This is an Author's Accepted Manuscript consisting of a post-peer-review, pre-copyedit version of a paper to be published by Springer in "30th International Conference on Advanced Information Systems Engineering" (CAiSE'2018). The final authenticated version will be available online.

Data-Driven Elicitation, Assessment and Documentation of Quality Requirements in Agile Software Development[1]

Xavier Franch[1], Cristina Gómez[1], Andreas Jedlitschka[2], Lidia López[1], Silverio Martínez-Fernández[2], Marc Oriol[1], Jari Partanen[3]

[1]Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{franch, cristina, llopez, moriol}@essi.upc.edu
[2]Fraunhofer IESE, Kaiserslautern, Germany
{silverio.martinez, andreas.jedlitschka}@iese.fraunhofer.de
[3]Bittium Wireless Ltd., Oulu, Finland
jari.partanen@bittium.com

Abstract. Quality Requirements (QRs) are difficult to manage in agile software development. Given the pressure to deploy fast, quality concerns are often sacrificed for the sake of richer functionality. Besides, artefacts as user stories are not particularly well-suited for representing QRs. In this exploratory paper, we envisage a data-driven method, called Q-Rapids, to QR elicitation, assessment and documentation in agile software development. Q-Rapids proposes: 1) The collection and analysis of design and runtime data in order to raise quality alerts; 2) The suggestion of candidate QRs to address these alerts; 3) A strategic analysis of the impact of such requirements by visualizing their effect on a set of indicators rendered in a dashboard; 4) The documentation of the requirements (if finally accepted) in the backlog. The approach is illustrated with scenarios evaluated through a questionnaire by experts from a telecom company.

Keywords: quality requirement; NFR; agile software development.

1 Introduction

Software quality is an essential competitive factor for the success of IT companies today [1]. Recent technological breakthroughs as cloud technologies, IoT and 5G, pose new quality challenges in software development. These challenges include quality aspects such as availability, reliability, security, performance, and scalability, which significantly influence the success of current and future software systems. Many well-known cases, which are worth millions of euros in losses, were due to bad quality or defective software (see an example in [2]).

Optimal management of software quality demands the appropriate integration of quality requirements into the software life cycle. Quality Requirements (QRs; also known as non-functional requirements) are the artefact that software engineers use to

[1] 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.

state conditions on, and analyse compliance of, software quality [3]. QRs are as generic as “The software service shall provide optimal response time” (in fact, more of a goal than a requirement) or as detailed as “Once the user completes the request, logging into the system should not take more than 3 seconds 95% of the time”. This level of detail will normally depend on the development stage. However, despite the competitive advantage of ensuring and maintaining high quality levels, software development methodologies still provide limited support to QR management [4].

One example is agile software development (ASD). Quality is essential in rapid software development processes happening in ASD: faster and more frequent release cycles should not compromise software quality. Still, a lack of mechanisms to support continuous quality assurance exists throughout the whole development process in the context of rapid releases [5]. Empirical studies conducted in the last few years in ASD have identified as a major challenge the consideration of QRs [6] whose deficient management was recently qualified as a reason for massive lapses and rework [7].

Given this situation, the present work addresses the following research goal:

Research goal: To formulate a tooled method supporting the effective management of QRs in agile software development.

We decompose this goal into four research questions, see Table 1. As a first step for managing QRs, we need to detect that some quality concern is not well covered in the current release of the system (RQ1). We adopt a data-driven approach in which we gather and analyse data from different sources to detect the need to improve any quality concern currently compromised. This identified need requires to be expressed in the form of one or more QRs (RQ2). We follow here a reuse-based approach in which a catalogue of QR patterns compiles the usual forms in which a QR can be specified. The implications of such QRs over the software system need to be assessed (RQ3) so that decision-makers may decide to accept or discard the QR. With this goal, we use a strategic dashboard that visualizes the effect of accepting the QR over several strategic indicators like product quality, customer satisfaction and team productivity. In case that the QR is finally accepted to be part of the system specification, it needs to be added to the backlog (RQ4). The way in which it is added may change from an epic, a user story or an acceptance condition.

Table 1. Research questions

Id	Question
RQ1	When is it necessary to improve some quality concern of a software system?
RQ2	How can this need be expressed as a quality requirement?
RQ3	How can the consequences of undertaking such quality requirement be assessed?
RQ4	In case that a quality requirement is accepted, how can it be added to the backlog?

As research method, we adopt a design science approach following the engineering cycle as described by Wieringa [8]. 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 in the context of a number of company-provided pilot cases.

2 State of the Art

Current approaches to manage QRs are usually inadequate in complex scenarios with highly-dynamic environments composed of fast-growing software systems. One the most unresolved challenges concerns their elicitation [9]. Techniques to elicit QRs include structured and unstructured interviews, quality models, checklists, and prioritization questionnaires, among others. These techniques do not exploit runtime data. Recently, data-driven requirements engineering has been advocated as the way to go [10]. In this area, modern approaches like crowdsourcing [11] or mining app stores [12] rely on feedback given by the user. But explicit feedback can be incomplete, biased or ambiguous. This contributes to the conceptual gap between user needs and the interpretation done by the development team of such needs. To overcome such communication problems, some approaches like Shekhovtsov et al. [13] propose a tool that aims at improving the communication in the software process development. In particular, their approach aims at identifying similar information communicated in the past, analyse its properties and, by using machine learning techniques, suggest communication-related decisions that would mitigate communication problems.

