How Can Quality Awareness Support Rapid Software Development? – A Research Preview

Liliana Guzmán[1], Marc Oriol[2], Pilar Rodríguez[3], Xavier Franch[2], Andreas Jedlitschka[1], Markku Oivo[3]

[1] Fraunhofer IESE, Kaiserslautern, Germany
{liliana.guzman, andreas.jedlitschka}@iese.fraunhofer.de
[2] Universitat Politècnica de Catalunya, Barcelona, Spain
{moriol, franch}@essi.upc.edu
[3] University of Oulu, Oulu, Finland
{pilar.rodriguez, markku.oivo}@oulu.fi

Abstract. [Context and Motivation] Rapid software development (RSD) refers to the organizational capability to develop, release, and learn from software in rapid cycles without compromising its quality. To achieve RSD, it is essential to understand and manage software quality along the software lifecycle. [**Question/Problem]** Despite the numerous information sources related to product quality, there is a lack of mechanisms for supporting continuous quality management throughout the whole RSD process. **[Principal ideas/Results]** We propose Q-Rapids, a data-driven, quality-aware RSD framework in which quality and functional requirements are managed together. Quality requirements are incrementally elicited and refined based on data gathered at both development time and runtime. Project, development, and runtime data is aggregated into quality-related indicators to support decision makers in steering future development cycles. **[Contributions]** Q-Rapids aims to increase software quality through continuous data gathering and analysis, as well as continuous management of quality requirements.

Keywords: software quality, quality requirements, rapid software development

1 Introduction

Agile software development (ASD) enables organizations to adapt to business dynamics by facilitating more flexible development through iterative methods that rely on extensive collaboration. ASD is prevalent in the software industry [1]. A recent evolutionary step from ASD is rapid and continuous software engineering, which refers to the organizational capability to develop, release, and learn from software in rapid cycles [2]. This capability is known as Rapid Software Development (RSD) [3]. RSD combines methods, practices and principles from ASD, lean software development, and continuous deployment to minimize time-to-market through reduced release cycles [4].

In RSD, faster and more frequent release cycles should not compromise software quality. Thus, understanding and managing software quality is essential to ensure that

new releases will lead to progressive improvement. But despite the numerous sources of information related to product quality that RSD provides (e.g., usage data), there is a lack of methods to support continuous quality management throughout the whole RSD process [4]. Recent empirical studies found the deficient management of quality requirements (QRs) [5] to be the main reason for rework in RSD [1].

In this research preview, we summarize the state of the art and challenges related to continuously managing QRs along the RSD process (Section 2). We also introduce Q-Rapids as a data-driven, quality-aware RSD framework that jointly manages QRs and functional requirements (FRs) throughout the RSD process (Section 3). Then we outline the strategy selected to develop and evaluate Q-Rapids (Section 4). Finally, we summarize our contributions and discuss future research (Section 5).

2 Challenges in Managing Quality Requirements

Quality Requirements and their Management. QRs are a subset of non-functional requirements that state conditions on "characteristics that make the product attractive, usable, fast or reliable" [6]. An example of a QR is: "The system should provide an availability of at least 98% for given time period". Thus, optimal management of software quality demands proper consideration of QRs in the software lifecycle. However, QRs have not received the same degree of attention as FRs [6]. Neglecting QRs is one of the top ten risks of requirements engineering [7], and errors in considering QRs are the most expensive and difficult to correct [8].

Another problem is the elicitation and specification of QRs. Modern approaches to elicit QRs rely on explicit user feedback [9]. However, explicit feedback may be incomplete, biased, or ambiguous. Implicit feedback (usage data) is a promising alternative to elicit QRs [10]. For example, QRs related to performance can be elicited by identifying how many users leave the system after waiting two seconds for a system response. So, QRs regarding response time can be specified by analyzing together the system and user behavior. Current approaches neither derive QRs automatically nor combine usage data with other data sources (e.g., software repositories). Furthermore, whereas FRs have clear-cut satisfaction criteria, QRs are initially elicited as "soft goals" [8] and need to be elaborated into measurable conditions. Finally, current tools for managing QRs do not manage them throughout the entire software lifecycle [11].

Thus, there is a need for (1) data-driven QR elicitation and specification; and (2) data-driven understanding of the strategic impact of QRs on management and business.

