., more complex than simple threshold-based conditions) that will trigger a quality alert. Users can use any timespan (e.g., range) or a specific date for executing the rule.
- *qr-generation*: This module automates the process of retrieving the candidate QR patterns that resolve a qr-alert. The module connects to PABRE through its RESTful interface and identifies whether a quality alert can be resolved by a QR pattern. If so, it provides a list of QR patterns that can resolve the quality alert to the qr-dashboard.
- *qrapids-backlog-**: Each module of this type is used to store the generated QRs in the backlog. The module defines a common RESTful interface that can have multiple implementations, enabling Q-Rapids to connect to multiple backlogs. The RESTful interface defines the method *addToBacklog*, which receives a QR as a parameter and returns a tuple with the identifier and URL of this QR after including it in the backlog, which enables traceability. At the current stage, there are two implemented services, connecting to the OpenProject (*qrapids-backlog-openproject*) and to the Jira (*qrapids-backlog-jira*) backlog, respectively.

The modules extended during this phase were:

- *PABRE*: implements the QR patterns catalog. It has been extended with the required functionalities for retrieving, from the catalog, the requirement patterns bound to the elements of the quality model that have raised an alert. Furthermore, it has been enhanced with further functionalities, such as the ability to easily extend the catalog through import/export functions as well as with methods to dynamically add, update, and delete existing QR patterns in the catalog.

- *qr-dashboard*: implements a quality-aware strategic dashboard with multiple functionalities, such as quality-assessment, forecasting techniques, and what-if analysis (Lopez et al. 2018). It has been extended to orchestrate the interaction between the alerts and the *qr-generation* module. In this regard, the *qr-dashboard* is able to maintain traceability of the whole process. Furthermore, candidate QRs obtained by *qr-generation* have been integrated with the what-if analysis functionality, providing the capability to show the impact that adding a QR would have on the quality factors and project indicators of the quality model. Figure 7 shows an example of the result of such a what-if analysis. As shown, the QR pattern is displayed on the left side of the screen, with a slider for modifying the values of its variable(s) in order to instantiate the QR pattern into a QR. The variable that instantiates a QR is bound to a specific element of the quality model (in this case, a quality metric), and the values of the upper layers of the quality model are recomputed. On the right side, the impact that such a QR would have on the quality factors and project indicators is shown in radar and gauge charts.



Goal	Reduce the blocking code
Requirement	Ratio of files without critical or blocker quality rule violations should be at least 0.99
Description	Ratio of files without critical or blocker quality rule violations with respect to the total number of files should be at least the given value in order to

Fig. 7. Screenshot of the what-if analysis feature with QRs.

The implementation and documentation of all components is available in GitHub. Table 2 lists the GitHub repositories of each of them. The first column indicates the task of the tool-supported generation and documentation of QRs (see Figure 3). The second column indicates the implemented component. The third column shows the repository where the tool is available in GitHub.

Table 2: GitHub repositories of the components

Implemented components		
Task	**Component**	**Repository**
Notify quality alert	qr-alert	https://github.com/q-rapids/qrapids-alert
Obtain QR patterns	qr-generation	https://github.com/q-rapids/qrapids-qr_generation

Store QR to backlog	qrapids-backlog-*	https://github.com/q-rapids/qrapids-backlog-openproject https://github.com/q-rapids/qrapids-backlog-jira
Extended components		
Task	**Component**	**Repository**
«repository» QR patterns catalog	PABRE	https://github.com/OpenReqEU/requirement-patterns
Decision makers' tasks	qr-dashboard	https://github.com/q-rapids/qrapids-dashboard

7. Evaluation of tool-supported QR generation and documentation

7.1 Goal and design

Regarding the aforementioned implementation of automating the QR generation and documentation process, we aimed at understanding the relevant information and system quality aspects of tool-supported generation and documentation of QRs mainly from the perspective of product owners, managers, and developers. Moreover, we aimed at identifying the key aspects that such tools should have according to practitioners.

