Data-driven Requirements Engineering in Agile Projects: The Q-Rapids Approach

Xavier Franch[1], Claudia Ayala[1], Lidia López[1], Silverio Martínez-Fernández[2], Pilar Rodríguez[3], Cristina Gómez[1], Andreas Jedlitschka[2], Markku Oivo[3], Jari Partanen[4], Timo Räty[4], Veikko Rytivaara[4]

[1]Universitat Politècnica de Catalunya (UPC)
Barcelona, Spain
{franch, llopez, cayala, cristina}@essi.upc.edu

[2]Fraunhofer IESE,
Kaiserslautern, Germany
{silverio.martinez, andreas.jedlitschka}@iese.fraunhofer.de

[3]University of Oulu
Oulu, Finland
{pilar.rodriguez, markku.oivo}@oulu.fi

[4]Bittium Wireless Ltd.
Oulu, Finland
{jari.partanen, timo.raty, veikko.rytivaara}@bittium.com

***Abstract*—Requirements identification, specification and management are key activities in the software development process. In the last years, many approaches to these activities have emerged, based on the exploitation of huge amounts of data gathered from software repositories and system usage. The Q-Rapids project proposes the collection and analysis of such data and its consolidation into a set of strategic indicators as product quality, time to market and team productivity. These indicators are visualized through a dashboard designed to support decision-makers. In this paper, we present the ongoing research undertaken in this project. We use the concept of blocking situation to exemplify the Q-Rapids approach.**

***Index Terms*—Data-driven software engineering; Agile software development; Requirements engineering; Strategic indicator; Waiting time.**

I. Motivation

Following lean and agile principles, just-in-time requirements analysis (JITRA) proposes that requirements should only be identified as needed and specified at the level of detail required for upcoming development[1]. While easy to state, JITRA principles are not as easy to apply in a particular project. In general, requirement engineers need to rely in their experience and mindset in order to identify requirements and decide the appropriate level of detail in their description.

In the last years, the requirements engineering community is witnessing the emergence of research approaches based on the exploitation of huge amounts of data gathered from software repositories and system usage [1]. These approaches tackle research questions such as identifying candidate features [2], predicting productivity [3] and planning releases [4].

The goal of this work is to present a summary on the ongoing research in the Q-Rapids (Quality-aware Rapid Software Development) project[2]. Q-Rapids is framed in this data-driven requirements engineering movement. The project aims at identifying candidate requirements from data, analyzing their impact on several selected strategic indicators and based on the results, deciding the most adequate action with this requirement: adding to the product or sprint backlog, postponing, discarding, or further refining in order to make the final decision. As the "Q" in the acronym hints, the emphasis of the project is on quality, meaning to make decisions always considering quality of either the final product or the development process, typically (but not only) by identifying appropriate quality requirements.

The rest of the paper is structured as follows. Section II briefly summarizes the idea of the Q-Rapids project. Section III presents a particular indicator that will be used to explain in depth the concept behind Q-Rapids. Sections IV to VI elaborate in detail the three main components of the approach using the example indicator. To finalize the paper, Section VII outlines a research agenda.

II. The Q-Rapids Approach

Nowadays, software quality is an essential key factor for the success of developed software. However, current software development methodologies still provide a limited support to ensure that adequate levels of quality are met, while complying with rapid development cycles.

Q-Rapids is a data-driven, quality-aware rapid software development approach in which quality and functional requirements are identified from available data and evaluated with respect to some selected indicators [5]. Q-Rapids aims to increase software quality through:

- Gathering and analyzing data from project management tools, software repositories, quality of service and system usage. The analysis of those data permits to systematically and continuously assess software quality using a set of quality-related indicators based on GQM+Strategies™ [6] Quamoco [7] and GQM [8].
- Providing decision makers with a highly informative dashboard to help them making data-driven, requirements-related strategic decisions in rapid cycles. The dashboard will aggregate the collected data into strategic indicators related to factors as time to market, team productivity, customer satisfaction, and overall quality.

[1] https://rthewitt.com/tag/requirements/
[2] www.q-rapids.eu

- Extending the agile software development process considering the comprehensive integration of quality and functional requirements and their management in a way that favors software quality and that brings a significant productivity increase to the software lifecycle.

III. An Example: The Problem of Blocking Situations

In this paper, we exemplify the Q-Rapids approach to manage requirements with one particular situation arising in software development in general, and agile projects in particular, namely the emergence of blocking situations while developing a feature or user story.