Quality Requirements in Rapid Software Development. Current RSD approaches are mostly driven by FRs. For example, in Scrum [12] requirements are specified as user stories stored in the product backlog. User stories are prioritized in each development cycle from a customer value perspective. Although QRs are usually included as acceptance criteria for user stories [13], mainly focusing on customer value when prioritizing requirements is problematic because other important factors (e.g. security, performance, and scalability) tend to be underestimated [1]. More recently, mechanisms such as automation, integration of R&D with operations and maintenance teams (also

referred to as DevOps), and post-deployment customer-data monitoring [4] have been introduced to further ensure quality and quick delivery in RSD. However, using mechanisms such as post-deployment data is not free from challenges. The growing size of data is a challenge, and systematic approaches for collecting, analyzing, and integrating data into the product development process are missing [14]. Moreover, there is a lack of methods and tools for integrating, analyzing, and visualizing collected data to make QRs transparent and support real-time decision-making on QRs [15], and to jointly manage QRs and FRs along the RSD process [4].

We conclude that there is a need for (1) seamless integration of QRs and FRs; (2) methods and tools for real-time monitoring of QRs; and (3) flexible, iterative, and dynamic generation and management of QRs in RSD.

Data-driven Quality Decision Making. As systems scale and their complexity increases, data generated and used during the software lifecycle is becoming increasingly important. Source code repositories, bug reports, and runtime logs contain a lot of hidden information about software quality. Applying analytics to extract this information can help decision makers to identify and monitor quality issues and to steer development activities in order to improve the overall quality in RSD.

However, integrating software analytics research results into tools established in practice is still challenging [17]. Regarding the analysis of historical data, Mining Software Repositories is an important research area to uncover information about software systems. An overview is given in [16]. Still, there are only a few reports (e.g., [23]) on the practical impact of data mining analytics on the development process. Regarding the analysis of runtime data, several approaches exist. For example, profiling and automated instrumentation techniques are usually used to study the runtime behavior of software systems [18]. Such techniques impose high overhead and slow down the execution. They also lead to a large volume of results that are impractical to interpret. Finally, analyzing heterogeneous and time-evolving streaming data can get very complex and lead to poor performance of analysis techniques. MapReduce [19], the Lambda Architecture [20], and modern analytics like Spark can help to address this problem.

We found there is a need for (1) in-time, scalable, and efficient QR-driven data analysis to support decision making; and (2) scalable and efficient gathering and monitoring of heterogeneous data at development time and runtime.

3 The Q-Rapids Framework

Based on the needs identified in Section 2, we conclude that the software industry needs methods and tools for handling software quality in the RSD context. To achieve this, we propose the Q-Rapids framework (cf. Figure 1). Q-Rapids relies on a generic data-driven, quality-aware, rapid development process characterized by integrated management of QRs and FRs. Q-Rapids aims to: (1) improve the software products' quality with an effective and seamless data gathering and analysis; (2) increase the productivity of the software lifecycle with a seamless integration of QRs into the development pro-

cess: and (3) reduce the time to market of software products by making optimal decisions based on strong evidence and solid experience-based decision making models. Q-Rapids also aims to be a generic and suitable for managing different types of QRs in different application domains and project settings. To attain these goals, Q-Rapids will be developed in collaboration with four European companies in the domains of health, defense, crisis management, and telecommunication. These companies develop software products with different QRs and FRs in different project settings.

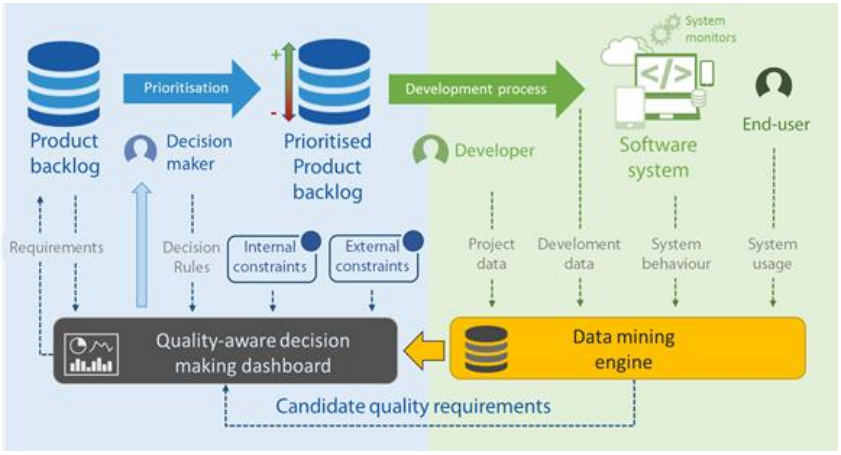

Fig. 1. The Q-Rapids framework.

Effective and seamless data gathering and analysis techniques. Q-Rapids will systematically and continuously track software quality based on quality-related indicators. It will combine different types of data sources to gather relevant indicators: project management tools, software repositories, and runtime data about quality of service and system usage by end users. This information will be selectively collected and pre-processed, and analyzed to support different decisions makers (e.g., product owners, developers, and testers). Q-Rapids will propose quality-critical indicators on the basis of a product-specific quality model based on the QUAMOCO approach [21]. Such approach will enable Q-Rapids to define a generic set of QRs as well as quality-related indicators that can be tailored according to the application domain, product characteristics, and project context. Examples of QRs that we plan to address in the Q-Rapids project include performance, reliability and usability.