A different approach that is not affected by these limitations is implicit feedback (usage data), which has been identified as a promising additional input for requirements elicitation [9]. Existing approaches to implicit feedback [14] neither aim at generating QRs, nor consider the possibility to correlate usage data with data gathered from software repositories or project management tools. Our positioning is that these correlations may be used to uncover relevant observations in the management of QRs.

Another problem with QRs is their refinement in terms of satisfaction criteria. Differently than functional requirements that have clear cut satisfaction criteria, QRs are initially elicited as “soft goals” [15] and need to be elaborated into measurable conditions. Evidence exists that the knowledge needed to determine these values is closer to the development team than to the customer, even to the extreme in which the end-user is not really involved in such process [16]. Some approaches (e.g. QUPER [17]) present techniques for such elicitation, but they are all based in stakeholder explicit involvement before the product is released. These techniques do not try to inject data once the project is in use in order to understand the feasibility and appropriateness of the stated thresholds. We envisage the wide use of different data sources and then complete a fully empirical-based, data-driven approach to QR elicitation.

Another aspect that needs to be emphasised is the use of QRs in decision making. Usually, quality aspects are surveyed by tools used by software developers. Current project management systems deal with scheduling, resource and budget management, etc., but they do not explicitly include support for QR management [18]. Nevertheless, some tools exist to support some activities on quality requirements. For instance, tools to support software architects and developers on decision making that may include quality aspects (e.g. WISED); tools intended to monitor the productivity of developers when implementing such QRs (e.g. JIRA); and tools to evaluate the technical debt based on software quality assessment (e.g. SQALE). Although they may support QRs to a certain extent, their management is not well integrated with the entire life cycle of the software process. Furthermore, they do not provide full support to the assessment of key prioritization criteria for QRs: costs, benefits and risks [19]. For instance, it is reported that they fail on considering risk factors [20]. We envisage the need to provide full support to strategic decision making based on the analysis of the impact of QRs in such key criteria, reconciling: 1) QR-specific universes of discourse with definition of the concepts that matter in their management and assessment [17], 2) reasoning frameworks as the NFR framework [15], 3) a highly informative dashboard that provides full transparency to the decision making process [21].

Last, existing requirements engineering practices fail short regarding QR documentation in ASD. Instead, its reliance on the continuous interaction with customers is thought to minimize the need for specifying QRs [22]. Another reason is that agile developers face challenges while using user stories for documenting NFRs such as security [23]. A recent industrial study reported QR documentation practices ranging from scarce documentation to documentation using agile assets (user stories, DoD, ...) and out-of-the-loop artefacts as wiki pages [24]. Other recent works propose quality criteria [25] and traceability [26] to improve QR documentation in ASD.

In summary, we advocate that the current state of the art calls for improved methods for the elicitation, assessment and documentation of QRs especially in ASD.

3 The Q-Rapids Method

Q-Rapids is a data-driven, quality-aware ASD tooled method in which QRs are identified from available data and evaluated with respect to some selected strategic indicators [27]. Q-Rapids aims at increasing software quality through (see Fig. 2):

- Gathering and analyzing data from project management tools, software repo-sitories, quality of service and system usage. Data analysis will permit to systematically and continuously assess quality and eventually suggest QRs.
- Providing decision makers with a highly informative dashboard to help them making data-driven, requirements-related strategic decisions. The dashboard will aggregate the collected data into strategic indicators related to factors as time to market, team productivity, customer satisfaction, and overall product quality.
- Extending the ASD process considering the comprehensive integration of QRs and functional requirements in the product backlog.

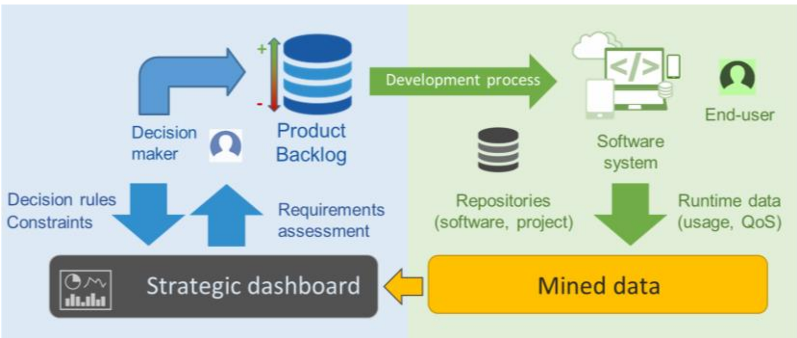

Fig. 2 The Q-Rapids method to QR management in ASD.