Blocking situations increase the waiting time, which is against the lean principle of "deliver fast" [9]. Waiting time can be defined as the cost of waiting for a previous upstream step to finish. It was one of the seven manufacturing waste types characterized in the Toyota Production System [10].

Several studies have identified the causes behind blocking situations and delays in projects. For instance, McConnell and Goldratt identified the top 11 cause of delays in IT projects[3]. In agile development, Sedano et al. identified a series of causes for waiting in agile development [11]. They mentioned: slow or unreliable tests; unreliable acceptance environment; missing information, people or equipment; and context switching from delayed feedback. This last reason has been reported as a cause of hidden waste [12] and productivity decrease [13].

The identification of blocking situations can be used to evaluate software quality and identify quality requirements. We will show several factors causing blocking situations. For instance, one of the causes related to requirements is the occurrence of reiterated failing tests of a feature. Among others, tests can fail for the following reasons: because the requirements were identified and included too early in the backlog, or because the level of detail of the requirement can be inappropriate for the current knowledge (e.g., a quality requirement with an unrealistic threshold required). We will show how such a blocking situation can be detected and ameliorated through the collection, aggregation and analysis of quantitative data gathered from software and project management repositories, and the possible actions to be taken in the process in order to mitigate the impact of waiting time.

IV. Data Gathering and Analysis

During meetings with the industry partners of the Q-Rapids project, we identified five factors being useful to identify blocking situations: feature definition completeness, delayed tasks, test failing, test performance, and low quality features. Table I shows metrics for these five factors, the data to gather for computing them, and their corresponding data sources. By measuring these five factors, we can have a quantitative vision of the blocking indicator, and therefore analyze whether quality requirements related actions should be suggested through the Q-Rapids dashboard. Next, we respectively explain the rationale of these factors.

[3] https://www.projectmanagement-training.net/appendix-1-causes-of-delays-in-it-projects

First, *feature definition completeness* refers to the state in which final information of a feature is included in the backlog, and hence it is ready to be developed. This factor enables to identify incomplete features, unrealistic requirements, and the time since someone knows a feature is needed until it is completely reported.

Second, *delayed tasks* refers to tasks blocking others. This factor enables to measure the total waiting time required to finish and close opened tasks blocking other tasks. It is important to note that the impact of these delays is greater when blocking either high priority tasks or many tasks. An example of alert in the dashboard is when a feature is delayed and other features depend on it. For this factor, we need to gather the tasks (a.k.a. features or user stories) from the issue tracking systems representing the product backlog (e.g., Redmine, GitLab, JIRA, Mantis), as well as the dependency information about these tasks during feature implementation.

Third, *test failing* refers to problems at testing of blocking features, as well as the quality of these tests (e.g., test coverage, independent tests, and test omitted). If a feature is not properly tested, it may block the deployment of depending features. The blocking impact depends on feature priority, number of depending features and their priority, and test quality. We need to gather the tests about different depending modules from continuous integration tools (e.g., Jenkins). Then, we can identify modules that are not ready for integration, causing delays in the deployment (i.e., the number of modules with failed tests blocking other modules deployment).

Fourth, *test performance* refers to the time consumed for the execution of tests (automated or manually). If the time is too long, it can cause delays.

Fifth, *low quality feature* refers to a feature already developed and tested, but having incurred technical debt. These features do not comply with code quality rules regarding maintainability, reliability, and security for static code analysis tool (e.g., SonarQube). If they have depending features, they can cause delays when maintainability actions are taken.

After data gathering and metrics calculation, data analysis approaches should prioritize the features or user stories that require urgent implementation to avoid waiting time, and identify the modules blocking continuous integration of other ready modules. The next section reports how such analysis is reported in the dashboard.

V. The Strategic Dashboard

The strategic dashboard is the component of the Q-Rapids approach that will interact with the decision-maker (typically, a requirements engineer, business analyst or product owner).

In general, the design principles of the dashboard are as follows:

- We aim at aggregating the factors into a single general value which provides a measure of the indicator under analysis. We plan to use Bayesian networks built as a combination of real data and experts' opinion in order to compute the value of such indicator. In the example given in this paper, we can define alerts when the blocking situation is reaching some thresholds.

TABLE I. CRITICAL FACTORS AFFECTING BLOCKING, TOGETHER WITH THEIR CORRESPONDING METRICS AND DATA SOURCES.

