This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732253.

D1.1 Data gathering and analysis specification

V2.0

Programme	**H2020**		
Funding scheme	RIA - Research and Innovation Action		
Topics	ICT-10-2016 - Software Technologies		
Project number	732253		
Project name	Quality-aware rapid software development		
Project duration	November 2016 – October 2019		
Project website	www.q-rapids.eu		
Project WP	WP1 – Data Gathering and Analysis		
Project Task	Task 1.1 – Specification of data gathering functionality Task 1.3 – Elaboration of data analysis requirements		
Deliverable ID & name	D1.1 Data gathering and analysis specification		
Deliverable type	X	R	Document, report
		DEM	Demonstrator, pilot, prototype
		DEC	Websites, patent fillings, videos, etc.
		OTHER	Material that does not belong to any specified category
		ETHICS	Ethics requirement
Deliverable version	V2.0		
Contractual delivery	V1.0: 30/04/2017; V2.0: 30/04/2018;		
Delivered	V1.0: 28/04/2017; V2.0: 27/04/2018;		
Responsible beneficiary	Organisation		
Dissemination level	X	PU	Public
		CO	Confidential, only for members of the consortium (including the Commission Services)
		EU-RES	Classified Information: RESTREINT UE (Commission Decision 2005/444/EC)
		EU-CON	Classified Information: CONFIDENTIEL UE (Commission Decision 2005/444/EC)
		EU-SEC	Classified Information: SECRET UE (Commission Decision 2005/444/EC)

Version history			
Version no.	Date	Description	Author
V0.1	31.03.17	Initial table of contents and structure of the document. Description of the research methodology. Q-Rapids "as-is situation". Initial Q-Rapids user stories. Initial Q-Rapids data gathering and analysis architecture specification.	Silverio Martínez, Andreas Jedlitschka, Henning Barthel (Fraunhofer IESE)
V0.2	12.04.17	Updates considering the reviewers' feedback about the outline. User stories for data gathering, generic Q-Rapids quality model, and user stories for data analysis (i.e., "to-be scenario"). Updated list of data sources in the architecture specification.	Silverio Martínez (Fraunhofer IESE)
V0.3	20.04.17	Updates considering first reviewer's (Lidia Lopez) comments.	Silverio Martínez (Fraunhofer IESE)
V0.4	26.04.17	Updates considering second reviewer's (Rafal Kozik) comments. Review of Sections 1, 2 and 5. Addition of annexes.	Silverio Martínez, Henning Barthel, Axel Wickenkamp (Fraunhofer IESE)
V0.5	27.04.17	Review of Sections 3 and 4. Final complete review before proofreading.	Silverio Martínez (Fraunhofer IESE)
V0.6	28.04.17	Proofreading.	Sonnhild Namingha (Fraunhofer IESE)
V1.0	28.04.17	Final version.	Silverio Martínez (Fraunhofer IESE)
V1.1	23.03.18	Initial update changes and new structure (Section 2).	Silverio Martínez (Fraunhofer IESE)
V1.2	18.04.18	Updating the document structure based on feedback from Michal Choras, Prabhat Ram, and reviewers.	Silverio Martínez (Fraunhofer IESE)
V1.3	25.04.18	Incorporation on new subsections explained in Section 2.	Silverio Martínez (Fraunhofer IESE), Rafal Kozik (ITTI), Prabhat Ram, Pilar Rodriguez (UOULU)
V1.4	26.04.18	Pre-final version	Silverio Martínez, Andreas Jedlitschka (Fraunhofer IESE)
V2.0	27.04.18	Final version based on feedback from reviewers.	Silverio Martínez (Fraunhofer IESE)

Authors	
Organisation	Name
Fraunhofer IESE	Silverio Martínez-Fernández
Fraunhofer IESE	Andreas Jedlitschka
Fraunhofer IESE	Henning Barthel
Fraunhofer IESE	Axel Wickenkamp
ITTI	Rafal Kozik
ITTI	Michal Choras
UOULU	Prabhat Ram
UOULU	Pilar Rodriguez
Reviewers	
UPC	Lidia Lopez
ITTI	Rafal Kozik
Fraunhofer IESE	Sonnhild Namingha (proofreading)

Disclaimer

The work associated with this report has been carried out in accordance with the highest technical standards and the Q-Rapids partners have endeavoured to achieve the degree of accuracy and reliability appropriate to the work in question. However, since the partners have no control over the use to which the information contained within the report is to be put by any other party, any other such party shall be deemed to have satisfied itself as to the suitability and reliability of the information in relation to any use, purpose or application.

Under no circumstances will any of the partners, their servants, employees or agents accept any liability whatsoever arising out of any error or inaccuracy contained in this report (or any further consolidation, summary, publication or dissemination of the information contained within this report) and/or the connected work and they disclaim all liability for any loss, damage, expenses, claims or infringement of third-party rights.

Contents

Executive summary 7
1. Introduction 8
1.1 Motivation 8
1.2 Intended audience 8
1.3 Scope 8
1.4 Relation to other deliverables 8
1.5 Structure of the deliverable 9
2. Update from previous (M6) version 9
3. Research Methodology 10
3.1 Research questions and steps 10
3.1.1 Presentation of the use cases 11
3.1.2 Interviews 12
3.1.3 A software quality workshop 12
3.1.4 Consolidation of the Quality Model 13
3.1.5 Second round of the software quality workshop for process metrics 14
4. Q-Rapids Data Sources at the Industry Partners 15
4.1 Summary of the data sources and tools at the industry partners (as-is scenario) 15
4.2 Integration of heterogeneous data sources 18
4.3 Critical Evaluation 20
5. User Stories for Data Gathering and Analysis 20
5.1 User stories from the use cases 21
5.2 Relevant user stories for data gathering 22
5.2.1 Product Factor: Code Quality (Maintainability) 23
5.2.2 Product Factor: Testing (Reliability) 24
5.2.3 Process Factor: Time-to-Complete Issues (Productivity) 24
5.2.4 Product Factor: Usage (Functional Suitability) 25
5.2.5 Product Factor: Bugs and Issues (Reliability) 26
5.3 Relevant user stories for data analysis 27
5.3.1 Preparation of data for analysis 27
5.3.2 Data analysis approaches 28
6. Results of Data Gathering and Analysis: Overview 30
6.1 Q-Rapids Quality Model by M18 30
6.2 Results on Data Analysis by M18 33
6.2.1 Using the Quality Model: Quality Alerts and Raw Data Visualization 33
6.2.2 Data Correlation 35
6.2.3 Time Series Analysis and Prediction 37
7. Specification of the Data Gathering and Analysis Architecture 41
7.1 The initial specification (M6) 41
7.1.1 The ingestion layer 42
7.1.2 Initial tools to be integrated into Q-Rapids for the proof-of-concept 44
7.2 Lessons learned and current data gathering and analysis specification (M18) 45
Conclusion 46
References 47
Annex A – Interview scripts for WP1 48
Annex B – Data storage: list of valuable attributes 54

The list of tables

Table 1. GQM workshops details 14
Table 2. Detailed results of data sources and tools of the industry partners. 15
Table 3. Summary of data available and their corresponding tools in the Q-Rapids industry partners. 17
Table 4. User stories of the Q-Rapids tool about gathering data during software development to assess quality. 21
Table 5. User stories of the Q-Rapids tool about gathering data at runtime to assess quality. 22
Table 6. User stories of the Q-Rapids tool about gathering generic data to assess software quality. 22
Table 7. Q-Rapids quality model 31
Table 8. Q-Rapids quality model: further process metrics 32
Table 9. Version Control Data (e.g. SVN, git) 54
Table 10. Metrics Data (Vector Version, example metrics) 54
Table 11. Metrics Data (Single Metric Version). For each metric e.g., DIT, LCOM 55
Table 12. Task Data 55
Table 13. Issue Data (useful if they write issue_id during commit info). Maybe to be merged with task 55
Table 14. Duplication Data (Software Clones) 56
Table 15. Testing data. 56
Table 16. Usage data. 56
Table 17. Crashes and errors data. 56