4 Conceptual Architecture of the Q-Rapids Method

The Q-Rapids method relies on several logical components that together conform a conceptual solution to the data-driven QR management problem. In this section, we present the main components of such solution, put together as shown in Fig. 3.

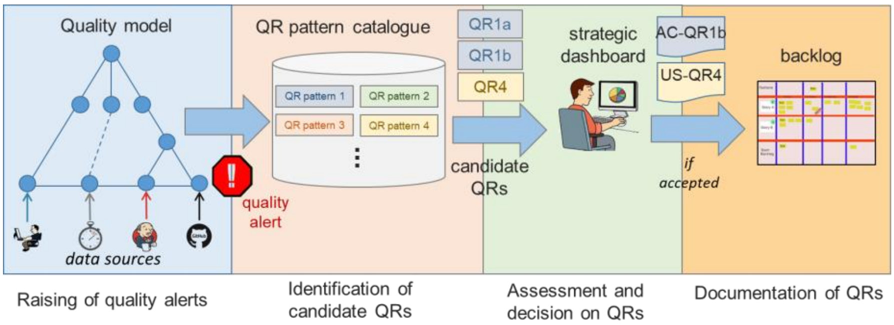

Fig. 3 Main logical components of the Q-Rapids conceptual architecture.

These components are rigorously described using a domain reference model expressed as a UML class diagram (see Fig. 4; the diagram also includes concepts presented later on) that is an extension of the decision-making ontology presented in [28]. The core concept of the reference model is the QR concept. We adopt an ontological interpretation of QR based on qualities in foundational ontologies [29] as done in [30]. A quality is defined as a basic perceivable or measurable characteristic that inheres in and existentially depends on its subject [29]. So, a QR is a need expressed by the stakeholders about a quality. Class and attribute names appear in italics when referenced in this section. A definition of all the concepts appearing in the reference model may be found in https://tinyurl.com/QRrefmod.

4.1 Strategic Dashboard

In the Q-Rapids method, the *Strategic Dashboard* is a logical component that visualizes information about a set of selected *Strategic Indicators* for a project developed under an ASD method. A strategic indicator is defined as an aspect that the company considers relevant for the decision-making process [28]. Examples of strategic indicators are: product quality, customer satisfaction, time to market and team productivity. They are measured through a *KPI*.

The strategic dashboard will also offer some techniques for analysing the indicators, e.g. trend and what-if analysis, failure prediction and suggestion of mitigation actions. These aspects are not covered in this paper.

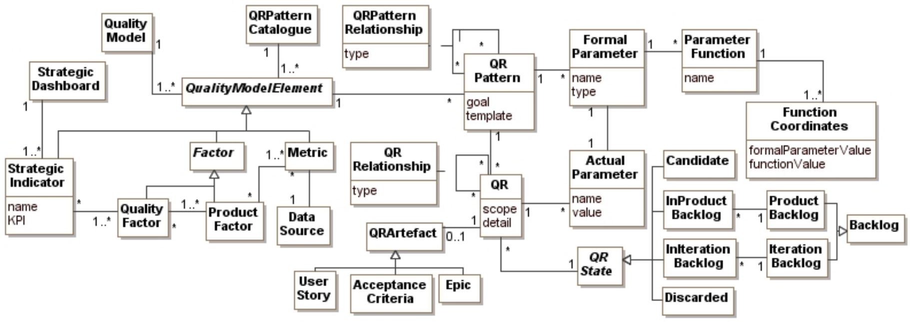

Fig. 4 Domain reference model for the Q-Rapids conceptual architecture (simplified).

4.2 Quality Model

In Q-Rapids, the main purpose of a *Quality Model* is to link the data gathered from some *Data Sources* to the strategic indicators rendered in the dashboard.

We have adopted the Quamoco approach [31] to define Q-Rapids quality models. In Quamoco, *Metrics* gather data from data sources using some software data collectors and are elaborated into *Product Factors* and ultimately aggregated into *Quality Factors*. We distinguish four types of metrics related to categories of data sources:

- Development metrics. Gathering data from software development repositories. E.g., test coverage gathers data from SonarQube using the SQALE plugin.
- Project management metrics. Gathering data from project management tools. For instance, personnel availability from Jira and MS Project.
- Feedback metrics. Gathering data from system users, either directly through given feedback, or indirectly through usage monitoring. For instance, the number of tweets referring to a particular (system-related) hashtag.
- Quality of service metrics. Gathering data from system behaviour through monitoring components. For instance, response time of a given system functionality.

4.3 Requirement Patterns

We envisage the use in Q-Rapids of a catalogue of *QR Patterns* that act as templates for deriving the concrete QRs. This *QR Pattern Catalogue* will be based on the PABRE approach [32]. Its structure complies the following principles:

- Its natural language *template* includes *Formal Parameters* that may have one or more *Parameter Function* defined as a set of *Function Coordinates*.
- QR patterns may present relationships (*QR Pattern Relationship*) of different *types* (e.g., conflict, dependency, synergy, …).
- QR patterns are bound to *Quality Model Elements*. This binding is fundamental in order to select the appropriate QR patterns once a quality alert is issued.