Factor affecting Blocking	*Metric*	*Data source and information to be gathered*
Feature definition completeness	- Number of features incomplete in the product backlog - Average time to complete feature definition	Features from the issue tracking system (e.g., JIRA, Redmine, GitLab) containing the product backlog. For each feature, the following fields should be gathered: its name, the duple its status (after changes) and corresponding timestamp, its type (e.g., development or tests), its estimated time, its real invested time, the assigned developer, its dependencies with other features (e.g., parent tasks), the progress reported, definition of done, linkage to main feature or sub/feature / task, and empty fields.
Delayed tasks	- Number of blocked tasks - Number of blocking tasks - Waiting time to finish blocking tasks (per tasks priority)	
Test failing	- Number of tests failed - Test coverage - Number of omitted/non-run tests	Tests from the continuous integration tool (e.g., Jenkins). For each test, the information to gather is: the duple of the result and the corresponding timestamp, test coverage, status of the test (i.e., active or skipped), and its execution time.
Test performance	- Time to execute tests - Prediction estimate of development until next release based on test status	
Low quality features	- Time to solve quality rule violations of the feature	From the static code analysis tool (e.g., SonarQube), the following information should be extracted: dependencies among tasks (e.g., fan-in/out to identify dependencies), and violations of quality rules, code complexity.

- The dashboard will provide drill-down capabilities making possible to visualize the behavior of an object under measure (feature, user story, component, ...) with a higher level of detail. This will allow to visualize, for instance, that the reason of an alert related to a blocking feature is more related to test failing than to definition completeness.
- The dashboard will integrate meaningful prediction rules in order to detect potential violations to the defined thresholds. For instance, in case there is a blocking situation involving higher priority tasks with several critical dependences, then the dashboard will raise the corresponding alarm.
- In order to allow the exploration of diverse blocking alternatives in the solution space, we plan to include in the dashboard what-if analysis techniques to visualize the potential impact of each alternative. For instance, the consequence of postponing a blocking test. Of course, this analysis shall reflect the consequences in all indicators, which can be in conflict and thus trade-offs will be explored.
- The dashboard will also suggest possible mitigation activities to improve some of the factors impacting on the strategic indicator. For example, if we have some testing problems in a blocking feature (high priority or with a high number of features depending on it), we can stop the development of lower-level features to use the resources to support the development team working on it.
- All in all, the dashboard will support decision makers to detect and visualize meaningful situations as well as to explore the impact of diverse solutions in order to take informed decisions.

Due to the intrinsic characteristic of a dashboard, the visualization is of utterly importance. For instance, a radar visualisation approach can be used by business high-level roles to have a more generic visualisation. On the other hand, a spider visualisation can be used by product owners, who can be more interested in the status of the separate factors generating a blocking situation. In the spider chart, where each vertex corresponds to one of the factors included in Table 1, the user can see that the possible blocking situation is related to the *Test failing* factor at the first sight.

For some factors, the users can need not only the status at some point in time, but analyzing trends. This is the case of the *Test failing* factor. Using the measures shown in Table 1, *Test failing* is characterized by the percentages of missing tests (*Test coverage*), failed tests (*Number of tests failed*) and non-run tests (*Number of omitted/non-run tests*). Having a high number of non-run tests at the beginning of an iteration is not a blocking situation. But, after an initial period of time, this situation can be considered blocking.

VI. THE PROCESS

One key element of the Q-Rapids process is flow. Flow is essential in agile and rapid software development because it allows a constant feature delivery pace [9][14]. The Q-Rapids process will focus on a continuous end-to-end flow of features by means of transparent development and automatic identification of blocking situations (i.e., elimination and reduction of waiting times). Automation is essential in this process as the goal is to eliminate as many manual steps as possible to identify blocking situations [15]. Moreover, the Q-Rapids strategic dashboard plays a fundamental role in this process, as it will visualize blocking situations as they emerge and make suggestions to decision makers on the different ways of solving the quality issues.

As an illustrative example, we develop in this section how the Q-Rapids process will support an optimal management of features by a real-time identification and understanding of 'blocking' features (i.e. feature which implementation is preventing flow). The information provides through the dashboard, and based on the five critical factors affecting blocking (see Table I), will serve as an input for managing product back-

logs. The strategic dashboard will inform decision makers at different organizational levels (e.g. business owners, product owners, etc.) about situations in which a certain feature is preventing flow because it is not being implemented on time (or it has not been selected for implementation yet). Thus, decision makers can use this information to include/drop items in product backlogs during decision-making meetings, reprioritize backlogs and, in the worst-case scenario, stop-the-line if needed (i.e. focusing exclusively on solving the blocking situation). Similarly, the strategic dashboard will visualize waiting times in decision-making when, for example, a feature that should be implemented is queuing because it misses certain information that is needed for its implementation (feature definition completeness).