To measure and evaluate this practical focus of the tool-supported generation and documentation of QRs, we used key information quality (i.e., the quality of the data included in a system to support users' tasks) and system quality (i.e., the system's functionalities and the user experience to support users' tasks) aspects (DeLone and McLean 2003). Such quality aspects have been proven to be suitable for collecting useful insights to assess and guide the further development and improvement of systems (Guzmán et al. 2017; McKinney et al. 2002; Nelson et al. 2005). Specifically, in this study, we focused on the degree to which product owners, managers, and developers perceive the tool-supported generation and documentation of QRs as easy to use, understandable, complete, efficient, reliable, satisfying, and useful. Thus, we defined the third research question:

RQ3. Information quality and system quality of the tool-supported generation and documentation of QRs – To what extent do product owners, managers, and developers perceive these functionalities as understandable, complete, useful, reliable, easy to use, efficient, and satisfying?

We complemented this evaluation by gathering qualitative data from practitioners. Those perceptions include the strengths of the tool-supported generation and documentation of QR as well as improvements. The goal is to understand the key aspects from the perspective of practitioners in order to answer our fourth research question:

RQ4. Relevant characteristics of tool-supported generation and documentation of QRs – What are the key characteristics that practitioners require in tools supporting the generation and documentation of QRs?

7.2 Execution

Procedure and Instruments: We performed the following activities with the participants: They (1) received a live demo based on the installation of the tool-supported generation and documentation of QRs in their real project; (2) performed a task independently trying out the tool-supported generation and documentation of QRs and commented on this task; (3) filled out a structured questionnaire on the information and system quality of the tool-supported generation and documentation of QRs; (4) participated in a debriefing session about the key aspects of the tool-supported generation and documentation of QRs, including strengths and suggestions for improvement. After each step, the experimenters asked for feedback from the participants and clarified any issues if necessary. Finally, once the functionalities had been presented and evaluated, there was a semi-structured feedback session to further collect more details on the key aspects, including strengths and suggestions for improvements.

We used reliable instruments for the different quality aspects to operationalize the appropriateness of the tool-supported generation and documentation of QRs based on the Likert scales described in the literature. We used the information quality instruments *understandability* and *usefulness* by McKinney et al. (2002) and *right level of detail* by Goodhue and Thompson (1995). For the system quality aspects, we used the instruments *reliability* by McKinney et al. (2002), *perceived ease of use* and *perceived usefulness* by Venkatesh and Bala (2008), and *efficiency* by Xu and Ramesh (2008). Each Likert scale includes up to four statements to be rated using a response scale from 1: strongly disagree to 5: strongly agree and an additional "I don't know" option. We instantiated the selected questions according to the purpose and content of the tool-supported generation and documentation of QRs. The instruments used during the evaluation (e.g., questionnaires given to participants to gather data) are available online (Oriol et al. 2019b).

Population and Sampling: The target users of the tool-supported generation and documentation of QRs are mainly product owners, managers, and developers. We communicated the target sample to the industry partners and they contacted and proposed a list of participants based on their suitability. Then, we drew a convenient sampling (Daniel 2012) including product owners, managers, and developers of the companies involved in the Q-Rapids project. At the time of the evaluation, the participants were team members of the pilot project selected in each company, who previously used the tool-supported generation and documentation of QRs.

Execution: Between July and October 2018, we deployed the tool-supported generation and documentation of QRs in the four Q-Rapids companies. In parallel, we trained the experimenters and observers responsible for performing the evaluation at each company. After collecting project data for the last ten months in each company (since January 2018), we evaluated the tool-supported generation and documentation of QRs following the procedures described above between October and November 2018. We scheduled each evaluation session for up to three hours, taking into consideration the availability of the participants. One researcher acted as the experimenter and at least one researcher acted as the observer.

7.3 Data Analysis

The experimenter and the observer transcribed the participants' answers (on the tasks, questionnaires, and cards from the open feedback sessions) and the observation notes into a protocol consisting of three standardized Excel templates (one for each type of answers by the participants). This served to keep the data analysis consistent among companies. Then we carried out the quantitative and qualitative analysis.

