The Quality Triage Method: Quickly Identifying User Stories with Quality Risks

Gunnar Brataas*, Inger Anne Tøndel*†, Eivind Okstad*
Ola Løkberg*, Martin Gilje Jaatun*, Geir Kjetil Hanssen*, Thor Myklebust*
*SINTEF Digital, Trondheim, Norway
{gunnar.brataas, inger.a.tondel, eivind.okstad, ola.lokberg, martin.g.jaatun, geir.k.hanssen, thor.myklebust}@sintef.no
†Norwegian University of Science and Technology (NTNU), Trondheim, Norway

***Abstract*—Quality requirements often receive insufficient attention, both in agile and in traditional software development. This paper describes the quality triage, a quick, agile method in which user stories or features with quality risks are identified. This paper shows how the four qualities scalability, security, safety, and availability are scored at short expert meetings — triages. In this way, quality risks are made explicit and can be immediately addressed. We illustrate the method with a scenario involving semi-autonomous cars.**

***Index Terms*—quality requirements, non-functional requirements, agile software development, DevOps, large complex systems, scalability, security, safety, availability, ATAM**

I. Introduction

Most software engineering methods have requirements management at their core. Numerous schools have tried to capture the best practice for such management. These practices range from methods that seek to capture and structure nearly all requirements early on, to methods that build the understanding of requirements as an integrated part of the development of the solution. However, most methods seem to share an excessive focus on capturing functional requirements, which is natural as we develop a system to solve tasks or perform operations. If we look at agile methods [1] — which seems to be the norm in the software industry today — the principle of the evolving product backlog and the frequent feedback from the product owner (PO) mostly emphasizes the needs for function and not so much the quality aspects.

Functional requirements are concrete, relatively easy to express, and have visible value. Since quality requirements often span the complete solution, they typically suffer from the "curse of the commons" where many functions and customers may benefit, but without anyone having clear ownership of the requirement [2]. Quality requirements commonly end up being implicit and unspecified, and may only really surface when the system is in operation [3], [4], [5], according to the fix-it-later approach [6].

Although functional requirements are commonly prioritised over quality requirements in agile projects [7], [4], [8], [3], this does not mean that quality is unimportant. A system offering all functions required by PO, but which fails to meet quality requirements, such as for scalability or security, is a poor system. However, this does not define the complete challenge. Mastering a specific system quality, for example how well it scales, or how secure it may be in given situations, is not enough. We postulate that quality requirements are related, and must be understood and managed together. This is especially important for complex systems.

For example, a remotely controlled oil-well system will have quality requirements addressing security (should withstand attacks), safety (should not cause harm or damage), availability (should not break down), and scalability (should not fail due to heavy workload). Each of these types of qualities can be managed by themselves, for example, by using a host of techniques for analysis and management, like hazard analysis, vulnerability analysis, or by modelling system performance. But what about their potential dependencies? If this example system — which needs to be both safe, available, secure and scalable — fails in one aspect (e.g. being hacked), what would that mean for the safety quality, as an example? Or, what if the system doesn't handle the workload, could that expose security vulnerabilities? For the system to be safe it may need to shut down, indicating it would break availability requirements. These complex questions with no simple answers serve as our motivation for this paper.

Another motivation is the need for a manageable method to handle quality requirements, to reduce the potential of being overwhelmed. Although one may have good techniques to handle individual qualities, having to integrate a plethora of such techniques into a project can result in a heavy approach, even though the techniques in themselves may be relatively light-weight [9].

In this paper, we propose the quality triage method to quickly identify areas to focus on when it comes to quality. The concept of a quality triage is borrowed from emergency medicine, where a doctor quickly determines if a person requires immediate treatment or can wait. Similarly, a quality triage is an expert group meeting to quickly identify the areas with most quality concern, where further effort and coordination are required. This helps the project to assign appropriate priorities both at the early stages and throughout. A concrete approach for addressing all qualities together makes the process manageable. This is particularly relevant for large, complex, agile projects with tough quality requirements.