The list of figures

Figure 1. Research Methodology. 11
Figure 2. (a) Quality model hierarchy from Quamoco [4], (b) Exemplar Enterprise Architect profile to represent quality models. 13
Figure 3. Summary of the tools used and data sources available in the use cases. 18
Figure 4. Heterogeneous data sources available and their connections in the Bittium use case. 19
Figure 5. Excerpt of Q-Rapids quality model for Code Quality (Maintainability). 23
Figure 6. Excerpt of Q-Rapids quality model for Testing (Reliability). 24
Figure 7. Excerpt of Q-Rapids quality model for Time to Complete Issues (Productivity). 25
Figure 8. Excerpt of Q-Rapids quality model for Usage (Functional Suitability). 26
Figure 9. Excerpt of Q-Rapids quality model for Bugs and Issues (Reliability). 27
Figure 10. Example of aggregation of metrics into product factors, and product factors into quality factors. 29
Figure 11. Example of utility functions for assessing the "maintainability" quality factor. 29
Figure 12. Example of utility functions to assess the "maintainability" quality aspect (adapted from Quamoco). 33
Figure 13. Example of actionable analytics for the "fulfilment of critical/blocker quality rules" assessed metric. It shows the issues divided by severity and normalized by lines code. Besides, a list of blocker and critical issues that Q-Rapids suggest to take care of. 34
Figure 14. Example of actionable analytics for the "ratio of open/in progress bugs" assessed metric. It shows the ratio of bugs with respect other issues types, the percentage of resolved issues that are bugs, and a list of open high&highest priority bugs that Q-Rapids suggest to take care of. 34
Figure 15 Correlograms backlog size vs. the number of duplicated lines (upper row) and the backlog size vs number of delayed tasks. The blue dashed lines indicate statistical significance thresholds. Bottom row shows the correlated series (backlog size on the left and duplicated lines density on the right). 36
Figure 16 Example of signal $y(t)$ (data) decomposition into seasonal fluctuation, trend, and the reminder. 37
Figure 17 Examples of moving averages (blue and green) for a signal X. 37
Figure 18 Seasonality in the average (per developer) number of completed tasks. 38
Figure 19 Example of prediction idea 38
Figure 20 ACF and PACF diagrams. 39
Figure 21 Forecasts and confidence thresholds for Holt-Winters and ARIMA models. 40
Figure 22. Prediction vs. true signal (left) and the absolute differences (right). 40
Figure 23 Distribution of errors. 40
Figure 24. Q-Rapids adjusted lambda architecture. 41
Figure 25: Abstract view on data ingestion. 42
Figure 26: Main parts of Apache Kafka. 43
Figure 27: Apache Camel architecture. 44
Figure 28. Q-Rapids adjusted lambda architecture at M18. See evolution with respect to Figure 24. 45

Executive summary

The deliverable D1.1 is an output of the "Specification of data gathering functionality" (T1.1) and the "Elaboration of data analysis requirements" (T1.3) tasks. This deliverable mainly has three contributions.

Firstly, the identification of available data sources at the four industry partners of Q-Rapids. We collected this information at our visits to the premises of these industry partners.

Secondly, we gathered the epics and user stories for the Q-Rapids data gathering and analysis tool. From these user stories, we have built a generic Q-Rapids quality model, which includes the industry partners' needs and uses available data sources.

Thirdly, we propose an architecture based on Big Data to integrate heterogeneous data sources. Current approaches focus on one specific type of data (e.g., static code metrics), but in order to gain better insights, we consider it necessary to use and combine data from several sources that complement each other (e.g., sources for development-related data as well as sources for the actual usage of software products). The proposed architecture includes potential tools and frameworks that can be used for the collection and analysis of the data.

1. Introduction

The overall goal of this document is to provide the specification of the data gathering and analysis architecture for the Q-Rapids tool. The Q-Rapids tool aims at providing integrated information about current quality issues from both development-related data and the actual system usage.

1.1 Motivation

Q-Rapids is a data-driven project. This means that in order to assess the level of software quality during development and at runtime, we firstly need to gather and analyse data about that software. This deliverable provides:

- The data sources available from the use case-specific software systems of each industry partner.
- User stories of the Q-Rapids data gathering and analysis tool, and a generic Q-Rapids quality model based on the use case-specific software systems of each industry partner.
- A specification of the Q-Rapids data gathering and analysis architecture.

1.2 Intended audience

This deliverable is a report produced for all the members of the Q-Rapids project. Specifically, the results of this report are interesting and useful for the following stakeholders:

- The industry partners (i.e., Bittium, Softeam, iTTi, NOKIA), who need to know the relevant data for the assessment of the level of software quality of their use cases.
- The WP2 researchers (UoO), who need the basic metrics reported in this document to support the planning of the next development steps.
- The WP3 researchers (UPC), who need the basic measures reported in this document to aggregate them into strategic indicators and to use them for decision-making.
- The WP4 integrators (ITTI), who need to integrate the data gathering and analysis architecture into the whole Q-Rapids solution.

1.3 Scope

The scope of this document is the entire Q-Rapids project. This version of the document is the result of the first phase of WP1: the project baseline (from month 1 to month 6). It is going to be used in the next phases of the Q-Rapids project for the development and deployment of the software solution (i.e., the Q-Rapids tool). The software solution will be delivered at milestones M15 (proof-of-concept), M24 (prototype) and M33 (consolidation). Therefore, this deliverable may be updated in case we receive feedback in the aforementioned phases or in M36 (final release).

1.4 Relation to other deliverables

Due to the initial state of the project, this deliverable does not relate to any other previously produced deliverables. However, the present document relates to contemporary or future deliverables:

- Deliverable 1.2 consisting of a demonstration of the implementation of the data gathering and analysis proof-of-concept.
- Deliverable 2.2 including further process metrics identified in the second round of workshops.
- Deliverable 3.1 includes an ontology containing terms used in WP1 and this document.
- Deliverable 4.2 describes the integration of the entire Q-Rapids solution.
- Deliverable 5.1 reports the evaluation strategy of the data gathering and analysis tool of WP1.
- The deliverables resulting from T3.3 (D3.2, D3.3, D3.4, D3.5) will cope with the aggregation of the quality factors reported in this document into strategic indicators.

1.5 Structure of the deliverable

This document is structured as follows. Section 2 indicates the changes with respect to the previous version of this document (i.e., changes from v1.0 to v2.0). Section 3 reports the applied research methodology. Section 4 presents the available data sources from the four use cases of the industry partners. Section 5 defines the user stories and the quality model for the data gathering and analysis tool of Q-Rapids. Section 6 reports latest data gathering and analysis results. Section 7 specifies the data gathering and analysis architecture of the Q-Rapids tool. Finally, the conclusions are reported.

2. Update from previous (M6) version

The first version of D1.1 was delivered in M6 (April 2017). Between M6 and M18 the following changes in the document have been made:

- Regarding the research methodology:
  - Consolidation of the Q-Rapids quality model (Section 3.1.4)
  - Second round of software quality workshops for process metrics (Section 3.1.5)
- Regarding data sources at the industry partners:
  - Integration of heterogeneous data sources (Section 4.2)
- Regarding data gathering and analysis results, we added the new Section 6 including:
  - The consolidated Q-Rapids quality model (Section 6.1)
  - Research results on actionable analytics, correlation, and prediction (Section 6.2)
- Regarding the specification of the data gathering and analysis architecture, we report the decisions made since M6, the lessons learned, and the current specification (Section 7.2)

3. Research Methodology

This section reports how the visits to the **use cases of the Q-Rapids industry partners** were performed. We use the term use case of a Q-Rapids industry partner to refer to each of the four software systems in which the Q-Rapids solution will be used. Each industry partner provides one use case. For data anonymization purposes, we refer to the use cases of the industry partners as UC1, UC2, UC3, and UC4.