A QR pattern is instantiated into a *QR* by binding *Actual Parameters* to formal parameters. QRs preserve the relationships (*QR Relationship*) established at the level of the patterns they are an instantiation of.

5 The Q-Rapids Process

We present below the four phases of the Q-Rapids process already depicted in Fig. 3.

5.1 Raising of Quality Alerts

In the first phase of the Q-Rapids process, we propose the analysis of data gathered from several sources as to raise quality alerts when some quality factor yields an evaluation outside the acceptable thresholds.

In Fig. 5, we exemplify the hierarchy of a quality model already described Fig. 4. We adopt the Quamoco approach [31], which proposes weighted sums to implement this aggregation. Albeit simple, this makes explicit the knowledge about quality, and allows transparency of quality-related decisions.

In Fig. 5, the *Cyclomatic Complexity* and *Comments Density* metrics are aggregated into the *Analyzability* product factor. This product factor is aggregated together with another one, *Adapatibility*, into the ISO 25010 *Maintainability* quality factor. Each product factor should have a specified weight.

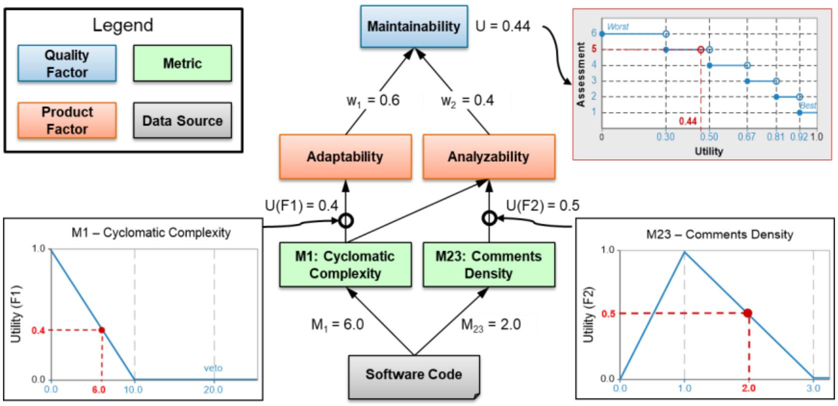

Fig. 5 Example of utility functions to assess the Maintainability quality factor –adapted from Quamoco [31].

Once the quality model is built, the next step is to apply utility functions as the way to embody knowledge on what is an acceptable level of quality. Utility functions model the preferences of decision makers with respect to the values of the factors, so that they can define an alert to be triggered by Q-Rapids when a basic metric or a factor has exceeded a defined threshold. In Fig. 5, we can see examples of utility functions for *Cyclomatic Complexity*, *Comments Density* and *Maintainability*. In a normalized value from 0 to 1, we can see that, in the current state of the system, *Maintainability* has the value of 0.44. If the goal had been set to have maintainability at its best value (this means to be greater than 0.92, as defined in the utility function), the corresponding alert would be triggered to inform that maintainability of the product should be improved. Quality alerts are how Q-Rapids identifies the need for new QRs.

5.2 Identification of Candidate Quality Requirements

Quality alerts are bound to elements from the quality model. As the reference model in Fig. 4 shows, QR patterns in the catalogue are bound to such elements, therefore Q-Rapids proposes the use of this binding to select the QR patterns whose instantiation will produce candidate QRs.

Fig. 6 shows an excerpt of a QR pattern that will be used as example in Section 6. We can see how the link with a factor, the *Availability* product factor (subfactor of the ISO 25010 *Reliability* quality factor), is explicitly written. The template is expressed in natural language and shows two parameters: the *entity* (the whole system, a service, a component, …) and the required *availabilityLevel* (a percentage). A cost function is attached to one of its parameters, measuring the relative cost of implementing the required availability level. This function will be defined as a set of pairs (parameter value, function value) (see Fig. 4).

Q-Rapids will select all the patterns bound to the factor for which the alert was triggered and for each, will propose one or more instantiations to be rendered to the decision maker. Having one or more instantiations will depend on the knowledge available to propose different alternative values to the formal parameters: these values should be determined smartly in terms of the utility function of the factor, the effect on the strategic indicators and the available parameter functions.

Pattern Keep Availability Level
Goal: To determine which is the appropriate availability level for a given component of the system
Bound to: Reliability::Availability
Template: The <*entity*> shall have a minimal availability rate of <*availabilityLevel*>
Parameter functions:
cost(<*availabilityLevel*>)

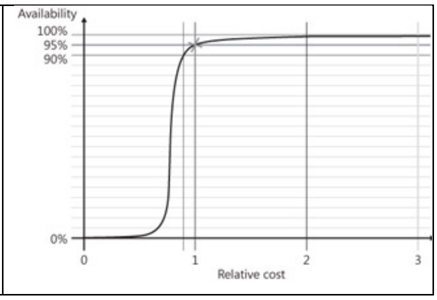

Fig. 6. Requirement pattern Keep Availability Level (left) and cost function of one its parameters (right; graphical representation).

