Data-driven elicitation of quality requirements in agile companies

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

[1] Universitat Politècnica de Catalunya, Barcelona, Spain
{moriol, farre, franch}@essi.upc.edu
[2] University of Oulu, Oulu, Finland
{pertti.seppanen, woubshet.behutiye, pilar.rodriguez}@oulu.fi
[3] ITTI Sp. z o.o., Poznań, Poland
{rkozik, mchoras}@itti.com.pl
[4] University of Science and Technology, UTP, Bydgoszcz, Poland
[5] Fraunhofer IESE, Kaiserslautern, Germany
silverio.martinez@iese.fraunhofer.de
[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 to ensure the quality and success of a software system. Despite its importance, QRs have not reached the same degree of attention as its functional counterparts, especially in the context of trending software development methodologies like Agile Software Development (ASD). Moreover, crucial information that can be obtained from data sources of a project under development (e.g. JIRA, github,…) are not fully exploited, or even neglected, in QR elicitation activities. In this work, we present a data-driven approach to semi-automatically generate and document QRs in the context of ASD. We define an architecture focusing on the process and the artefacts involved. We validate and iterate on such architecture by conducting workshops in four companies of different size and profile. Finally, we present the implementation of such architecture, considering the feedback and outcomes of the conducted workshops.

Keywords: Quality Requirements, Non-Functional Requirements, Agile Software Development, Data-driven Requirements Engineering.

1 Introduction

Quality management is known to be one of the critical success factors for software projects [1]. There are many examples of software with poor quality (e.g. software with bugs, security issues, technical debt, low quality of service, poor code quality, etc.) that have caused millions of euros of losses. A report conducted by the software testing company Tricentis revealed that software failures caused more than $1.7 trillion in financial losses in 2017 [2].

To be successful, software development companies must understand and manage software quality to ensure that new releases lead to progressive improvement [3]. For such a reason, many approaches have emerged to try to improve the software quality in different phases of the software development lifecycle. In this regard, market studies show a steady increase in the proportion of software development companies' budget being spent on dealing with software quality [4].

It is argued that an optimal approach to ensure a good software quality, should consider and address such quality starting early from the requirements [5]. The artifact that requirements engineers use to state conditions on, and analyse compliance of, software quality are the Quality Requirements (QRs; also known as non-functional requirements) [6].

A QR is defined as "*a requirement that pertains to a quality concern that is not covered by functional requirements*" [6].

QRs play an essential role in the success of software systems, and neglecting or failing to satisfy QRs can lead to critical or even catastrophic consequences [5][7].

Despite their importance, QRs have traditionally not received the same degree of attention than their functional requirements counterpart [8]. 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 [9].

To address this problem we presented in a previous work an explorative position paper at [10] where we envisaged a conceptual framework, named Q-Rapids, to generate and document QRs using a data-driven approach in the context of ASD. The goal of this paper is to continue on such work and present an operational implementation of the approach for generating and documenting QRs in ASD. The contributions of this paper are as follows:

1. A software architecture for a tool implementation, explained with a running example impacting the companies.
2. Workshops to refine and validate the architecture for a tool implementation with prospective end-users.
3. Discussions on the results and current tool support.

The research has been conducted in the context of the Q-Rapids H2020 project (www.q-rapids.eu) which has given us the opportunity to elicit real scenarios and evaluate the results on different company-provided scenarios.

The rest of the paper is organized as follows: Section 2 presents the Related Work. Section 3 provides an overview of the overall Q-Rapids Approach. Section 4 describes our proposal, detailing the architecture and artifacts for the QR generation and documentation process. Section 5 evaluates such proposal by means of workshops conducted in four companies. Section 6 reports and discusses the findings of such

evaluation, which are then used for the implementation of the tools, described in Section 7. Finally, Section 8 provides the conclusions and the future work.

2 Background and related work

In ASD, the development process is mostly driven by functional requirements. For example, in Scrum [11], requirements are specified as user stories in the product backlog and prioritized based on a customer perspective. This way of eliciting and managing requirements tends to favour functional requirements over QRs [12][13]. As a result, QRs are not properly documented and only managed in a tacit way [14]. Moreover, despite the numerous sources of information related to product quality that ASD provides (e.g. continuous integration systems and user feedback), there is a lack of methods to support continuous elicitation and management of QRs, throughout the whole software development lifecycle [13].