The way to handle blocking situations caused by blocking features will depend on the concrete organizational structure and agile software development method being applied. Companies applying Kanban will be in the best position to benefit the best from Q-Rapids solutions to support flow. Blocking situations will be identified at real-time, allowing fast identification of waiting times and bottlenecks that can be used as an input by the Kanban team to update its Kanban board. A typical situation in Scrum teams would be that the Product Owner uses the Q-Rapids strategic dashboard as an input for prioritizing the product backlog and guiding discussions during sprint planning meetings. Development teams will particularly benefit from the Q-Rapids dashboard as blocking situations due to quality issues, such as neglecting internally generated backlog items (e.g. quality requirements), will be transparent for everyone. Such increasing in transparency will help solve natural tensions between the desire to deliver functionalities quickly and the need for reliable products. Consequently, decision makers can, then, decide upon different strategies from solving the blocking situation, from stopping-the-line, if the blocking situation is really critical, to reprioritizing existing backlog items, or adding new features.

VII. Research Agenda

The ongoing research work in Q-Rapids project shaped a research agenda including topics as:

- Automatic identification of blocking situations preventing flow and threating product quality. As illustrated in this paper, development of techniques to make development flow transparent is essential to enable quick and easy identification of blocking situations.
- Development of practices for seamless integration of quality requirements in agile product backlogs. Techniques that allow practitioners to ensure that focus on customer when prioritizing backlog items does not compromise quality levels need further research.
- Enhancement of agile and rapid software development processes by incorporating technical infrastructure for supporting continuous quality monitoring. The aim is to provide a real-time understanding on quality so to react as quickly as possible upon identified quality challenges.
- Identification and definition of strategic indicators that provide information related to quality in real-time. A versatile dashboard presenting these indicators as proposed in the paper should be a key asset in this approach. Such dashboard should not be invasive to agile teams, on the contrary it needs to be seamless integrated with their current repositories and assessment tools.
- Integration and measurement of quality in agile and rapid processes during development and at runtime. We believe that it is possible to create a quality model in rapid software development, consolidating the usually available data and the quality issues to be solved, which could be the starting point for agile organizations adopting the Q-Rapids vision.

Acknowledgment

This work is a result of the Q-Rapids project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732253.

References

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

[2] C. Iacob, R. Harrison, "Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews". MSR 2013.

[3] M. Choetkiertikul, H. Khanh Dam, T. Tran, A. Ghose, J. Grundy. "Predicting Delivery Capability in Iterative Software Development". *Transactions on Software Engineering*, online.

[4] M. Nayebi, G. Ruhe, "Analytical Product Release Planning". In *The Art and Science of Analyzing Software Data*, Morgan Kaufmann, 2015.

[5] L. Guzmán, M. Oriol, P. Rodríguez, X. Franch, A. Jedlitschka, M. Oivo. "How Can Quality Awareness Support Rapid Software Development? - A Research Preview". REFSQ 2017.

[6] V. Basili et al. *Aligning Organizations through Measurement - The GQM+Strategies Approach*. Springer, 2014.

[7] V. Basili et al. *The Goal Question Metric Approach. Encyclopedia of Software Engineering*. Wiley, 1994.

[8] S. Wagner et al. "Operationalised Product Quality Models and Assessment: The Quamoco Approach". *Information and Software Technology* 62, 2015.

[9] M. Poppendieck, T. Poppendieck. *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley, 2006.

[10] S. Shingo, A.P. Dillon. *A Study of the Toyota Production System from an Industrial Engineering Viewpoint*. CRC Press, 1989.

[11] T. Sedano, P. Ralph, C. Péraire. "Software Development Waste". ICSE 2017.

[12] T. Ohno. *Toyota Production System: Beyond Large-scale Production*. Productivity Press, 1988

[13] S. Monsell. "Task Switching". *Trends in Cognitive Sciences* 7(3), 2003.

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

[15] V. Mandić, M. Oivo, P. Rodríguez, P. Kuvaja, H. Kaikkonen, B. Turhan. "What is flowing in Lean Software Development?". In *Lean Enterprise Software and Systems*, 2010.