We first carried out within-case analyses of the quantitative and qualitative data collected for each company. Then we compared, related, and integrated the results among the companies (cross-case analysis) (Miles and Huberman, 1994). We report descriptive statistics including *sample size (N)*, *median (Mdn)*, *minimum (Min)*, *maximum (Max)*, and *modal value (Mode)* for the quantitative analyses. In addition, we performed a *One-Sample Wilcoxon Signed Ranks Test* (Wilcoxon 1945), as it is suitable for testing with small samples whether the participants rated the quality aspects more positively or more negatively, i.e., for checking whether or not the answers are significantly lower or higher than a selected middle point, thus H_0: Mdn (x) = 3 (the neutral point). In addition to our descriptive results, we report the *standardized test statistic (T*)* and the *significance levels (p)* of the One-Sample Wilcoxon Signed Ranks Test. We used IBM SPSS Statistics 19 (including IBM SPSS Exact Tests for analyzing small samples) and set the confidence level of the test at 95% (i.e., $\alpha = 0.05$).

Regarding the qualitative analysis, we used data-driven thematic analysis (Braun and Clark 2016) to analyze participants' feedback on the tool-supported generation and documentation of QRs. At least two researchers derived themes – i.e., explicitly mentioned suggestions of improvement – inductively by coding and interpreting all observation protocols independent of each other. Then they compared their results and resolved any deviations. Moreover, we performed several peer-review meetings including all experimenters, observers, and analysts to review the interpretations of the elicited qualitative data. This served to keep the qualitative analyses grounded on the collected evidence and ensured scientific rigor.

7.4 Results

In total, 22 persons participated in the evaluation of the tool-supported generation and documentation of QRs. Among these participants were three product owners, seven project managers, six managers (including quality and technical managers), three developers, and three participants who did not check their role in the demographics questionnaire but who belong to one of these categories. All participants had at least half a year of work experience in their companies (Mdn = 10, Min = 0.5, Max = 32) and at least three months of work experience in their current role (Mdn = 5, Min = 0.25, Max = 30).

The results are supported by the data collected during the structured questionnaires (mainly quantitative data about different quality aspects of the tool and included features), and semi-structured moderated sessions and observations (mainly qualitative data analyzed as strengths and suggestions for improvement).

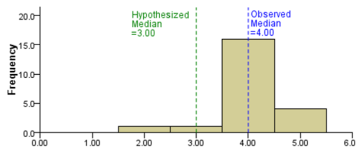
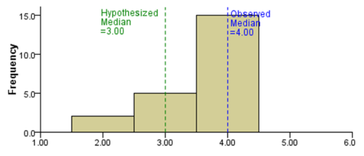
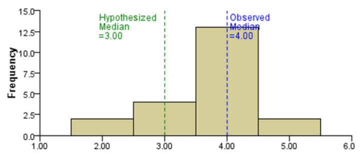
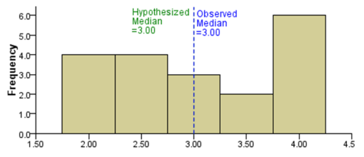
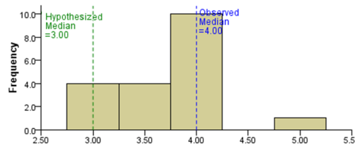
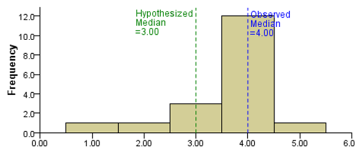
7.4.1 Information quality and system quality of the tool-supported generation and documentation of QRs (RQ3)

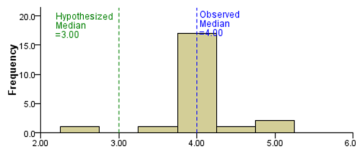
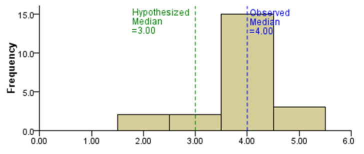
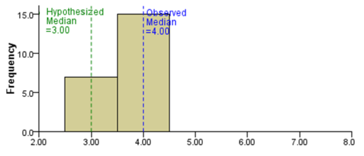
Table 3 summarizes the results. For the information quality aspects, the questions focused on the QR patterns and the corresponding catalog as part of the tool-supported generation and documentation of QRs. Overall, almost all participants perceived the QR patterns as *understandable* (N = 22, Mdn = 4, Min = 2, Max = 5, Mode = 4, p = .000, T* = 3.824; see Table 3). They also perceived them as complete in the sense that the provided information content is at the *right level of detail,* even though some answers differ (Mdn = 4, p = .002). Moreover, the majority of the participants considered the information within the QR patterns *useful* for their work (Mdn = 4, p = .002).