On the other hand, traditional approaches for eliciting and managing QRs are usually inadequate in the highly-dynamic scenarios in which ASD is more suitable. Traditional techniques to elicit QRs include structured and unstructured interviews, quality models, checklists, prioritization questionnaires, and the like. None of them exploits runtime data. In this context, data-driven requirements engineering [15] is advocated as the proper way to go for eliciting QRs. Some recent proposals in this direction aim at exploiting end-user explicit feedback data [16][17][18].

Explicit feedback requires user commitment and can be incomplete and/or biased. Implicit feedback can be considered as an alternative/complementary data source for requirements elicitation [15]. As an example, [19] exploits implicit feedback but does not aim at generating QRs but at discovering user preferences and usage patterns. The SUPERSEDE data-driven approach [20] combines both explicit and implicit end-user feedback with other sources like run-time monitors to detect and address different kinds of issues: bugs, new features, QoS violations.

However, none of the aforementioned approaches exploit 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 could hardly be elicited.

3 The Q-Rapids Approach to Quality Requirements elicitation and documentation

Q-Rapids is a quality-aware ASD framework in which QRs are elicited using a data-driven approach. Data from multiple data sources are gathered and evaluated against a Quality Model to generate QRs if an issue is identified.

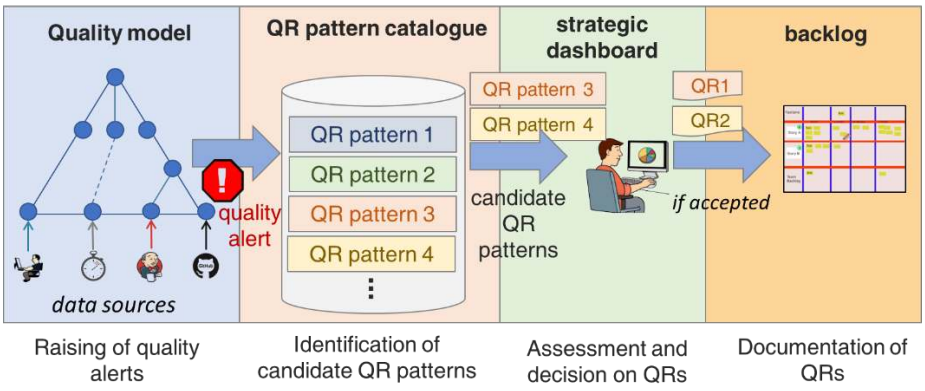

Fig. 1. Main logical components of the Q-Rapids conceptual architecture [10]

Fig. 1 depicts the Conceptual Architecture of Q-Rapids with its different phases to generate and document QRs:

- *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. These elements of the *quality model* have customizable thresholds, that, if violated, automatically raise a *quality alert*
- *Selection of candidate QR patterns:* When a *quality alert* is raised, Q-Rapids identifies *candidate QR patterns* that, 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 QR* is the *QR pattern catalogue* [21]. The *QR pattern catalogue* consists of a set of *QR patterns* that 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 catalogue*. This binding is fundamental in order to match the appropriate *candidate QR patterns* with the raised *quality alert*.
- *Assessment and decision on QRs:* The *candidate QR patterns* are presented to the decision makers —Product Owners, Project Managers or other members of the development team— through a *strategic dashboard*. The decision makers assess the *candidate QR patterns* and instantiates them to particular *QRs* by setting the values of the formal parameters of the *QR pattern*. To support the assessment of the instantiated QRs, the strategic dashboard includes simulation techniques. Such simulations predict the impact that the QRs would have on the values of the different elements of the *quality model* if such *QRs* were implemented.
- *Documentation of QRs:* In case the QR is accepted by the decision maker, such QR is forwarded to the backlog. The strategic dashboard provides the user with a link through which the accepted QR is automatically moved to the organization's requirements repository. The strategic dashboard itself does not

depend on any fixed repositories or tools but utilizes the link mechanism to transfer the data content of the accepted QR to the requirements repository. Building the linkage between the strategic dashboard and the used requirements repository is a task done in the Q-Rapids setting-up actions.