3.1 Research questions and steps

In line with the objectives of the first phase of WP1 (project baseline, from month 1 to month 6), we stated the following three research questions:

- RQ1. What is the current state of quality management at the industry partners?
  - Objective: getting to know their use cases (i.e., the software system and project in which they will use the Q-Rapids tool) and checking their data sources to gather information about the identified quality issues.
  - Step 1: Understanding the as-is scenario of the use cases (see Section 4).
  - Instruments: two instruments:
    - The industry partners presented their use cases through slides (see details in Section 3.1.1).
    - The WP1 researchers conducted face-to-face semi-structured *interviews* (see details in Section 3.1.2).
- RQ2. Which quality factors should be measured to support rapid software development?
  - Objective: discovering the industry partners' quality issues and their expectations regarding Q-Rapids.
  - Step 2: Identifying the to-be scenario of the use cases (see Section 5).
  - Instrument: The WP1 researchers conducted a *software quality workshop* (see details in Section 3.1.3).
- RQ3. How can we use Big Data technologies to gather data from heterogeneous sources of software development and analyse the relevant quality requirements?
  - Objective: specifying an architecture for the data gathering and analysis tool of Q-Rapids.
  - Step 3: from "as-is" to "to-be" (see Section 7).
  - Instrument: architectural synthesis.

During the first month of the project, we devised the research methodology. During months 2 to 4 of the project, we visited the four industry partners. During months 5-6, we analysed and synthesised the collected data.

To answer the three research questions, we followed the research methodology depicted in Figure 1, which consists of three sequential steps (one for each research question). The first two steps relate to the "empirical acquisition of data", and the third step to the "construction of the architecture" (these steps are proposed by Galster et al. [1] for building empirically-grounded architectures). The three steps can also be mapped to the process for creating architectures proposed by Nakagawa et al. [2]: information source investigation, architecture analysis, and architecture synthesis. Figure 1 shows on the right side of each step the output (i.e., contribution) of each step. In the next phases of the Q-Rapids project, we are going to conduct a fourth step for "validation and evaluation" of the architecture, which is also the next step in the proposals of Galster et al. [1] and Nakagawa et al. [2].

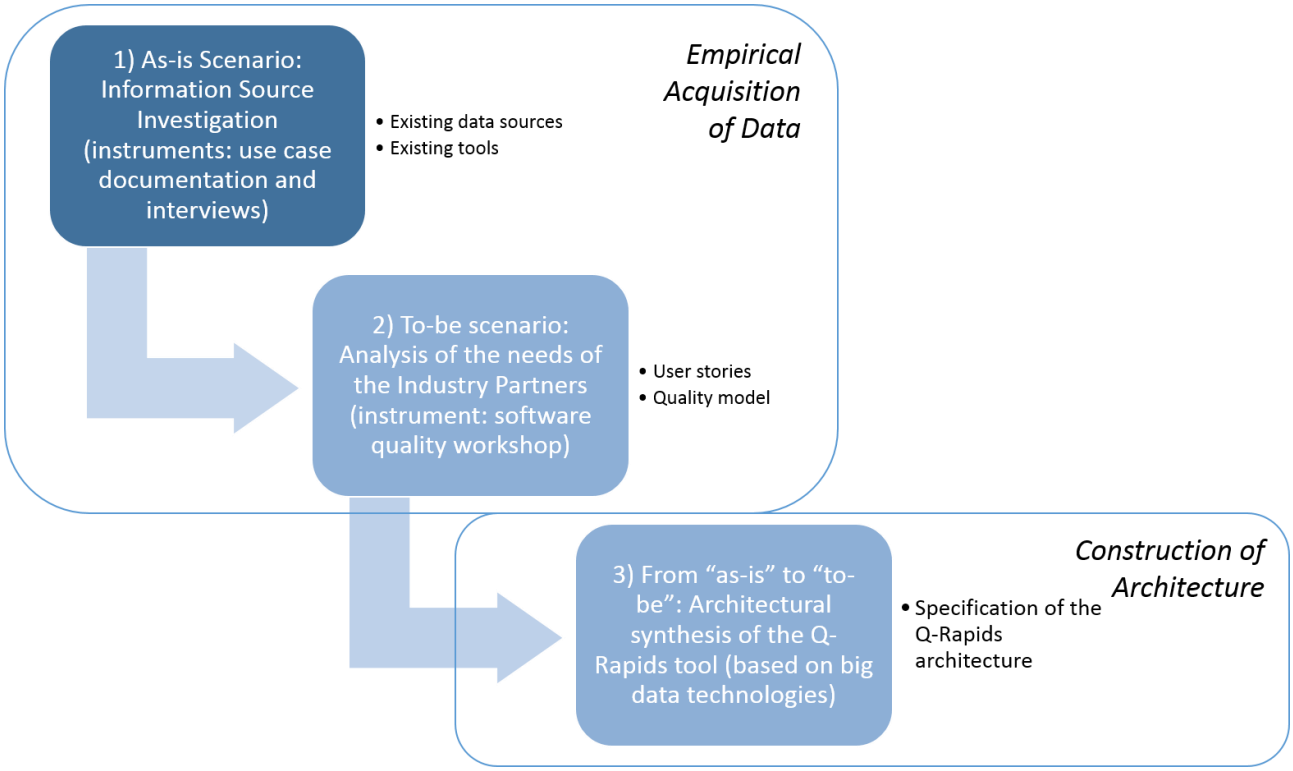


Figure 1. Research Methodology.

We will report the instruments used to answer our research questions in the following subsections.

3.1.1 Presentation of the use cases

Each industry partner showed its use case by presenting a set of slides. The goal was twofold: firstly, to present the context of the use case at the respective company, and secondly, to foster a shared mindset about the use case by both the team members of the use case (between 3 and 10 members) and the academic partners of Q-Rapids (between 3 and 8 members). The presentations followed the structure indicated below:

- Context & Current State: "as-is"
  - Environment and scope: brief description of the *type of projects and products* at the company.
  - Use case context: brief presentation of the *context of the selected project and product(s)* under study in the use case of the company.
  - Development process: brief description of the current *software development process* and the *main roles* in the use case.
  - Quality requirements: brief description of 1 to 3 *quality requirements* and how they are currently managed in the use case.
  - SWOT analysis and motivation: highlighting of the current problem for the selected project and product(s) in the use case.
- Expectations regarding Q-Rapids: "to-be"
  - Brief summary of the industry partner's expectations regarding Q-Rapids.
  - Description of one example of how Q-Rapids could help to manage quality requirements in their use case.

The presentation of the use case typically lasted between 45 minutes and one hour.

3.1.2 Interviews

We conducted one-hour semi-structured interviews with quality managers and developers to gather the following information:

- Section 1: As-is scenario of the use case. To identify data sources:
  - *Which different data sources and their corresponding (repeatable) technologies do you have at the company for:*
    - Projects (e.g., planning and effort sheets)?
    - Development (e.g., code repositories, bug reports)?
    - System behaviour (e.g., system logs)?
    - Software usage (e.g., profiling data, performance measures)?
    - User feedback (e.g. questionnaires, complaints, maintainability/change requests)?
    - Any other important sources (not to lose any data)?
- Section 2: To-be scenario of the use case. To identify user stories related to data analysis:
  - *What information about actual quality issues (of the software system(s) of the use case) would Q-Rapids provide to you?*
- Section 3: Acceptance and impact of Q-Rapids. To identify user stories related to the acceptance of the architecture.
  - *What quality characteristics should the Q-Rapids tool have to be accepted and used in your company?*

The complete interview scripts are available in Annex A.