The paper is structured as follows: Section II gives an

978-1-7281-9690-9/20/$31.00 ©2020 IEEE

overview of related work. In Section III, we describe the quality triage approach. Section IV applies the method to a scenario from autonomous cars, where we address four qualities: scalability, security, safety and availability. The contribution of the paper is discussed in Section V. Section VI concludes the paper and outlines further work.

II. State of the Art

It is well documented that neglect of non-functional requirements is common in agile development [7], [4], [8], [3]. Studies point to different reasons why this is the case. Behutiye et al. [3] performed a systematic mapping study (N=156) of quality requirements in agile and rapid software development [3] identified the following top five challenges: a limited ability of agile software development to handle quality requirements, time constraints due to short iteration cycles, limitations in testing quality requirements, neglect of quality requirements, and lack of an overall picture of quality requirements.

Ramesh et al. have pointed to customers' focus on core functionality and their lack of recognition for the importance of non-functional requirements at an early stage [4]. Ramesh et al. described the state of affairs in the following way: *"Of specific concern to a majority of the organizations in the study are operational scalability and security of systems. With the focus primarily on delivering functionality to the users as early as possible, these concerns typically do not receive much attention during early development cycles. Also, in the absence of a detailed design, the non-functional requirements were either typically not understood or explicitly specified at the appropriate time, or negotiated carefully to understand the trade-offs involved. The inadequate attention given to understanding and implementing non-functional requirements makes it harder to incorporate them as the system grows through successive development cycles. Also, without clear specification of the quality requirements, developers may make design choices that are arbitrary, and this makes it difficult to assess whether the system meets the real requirements"* [4].

Based on a systematic literature review, Alsaquaf et al. [9] point to limitations in the available techniques but also to characteristics and what is often considered strengths of the agile development process itself (*"the role of the product owner, the use of user stories"*), suggesting that agile development projects are prone to neglecting quality requirements. There is a potential over-reliance on the Product Owner (PO) [9], with the risk of the PO hampering inclusion of quality concerns. Additionally, the format of user stories may not be well suited to document quality requirements and their dependencies. Alsaquaf et al. point out the need for a holistic approach to quality and non-functional requirements overall. In their systematic literature review they identified many proposals for integrating specific types of quality requirements into agile development, but they point out that adopting separate methods for the different qualities is not a viable solution as it will lead to a heavy approach to quality requirements management. Limitations in the available techniques may also be a concern [9], and coming up with adequate techniques is challenging.

Alsaqaf et al. [10] additionally performed an interview study with 17 practitioners working on large-scale agile projects in six organisations. The study resulted in identification of 15 challenges grouped into five classes, 13 mechanisms that underlie these challenges, and nine practices to meet these challenges. Several of the challenges and underlying mechanisms are related to the relation between different types of quality requirements. Examples include hidden assumptions, sub-optimal priority assignment, late detection of quality requirements' infeasibility, focusing on a specific viewpoint or component, and losing sight of the big picture. Additionally, there were challenges related to elicitation and overlooking sources of quality requirements. The challenges were managed using assumption wiki-pages, multiple product backlogs, automated monitoring tools, by reserving parts of the sprint for important quality requirements, by doing sprint allocation based on multiple product backlogs, by establishing a preparation team, a components team and a quality requirements specialist team, and by having innovation and planning iterations.

Poller et al. [5], in their study of one organisation and the effects of external security audit, found that security was considered a quality aspect among other quality aspects, and developers was expected to deal with such quality issues without this responsibility being explicit and visible. Responsibility for quality requirements is mentioned as a challenge in a variety of studies [11], [12], [13]

Knauss et al. [14] point out the need to consider both Just in Time and long term considerations for quality requirements. They use safety and security requirements as examples to illustrate that a longer perspective is needed to complement the just in time considerations. This is needed since quality requirements "are typically related to architecture, and tend to build on knowledge as much as on software structures, although both need to come together."