In [10], we introduced the ideas of such QR generation process. We have further elaborated on those ideas and provided a first design, which was evaluated in different companies as part of a co-creation activity for its implementation.

4 QR generation and documentation architecture

4.1 QR generation and documentation process

To design the QR generation and documentation process, we have formalized the different steps and elements required by means of a Business Process Model and Notation (BPMN) process model. The QR generation process are the activities conducted from the raise of an alert until the QR is generated, whereas the QR documentation is the activity to include such QR into the backlog. Such BPMN process model is shown in Fig. 2 and defines the main tasks and artifacts involved. 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 Catalogue* the *«artifact» QR patterns* able to resolve the quality alert. The decision makers select and instantiate the QR pattern, generating the *«artifact» QR*, which is finally stored to the *«repository» Backlog*.

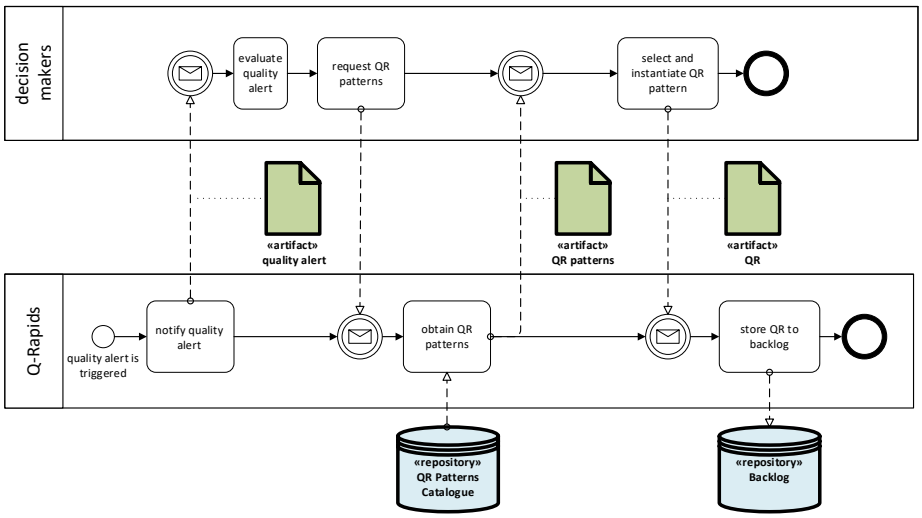

Fig. 2. BPMN process model of the QR generation and documentation process

In the following subsections we describe in detail how we have designed those artifacts. Namely: *quality alert, QR patterns* and *QR*. We also describe and discuss the repositories that we will use for the *QR patterns Catalogue* and *Backlog*.

Finally, the presented artifacts are evaluated in multiple companies (see Section 5 and Section 6). Following the evaluation of the artifacts, we implemented a first prototype of the tools that automate the defined tasks in Q-Rapids (see Section 7).

4.2 Quality alerts

In a previous work, we had defined the quality model based on expert knowledge from the companies of the Q-Rapids consortium [22]. The nodes of the quality model are of different type 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 decomposed into quality metrics (e.g. duplicated lines of code). Starting from this quality model, we have defined customizable thresholds on each of the different nodes in order to raise a quality alert if such threshold is violated. We defined the quality alert artifact in JSON with the following metadata:

- **Element id:** a unique identifier for the alert.
- **Name**: name describing the alert.
- **Type**: identifies if the alert is at the quality metric, quality factor or project indicator level.
- **Category**: it is 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 in 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 those quality factors is *Code Quality* that 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 (such threshold was defined by the company based on historical data from similar projects and their experience on it). Because of such situation, a quality alert is raised for the *Code Quality* quality factor (see Fig. 3).

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. 3. Example of quality alert in Q-Rapids

4.3 QR Patterns Catalogue

To design and instantiate the QR Patterns Catalogue, we used the PABRE framework [21] and extended it to support the QR generation process. PABRE is a tooled framework that facilitates the reuse in requirement 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 in a schema following a tree-structured form.