3.1.3 A software quality workshop

Bearing in mind that relevant quality factors of software systems should be considered during the software development process, we proposed and conducted at each industry partner an integrated workshop for the creation of a measurable quality model, to proceed from strategic business goals to quantifiable metrics. The workshops demonstrated a workflow and corresponding moderation methods that allow using GQM+Strategies™ [3], Quamoco [4] and GQM [5] to build such a quality model and to visualise the findings. A brief explanation of these three approaches is given below.

GQM+Strategies™ aligns the goals and strategies of an organisation across different units through measurement. Besides a clear understanding of what the goals of the organisation are, the use of GQM+Strategies™ facilitates communication between different units by creating a common understanding. It helps to show the developers their contribution to the higher-level key performance indicators. Usually, this is possible because there are enough goals or strategies depending on the product quality. During the workshop, we identify the organisational goals underlying software quality.

Quamoco solves the problem of traditional software quality models, which provide either abstract quality characteristics or concrete quality measurements, by integrating both aspects. It provides a generic quality model that needs to be tailored to a company's specific strategic goals. We use ISO/IEC 25010[1] as the generic model to match the specific quality goals identified during our workshop sessions.

GQM provides an approach for goal-oriented measurement. Starting from the goals, questions are derived. By answering these questions, the respective metrics quantifying the goals are defined. GQM thus provides a way to define a metric and interpret it. This allows demonstrating how to make quality aspects measurable and where to get the data from. In our workshops, we examined how to integrate the model, i.e., the

[1] ISO/IEC 25010:2011. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. https://www.iso.org/standard/35733.html

measurement, the analysis and the derived feedback, into the software development process of the respective industry partner.

Representing Quality Models: An Introduction to Quamoco

To represent the quality models, we use the quality model concepts defined in Quamoco [4]. These concepts are explained in the ontology of Deliverable 3.1. Briefly stated and as shown in Figure 2 (a), we have four levels: quality aspects, product factors, measures and instruments/software products. We created the quality models of this document by using a customised Enterprise Architect profile for quality models. Figure 2 (b) shows an example with the four levels mapped to Quamoco: quality factor, product factor, metric and data sources. In Q-Rapids, abstract *quality aspects* are broken down into *product factors* (attributes of parts of the product that are concrete enough to be measured), *assessed metrics* (concrete descriptions of how specific product factors should be quantified and interpreted for a specific context), and *raw data* (i.e. the data as it comes from the different data sources, without any modifications). Nowadays, operationalised quality models offering actionable analytics for multiple purposes (system, process, and usage) are still a challenge.

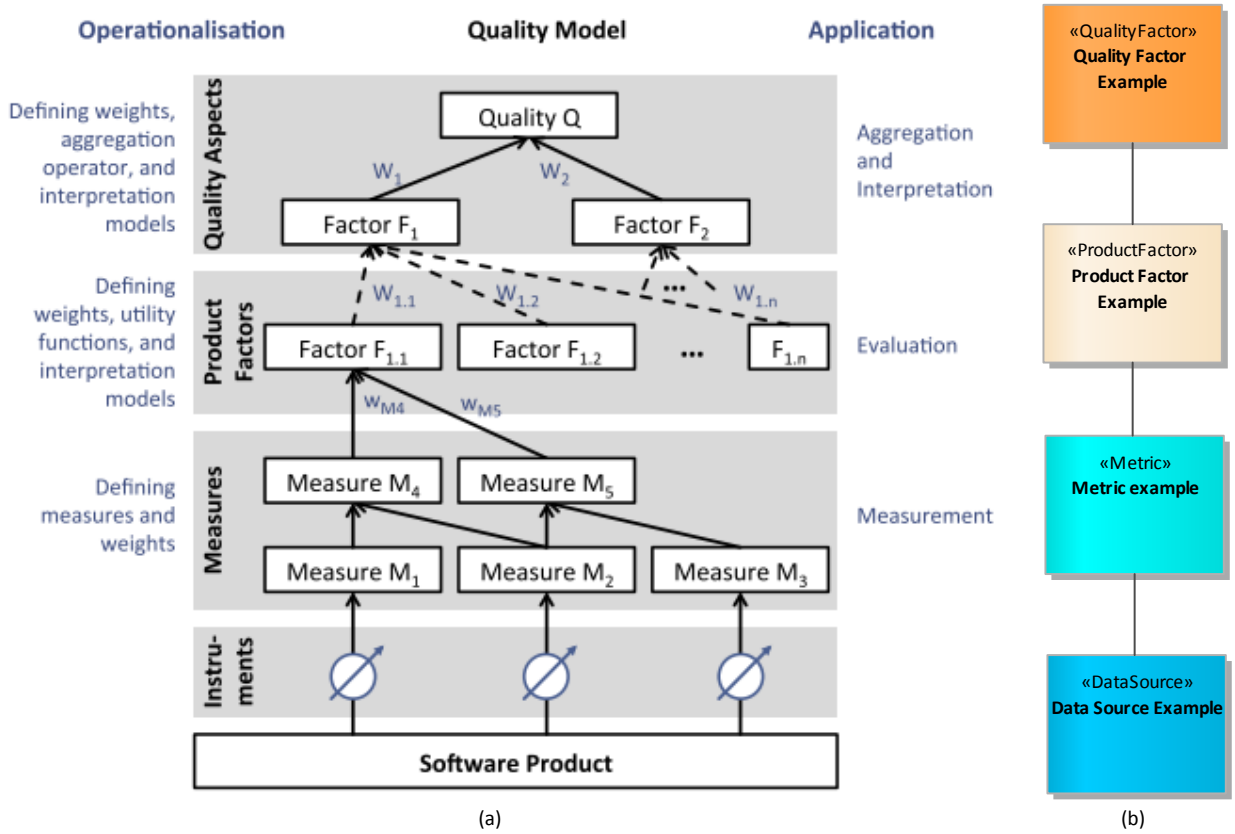


Figure 2. (a) Quality model hierarchy from Quamoco [4], (b) Exemplar Enterprise Architect profile to represent quality models.

3.1.4 Consolidation of the Quality Model

Based on the workshops' results, we specified practitioners-relevant user stories. Moreover, we created the Q-Rapids quality model by comparing, relating, and integrating the company-specific quality models. Thus, we identified commonalities and variabilities regarding relevant product and process factors as well as metrics among the four companies. We also checked those factors and metrics regarding to its feasibility to be automatically measured. Then, we presented the Q-Rapids quality model in two face-to-face meetings to a subset of the participants who attended the quality workshops. These meetings were intended for getting

further feedback and improving the Q-Rapids quality model. Although there were more metrics identified in the workshops, the Q-Rapids quality model includes an initial list being applied in the four companies.

We elicited and implemented the Q-Rapids quality model between December 2016 and December 2017. In total, 20 practitioners working in RSD attended to the quality workshops and were involved in the review of the company-specific quality models, and eight participants attended the face-to-face workshops for providing feedback on the Q-Rapids quality model. The Q-Rapids tool includes the implementation of the Q-Rapids quality model as well as the gathering, integration, and analysis of the required data.

Section 6.1 includes the consolidated version of the quality model, which could evolve in further validations in the companies.