Approaches to handling quality requirements in software development exist. One of the more prominent is the Architecture Tradeoff Analysis Method (ATAM) [15], a comprehensive method for evaluating software architectures relative to quality goals. ATAM is performed by experts taking part in long meetings (a lightweight evaluation meeting is estimated at four to six hours) with the goal of discovering risks, non-risks, sensitivities, and tradeoffs. One part of ATAM is the utility tree, where each quality attribute (such as performance) has a few associated attribute refinements (such as throughput). For each of the attribute refinements, architecturally significant requirements (ASRs) are identified. The ASRs are evaluated by business value and architectural impact, often using values like high, medium, or low.

When it comes to agile development, there are many suggestions for how to address challenges related to quality requirements [3], but these are generally less mature than e.g. ATAM or are targeted towards only one quality dimension. One example is Lemmetti et al. [16] that suggest a one day workshop in the first iteration of the project to identify quality requirements. Another example is ScrumScale [17], which includes the concept of a scalability triage at iteration

TABLE I
QUALITY FACTORS TO RATE

Quality	Factor	Description
Security	Asset values	How valuable are the assets that this functionality touches upon?
	Exposure	To what extent does this functionality open up for attacks?
Scalability	Workload	Amount of work to be done in a given period
	Response time	How fast must the result be available?
Safety	Possibility	The possible occurrence of the safety critical situation (event)
	Safety consequence	Potential loss in terms of loss of lives, expenses or damage to the environment
Availability	Probability of failure	Probability of functional failure given component redundancy
	Restoration time	Time for the restoration of a failed system (unplanned maintenance)

zero and if required, also for a sprint. Such an approach has been found to reduce the amount of analysis required [18], and is thus a motivation behind the work on quality triage presented in this paper. A number of studies, both theoretical [19] and empirical [20], [21], [22], point to the importance of considering quality requirements early on in an agile development project. However, others point to this early capturing of quality requirements as a challenge, since underlying assumptions may change [23]. The Protection Poker [24] game is an example of a technique that evaluate security risk at the beginning of each iteration.

Involvement of quality experts is commonly suggested, such as establishing a quality requirements specialist team [10] or including experts on the team (e.g. security champion [22] or a usability expert [13]). Selection and utilization of experts to support the development team has been included in elicitation guidelines for non-functional requirements [25].

III. THE QUALITY TRIAGE METHOD

In the this section, we explain the concept of a quality triage in more detail, how it can be performed in practice and how it would fit into an agile development practice. Figure 1 gives an overview of the quality triage method, exemplified by the four quality dimensions scalability, security, safety and availability. The quality triage method follows these high-level steps:

1) In an expert group meeting user stories/features are evaluated from each quality's point of view.
2) Identifying and evaluating where different qualities may have an influence on each other, and help to decide on solutions.

The initial evaluation of user stories/features in this figure is represented by individual triages (scalability triage, security triage, etc.). In for example the scalability triage, the user stories/features are evaluated by experts on scalability with the goal to identify the most critical user stories/features from a scalability point of view to understand where additional

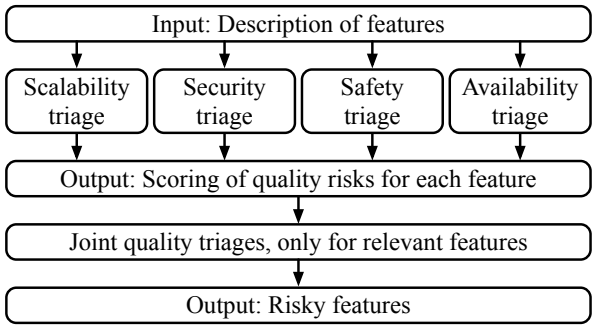

Fig. 1. Major steps, inputs and outputs in the quality triage method

measures may be needed. Additionally, the scalability experts would be expected to come up with potential suggestions for how to improve the identified issues. After all the individual triages have been performed, the experts gather for a short meeting (quality triage) where issues are flagged, potential solutions proposed and any dependencies identified. The quality triage will then point to areas where quality experts need to work together to obtain solutions that will enable the project to meet all their quality goals.

The quality triage method is light-weight and flexible and can be adjusted to the needs of different types of organizations and projects. Such a quick evaluation is needed in projects where there are insufficient resources (time, money, expertise) available to do a full analysis of the whole system related to each quality dimension, and thus, a more targeted and time-efficient evaluation is needed. In addition, it meets the needs to explore and handle dependencies among quality dimensions.