In this regard, we defined a schema with the tree structure following the same structure defined in the quality model. In such a manner, there is a clear mapping between the categories of the quality alerts generated and the QRs that can solve them. A generic catalogue of QR patterns is available at the supporting material of this paper [23].

It is worth to remark that such approach enables to have multiple QR patterns for a given quality alert, or that a quality alert at the quality factor level can be solved by the QR patterns bounded to the quality metrics that decompose such quality factor.

For instance, from the previous example, PABRE can retrieve the QRs able to resolve the alert of the *Code Quality* factor. In this case, the QR patterns are: *ComplexFilesReq* (which aims to reduce the ratio of files with a high cyclomatic complexity), *CommentedFilesReq* (which aims to reduce the ratio of files with a high number of commented lines of code) and *DuplicationsReq* (which aims to reduce the ratio 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 decompose *Code Quality*.

4.4 QR patterns

The internal structure of a QR pattern is also based on PABRE [21] and it has been tailored to the specific needs of Q-Rapids. A requirement pattern includes several metadata as described and specified at [24]. But from the point of view of the decision makers, just the following information is visible:

- **Goal**: it 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 ig. 4.

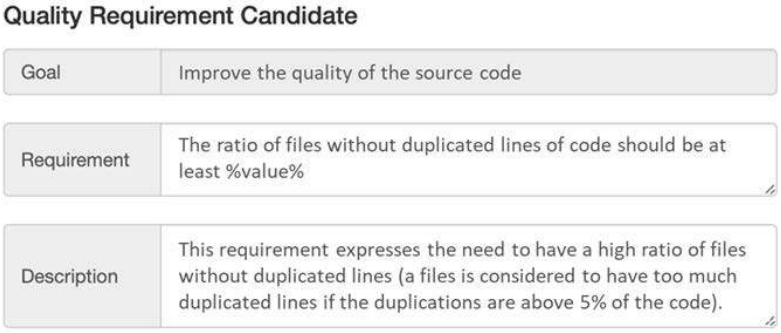

Fig. 4. QR pattern – DuplicationsReq

4.5 Quality Requirements

The QR is an instantiation[8] 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. To assist the decision maker on instantiating the parameter(s) with appropriate values to solve the quality alert, Q-Rapids will provide simulation techniques that will show the impact that the instantiated QR will have on the elements of the quality model. Such simulation techniques will be based on bayesian networks as proposed in the VALUE framework [25].

Following the example previously described, the decision maker could instantiate the QR patterns provided by Q-Rapids (i.e. *ComplexFilesReq, CommentedFilesReq, DuplicationsReq*) with different values on its parameters and evaluate the impact that those instantiations have on the quality model. During this process of simulation, the decision maker could play with different alternatives and combinations in order to decide the QRs to add. For instance, after playing with different combinations, the decision maker might choose to instantiate the *DuplicationsReq*, setting its value to 85% (leading to the QR "The ratio of files without duplicated lines of code should be at least 85%"), and *ComplexFilesReq*, setting its value to 70% (leading to the QR "The ratio 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, and hence resolving the quality alert.

4.6 Backlog

In a previous study [26], we identified that companies adopt different practices and tools for documenting QRs. Hence, the Q-Rapids approach for integrating generated QRs in the projects' requirements backlog needs to consider varying 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 such heterogeneity, we propose a generic service interface to link the generated QR in Q-Rapids to the projects' backlog. Such service interface can have multiple implementations to meet the needs of each requirements management tool (e.g. JIRA, openProject) and can be tailored to the specific companies' needs. Hence the generated QRs can be added to the project' requirements backlog following specific practices adopted by each company.

5 Evaluation design, execution and analysis

In order to evaluate the artifacts and the process defined, we designed an evaluation that involved the participation of four companies following a structured workshop format. The evaluation was conducted in the four companies of the Q-Rapids consortium, which have different profiles (one large corporation, two large/medium companies and one SME) and produce different types of systems (e.g., from modelling tools to telecommunication software).

The goal of the workshop was twofold. On the one hand, to validate the Q-Rapids QR generation (i.e. the process and its artifacts) and, on the other hand, to conduct an exploratory study of the Quality Requirements documentation process (i.e. the step that documents the QR into the backlogs).