5.3 Assessment of, and Decision on, Candidate Quality Requirements

Once a candidate QR is derived from the pattern catalogue, it will be assessed by the decision-maker using the dashboard. To assess the impact of the candidate QRs on the indicators, we plan to use Bayesian networks as proposed in the VALUE framework [33]. Models are to be built with a mix of expert knowledge and data, so that they will evolve as the system learns from decisions made. In addition to the indicators, the dashboard will show those other functions bound to QR (coming from the corresponding QR pattern, see Fig. 4), e.g. implementation cost, so that it will be possible to visualize the fluctuation of these functions if the candidate QR were selected.

As an outcome of this assessment, several actions are possible onto the candidate QR (see state diagram for candidate QRs at Fig. 7):

- *Consider in the current iteration*. The QR is added into the current iteration backlog and correctly documented (see next subsection). This action will be applied only in the case of detecting a critical quality concern violation (e.g., resulting in massive negative feedback from users in a short period of time).
- *Consider in future iterations*. Usual action when the QR is accepted but it is moved to the product backlog waiting for its future inclusion in the iteration backlog. As in the previous action, the QR should be correctly documented.
- *Postpone decision*. In this case, the QR is considered plausible but no action is taken yet. The QR is included in the product backlog but with the minimal possible metadata (e.g., if it a user story, its effort is not computed) and it may be removed in the future or even reconsidered using the same casuistic described here.
- *Negotiate*. A QR that is considered necessary may not be possible to just accept, but requires joint analysis with conflicting requirements, eventually calling for a negotiation among involved stakeholders. The QR will not be included in any backlog until the dependencies are managed, and eventually it may end up slightly modified or even just rejected.
- *Detail*. Since the QRs are derived from a pattern catalogue, it may eventually happen that a template misses some detail that is necessary in the current situation or even because Q-Rapids has not access to enough knowledge as to generate values for the formal parameters. In this case, the QR is not ready to be included in the backlog and further refinement is requested.
- *Rephrase*. It may happen that the QR is not representing the quality concern in a valid way. This action asks for an alternative formulation. In this action and the previous one, the pattern catalogue needs to be examined to decide whether it can be improved with the knowledge stemming from this candidate QR.
- *Discard*. The QR analysis using the dashboard concludes that its acceptance would be harmful because some strategic indicators would be seriously damaged.
- *Redefine quality thresholds*. Candidate QRs are proposed considering utility functions and thresholds defined by decision makers for the factors. This action is to be taken when the analysis of the QR points out that the identified quality misbehaviours is in fact not such. Therefore, it is necessary changing the utility functions and thresholds in the QM so that the reason for triggering a QR disappears.

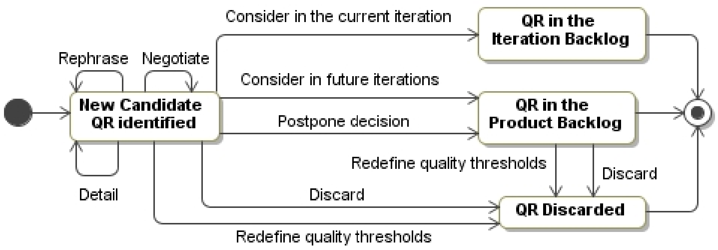

Fig. 7 State diagram of candidate QRs.

5.4 Representation of Quality Requirements

In case the candidate QR is accepted as a valid requirement for the system, it needs to be correctly included in the backlog (either iteration or product backlog). As stated in Section 2, the documentation of QRs in ASD is a well-known challenge [7][34].

In [24], we analysed different agile artefacts to represent QRs in ASD: user stories, epics and acceptance criteria. From a practical perspective, the type of artefact impacts on the information that is required to determine when entering the QR into the backlog. For instance, if it is represented as a user story, it will require fixing its priority (according to its business value and effort required), determining its acceptance criteria, etc.

6 Example Scenarios

The following scenarios are inspired on real-based challenges and problems the companies in the Q-Rapids project's consortium face. The scenarios aim at illustrating situations in which a company using ASD would eventually benefit from using Q-Rapids. They are presented with increasing ambition. In the first scenario, the quality alert is not automatically triggered by Q-Rapids but it is the quality expert who raises it through inspection of the dashboard. In the second scenario, the process is automatized. In the third scenario, in order to identify the root cause of the quality alert, two sources of data need to be correlated. Fig. 8 shows the indicators, factors and metrics used.

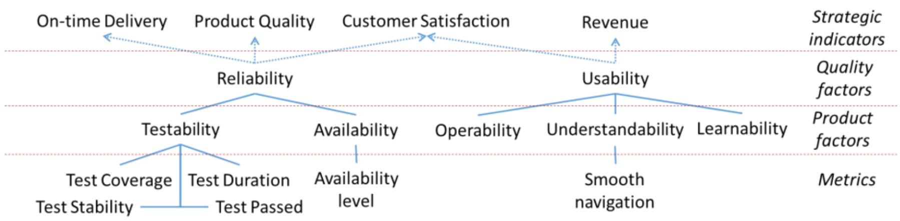

Fig. 8 Strategic indicators and quality model elements used in the three scenarios.