In the ScrumScale method [17], a scalability triage was envisioned to take place in sprint zero at the beginning of each project or release, and if required, also for each iteration. Similarly, we propose that the PO initiates quality triages when planning new releases. Performing this activity for all sprints will introduce too much overhead, but it should be done for all iterations when new features are introduced or if old features are drastically changed. Having quality triage when planning for a new release makes it possible to take quality dimensions into account for estimation and planning.

We propose that each quality dimension selects one to three factors to aid in assigning importance level scores. In Table I, we provide suggestions for the quality dimensions scalability, security, safety, and availability. The suggestions for scalability are the conventional concepts of workload and response time (see ScrumScale [2] for a more elaborate description of scalability requirements). The suggestions for security are inspired by Protection Poker [24], [26], which replaced the more traditional factors 'consequence' and 'likelihood' that is used in security risk analysis, with 'exposure' and 'value', to be better aligned to *the feature* as a unit of analysis as opposed to a potential security incident. Consequence and likelihood are again used as a basis for the safety factors by selecting 'possibility' instead of 'likelihood' to convey a high-level evaluation (not quantified). For availability we build on

the Reliability, Availability, Maintenance, and Safety (RAMS) process [27] in which in-service availability can be increased by optimizing reliability and maintainability. Redundancy and restoration time are factors related to maintainability. Note, redundancy is included in the description of probability of failure to include the significance of redundancy for the subsystem's functional availability, e.g. in relation to a sensor system. It is the reliability of the sensor system as a whole that is considered in Table I, and not the sensors individually.

Based on experiences from applying ScrumScale [17], [18] and Protection Poker [26], we suggest using a five point scale for each factor: VL (very low), L (low), M (medium), H (high) and VH (very high). Note that, similar to what is done in Protection Poker, this scale needs to be calibrated to the project at hand, which means that the quality experts doing the analysis need to consider what is very high or very low related to *this* project, and evaluate features/user stories related to that consideration.

The scores assigned by the quality experts can be used in the quality triage meeting to aid the discussions on dependencies and can help flag user stories/features where quality issues are prominent. Several things can happen:

- A user story/feature has no unacceptable risk for any of the quality dimensions, and thus can be developed without further analysis related to quality.
- A user story/feature has unacceptable risk for one quality dimension, and the mitigation causes no problems for other quality dimensions. The PO then makes the decision on mitigation based on risk and other project factors.
- A user story/feature has unacceptable risk for one quality dimension, and the mitigation creates more risks for other quality dimensions. This points to a need for the related quality experts to work together to find acceptable solutions to be considered by the PO.
- A user story/feature has unacceptable risk for two or more quality dimensions. This points to a need for the related quality experts to work together to coordinate suggestions for solutions, which are to be considered by the PO.

IV. Semi-Autonomous Car Scenario

This scenario primarily considers level 4 of semi-autonomous driving, as defined in SAE J3016 [28]. An autonomous car has several sensors, cameras and LIDARs (light detection and ranging), radar, GPS (Global Positioning System), and weather data. The software in an autonomous car has billions of lines of code and thousands or millions of features. We are interested in the situation where a new release shall be produced. This release may have in the order of 10 to 100 new functions/features. In this case, it will simply be too much work to analyse in detail all the quality implications of all of these new features. The idea of this paper is to be able to focus the attention on the critical features. This focus does not mean that the other features are ignored, and they may also later become critical features. Besides, it is vital to discover how new features have quality implications for old features.

A. Description of Features

In this paper, we consider the following three epics (EX) with underlying features (FX.Y).

E1 - Improved collision sensor system: Combine information from several sensors, cameras and LIDARs in the front left and right of the car, to avoid collisions with, for example, small dogs.

- **F1.1** Collect information from different sources.
- **F1.2** Prune information from different sources, because we cannot deal with every entering snow flake or leaf coming.
- **F1.3** Make decision: Find out what to do and then actually brake or turn. If the decision does not have the intended effect, repeat all features in this epic.