5.1 Workshop design

The workshop was structured in two parts, following the two goals defined above.

The first part was the validation of the QR generation process. This validation followed a user-oriented perspective involving the representatives of the aforementioned companies. In this regard, the validation focused on the generated artifacts that need to be processed and analysed by the Decision Makers in their tasks of defining and deciding about QRs. Namely: the quality alerts, the QR patterns catalogue and the QR patterns along with the instantiated QRs.

The first part started with a short presentation by the researcher describing the workflow and the structure and contents of each of those artifacts, presenting as well an illustrative example akin to the one presented in Section 4, by means of mock-ups. After the description of each artifact, the researcher asked the following questions to retrieve the feedback from the participants. Questions were asked orally to motivate discussion within the company representative participants. The particular questions for each artifact that were investigated are:

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

The participants were also invited to provide at any time any feedback or comment they wanted to raise.

The second part of the workshop focused on exploring the QRs documentation practice of the companies and identifying important aspects for documenting the generated QRs in the projects' requirements backlogs. Researchers used findings from earlier study with the companies regarding requirements documentation [26] to initiate the discussion. We used requirements documentation templates based on requirements management tool applied in the projects (e.g. JIRA), to guide the discussion and asked the participants to identify aspects they find important while documenting QRs, with a purpose to achieve lightweight and informative QRs documentation.

5.2 Workshop execution

The workshops were conducted in 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 pilot test, and they acted as representatives of their respective development teams. Each workshop had between 1 and 3 members representing the company. Due to the limited amount of participants, analysis was limited to a qualitative assessment and no quantitative study was conducted. Three of the four workshops were conducted in the premises of the company, whereas one workshop was conducted on-line. The workshops were conducted between June 12th 2018 and September 7th 2018. The duration of the workshops range from 124 minutes to 202 minutes. Details are summarized in Table 1.

Table 1. Summary of workshops execution.

Company	Bittium	iTTi	NOKIA	Softeam
Country	Finland	Poland	Finland	France
Number of participants	2	1	3	2
Date of the workshop	June 12th 2018	September 7th 2018	June 13th 2018	June 19th 2018
In premises / On-line	in premises	on-line	in premises	in premises
Duration of workshop	196 min	202 min	190 min	124 min

5.3 Data Analysis

The research data were gathered in the workshops by recording the discussions. The recordings were transcribed in a professional transcriptions company in Finland to MsWord documents.

The research data were analyzed by using a combination of thematic synthesis and narrative synthesis [27][28]. The combination of two synthesis practices was opted because, at a detailed level, the practices of the case companies were very company specific.

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

The documentation-specific sections of the MsWord documents were gathered to Excel tables, one for each case company, the sections were coded and the codes were gathered to higher-order themes according to the thematic synthesis principles [27]. 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 three case companies were summarized and the consistency of the summarized themes were checked by using the principles of the narrative synthesis [28].

6 Results and findings

6.1 Results on QR generation

Quality alerts. All case companies answered that the amount of information provided in the alerts was adequate and not overwhelming. As a respondent summarized, “to me it looks like the most important information”. Most companies’ representatives provided also valuable feedback and ideas based on their needs in order to improve such 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”, or, “the 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), to be able to visualize the values of the upper levels 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 its learnability.

QR patterns and QRs. All companies answered that the amount of information provided in the QRs was adequate and not overwhelming.

Regarding information missing, some companies’ representatives requested to make more explicit the terminology of what is commonly understood as QRs, as a participant requested: “something like stability or security or maintainability”. For that, such participant suggested that “non-functional requirement-related keywords could be somehow highlighted in the text. So, that would give clearer understanding that this relates for example to performance issues”.

QR Patterns Catalogue. All companies considered the QR patterns catalogue adequate, complete and not overwhelming. As valuable feedback, they pointed out the need to easily “have the ability to add a new quality requirement pattern” as they evolve the quality model. One company, went one step further in this direction, and suggested to be able to extend the QR patterns catalogue on demand. That is, if there is no QR pattern able to solve a particular quality alert there should be the possibility to extend the QR patterns catalogue dynamically.