6.1 Scenario 1: Test Coverage

Context. ACME is a software-intensive company continually evolving its software products, with rapid release environment. Currently, they are releasing software every four weeks. To improve time-to-market, the company decides reducing the release period from four to three weeks. They enter the new release time in the Q-Rapids tool.

Raising of quality alerts. All elements of the ACME quality model are regularly monitored, assessed and visualized through the dashboard. Using the Q-Rapids' dashboard, release engineers see the impact of the release decision on the quality model assessment: *On-time delivery* strategic indicator is positively impacted whereas *Product Quality* is negatively impacted, even under the targeted threshold.

Identification of candidate QRs. By browsing patterns in the catalogue related to factors that influence *Product Quality*, Q-Rapids finds *Keep High Coverage Tests*, linked to *Test Coverage*, whose template is: The <*entity*> shall have a test coverage not lower than <*percentage*>. The parameter <*percentage*> is key for assessing the benefit of executing a test. Given his domain knowledge, the engineer predicts a solution to reduce testing time without dramatically decreasing product quality: the prioritization of test cases with greater coverage that cover a greater number of potential bugs [35].

Assessment of, and decision on, candidate QRs. With the support of the platform, the engineer generates the QR from the pattern: The *prioritized tests* shall have a minimal condition coverage rate of *70%*. The engineer can visualize the effect of this QR using the drill down feature of the dashboard, allowing to see prioritized tests and their characteristics (i.e., Test Coverage, Test Duration, …). Release engineers can use this information to postpone the execution of tests with lower or redundant test coverage.

Representation of the QR. Three QRs may be inserted into the backlog. First, an issue indicating the reduced set tests to be executed by release engineers because they potentially cover more bugs. Second, the improvement of tests with low quality levels. Third, the test quality product factor also shows the untested code. As Martin Fowler's argue: "It's worth running coverage tools every so often and looking at these bits of untested code" (https://martinfowler.com/bliki/TestCoverage.html). Therefore, another QR should be inserted: manually check the parts of code that are untested.

6.2 Scenario 2: Availability

Context. ACME provides regular over-the-air (OTA) updates of the software installed in their smartphones. However, due to scalability problems, the service that provides such updates, the *update service*, may experience availability issues. This is especially true when there is a critical update and multiple devices try to download such update at the same time. These problems may lead to customer dissatisfaction. The availability rate of the update service is monitored through a data collector that regularly checks if the service is up-and-running. Unfortunately, the current implementation of the system has not considered a QR related to availability specific for this functionality; just a system-wide availability requirement exists, used to design the utility function.

Raising of quality alerts. In this context, ACME releases a critical OTA update, and the data collector detects that the *update service* availability goes down from 95% to 60%. This value is checked against the quality model, which links this gathered data to the quality factors and strategic indicators. To do so, the availability is normalized using the utility function: availability of 60% corresponds to a utility value of 0.35. Such utility value is used to compute the quality factor *Reliability*, and ultimately the strategic indicator *Customer Satisfaction*, whose value falls below the required threshold.

Identification of candidate QRs. Due to the indicator's violation, Q-Rapids browses the pattern catalogue and finds the *Keep Availability Level* pattern (see Fig. 6) linked to the factor whose utility function is failing. In this case, the parameter <*system*> is instantiated as *update service*. For the instantiation of <*availabilityLevel*>, Q-Rapids uses the cost information embodied in the pattern and selects 3 alternatives, 99%, 90% and 80%, which yields to 3 different candidate requirements, one for each value.

Assessment of, and decision on, candidate QRs. The Q-Rapids dashboard offers the three QRs so that the decision-maker can analyse the consequences of each of them by inspecting the fluctuations of the *Customer Satisfaction* strategic indicator value. The decision maker can see that availability of 80% reaches adequate results in terms of *Customer Satisfaction* and cost. Availability of 90% improves significantly the *Customer Satisfaction* with a minimum increase of cost compared to the first option. The last option (availability of 99%) provides a *Customer Satisfaction* only slightly better than the previous option but doubling the cost. After considering these facts, the decision maker selects the QR with 90% availability rate and discards the other two.

Representation of the QR. Q-Rapids considers that the *update service* was implemented in response to a user story identified in some past iteration and marked as "done": "As a user, I want to download OTA updates so that the smartphone is secure". The QR refers to this user story and no other functionality, therefore Q-Rapids adds the QR as new acceptance criteria. By adding a new (unsatisfied) acceptance criteria, such user story is ready to go back to a backlog. Given the importance to fix quickly this problem, Q-Rapids suggest to include it in the current iteration backlog.

6.3 Scenario 3: Usability

Context. ACME has just released a major update on the user interface of an *e-commerce platform* used to sell their mobile phones and they expect that this update will improve the user experience and hence increase the number of sales.