E2 - More V2V communication: Receive direct information about risky situations from the cars ahead using vehicle to vehicle (V2V) communication:

- **F2.1** Open up for communication with more neighbouring cars (not only the car ahead).
- **F2.2** Compare against other available information, such as speed, rotating wheels, and temperature.
- **F2.3** Make decision, similar to **F1.3**.

E3 - Improved accident warnings (V2X): React automatically to more warnings about congestion, accidents or changes in road condition from traffic control servers using V2X, vehicle-to-everything communication. Messages are authenticated via a public key infrastructure (PKI).

- **F3.1** Receive messages from a central server.
- **F3.2** Understand message, compare messages from different cars etc.
- **F3.3** Make decision, similar to **F1.3** and to **F2.3**.

B. Scoring of Quality Risks for each Feature

Table II shows the scoring of the different quality criteria for the features. In the following section, we provide a brief justification for these scores.

a) Scalability: With sensor fusion, the amount of sensor data will increase in terms of processing, storage, and communication. Especially feature F1.1, but also feature F1.2 and F2.1, will only be feasible with breakthroughs in scalability. As a result, feature F1.1 has very tough workload and response time requirements, whereas the workload and response time requirements in feature F1.2 and F2.1 are high, but less rigorous. The features handling decisions, F1.3, F2.3 and F3.3 will be similar to existing decision features with a medium to low amount of data and and a medium to low response time requirement. Also, feature F2.2 will be manageable in terms of scalability.

b) Security: Opening up for more vehicle-to-vehicle (V2V) communication, as in E2, can highly increase the exposure of the system. The message that is received from a neighbouring vehicle may not be highly valuable in and of itself, but the functionality related to receiving messages may have a relationship to more valuable assets (such as lack of input validation may open up for attackers being able

TABLE II
SCORING OF QUALITY RISKS FOR EACH FEATURE, FROM VERY LOW (VL), VIA LOW (L), MEDIUM (M) AND HIGH (H), TO VERY HIGH (VH)

User story	Workload	Resp. time	Asset value	Exposure	Consequence	Possibility	Probability of failure	Restoration time
F1.1	VH	VH	M	L	VL	M	VL	VL
F1.2	H	H	H	VL	M	H	VL	VL
F1.3	M	M	VH	VL	H	L	H	H
F2.1	H	H	H	VH	H	M	M	H
F2.2	M	M	M	VL	M	H	M	H
F2.3	M	M	H	VL	M	H	M	H
F3.1	M	M	H	L	M	M	H	L
F3.2	L	M	L	VL	M	H	H	L
F3.3	L	M	H	VL	L	H	H	VL

to run code), thus a high score is given. As a result, F2.1 is considered important to consider further from a security perspective. There are similar concerns related to F3.1, but as a PKI is in place for authentication of messages from central servers the risk is drastically reduced. The assets related to making decisions (especially F1.3, F3.2 and F3.3) are considered highly valuable, but these concern well-placed internal processes that are believed to be difficult for an attacker to reach. We assume that we can trust the sensors and the internal communication in the car.

c) Safety: Though errors in single sensor readings are to be expected, these errors will likely not lead to safety consequences due to availability of a large amount of sensor readings. The consequence is therefore scored very low to medium for feature F1.1 to F1.2. Decisions based on such sensor data are regarded as safety critical functions, and safety requirements apply [29]. Therefore, a high safety consequence but low possibility of errors for F1.3 is assumed. Communication between cars is vulnerable and safety critical; therefore, F2.1 to F2.3 have been given medium to high consequence and possibility of errors. Early accident warning from a traffic control center are less safety critical because more time is available before situations become safety critical. Therefore, there will be less consequences. The same possibility scores as for E2 are given to F3.1 to F3.3.

d) Availability: In feature F1.1 and F1.2, minor effects on availability from single sensor failures are assumed as a high amount of total sensor readings are expected. Most of the signal failures from sensors will likely be caused by bad weather conditions, like heavy snow or rain. Thus, only minor efforts to regain function are expected. Possible errors in the decision software are assumed to require service, i.e. spending time out of service for the car and thus a high restoration time in feature F1.3. The same argument for restoration time applies to failure in communication software with cars further ahead in feature F2.1 to F2.3. However, here we assume a more reliable system than for F1.3, i.e. medium probability of failure scores of F2.1 to F2.3. For communication with the traffic control center, possible errors will generally cause delays for the autonomous vehicles, i.e. high probability of failure score for F3.1 to F3.3. The restoration time for F3.1 to F3.3 are however, assumed to be low because the failures most likely are external to the vehicles.