6.2 Results on QR documentation

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 derive 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 requirement we already have, we should be able to trace back what requirement we are violating”

The companies had established, well-implemented processes and practices for ASD and quality assurance, and several tools gathering and reporting quality-related information were in use. That sets requirements to the documentation of QRs - the information content of the QRs must be exact and fitted to the processes and practices of the company. The 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 requirement repository tools, one was in a middle of change to a new tool, and one company was improving the processes and tools in a continuous manner resulting in a situation where several requirement repository tools were in use in different parts of the organization. Such situation generates challenges to the automatic link for QRs between the Q-Rapids strategic dashboard and the requirement repositories, meaning that there would not be any one-fit-all solution: “But then the question is that 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?”.

7 Tool-support implementation

Based on the results of the workshop, we were able to refine the evaluated artifacts and start the implementation of the tools that automate the QR generation and documentation process described in Section 4. The modules implemented as a result of those workshops were:

qr-alert module: This module automates the process of evaluating the elements of the quality model and raise an alert if a threshold is violated. Decision makers can specify the threshold for each of these nodes and receive a notification once a violation is triggered. The service can be triggered in time-based manner and configured in terms of running intervals during the day. Moreover, the module allows the user to specify more complex (than simple threshold-based conditions) activation rules that will trigger a quality alert. Users can use any timespan (e.g. range) or a specific date for executing the rule.

qr-generation module: 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 if a quality alert can be resolved by a QR pattern. If so, it provides the list of QR patterns that can solve the quality alert throughout the tree-based structure.

qrapids-backlog-*: This module is used to store the generated QRs to the backlog. The module defines a common RESTful interface that can have multiple internal implementations, enabling the capability for Q-Rapids to connect to multiple backlogs (e.g. OpenProject, Jira,...).

On top of those components, the workflow was integrated in the Q-Rapids strategic dashboard, which offers to the decision makers an easy-to-use user interface to generate and document QRs, providing also the traceability functionalities requested by the companies through its navigable interface.

Finally, PABRE was also extended to provide the functionality to easily extend the catalogue through import/export functions as well as RESTful methods to dynamically add, update and delete existing QR patterns in the catalogue.

The implementation and documentation of such components is available in github[1].

8 Conclusions and future work

In this paper we have presented a data-driven approach for generating and documenting QRs in ASD. Our proposed solution is part of the Q-Rapids framework, which aims at improving QR management in the agile ecosystem. In a nutshell, Q-Rapids collects data from multiple sources of a project (e.g. Jira, SonarQube, github, etc.) and feeds a quality model that computes the quality of the software. Quality alerts are triggered when defined thresholds in the nodes of the quality model are surpassed, which, in turns triggers the QR generation and documentation process. We have formalized such process by means of a BPMN process model where the different tasks, artifacts and repositories involved were defined. To refine and validate our proposal, we have conducted a workshop in four companies of different size and profiles. The results of such workshop have enabled us to iterate on the proposal and implement the tools that automate the activities of the proposed QR generation and documentation process.

As Future work, on the one hand, we plan to improve the current implementation in several directions, such as the simulation techniques to compute the impact of the QRs over the quality model, adding cost functions to estimate the effort to implement such QRs, and improving the overall user experience to facilitate its adoption. On the other hand, we plan to deploy and evaluate the implemented tools in different companies, considering also companies beyond the consortium. For those evaluations, we plan to conduct a quantitative analysis involving a higher number of participants in the study, as well as more complete interviews to obtain additional insights.

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

[1] N. Abbas, A. M. Gravell, G. B. Wills: The Impact of Organization, Project and Governance Variables on Software Quality and Project Success. In: Procs of Agile Conference, 2010.

[2] Tricentis: Software Fail Watch: 5th Edition. White Paper, 2018. http://www.tricentis.com/resources/software-fail-watch-5th-edition/, last accessed 2019/05/19.

[3] P. Behnamghader, R. Alfayez, K. Srisopha, B. Boehm: Towards Better Understanding of Software Quality Evolution through Commit-Impact Analysis. In: Procs of *QRS*, 2017.