With respect to system quality, the questions focused on the technical functionalities of the tool-supported generation and documentation of QRs. The participants agreed that the generation of concrete

QRs by the tool-supported generation and documentation of QRs based on these QR patterns and the available data is *reliable*, as this result is also statistically significant (Mdn = 4, p = .000). Their perceptions regarding the appropriateness of the tool-supported generation and documentation of QRs for their project context were, on average, neutral, but they had opposite opinions and the result is not statistically significant (Mdn = 3, p = .668). In addition, the results of the perceived *ease of use* regarding the procedure of the tool-supported generation and documentation of QRs, i.e., receiving an alert, inspecting the alert in the list of alerts, clicking the QR button and deciding what to do with the suggested QR (i.e., to integrate it in the repository or not), are positive and statistically significant (Mdn = 4, p = .000). The participants perceived the tool-supported generation and documentation of QR as *efficient* (Mdn = 4, p = .001). In general, the participants agreed that the tool-supported generation and documentation of QRs closes the loop from data to QRs in a *satisfying* way (Mdn = 4, p = .000).

Table 3. Quantitative analysis of the tool-supported generation and documentation of QRs

Participants' perception regarding …*	N	Mdn	Mode	Min	Max	One-Sample Wilcoxon Signed Ranks Test	p	T*
Understandability	22	4	4	2	5		.000	3.834
Right level of detail	22	4	4	2	4		.002	3.153
Usefulness	21	4	4	2	5		.002	3.120
Appropriateness	16	3	4	2	4		.668	.428
Reliability	19	4	4	3	5		.000	3.535
Perceived usefulness	18	4	4	1	5		.018	2.368

Perceived ease of use	22	4	4	2,5	5		.000	4.296
Efficiency	22	4	4	2	5		.001	3.244
Satisfaction	21	4	4	3	4		.000	3.873

*: Each quality aspect was measured using a valid and reliable Likert scale. Each item was rated using a five-point response scale from 1: strongly disagree to 5: strongly agree and included the option "I don't know".
One-Sample Wilcoxon Signed Ranks Test: H_0: Mdn(x) = 3 with 95% confidence level

7.4.2 Key characteristics of tool-supported generation and documentation of QRs (RQ4)

During the evaluation, the participants also provided more feedback during the semi-structured group feedback session and mentioned aspects that are important for the tool-supported generation and documentation of QRs in their pilot projects. A table detailing those key aspects is available at (Oriol et al. 2019b).
In general, five participants from three use cases emphasized the usefulness and relevance of the alert and QR functionalities because *"[the tool] includes "closing the loop" [between collected data and QR and] making items actionable". "Overall the alert /QR approach is excellent"* and *"[the alerts functionality] will prevent from actively checking [the] tool."* Furthermore, the data-driven approach enables early detection of issues and systematic QR generation and documentation. Four participants from two use cases highlighted that aspect and explained that for them, it is now easier to address a QR with information support due to the data considered for the generation and documentation of this QR. Overall, the data-driven approach enables *"early detection of issues"* and *"better detection of quality issues which arrives to the customer"*. Another participant summarized: *"[the] systematic way of creating QR based on what we measure is good"*. Besides this aspect, three participants from two use cases also emphasized the traceability between the dashboard and the respective backlog, i.e., *"the link from the [Q-Rapids] dashboard to the respective backlog where the generated QR is saved"*. This *"enables traceability and an easy access to the generated QR"*. Furthermore, one participant emphasized that the alerts and QRs are tailorable because it is possible to develop and add new QRs based on new factors, metrics, etc.