Raising of quality alerts. In this context, the data collector detects a significant amount of users who get stuck in a particular step of the process and quit without buying the phone. The data collector informs that the *Smooth Navigation* metric (which measures how many users can complete the process of buying the phone smoothly) goes from 90% down to 80%. This metric is normalized using the utility function, yielding to a utility value of 0.45, which impacts negatively on the upper layers in *Understandability*, *Usability*, and finally, *Revenue*. Because of this process, the value of the strategic indicator *Revenue* goes below the required threshold.

Identification of candidate QRs. A drop in the utility value of *Smooth Navigation* causes the violation of the *Revenue* indicator. However, this information does not suffice to identify a candidate QR. This is because a drop in *Smooth Navigation* could be caused by several factors (e.g., is there a problem related to the browser of the user? Or is it related to an issue in the internet connection?). In this scenario, the violation of the indicator triggers Q-Rapids to correlate *Smooth Navigation* with other metrics to detect the cause of the problem. After inspecting different metrics from the multiple data sources under monitoring, Q-Rapids detects a strong correlation between *Smooth Navigation* and the user device type. In particular, it detects that the device of those users who present that problem is a smartphone. Q-Rapids browses the pattern catalogue and finds the *Interface Type* pattern: The <*system*> shall be supported by <*device-types*>

devices, which is linked to the *Understandability* factor. The instantiation yields to the candidate QR: The e-commerce platform shall be supported by smartphone devices.

Assessment of, and decision on, candidate QRs. The Q-Rapids dashboard offers the QR so that the decision-maker can analyse the consequences of applying it by inspecting the fluctuations of *Revenue* and other impacted indicators, e.g. *On-Time delivery*. Q-Rapids shows that considering the QR would return the utility value of *Smooth Navigation* close to 1, and hence a very good *Revenue*. *On-Time delivery* would worsen but, by computing different metrics, Q-Rapids demonstrates that not significantly. After considering these implications, the decision-maker decides to apply the QR.

Representation of the QR. Since the QR is asking for a well-defined implementation task, Q-Rapids suggests a new user story: "As a user, I want to be able to use the e-commerce platform using my smartphone in order to ensure user's experience for this kind of device". Finally, Q-Rapids adds the user story to the product backlog, ready to be scheduled for the next iteration.

7 Evaluation

In order to get early feedback from our exploratory ideas, we ran a questionnaire in one of the companies of the Q-Rapids consortium, Bittium Wireless Ltd. We simplified the TAM evaluation questionnaire [36] since the technology is not available; the-refore, we asked them to evaluate their vision on the scenarios, using the questions in Table 2, scored from 1 (strongly disagree) to 7 (strongly agree). We got 3 responses.

All the responses, without exceptions, score positively. In addition, we got free text for most responses. Among the feedback, we can mention: 1) acknowledgment that a lot of knowledge needs to be managed inside Q-Rapids ("When thinking about all possible scenario in real life, quite a "machine" is needed!"); 2) statement that even if the system behaves well, decision-makers want to have the opportunity to confirm themselves the recommendations by "digging deeper in the data manually"; 3) Scenario 3 is really valuable ("the third scenario is really genuine, if it works").

Table 2. Questions to practitioners for evaluating Q-Rapids.

Criteria	Question	R1	R2	R3
Perceived usefulness	Using Q-Rapids for eliciting, deciding upon and documenting QRs as described in the scenarios above, could be useful for my job	6	5	7
Perceived ease of use	I find easy using Q-Rapids for eliciting, deciding upon and documenting QRs as described in the scenarios above	6	6	7
Output quality	The output given by Q-Rapids as described in the scenarios above, is high	7	6	7
Result demonstrability	It would be easy for me telling others about the results of using Q-Rapids as described in the scenarios above	5	6	7
Feasibility	I find that the scenarios described above can be implemented in my work environment	7	5	6

8 Final Discussion

In this exploratory paper, we have presented a data-driven method, Q-Rapids, for improving QR management in ASD. Q-Rapids argues for the intensive use of data to detect quality misbehaviours, then to use the knowledge embedded in the method to propose candidate QRs, to visualize the consequences of their acceptance in a dashboard and, if the decision-maker accepts them, to document them in a backlog.

We are aware that the main goal of the method is very ambitious and can be the case that not all QRs can be detected as proposed in Q-Rapids; one of the main results of the method evaluation will be to understand its limitations and identify which QR types are the best suited for this data-driven analysis. The Scenario 1 has tried to illustrate this eventual need of putting the human in the loop, which in fact has been one of the points highlighted in the preliminary evaluation presented in Section 7. Other threats also exist: e.g., privacy issues are always delicate in data-driven approaches.

Q-Rapids has been conceived to focus on QRs. However, as the project progresses, we see that the main concepts could eventually be used both for suggesting new features and for dealing not only with product factors, but process factors (e.g., factors affecting team performance). This can become a further goal in later stages of our research. Also, the implementation, validation and a complete evaluation of the proposed method are planned for the near future. Last, the adoption of Q-Rapids beyond ASD (e.g., in organizations adopting DevOps) is part of our research agenda.

References

[1] Capgemini: World Quality Report 2015-16, 7th Edition. Available at https://www.capgemini.com/thoughtleadership/world-quality-report-2015-16.