Data gathering will be seamlessly integrated into the software lifecycle and later system usage. Q-Rapids will integrate different data collection instruments. For example, we plan to gather project data through a monitor of the project management, development data through a monitor of GIT, data on the system behavior through QoS monitors, and usage data through an event-tracking monitor. Deployment should be as easy as providing the URLs or directories for the software project repositories and a specification of the quality attributes that are of interest for the particular project in order to deploy only the needed monitors. Monitoring instruments will be deployed in different

contexts considering the lifecycle phase in which they apply [22] and the architecture type (e.g., service-based architectures or cloud deployments).

Through the application of data analytics, elicited data will be analyzed, making it possible to support the comprehension of quality issues that will steer subsequent development activities, thus improving the overall software quality in a timely manner. An essential part of the analysis will be to find correlations. For instance, the correlation analysis between bug rate and QR types may help to understand which QR types are more error-prone and require more effort allocation when planning releases.

Quality-aware rapid software development process. Q-Rapids will extend the RSD process with the comprehensive integration of QRs and FRs. Thus, we will focus on rapid practices such as product backlogs, release planning, and sprint planning but with seamless integration of both FRs and QRs. Our goal is to define a generic software development process based on the principles of RSD and, therefore, being lightweight, flexible, and adaptable to market fluctuations and customer changes, still properly considering the management of QRs in a way that rapid releases do not have negative repercussions on software quality. In particular, a quality-aware RSD process will be defined including existing practices, tools, and methods to be used in rapid development cycles and complex scenarios such as Scrum, Kanban and DevOps. The process will be based on key characteristics of agile methods and RSD, including the management of FRs and QRs using a holistic management of product backlogs, continuous integration, and short release cycles [1, 4, 13]. The quality-aware RSD process will provide the means needed to elicit, derive, and manage QRs in rapid cycles by addressing questions like "how should QRs be processed in RSD so that the result will be high-quality products?" Q-Rapids will focus on success factors for software companies and help managers to balance such issues as time to market and product quality. It will also consider business-related constraints and domain-specific requirements and regulations so that the framework can easily be tailored to different company's needs.

Q-Rapids will provide a novel rapid requirements engineering approach that will elicit QRs using a data-driven approach, followed by the implementation and assessment of QRs in rapid cycles. It will provide a generic quality-aware RSD process that can be customized based on the software company setting and their quality demands.

Quality-aware decision making dashboard. Q-Rapids will extend current tools for measuring and analyzing software quality (e.g., SonarQube™) by providing decisions makers with a highly informative dashboard to help them make data-driven strategic decisions related to QRs in rapid cycles. The Q-Rapids dashboard will aggregate the collected data into key strategic indicators related to, e.g., time to market, development costs, and overall quality. It will also comprise the product and iteration backlogs that contain the project requirements. Thus, the dashboard will help decision makers to analyze, e.g., the impact on time to market of selecting, leaving out, or discarding a QR. In addition, the dashboard will allow defining project-specific decision rules (e.g., how to handle conflicts between time and quality levels) as well as external and internal constraints. External constraints are conditions beyond the control of decision makers,

e.g., a fixed budget. Internal constraints are development and organizational conditions influencing decision making, e.g., a maximum number of tasks per developer per week.

The Q-Rapids dashboard will provide models and advanced capabilities to (1) analyze and evaluate alternative solutions to current QR management decisions; (2) predict and analyze the impact of violations related to key strategic indicators; and (3) suggest mitigation actions when violations are identified. The underlying rationale of previous analyses will be transparent to decision makers.

4 Development and Evaluation of the Q-Rapids Framework

Q-Rapids will be developed as part of the H2020 European project Q-Rapids following an iterative and incremental approach applying RSD principles. Its development will be driven by four use cases defined in collaboration with four European companies. The use cases were chosen to show the generalizability and suitability of Q-Rapids. They will serve as basis for understanding how QRs are managed as well as for evaluating Q-Rapids. The use cases cover managing QRs of single and multiple product lines in application domains such as health, defense, crisis management and telecommunication. The use cases also vary regarding the RSD approaches been used. The Q-Rapids development approach includes four phases: (1) requirements elicitation, (2) proof-of-concept, (3) consolidated approach, and (4) final solution.