3.1.5 Second round of the software quality workshop for process metrics

A second round of software quality workshop was conducted to further explore the metrics that industry partners find relevant from software development process perspective. Twelve GQM workshops were held (three sessions with each industry partner) to enquire about the process metrics that could supplement the quality model developed during the first workshop. Table 1 provides details of these GQM workshops:

Table 1. GQM workshops details

Workshop Session	Use Case	# of Participants	Role (# participants)
GQM Workshop 1	UC1	4	Architect/Developer, Project Manager, R&D Manager, CEO/Product Owner
	UC2	4	Quality Lead, Developers (2), Requirement & Process Lead
	UC3	8	Quality Manager (2), Project Manager, Developer (3), Development Manager (2)
	UC4	3	Product Owner, Project Manager, System Designer
GQM Workshop 2	UC1	2	Architect/Developer and R&D Manager
	UC2	4	Quality Lead, Developers (2), Requirement & Process Lead
	UC3	5	Quality Manager, Project Manager, Developer (2), Development Manager
	UC4	2	Product Owner, Project Manager
GQM Workshop 3	UC1	1	Architect/Developer
	UC2	4	Quality Lead, Developers (2), Requirement & Process Lead
	UC3	4	Quality Manager, Project Manager, Developer (2)
	UC4	1	Project Manager

The existing quality model at M12 was presented to the industry partners while eliciting the process metrics. The aim was to build upon the existing quality model. Thus, looking from a process perspective, practitioners discarded those factors/metrics that were not relevant from a process point of view and elaborated upon those that were of interest to them. Thus, new metrics/factors also emerged during these workshops, to be incorporated into the quality model. Further details on the way in which these workshops were conducted are presented in D2.2 (Section 2).

4. Q-Rapids Data Sources at the Industry Partners

This section presents the data sources available from the four use cases of the industry partners. This information was collected during the visits to the industry partners (see Section 2).

4.1 Summary of the data sources and tools at the industry partners (as-is scenario)

During the visit to the industry partners, we could see the current state (i.e., as-is scenario) of the respective use case regarding the available data sources. Table 2 reports the data sources available for the different types of data (first column) of each use case (columns 2 to 5).

Table 2. Detailed results of data sources and tools of the industry partners.

Data about….	UC1	UC2	UC3	UC4
Data sources and tools for project data				
Effort (planned)	None	JIRA (for epics and user stories)	Planned in the backlogs, FocalPoint, DOORS	GitLab (Each task is assigned to a particular person and is estimated according to the knowledge of the product owner)
Effort (real/tracked)	Effort sheets weekly (difficult to get)	Elbic (own software for invoicing)	Attlassian JIRA (real effort is not that reliable/trusted)	GitLab (ticketing system, how much time was spent on a particular task)
Planning (backlog)	Redmine (containing features), Word (cycles of 6 months)	JIRA (requirements management)	Many sites (e.g., multilevel backlogs, "platform service"), mainly Excel (pain point), e.g., R&D backlog in Excel. Moving to a real database.	Scoping sections with customers → GitLab (user stories and Kanban issue management)
Data sources and tools for development data				
Code	SVN repository	Git, Gerrit (code collaboration)	Git (SVN tool)	GitLab
Documentation	Models (code is reversed into models), API	Excel-based documents	"SerNet"	Wiki page
Bugs	Mantis (bug reporting)	JIRA (bugs and errors)	"Pronto" tool, "Normal" tool	GitLab
Data sources and tools for system behaviour data				
Log files	Yes (different levels, can be collected for end users, contain systems actions)	Yes (collected from tests in real devices (SpiraTest with physical product information), difficult to collect from end users)	Yes (different levels, different types, e.g., system, memory dumps). Read with text editors (no further tools)	Yes (in the production environment)
Performance	Java Engineering tools	SpiraTest	Tests (for stability and performance) need to be fulfilled before release	-
Network monitoring	-	Kibana, Elasticsearch	-	Zabbix, Nagios

Data sources and tools for system usage data				
Usage statistics	No	Kibana, Elasticsearch (complete?)	No; platform services (the end users are developers of applications using the platform). In applications, yes.	Partially in Zabbix, e.g., web scenarios execution stats
Usage feedback	Gforce, Salesteam (Salesforce) for customers (connected with Mantis), and Forum for open-source users (connected with Redmine)	Bittium Customer Care (connected with JIRA)	Complaints, usage reports	-
Other data sources and tools				
Code quality	SonarQube	SonarQube	They focus on faults. Different portals: SonarQube, Klocwork	SonarQube, GitLab
Tests	JUnit	SpiraTest (connected to JIRA)	(static and dynamic)	Spring framework (dedicated library)
Continuous Integration (CI)	Jenkins	Many tools (they have a complete list in CITA tools): JIRA, Jenkins	Jenkins	Jenkins
Scenarios/IU validation	TestLink (previously used), QFS: Quality First Software (new tool to be used)	Yes	-	-
API	Yes	-	-	-
Others	-	-	-	Flyway (database migrations)

We classify the data sources in two categories depending on when data is gathered: during development or at runtime. Table 3 summarises each type of data (i.e., the different entities of interest) and their corresponding quality-relevant tools used in the four use cases. As in [4], we use the term 'entities' to describe the things that are important for quality and 'properties' for the attributes of the things we are interested in. Table 3 also indicates in how many use cases (out of 4) the tools are used.

During software development, we found data sources about the project and development. This can be respectively mapped to the topic of improving the software development process productivity and software system quality by Zhang et al. [6].

Project data is on tools for managing the backlog containing the tasks and issues to be done at each sprint as well as estimated and real effort (e.g., Redmine, JIRA, GitLab), and tools for monetary investment (mostly proprietary). Issue tracking tools are common for the four use cases whereas they use different ones. Therefore, basic information about issues is available in all use cases. However, there is also diversity in the data introduced in these tools. For instance, estimated time and real effort is not introduced in one use case. In addition to these tools, a use case also uses an instance of a bug tracking system (called Mantis) just for bugs. It is important to notice that the industrial partners also use other generic data containing indirect metrics for quality factors and variation factors, such as the employees portfolio containing their experience.

Development data can be found in repositories (e.g., SVN, git, GitLab), and documentation (e.g., diagrammatic models, Excel files, Wiki pages, Confluence). Again, software versioning systems are common, meaning that commit information is available, but use cases have their own preferences concerning tools. Other development data relates to code quality (e.g., SonarQube, Klocwork, GitLab), testing and continuous integration (e.g., Jenkins, SpiraTest, QFS). Here we found that SonarQube and Jenkins are very popular and used in the four uses cases. Still, the configuration and importance of quality profiles in SonarQube is different among uses cases (for instance, which quality rules are analysed or which thresholds about testing coverage should be accomplished). Jenkins is used in a similar way in all use cases for the continuous integration pipeline. Industry partners with many software products have many Jenkins instances.

At runtime, we find data sources about the system behaviour and its usage. This can be mapped to improving the software users experience topic by Zhang et al. [6]. System behaviour data may be collected from logs and monitoring tools (e.g., Kibana, elastic, Naggios). Logs are heavily used in all uses cases at different levels, although it is challenging to get all end users' logs due to privacy issues. For the use cases involving different hardware needs, it is relevant to have logs' meta-data consisting of the device in which software is running. Besides, the usage of the software generates useful information such as statistics about how it is used, and feedback directly provided by the end user (e.g., hotlines, Gforce, proprietary software). The tools used here are completely heterogeneous among use cases.

Table 3. Summary of data available and their corresponding tools in the Q-Rapids industry partners.