In addition, the participants provided feedback to further increase the usefulness of the implemented functionalities: Six participants across all use cases mentioned that QRs should be simple and linked to concrete/actionable tasks. Currently, QRs are not meaningful in terms of concrete actions to do for some participants. One person said: *"[The] QR "decrease complexity" means basically nothing; it is not understandable for people. It should be more precise how to do it (e.g., reduce LOC, use different patterns, etc.). [A] tool should help to identify where to address the complexity. [...] [For example],* *tasks for Git needs to be more specific, [e.g. a] list of specific files with a too high complexity would be fine."* QRs should include precisely how they can be resolved and what their fit criteria are (e.g., *"[The elicited] QR is a good epic to Jira but developers need more detailed tickets for specific tasks"*). Therefore, QRs should be split into concrete actions, including upper and lower limits, the components involved, and the cause of the issue. These improvements should enable the user to select the best possible alternative (when QRs are generated). Five participants from three use cases reported that the QR patterns catalog should be enhanced by improving correctness. For instance, the participants recommended checking whether the calculations are correct or whether the descriptions are complete because the text of *"some [patterns] might be dis-leading, e.g., test coverage [or] some needs further definition, e.g., critical error in errors at runtime"*. Furthermore, they proposed including only what is understood as a QR in the field of software engineering, i.e., *"common knowledge in the SE domain"*. On the other hand, seven participants from two use cases suggested providing information when adding or ignoring QRs, i.e., *"some more data probably needs to be input when filling in the QR"*. New fields could ask for information about the person who created the QR, the kind of issue (e.g., "error"), the definition of done, and the target release/date. Also, *"sometimes one may want to pinpoint/assign responsible component/team"*. In addition, *"when one ignores a proposed QR, there should be a provision for that person to document why exactly that QR is being ignored"*. Further support for the user in terms of more guidelines and more explanations of the tool-supported generation and documentation of QRs was suggested by two participants from two use cases. In particular, guidelines for the usage of the alerts and the requirements patterns were desired, for example, guidance on how to set proper values for thresholds and inclusion of explanations of the logic underlying the generation and documentation of a QR. The transparency of the tool-supported generation and documentation of QRs would be improved based on such explanations.

8. Threats to validity

During the design of the evaluation of the QR generation and documentation process and its tool support, we identified several threats to validity. Based on them, we emphasize below the mitigation actions applied:

Construct validity: During the evaluation of the QR generation and documentation process, in order to minimize threats from construct validity that may arise due to misunderstanding of concepts among researchers and participants of our study, the researchers moderating the workshops clarified concepts for the participants during the workshops and ensured that there was a common understanding of the concepts that were discussed. Additionally, we used key informant techniques to involve participants with varying roles and backgrounds in our study. The selection proved positive, as it enabled us to collect relevant feedback from varying points of view of the intended users of the artifact. For instance, we collected viewpoints from product owners, project managers, and test leads.

In the evaluation of the tool-supported QR generation and documentation, we used validated, reliable constructs (see Section 7.2) from the literature to correctly operationalize the identified key information and system quality aspects. Complementarily, a later debriefing session enabled us to further elaborate the practical relevance and perceptions of the users of the tool for eliciting QRs. The use of quantitative and qualitative measures and observations reduced mono-method bias. Furthermore, we aimed at creating a safe environment, encouraging the participants to highlight any negative aspects and make suggestions for the improvement of the tool-supported QR generation and documentation.

External validity: This entails the generalizability of the findings of a study to external contexts. regarding both evaluations, the generalizability of the findings is limited to the context of the studied use cases. Still, our goal was to better understand practitioners' perception. Because the companies involved in the study are from different domains, countries and size, we believe that the findings from our study can be extended to multiple companies with an Agile context, which can adopt the tasks and artifacts of the QR generation and documentation presented in Figure 3.

Internal validity: In both evaluations, to mitigate threats from internal validity, more than one researcher was involved during the data analysis steps to avoid any human error or possible bias. However, we drew a convenient sample. Therefore, one limitation of our work that could not be avoided is that we were not able to get a random sample of participants in the pilot projects of the companies.

In addition, we defined an evaluation protocol in advance. We included, for instance, a specific description of our planned procedure and the order for using the materials for the evaluation of the tool-supported QR generation and documentation, i.e., a script of the demonstrations to the participants, the tasks, the questionnaire, and an explanation with all the steps that had to be performed by the experimenter and the observer. After all the involved parties had agreed on the final version of the evaluation guidelines, we executed the evaluation accordingly. This should mitigate the fact that we needed to split the work of conducting the evaluation among different researchers and companies. Some of the researchers who conducted the evaluation were involved in developing the tool-supported QR generation and documentation. To minimize that bias, we made sure that in each session, at least two researchers were present, one acting as the moderator/experimenter and one as an observer, to emphasize that the participants could speak freely.