[4] Capgemini: World Quality Report 2015-16. Tech Report, 2015. www.capgemini.com/resources/world-quality-report-2015-16/, last accessed 2019/05/19.

[5] X. Franch: Why Are Ontologies and Languages for Software Quality Increasingly Important?. In: SERC Talks, 2018. sercuarc.org/event/serc-talks-why-are-ontologies-and-languages-for-software-quality-increasingly-important, last accessed 2019/05/19.

[1] https://github.com/q-rapids (modules qr-alert, qr-generation and qrapids-backlog-*)

[6] K. Pohl, C. Rupp: Requirements engineering fundamentals A Study Guide for the Certified Professional for Requirements Engineering Exam. 2nd ed. Rocky Nook, 2015.
[7] D. Spinellis: Code quality : the open source perspective. Addison-Wesley, 2006.
[8] S. Wagner: Software Product Quality Control. 2nd ed. Springer Berlin Heidelberg, 2015
[9] P. Rodríguez, J. Markkula, M. Oivo, K. Turula: Survey on agile and lean usage in finnish software industry. In: Procs of ESEM, 2012.
[10] X. Franch *et al.*: Data-Driven Elicitation, Assessment and Documentation of Quality Requirements in Agile Software Development. In: Procs of CAiSE, 2018.
[11] K. Schwaber, Agile project management with Scrum. Microsoft Press, 2004.
[12] E.-M. Schön, J. Thomaschewski, M. J. Escalona: Agile Requirements Engineering: A systematic literature review. Computer Standards and Interfaces, vol. 49, 2017.
[13] P. Rodríguez *et al.*: Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, vol. 123, 2017.
[14] S. Bartsch, Steffen: Practitioners' Perspectives on Security in Agile Development. In: Procs of ARES, 2011
[15] W. Maalej, M. Nayebi, T. Johann, and G. Ruhe: Toward Data-Driven Requirements Engineering. IEEE Software, vol. 33(1), 2016.
[16] E. C. Groen *et al.*: A Study on How App Users Report Quality Aspects in Online Reviews. In: Procs of RE, 2017.
[17] Z. Kurtanovic, W. Maalej: Mining User Rationale from Software Reviews. In: Procs of RE, 2017
[18] M. Lu, P. Liang: Automatic Classification of Non-Functional Requirements from Augmented App User Reviews. In: Procs of EASE, 2017.
[19] X. Liu *et al.*: Deriving User Preferences of Mobile Apps from Their Management Activities. ACM Transactions on Information Systems, vol. 35(4), 2017.
[20] X. Franch *et al.*: A Situational Approach for the Definition and Tailoring of a Data-Driven Software Evolution Method. In: Procs of CAiSE, 2018.
[21] C. Palomares, C. Quer, X. Franch: PABRE-Proj: Applying patterns in requirements elicitation. In: Procs of RE, 2013
[22] S. Martinez-Fernandez, A. Jedlitschka, L. Guzman, A. M. Vollmer: A Quality Model for Actionable Analytics in Rapid Software Development. In: Procs of SEAA, 2018.
[23] M. Oriol *et al.*: Appendix of: Data-driven elicitation of quality requirements in agile companies, 2019. http://www.essi.upc.edu/~moriol/qr_elicitation/.
[24] PABRE API Documentation. http://gessi3.cs.upc.edu/pabre-ws/doc/#/.
[25] E. Mendes, P. Rodriguez, V. Freitas, S. Baker, M. A. Atoui: Towards improving decision making and estimating the value of decisions in value-based software engineering: the VALUE framework. Software Quality Journal, vol. 26(2), 2018.
[26] W. Behutiye *et al.*: Non-functional Requirements Documentation in Agile Software Development: Challenges and Solution Proposal. In: Procs of PROFES, 2017.
[27] D. S. Cruzes, T. Dyba: Recommended Steps for Thematic Synthesis in Software Engineering. In: Procs of ESEM, 2011.
[28] D. S. Cruzes, T. Dybå, P. Runeson, M. Höst: Case studies synthesis: a thematic, cross-case, and narrative synthesis worked example. Empirical Software Engineering, vol. 20 (6), 2015.