When is data collected?	**Topic (see [6])**	**What is collected (i.e., entities)?**	**Where is the data (i.e., tools)?**	**Data sources and tools (frequency)**
During Software Development	Software development process	Product backlog, issues, iterations (a.k.a. sprints), effort, milestones, releases	Issue tracking system	Jira (2/4), Redmine (1/4), GitLab (1/4), Mantis (1/4), and proprietaries tools for invoicing
		Code reviews	Code reviews system	Gerrit (2/4), GitLab (1/4)
		Employees experience	Diverse proprietary tools	n/a
	Software system	Source code, commits	Code repository	SVN, Git, GitLab
		Software documentation, requirements	Diverse tools	Confluence (2/4), DOORS (1/4), and others (e.g., diagrammatic models, Excel files, Wiki pages)
		Static code analysis: software metrics and quality issues	Static code analysis tools	SonarQube (4/4), Coverity (1/4), CodeSonar (1/4), Klocwork (1/4)
		Exectuted tests, builds	Testing and continuous integration tools	Jenkins (4/4), Robot framework (2/4), Spiratest (1/4), QFS (1/4)
At runtime	Software users	System behaviour	Logs, network monitoring tools	Logs (4/4), Zabbix (1/4), Naggios (1/4), Elastic stack (1/4)
		System usage	Logs, monitoring plugins	Logs (4/4), SafeMove Analytics (1/4)
		Customer feedback	Proprietary questionnaires, hotlines, forums	Forum (1/4), ServiceDesk (1/4), Digium Enterprise (1/4), and proprietaries tools

Figure 3 aggregates the tools used by all industry partners for each type of data.

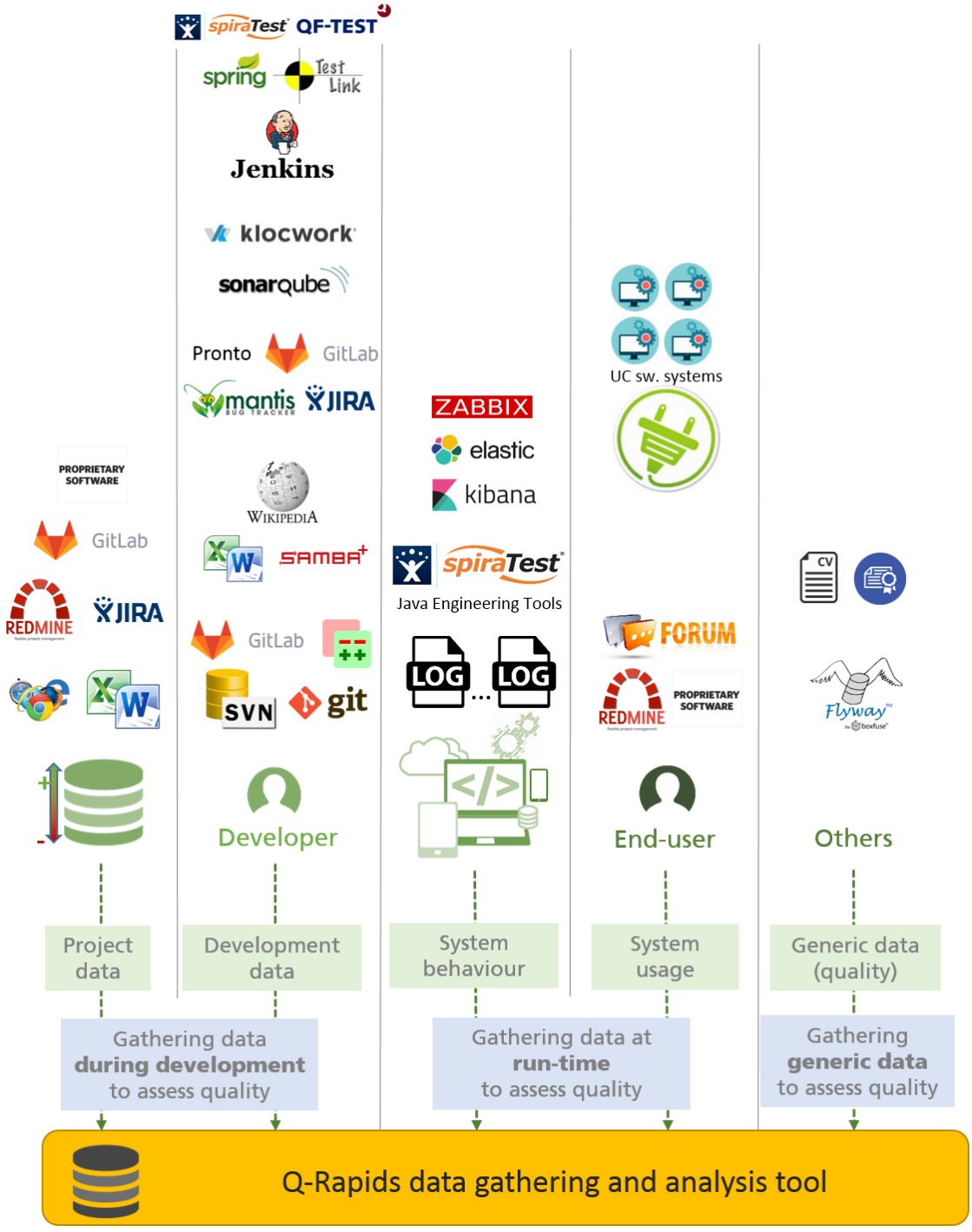


Figure 3. Summary of the tools used and data sources available in the use cases.

4.2 Integration of heterogeneous data sources

Now that we know the data sources and tools in each company, one question arises: how is data linked among these data sources? To answer this question, we study each use case. In this section, we detail the scenario in Bittium use case. Figure 4 shows the different entities (e.g., issues, commits, and logs) under study in the Bittium use case. Following the Table 3 format, the entities are divided regarding their topic (software development process, software system, or software users). Next, we report the flow about how the data from these entities is connected.