C. Joint Quality Triage, only for Relevant Features

With the scoring of Table II as input, the quality triage meeting identified features where further coordination is needed:

- Availability and safety: Several features (F1.3, F2.2 and F2.3) scores high on both availability and safety. Additionally, the quality triage identified a potential trade-off between safety and availability in this system, as incorporating safety requirements like in ISO 26262 [29] may result in the need to shut down the sensor system early when facing challenging situations, such as bad weather with snow or heavy rainfall.
- Scalability and security: Feature F2.1 scores high on both scalability and security. As security mitigations often cause scalability risks, further coordination between scalability and security experts is necessary.

Thus, additional meetings are set up to coordinate these aspects. In addition, the features F1.1 and F1.2 present scalability challenges, but they seem to be manageable for the other qualities. The scalability mitigation required for feature F1.1 and F1.2 is not likely to give any effects on the other risks.

As a result of this quality triage meeting, the product owner knows for which features additions are needed in the backlog, and which experts are responsible for suggesting solutions. Just as important, the product owner can proceed with the features where no quality additions are needed.

When the quality experts at a later stage recommend mitigations for the quality issues identified, the scoring in Table II can be used to identify quality experts that may need to be consulted to check that the mitigations do not cause further problems. In this case, one example is the mitigations that end up being suggested by the safety and availability experts concerning feature F1.3. For this feature, this quality triage identified the coordination need but did not provide any suggestion for mitigation that fulfilled the needs. When the experts later suggest a solution, this solution may need to be checked by availability and security experts to ensure it does not cause problems, as these qualities have moderate scores for this feature.

V. Discussion

In the introduction, we explained our goal of a manageable method to handle quality requirements for agile projects; to move from quality requirements being largely implicit to a practise of understanding and managing quality requirements in relation to each other. Our method is a compromise between the two extremes of on the one side fix-it-later [6] and on the other side a rigorous approach like a full model-based approach or something like ATAM [15]. In the following we discuss our relation to ATAM before moving on to discussing the costs and benefits that we see of the quality triage method. Finally, we point to critical success factors for projects wanting to adopt this method.

A. Relation to ATAM

We mentioned ATAM in Section II. The quality triage approach suggested in this paper is inspired by ATAM [15], and in particular the utility tree. The utility three utilizes factors to refine the quality dimensions, thus making it more manageable. Additionaly, the goal of the quality triage meeting(s) is similar to that of ATAM; identifying risk, non-risk and needs for further coordination. Building on the ideas of a well-recognized method such as ATAM is a strength of our approach. However, our approach is different from ATAM in several ways. Most importantly, it is a more lightweight approach than ATAM and more in line with the agile way of working. Our method takes as input the user stories or features already in the backlog, and not an architecture. Furthermore, the goal of the quality triage is to score the quality risks of user stories or features, not to identify all architecturally significant requirements when it comes to quality.

This has implications for the complexity of the method. Several qualities can be affected by the same features, whereas the architectural significant requirements (ASRs) in ATAM are unique for each quality. In the quality triage method, we do not have to deal with business value, since this is taken care of by the selected features (often with the highest business value). Most importantly, we are not analyzing architectural approaches, which in the lightweight version of the ATAM consumes the bulk of the time. Of course, the result of our quality triage approach is not a full walk through of the whole architecture. But this is not the point. The key value of the quality triage method is to direct the attention of the development team and the quality experts towards the most critical user stories or features using a continuous and lightweight approach.