To assess the impact of Q-Rapids, we plan a formative and summative evaluation. The outcomes of the proof-of-concept phase will be evaluated by themselves in small-scale, lab-like environments, with the goal of obtaining information on their general functionality (formative evaluation). Then, the (ready-to-integrate) intermediate components and the final framework will be evaluated in real context. Q-Rapids, as a whole, will be integrated into the industrial partners' development environments and evaluated regarding to predefined criteria (summative evaluation). Evaluation criteria will be derived from the Q-Rapids goals and selected use cases.

5 Summary

In this paper, we identified the challenges that need to be overcome to support decision makers in managing QRs throughout the whole RSD process. As a response to these challenges, we introduced the Q-Rapids framework, developed as part of the H2020 European project Q-Rapids. This project will follow an iterative and incremental approach applying RSD principles itself. A full 3-year validation plan has been designed, including a formative and summative evaluation involving four European companies, which will provide real projects that will allow scaling initial small-scale results produced in lab-like environments to ready-to-transfer solutions.

Acknowledgements. This work is a result of the Q-Rapids project, which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement N° 732253.

References

1. Inayat, I., Salim, S.S., Marczak, S., Daneva, M., Shamshirband, S.A.: Systematic Literature Review on Agile Requirements Engineering Practices and Challenges. Computers in Human Behavior, 51(B), pp. 915-929 (2014)
2. Fitzgerald, B., Stol, K.J.: Continuous Software Engineering: A Roadmap and Agenda. Journal of Systems and Software, 123, pp. 176-189 (2017)
3. Mäntylä, M.V., Adams, B., Khomh, F., Engström, E., Petersen, K.: On Rapid Releases and Software Testing: A Case Study and a Semi-Systematic Literature Review. Empirical Software Engineering, 25(2), pp. 1384-1425 (2015)
4. Rodríguez, P., Haghighatkhah, et al.: Continuous Deployment of Software Intensive Products and Services: A Systematic Mapping Study. Journal of Systems and Software, 123, pp. 263-291 (2017)
5. Ramesh, B., Baskerville, R., Cao, L.: Agile Requirements Engineering Practices and Challenges: An Empirical Study. Information Systems Journal, 20(5), pp. 449–480 (2010)
6. Wagner, S.: Software Product Quality Control. Springer (2013)
7. Lawrence, B., Wiegers, K., Ebert, C.: The Top Ten Risks of Requirements Engineering. IEEE Software 18(6), pp. 62-63 (2001)
8. Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering, (Vol. 5). Springer Science & Business Media (2000)
9. Dalpiaz, F., Korenko, M. Salay, R., Chechik, M.: Using the Crowds to Satisfy Unbounded Requirements. CrowdRE 2015, pp. 19-24. (2015)
10. Maalej, M., Nayebi, M., Johann, T., Ruhe, G.: Toward Data-Driven Requirements Engineering. IEEE Software 33(1), pp. 48-54 (2016)
11. Caracciolo, A., Lungu, L.F., Nierstrasz, O.: How Do Software Architects Specify and Validate Quality Requirements? In ECSA 2014, pp. 374-389 (2014)
12. Schwaber, K.: Agile project management with Scrum. Microsoft Press. (2004)
13. Leffingwell, D.: Agile software requirements: Lean requirements practices for teams, programs, and the enterprise. Addison-Wesley Professional (2010)
14. Sauvola, T., Lwakatare, et al.: Towards Customer-centric Software Development: A Multiple-case Study. In Proceedings of Euromicro Conference on SEAA, pp. 9-17 (2015)
15. Yaman, S.G., Sauvola, T., et al.: Customer Involvement in Continuous Deployment: A Systematic Literature Review. In: Proceedings of REFSQ 2016, pp. 249-265 (2016)
16. Kwan, I., Damian, D.: A Survey of Techniques in Software Repository Mining. In: Technical Report DCS-340-IR, University of Victoria. (2011)
17. Zhang, D.: Software Analytics in Practice – Approaches and Experiences. Microsoft Research. In: Keynote PROMISE (2015)
18. Thomas, S.W., Hassan, A.E., Blostein, D.: Mining Unstructured Software Repositories. In: Evolving Software Systems, pp. 139-162 (2014).
19. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Communications of the ACM, 51(1), pp. 107-113. (2008).
20. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Real Time Data Systems. In: Manning Publications Co. (2015)
21. Wagner, S., Goeb, A., et al.: Operationalised product quality models and assessment: The Quamoco approach. Information and Software Technology, 62, pp. 101-123 (2015)
22. Oriol, M., Franch, X., Marco, J.: Monitoring the Service-Based System Lifecycle with SALMon. Expert Systems with Applications 42(19), pp. 6507-6521 (2015)
23. Shihab, E., Hassan, A. E., Adams, B., Jiang, Z.M.: An Industrial Study on the Risk of Software Changes. In: Proceedings of the Symposium on the FSE, article 62 (2012).