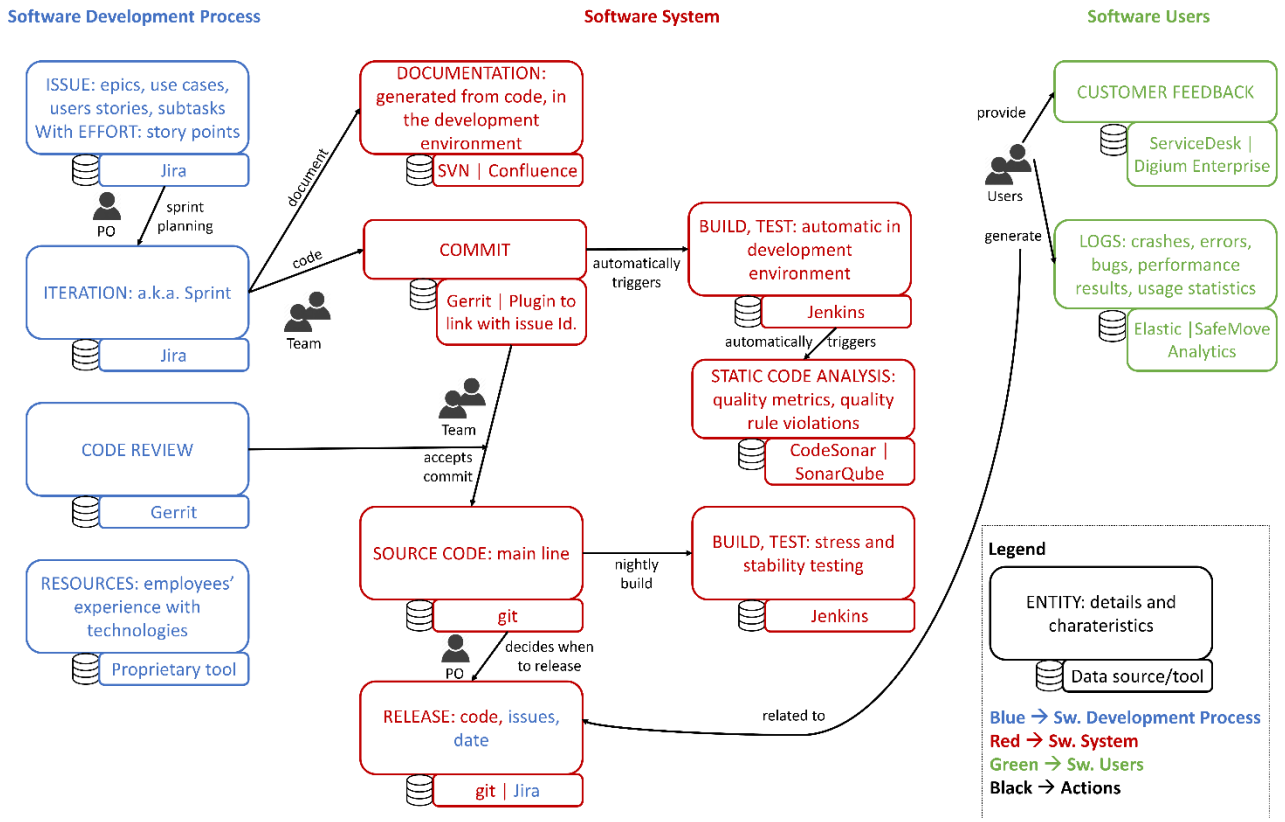


Figure 4. Heterogeneous data sources available and their connections in the Bittium use case.

First of all, the product owner together with project manager assigns the people to certain project, by checking the employees experience with technologies. This information is available from a proprietary tool called comma and the resourcing meetings.

When the project starts, the epics are defined in Jira. The epics (main requirements and product requirements) are just an issue type together with the use cases. As the project evolves, more concrete issues are created, from user stories to subtasks. During sprint planning, the product owner indicates the issues to be resolved in the current sprint, and estimate the effort with story points. Therefore, it is known to which iteration(s) an issue is assigned. This is handled by the Jira.

After sprint grooming and planning, the team is ready to start working in their assigned issues. As they work on issues, they generate documentation, which is stored either in SVN (normally documentation generated from code) and in Confluence (relevant manual documentation in the development environment). The actual code is stored in Git. Indeed, the documentation and Jira issues provide a good overview of what parts of the architecture are already done in each version. Besides documentation, the team commit source code to Gerrit. An important note is that by using a plugin in Gerrit, each commit comment has a template including the identifiers of the issues in Jira. Therefore, issues commits can be reached via hyperlinks from Jira.

After a commit is performed, automatic module and system tests are triggered in Jenkins. Therefore, the tests results can be associated to a commit. Other test frameworks are used in a lower extent, like RobotFramework and SpiraTest. Still, in most of the projects, Jenkins includes all the relevant pipeline. The builds in Jenkins also trigger the static code analysis to be performed either by SonarQube or CodeSonar. When a developer decides to handle a quality rule violation from CodeSonar, it is added in Jira with a link.

Once the committed source code includes automatic builds/tests results and static code analyses, the team colleagues start with the code reviews. Code reviews are not done over intermediate results, requiring the

results of automatic build and tests, and metrics and quality rules violations from static code analysis. However, there are no enforced rules while reviewing, except that an author cannot accept her own changes. There are some Key Performance Indicators (KPIs) to be achieved, related to test coverage, code complexity, etc. However, these KPIs are not really listed anywhere and can change. Reviewers' normal activities include running new/own tests themselves, and other activities like peer reviews if they consider it necessary. Information of the code reviews is stored in Gerrit. Code reviews can be read from Jira thanks to a Gerrit plugin. If the commit is rejected, the assignee has to continue working in order to push another patchset. When the commit is accepted, then it is moved to the source code main line, which is in Git. Therefore, when source code is accepted, it can be known under which test results and static code analysis characteristics was accepted.

The approved source code in git is heavily tested in night builds. Contrary to the aforementioned tests, these night builds additionally include stress testing and stability testing (i.e., tests executed several times in a row).

Bittium aims for continuous delivery (not continuous deployment). Therefore, after successful night builds, the product owner may decide to create a new release. Then, the new release is created in Git and associated to Jira.

When the new release is being used, it is when runtime data becomes necessary. Bittium receives customer feedback through the Atlassian JIRA ServiceDesk tool and Digium Enterprise. The customer feedback normally relates to bugs, and the device functionalities. The logs from both end users and executed tests are also gathered and push to Elastic for further mining. It is important to identify crashes, errors, bugs, and performance problems. Also, with SafeMove Analytics, more runtime aspects are monitored, such as crash logs. This information is linked to Jira when new issues of type 'error' are immediately created.

4.3 Critical Evaluation

The current state of the industrial partners is that they gather data from different data sources, but they do not aggregate this data in order to use it to assess the software quality. There is one notable exception: UC2 has a dashboard integrating different tools, namely Jenkins, SonarQube, JIRA and Kibana. However, no use case integrates and combines these data sources to offer combined or aggregated metrics and indicators. Therefore, we decided to investigate this gap to determine whether it would be useful for them to have combined or aggregated metrics and indicators. We will explain this investigation in the next section.

5. User Stories for Data Gathering and Analysis

In this section, we describe the "to-be" scenario, where we aim at being able to assess the quality of software in rapid software development processes with the Q-Rapids tool. To create such a "to-be scenario", we conducted three activities.

Firstly, we gathered the user stories elicited during the interviews and the workshops and identified from our own research into the use cases. They are defined in the Redmine instance of Q-Rapids[2] (see Section 5.1), to which the reader is referred for details. Secondly, we analysed the commonalities (i.e., data available from all partners) among the four use cases to select the user stories for the data gathering tool of Q-Rapids (see Section 5.2). Thirdly, we elicited the requirements for the data storage, aggregation and analysis of the Q-Rapids tool (see Section 5.3).

It is important to note that other relevant user stories about quality requirements and constraints of the tool are defined in "Table 7: Enablers for successful adoption of the Q-Rapids framework" of Deliverable 5.1. An

[2] Redmine instance of Q-Rapids: http://193.142.112.120/redmine

example is integration with existing systems: "As product owner, I want to have integration with existing systems, so that no new systems will be needed for data gathering".

5.1 User stories from the use cases

The user stories elicited during the visits to the use cases are divided by epics:

- Epic 1: As a Q-Rapids researcher, I want to gather data during software development, so that we can systematically assess the level of software quality (see Table 4).
- Epic 2: As a Q-Rapids researcher, I want to gather data at runtime, so that we can systematically assess the level of software quality (see Table 5).
- Epic 3: As a Q-Rapids researcher, I want to gather other data (besides data gathered during software development and at runtime), so that we can systematically assess other aspects of the level of software quality (see Table 6).

Table 4. User stories of the Q-Rapids tool about gathering data during software development to assess quality.