B. Cost

The quality triage method comes with some costs, mainly involving the work needed to perform quality triages. Quality experts need to spend time to evaluate user stories/features from the point of view of their quality dimension, and discuss quality concerns with other quality experts. This is a challenging task, and the cost of these quality triages will depend on the experts' abilities to be brief in their evaluations and discussions. We expect that the costs of the quality triages will drop as experts become more familiar with this way of working. Note, however, that although there is a cost of estimating risk for all the features, this is a task that strictly speaking is required anyway (although it may be neglected in many projects). The quality triage method is a way to make this task more cost-effective in that the overall evaluation is done quickly, to provide more ability to focus on the key areas. There is potential to make this process even more efficient in some cases; if a whole epic seems simple for many qualities, there may be no need to do a detailed scoring.

The number of experts involved may vary depending on the projects, and this will also impact the costs. Note however that in reality there may well be overlap in the experts on the different qualities, such as the same person being both the scalability and the security expert.

C. Benefit

Through a scenario on semi-autonomous cars, we have demonstrated the potential usefulness of this method to quickly identify where to put effort on quality and where the different quality dimensions need to coordinate their effort. A study of a similar method has shown a reduced need for analysis; being able to restrict scalability analysis to high-risk features effectively reduced the analysis need from ten to three features [18]. In practice, this large reduction made the analysis manageable. Since risks are identified earlier, mitigation can also happen while the cost of fixing is low. The amount of testing may be relaxed for features with a low risk. To do an explicit analysis of quality risks will increase the quality of the solution. Thus, the need for costly and time-consuming fixing after introducing the solution to the public is avoided.

The quality triage approach introduces a common way to manage qualities, independent of which quality dimensions are most prominent. This makes the process more manageable and less costly for the product owner, as it does not require the product owner to relate to a myriad of quality approaches and ad-hoc and unstructured quality expert interactions. Especially for complex projects, these benefits should be important.

D. Success factors for adoption

We argue that the quality triage is a light-weight method in line with the the agile manifesto [1] (Individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; and responding to change over following a plan). Still, it is an add-on to current agile development methodologies, and this poses challenges on how to integrate it with current practice.

One main challenge that we see is that of ensuring that quality triages become a regular activity in the project. We have in this paper suggested that the PO should be responsible for the quality triage process, and thus, be the one to initiate a new round of quality triages and follow up the result. This is because the PO is in a position to know when during a project timeline that quality triages will be most useful and to integrate the results into the project. However, previous research has

shown that the PO in some cases acts as an obstacle for quality [9]. This could be because of a lack of competence or interest, or simply because of too many responsibilities. As an add-on to the agile process, the quality triage is dependent on someone "championing" it, at least in the early phases of adoption, to ensure it becomes a part of practice. If in a given organization and project this is not likely to be a role taken by the PO, then it is essential that someone else is given responsibility for following up both the initiation of quality triages and the integration of results into development.

VI. Conclusion and Further Work

We have described a light-weight and integrated method for identifying risk features with respect to four quality requirements: scalability, security, safety and availability. The recommendations made in this paper are based on the authors' experiences with working with companies on scalability, security and safety in agile development. More studies are needed to further improve this promising quality assessment method and gain knowledge on effects and factors important for adoption.

Acknowledgment

This work was supported by the SINTEF strategic project *Multilities: Agile Mastering of Security, Safety, Scalability and Availability Requirements*, and by two projects funded by Research Council of Norway: SoS-Agile: Science of Security in Agile Software Development (grant # 247678) and SMED: Smarter Innovation with Digital Transformation of Innovative Procurement (grant # 285542).

References

[1] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[2] G. Brataas and T. E. Fægri, "Agile Scalability Requirements," in *International Conference on Performance Engineering*. ACM, 2017.

[3] W. Behutiye, P. Karhapää, L. Lopez, X. Burgués, S. Martínez-Fernández, A. M. Vollmer, P. Rodríguez, X. Franch, and M. Oivo, "Management of quality requirements in agile and rapid software development: a systematic mapping study," *Information and Software Technology*, p. 106225, 2019.

[4] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: an empirical study," *Information Systems Journal*, vol. 20, no. 5, pp. 449–480, 2010.

[5] A. Poller, L. Kocksch, S. Türpe, F. A. Epp, and K. Kinder-Kurlanda, "Can security become a routine? a study of organizational change in an agile software development group," in *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 2017, pp. 2489–2503.