Conclusion validity: This relates to the repeatability of the procedures followed in the study. In our study, the data collection and analysis procedures, including the creation of instruments for the implementation and execution, were documented in detail and carried out systematically. Therefore, we applied these procedures and instruments to execute the evaluation and conduct the analysis of the findings from all the use cases. During the analysis, we involved researchers who had not been involved in the creation of the tool support for QR generation and documentation. In this way, we mitigated risks such as using poorly designed instruments or fishing for results during the analysis, which would have led to a subjective analysis. Furthermore, we were aware that we would only get a small sample size (i.e., 22 participants) and looked for appropriate statistical tests.

9. Conclusions and future work

In this paper, we presented a data-driven, tool-supported approach for semi-automatically generating and documenting QRs in the context of ASD.

The approach is based on a quality model with project indicators, quality factors, and quality metrics obtained from expert knowledge. All those elements of the quality model have a customized threshold that, if violated, triggers a quality alert. Subsequently, the quality alert triggers the selection of candidate QR patterns obtained from a catalog to resolve the issue. Decision makers can instantiate the QR patterns into concrete QRs and, by applying what-if analyses, see the impact that this QR would have on the quality model. If the QR is selected, the QR is stored in the product backlog.

Our proposed solution is part of the Q-Rapids framework, which aims at improving QR management in an Agile ecosystem. Over the course of one year, four companies differing in size and profile followed the presented approach and deployed the tool to generate QRs in several projects. We studied the perception of practitioners regarding the tool-supported generation and documentation of QRs, i.e., their perception with regard to the tool's understandability, reliability, completeness, efficiency, usefulness, and satisfaction. The results of this evaluation show that the participants perceived the tool-supported generation and documentation of QRs as positive. For instance, getting alerts to ensure that the quality requirements are maintained throughout the project lifecycle and the possibility to predict and simulate the quality was perceived as a great feature highly valued by the participants. Furthermore, data-driven decision-making reduces manual guess work. Finally, we also elicited key aspects of tools supporting the generation and documentation of QRs.

The evaluations and experiences in four pilot projects in realistic Agile contexts show evidence of the feasibility of our proposal in industry. Agile companies that have the required heterogeneous data sources that can flexibly adapt to the QR generation and documentation process presented here could benefit from having semi-automatically elicited QRs directly in their backlogs. However, there still remain challenges, such as customization of the QR patterns catalog, which should be aligned with company-specific objectives, consideration of false positives with historical data, and more importantly, the initial investment required to start using this type of tool in projects.

Future work will consider the aforementioned challenges regarding the tool-supported generation and documentation of QRs in realistic Agile contexts. Moreover, we plan to include advanced functionalities in our tool, such as adding cost functions to estimate the effort to implement QRs, adding mechanisms to estimate the severity of a QR, and advanced Artificial Intelligence techniques to predict the violation of thresholds in order to trigger quality alerts of possible future issues in advance. Regarding this last functionality, the alerts would enable decision makers to generate QRs with enough time for their implementation before a quality issue is noticeable or surpasses any threshold.

Acknowledgments. This work is a result of the Q-Rapids project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement no. 732253.

References

Abbas, N., Gravell, A. M., Wills, G. B. (2010). The Impact of Organization, Project and Governance Variables on Software Quality and Project Success. In: Proceedings of the 2010 Agile Conference.

Bartsch, S. (2011). Practitioners' Perspectives on Security in Agile Development. In: Proceedings of the 6th International Conference on Availability, Reliability and Security (ARES).

Behnamghader, P., Alfayez, R., Srisopha, K., Boehm, B. (2017). Towards Better Understanding of Software Quality Evolution through Commit-Impact Analysis. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS).

Behutiye, W. *et al.* (2017). Non-functional Requirements Documentation in Agile Software Development: Challenges and Solution Proposal. In: Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES).

Braun, V., Clark, V. (2016). Using thematic analysis in psychology. Qualitative research in psychology Journal, Vol. 3 (2), pp. 77–101.

Brill, O., Knauss, E. (2011). Structured and unobtrusive observation of anonymous users and their context for requirements elicitation. In: Proceedings of the IEEE 19th International Requirements Engineering Conference (RE).

Caivano, D. *et al.* (2018). Artifact-based vs. human-perceived understandability and modifiability of refactored business processes: An experiment. Journal of Systems and Software, Vol. (144), pp. 143-164.