[2] Reuters: http://www.reuters.com/article/us-hsbc-it-idUSKBN0UJ0ZB20160105, 2015.

[3] Pohl, K.: *Requirements engineering: fundamentals, principles and techniques*. 2010.

[4] Wagner, S.: *Software Product Quality Control*. Springer, 2013.

[5] Rodríguez, P. *et al.*: Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study. *Journal of Systems and Software*, 123, 2017.

[6] Ramesh, B., Baskerville, R., Cao, L.: Agile Requirements Engineering Practices and Challenges: An Empirical Study. *Information Systems Journal*, 20(5), 2010.

[7] Inayat, I. *et al.*: A Systematic Literature Review on Agile Requirements Engineering Practices and Challenges. *Computers in Human Behavior*, 51(B), 2014.

[8] Wieringa, R.: *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.

[9] Berntsson-Svensson, R., Host, M., Regnell, B.: Managing Quality Requirements: A Systematic Review. SEAA 2010.

[10] Maalej, M., Nayebi, M., Johann, T., Ruhe, G.: Toward Data-Driven Requirements Engineering. *IEEE Software* 33(1), 2016.

[11] Groen, E.C. *et al.*: The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software* 34(2), 2017.

[12] Kurtanovic, Z., Maalej, W.: Mining User Rationale from Software Reviews. RE 2017.

[13] Shekhovtsov, V.A., Mayr, H.C., Kucko, M.: Implementing Tool Support for Analyzing Stakeholder Communications in Software Development. ICSTW 2015.

[14] Liu, X. *et al.*: Deriving User Preferences of Mobile Apps from Their Management Activities. *ACM Transactions on Information Systems* 35(4), 2017.
[15] Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J.: *Non-Functional Requirements in Software Engineering*. Springer, 2000.
[16] Ameller, D., Ayala, C.P., Cabot, J., Franch, X.: Non-functional Requirements in Architectural Decision Making. *IEEE Software* 30(2), 2013.
[17] Berntsson-Svensson, R., Regnell, B.: A Case Study Evaluation of the Guideline-Supported QUPER Model for Elicitation of Quality Requirements. REFSQ 2015.
[18] Caracciolo, A., Lungu, L.F.,. Nierstrasz, O.: How Do Software Architects Specify and Validate Quality Requirements?. ECSA 2014.
[19] Daneva, M. Buglione, L. Herrmann, A.: Software Architects' Experiences of Quality Requirements: What We Know and What We Do Not Know? REFSQ 2013.
[20] Letier, E., Stefan, D., Barr, D.T.: Uncertainty, Risk, and Information Value in Software Requirements and Architecture. ICSE 2014.
[21] Franch, X. *et al.*: The RISCOSS Platform for Risk Management in Open Source Software Adoption. OSS 2015.
[22] Sillitti, A., Succi, G.: Requirements Engineering for Agile Methods. In: *Engineering and Managing Software Requirements*, Springer, 2005.
[23] Martakis, A., Daneva, M.: Handling Requirements Dependencies in Agile Projects: A Focus Group with Agile Software Development Practitioners. RCIS 2013.
[24] Behutiye, W. *et al.*: Non-functional Requirements Documentation in Agile Software Development: Challenges and Solution Proposal. QUASD@PROFES 2017.
[25] Heck, P., Zaidman, A.: A systematic literature review on quality criteria for agile requirements specifications. *Software Quality Journal*, 2018.
[26] Furtado, F., Zisman, A.: Trace++: A Traceability Approach to Support Transitioning to Agile Software Engineering. RE 2016.
[27] Guzmán, L. *et al.*: How Can Quality Awareness Support Rapid Software Development? - A Research Preview. REFSQ 2017.
[28] Gómez, C. *et al.*: Towards an Ontology for Strategic Decision Making: The Case of Quality in Rapid Software Development Projects. MREBA@ER 2017.
[29] Guizzardi, G.: Ontological foundations for structural conceptual models. CTIT, Centre for Telematics and Information Technology, 2005.
[30] Li, F.-L., Horkoff, J., Mylopoulos, J., Guizzardi, R.S., Guizzardi, G., Borgida, A., Liu, L.: Non-functional requirements as qualities, with a spice of ontology. RE 2014.
[31] Wagner, S. *et al.*: Operationalised Product Quality Models and Assessment: The Quamoco Approach. *Information and Software Technology*, 62, 2015.
[32] Quer, C. *et al.*: PABRE: Pattern-Based Requirements Elicitation. RCIS 2009.
[33] Mendes, E. *et al.*: Towards improving Decision Making and estimating the Value of Decisions in Value-based Software Engineering: The VALUE Framework. *Software Quality Journal* (available online).
[34] Heikkilä, V.T., Damian, D., Lassenius, C., Paasivaara, M.: A Mapping Study on Requirements Engineering in Agile Software Development. SEAA 2015.
[35] Noor, T.Z., Hemmati, H.: Studying Test Case Failure Prediction for Test Case Prioritization. PROMISE@ESEM 2017.
[36] Davis, F.F.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, 13(3), 1989.