[6] C. U. Smith and L. G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, 2001.

[7] L. Cao and B. Ramesh, "Agile requirements engineering practices: An empirical study," *IEEE software*, vol. 25, no. 1, pp. 60–67, 2008.

[8] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in human behavior*, vol. 51, pp. 915–929, 2015.

[9] W. Alsaqaf, M. Daneva, and R. Wieringa, "Quality requirements in large-scale distributed agile projects–a systematic literature review," in *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Springer, 2017, pp. 219–234.

[10] ——, "Quality requirements challenges in the context of large-scale distributed agile: An empirical study," *Information and software technology*, 2019.

[11] I. A. Tøndel, M. G. Jaatun, D. S. Cruzes, and N. B. Moe, "Risk centric activities in secure software development in public organisations," *International Journal of Secure Software Engineering (IJSSE)*, vol. 8, no. 4, pp. 1–30, 2017.

[12] E. Terpstra, M. Daneva, and C. Wang, "Agile practitioners' understanding of security requirements: Insights from a grounded theory analysis," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 439–442.

[13] Å. Cajander, M. Larusdottir, and J. Gulliksen, "Existing but not explicit-the user perspective in scrum projects in practice," in *IFIP Conference on Human-Computer Interaction*. Springer, 2013, pp. 762–779.

[14] E. Knauss, G. Liebel, K. Schneider, J. Horkoff, and R. Kasauli, "Quality requirements in agile as a knowledge management problem: More than just-in-time," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2017, pp. 427–430.

[15] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Addison-Wesley, 2012.

[16] J. Lemmetti, M. Raatikainen, V. Myllärniemi, and T. Männistö, "Comparison of software architecture design and extreme programming," in *Work-in-Progress Session in IFIP Central and East European Conference on Software Engineering Techniques (CEE-SET)*, vol. 30, 2007, p. 44.

[17] G. Brataas, G. Hanssen, N. Herbst, and A. van Hoorn, "Agile Scalability Engineering: The ScrumScale Method," *IEEE Software*, September 2020.

[18] G. K. Hanssen, G. Brataas, and A. Martini, "Identifying Scalability Debt in Open Systems," in *Int. Conf. on Technical Debt*. IEEE, 2019.

[19] R. Goetz *et al.*, "How agile processes can help in time-constrained requirements engineering," in *Proceedings of the international workshop on time constrained requirements engineering*. Citeseer, 2002.

[20] V. Sachdeva and L. Chung, "Handling non-functional requirements for big data and iot projects in scrum," in *2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence*. IEEE, 2017, pp. 216–221.

[21] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. Zelkowitz, "Empirical findings in agile methods," in *Conference on extreme programming and agile methods*. Springer, 2002, pp. 197–207.

[22] A. Alnatheer, A. M. Gravell, D. Argles, and L. Gilbert, "Agile security methods: an empirical investigation," in *Proceedings of the IASTED International Conference, Software Engineering. DOI*, vol. 10, 2013, pp. 2014–810.

[23] T. N. Graham, R. Kazman, and C. Walmsley, "Agility and experimentation: Practical techniques for resolving architectural tradeoffs," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 519–528.

[24] L. Williams, A. Meneely, and G. Shipley, "Protection poker: The new software security game," *IEEE Security and Privacy*, vol. 8, no. 3, pp. 14–20, 2010.

[25] M. Younas, D. Jawawi, I. Ghani, and R. Kazmi, "Non-functional requirements elicitation guideline for agile methods," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 9, no. 3-4, pp. 137–142, 2017.

[26] I. A. Tøndel, M. G. Jaatun, D. S. Cruzes, and L. Williams, "Collaborative security risk estimation in agile software development," *Information & Computer Security*, 2019.

[27] EN50126-1:2017, "The Specification and Demonstration of Reliability, Availability, Maintainability and Safety (RAMS) - Part 1: Generic RAMS process," 2017.

[28] SAE, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2018.

[29] ISO26262, "Road Vehicles – Functional Safety," 2011.