Capgemini (2015). World Quality Report 2015-16. Technical Report. http://www.capgemini.com/resources/world-quality-report-2015-16/. Accessed 15 November 2019.

Cronholm, S., Göbel, H. (2015). Empirical Grounding of Design Science Research Methodology. In: Proceedings of the New Horizons in Design Science: Broadening the Research Agenda (DESRIST).

Cruzes, D. S., Dyba, T. (2011). Recommended Steps for Thematic Synthesis in Software Engineering. In: Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM).

Cruzes, D. S. , Dybå, T., Runeson, P. , Höst, M. (2015). Case studies synthesis: a thematic, cross-case, and narrative synthesis worked example. Empirical Software Engineering, Vol. 20 (6), pp. 1634–1665.

Daniel, J. (2012). Sampling essential. Practical guidelines for making sampling choices. SAGE Publications.

DeLone, W.H., McLean, E.R. (2003). Information Systems Success Revisited. In: Proceedings of the 35th Hawaii International Conference on System Sciences (HICSS).

Franch, X., Palomares, C., Quer, C., Renault, S., De Lasser, F. (2010). A Metamodel for Software Requirement Patterns. In: Procedings of the International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ).

Franch, X. (2018). Why Are Ontologies and Languages for Software Quality Increasingly Important?. In: SERC Talks. http://sercuarc.org/event/serc-talks-why-are-ontologies-and-languages-for-software-quality-increasingly-important. Accessed 15 November 2019.

Franch, X. et al. (2018a). Data-Driven Elicitation, Assessment and Documentation of Quality Requirements in Agile Software Development. In: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE).

Franch, X. et al. (2018b). A Situational Approach for the Definition and Tailoring of a Data-Driven Software Evolution Method. In: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE).

Franch, X, López, L., Martínez-Fernández, S., Oriol, M., Rodríguez, P., Trendowicz, A. (2019). Quality-Aware Rapid Software Development Project: The Q-Rapids Project. In: Proceedings of the International Conference on Objects, Components, Models and Patterns (TOOLS).

Glinz, M. (2007). On Non-Functional Requirements. In: Proceedings of the IEEE 15th International Requirements Engineering Conference (RE).

Goodhue, D. L., Thompson, R. L. (1995). Task technology fit and individual performance, MIS Quarterly. Vol. 19 (2), pp. 213–236, 1995.

Groen, E. C. et al. (2017): The Hidden Software Product Quality Experts?: A Study on How App Users Report Quality Aspects in Online Reviews. In: Proceedings of the IEEE 25th International Requirements Engineering conference (RE).

Guzmán, L., Alkadhi, R., Seyff, N. (2016). A Needle in a Haystack: What Do Twitter Users Say about Software? In: Proceedings of the IEEE 24th International Requirements Engineering conference (RE).

Guzmán, L., Vollmer, A.M., Ciolkowski, M., Gillmann, M. (2017). Formative evaluation of a tool for managing software quality. In: Proceedings of the 11th International Symposium on Empirical Software Engineering and Measurement (ESEM).

Krasner, H. (2018). The Cost of Poor Quality Software in the US: A 2018 Report. Technical Report, CISQ Consortium for IT Software Quality.

Kurtanovic, Z., Maalej, W. (2017). Mining User Rationale from Software Reviews. In: Proceedings of the IEEE 25th International Requirements Engineering conference (RE).

Liu, X. *et al.* (2017). Deriving User Preferences of Mobile Apps from Their Management Activities. ACM Transactions on Information Systems, Vol. 35(4).

López, L., Martínez-Fernández, S., Gómez, C., Choraś, M., Kozik, R., Guzmán, L., Vollmer, A. M., Franch, X., Jedlitschka, A. (2018). Q-Rapids Tool Prototype: Supporting Decision-Makers in Managing Quality in Rapid Software Development. In: Proceedings of the International Conference on Advanced Information Systems Engineering (CAiSE).

Lu, M., Liang, P. (2017). Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE).

Maalej, W., Nayebi, M., Johann, T., Ruhe, G. (2016). Toward Data-Driven Requirements Engineering. IEEE Software, Vol. 33(1), pp. 48-54.

Martínez-Fernández, S., Jedlitschka, A., Guzman, L., Vollmer, A. M. (2018a). A Quality Model for Actionable Analytics in Rapid Software Development. In: Proceedings of the Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA).

Martínez-Fernández, S., Jovanovic, P., Franch, X., Jedlitschka, A. (2018b): Towards Automated Data Integration in Software Analytics. In: Proceedings of the 13th International Workshop on Real-Time Business Intelligence and Analytics (BIRTE).

Martínez-Fernández, S., Vollmer, A. M., Jedlitschka, A., Franch, X., López, L., Ram, P., Rodríguez, P., Aaramaa, S., Bagnato, A., Choras, M., Partanen, J. (2019): Continuously Assessing and Improving Software Quality With Software Analytics Tools: A Case Study. IEEE Access Vol. 7: 68219-68239.

McKinney, V., Yoon, K., Zahedi, F. M. (2002). The measurement of web-customer satisfaction: An expectation and disconfirmation approach. Information Systems Research, Vol. 13(3), pp. 296–315.

Mendes, E., Rodriguez, P., Freitas, V., Baker, S., Atoui, M. A. (2018). Towards improving decision making and estimating the value of decisions in value-based software engineering: the VALUE framework. Software Quality Journal. Vol. 26(2), pp. 607-656.

Miles, M., Huberman, M. (1994). Qualitative data analysis, 2nd edition. London: Sage Publications.

Nelson, R. R., Todd, P. A., Wixom, B. H. (2005). Antecedents of information and system quality: An empirical examination within the context of data warehousing. Journal of Management Information Systems, Vol. 21(4), pp. 199–235.

Oriol, M. et al. (2019a). Data-Driven Elicitation of Quality Requirements in Agile Companies. In: Proceedings of International Conference on the Quality of Information and Communications Technology (QUATIC).

Oriol, M. et al. (2019b). Supporting material of Data-Driven Elicitation of Quality Requirements: Tool Support and Experiences in Agile Companies. figshare. Online resource. https://doi.org/10.6084/m9.figshare.10308299.v2

Palomares, C., Quer, C., Franch, X. (2013). PABRE-Proj: Applying patterns in requirements elicitation. In: Proceedings of the IEEE 21st International Requirements Engineering conference (RE).

Peffers, K., Tuunanen, T., Rothenberger, M. A., Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems. Vol. 24(3), pp. 45-78.

Pohl, K., Rupp, C. (2015). Requirements engineering fundamentals A Study Guide for the Certified Professional for Requirements Engineering Exam. 2nd edition. Rocky Nook.

Renault, S., Méndez-Bonilla, Ó., Franch, O., Quer, C. (2009). PABRE: pattern-based requirements elicitation. In Proceedings of the 3rd International Conference on Research Challenges in Information Science (RCIS).

Rodríguez, P., Markkula, J., Oivo, M., Turula, K. (2012). Survey on agile and lean usage in Finnish software industry. In: Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM).

Rodríguez, P. *et al.* (2017). Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, Vol. 123, pp. 263-291.

Schön, E.M., Thomaschewski, J., Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. Computer Standards and Interfaces. Vol. 49, pp. 79-91.

Schwaber, K. (2004), Agile project management with Scrum. Microsoft Press.

Spinellis, D. (2006). Code quality: the open source perspective. Addison-Wesley.

Tricentis (2018). Software Fail Watch: 5th Edition. White Paper. http://www.tricentis.com/resources/software-fail-watch-5th-edition/. Accessed 15 November 2019.

Venkatesh, V., Bala, H. (2008). Technology acceptance model 3 and a research agenda on interventions. Decision Sciences. Vol. 39(2), pp. 273–315.

Wagner, S. (2015). Software Product Quality Control. 2nd edition. Springer Berlin Heidelberg.

Wilcoxon, F. (1945). Individual Comparisons by Ranking Methods. Biometrics Bulletin, Vol. 1(6), pp. 80-83.

Wohlin, C. *et al.* (2012). Experimentation in software engineering. Springer Science & Business Media.

Xu, P., Ramesh, B. (2008). Impact of knowledge support on the performance of software process tailoring. Journal of Management Information Systems. Vol. 25 (3), pp. 277–314.

Zowghi, D., Coulin, C. (2005): Requirements Elicitation: A Survey of Techniques, Approaches, and Tools. In: Engineering and Managing Software Requirements. Springer, Berlin, Heidelberg