#[3]	Description
44	As a developer/product owner, I want to gather the time/effort needed to implement the same or a similar functionality, so that we can estimate how much time is needed to implement similar functionalities.
41	As a project manager, I want to gather the time/work effort used for variants in their first software release, so that we can predict variant reusability.
40	As a developer, I want to gather data about the impact a code change has on complexity, so that we can identify how rising complexity deteriorates maintainability.
39	As a product owner/project manager, I want to gather data about content delivered at Feature Build (FB) compared to planned content on exit so that we can see the planning capability and accuracy of the team.
38	As a quality manager, I want to gather data about requirements quality (e.g., percentage of unfilled data fields, standard tool for requirements), so that we can identify inadequate information or missing information in requirements.
37	As a test manager, I want to gather data about the quality and stability level of regression testing, so that regression tests cases are not failed or skipped (ensuring reliability and integratability of the platform)
36	As a code guardian, I want to gather data about how coding guidelines are followed (e.g., complexity and function size, coverage and duplications, and technical debt), so that we can manage resources for maintainability.
35	As a program manager, I want to gather data about how many individual software components can be initially deployed, so that we can analyse the possibility to deliver software in smaller entities.
34	As a product owner/release manager/program manager, I want to gather data about the completeness of sub-features in iteration (i.e., what is planned and what is actually done), so that we can know how many features are ready only in the last iteration before branching for stabilisation or even later.
15	As a quality assessor and integrator, I want to gather data about the current state of the integration process, so that we can improve the quality of the code delivered by the developers.
14	As a developer and integrator, I want to gather data about the quality of the code committed by the developers, so that we can improve the quality of the code delivered by the developers.
13	As a quality assessor, I want to gather data about the number of defects discovered during validation and in operation, so that the proportion of bugs discovered in validation and in operation is completely clear.

[3] We show the identifier of each user story (automatically assigned by Redmine) for traceability reasons.

Table 5. User stories of the Q-Rapids tool about gathering data at runtime to assess quality.

#	Description
45	As a product owner/UX designer, I want to gather data about product usage (e.g., total time spent on functionalities, and most/least used functionalities), so that we can know if an application is used and to what extent.
42	As a quality manager, I want to gather product reliability data for KPIs (e.g., MTBF), so that we can maintain efficient service capability/quality/prioritisation.
12	As a product director, I want to gather data about the most critical issues in operations, so that it is completely clear what the most critical issues are (we need stats over time and over user base).
11	As a product director, I want to gather data about the features heavily used by customers, so that it is completely clear how heavily each feature is used by customers and why.

Table 6. User stories of the Q-Rapids tool about gathering generic data to assess software quality.

#	Description
43	As a project manager, I want to gather data about the team skill level, so that I can evaluate team composition.

5.2 Relevant user stories for data gathering

We identified five common and relevant **product or process factors** for the industry partners of Q-Rapids. As we showed in the user stories above, there are more product factors, but we chose this subset because of their relevance and the commonalities among the industry partners. Three factors are relevant during software development, whereas two factors need data at runtime. We list them below, also indicating between brackets to which quality factor they are related and their main data sources.

- During software development:
  - Product Factor: Code Quality (Maintainability)
    - Main data sources: SonarQube, code repositories (e.g., SVN, Git, GitLab), Jenkins, Gerrit
  - Product Factor: Testing (Reliability)
    - Main data sources: Jenkins, SonarQube, other testing tools (e.g., TestLink)
  - Process Factor: Time to complete issues (productivity from the process point of view).
    - Main data sources: issue tracking tools (e.g., Redmine, GitLab, JIRA)
- At runtime:
  - Product Factor: Usage (Functional Suitability)
    - Main data sources: log files from software used at runtime
  - Product Factor: Bugs and Crashes (Reliability)
    - Main data sources: log files from software used at runtime, network monitoring tools (e.g., Kibana, Zabbios, Nagios)

We will describe these five factors in detail below. For each relevant product or process factor, we will describe related user stories, a visual representation of a quality model containing exemplar metrics for that product factor and some relevant attributes that could be stored in the Q-Rapids tool.

5.2.1 Product Factor: Code Quality (Maintainability)

The industry partners are interested in the code quality of the current build, in coding guidelines and in code complexity:

- As a developer and integrator, I want to gather data about the quality of the code committed by the developers, so that we can improve the quality of the code delivered by the developers.

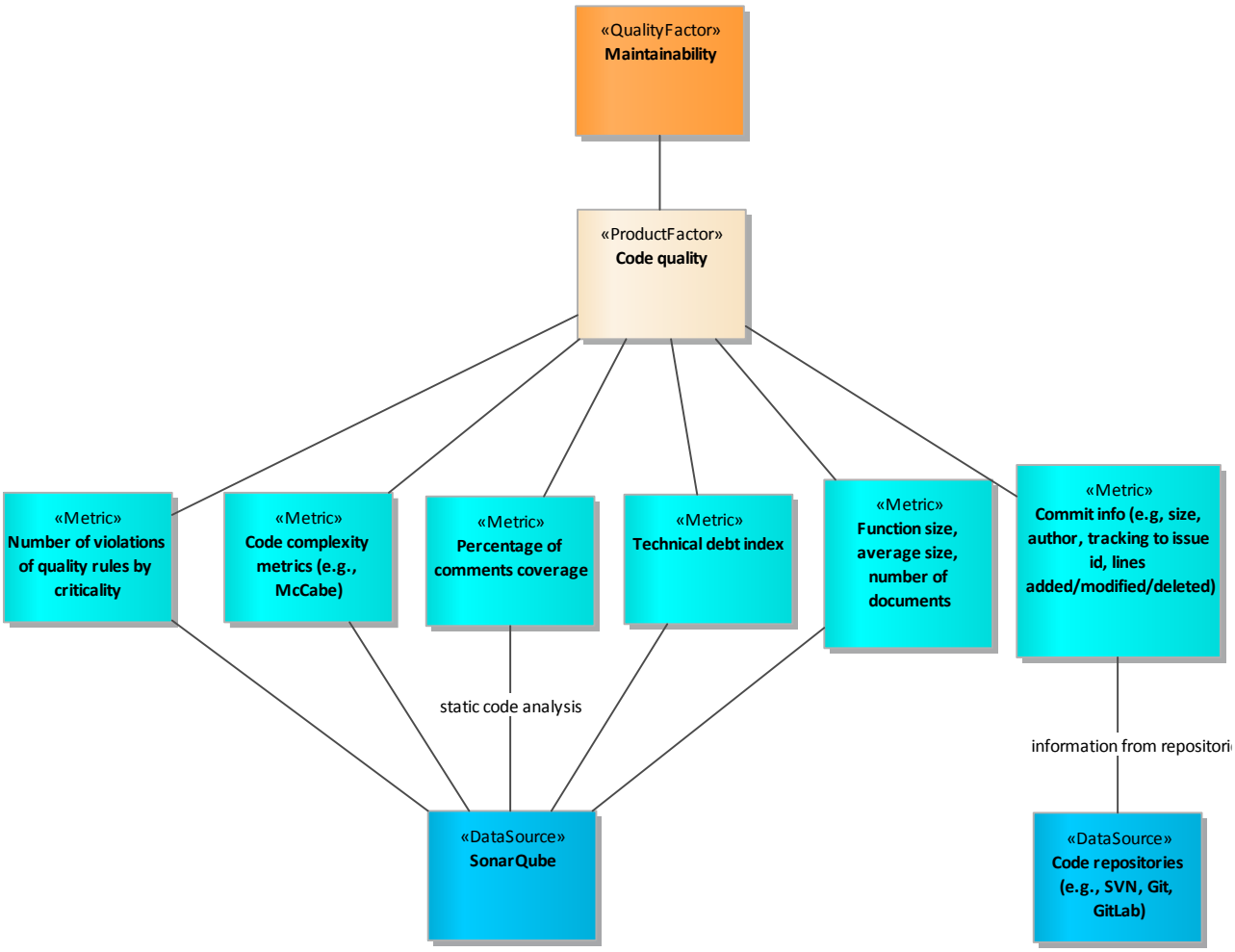


Figure 5. Excerpt of Q-Rapids quality model for Code Quality (Maintainability).

- As a code guardian, I want to gather data about how coding guidelines are followed (e.g., complexity and function size, coverage and duplications, and technical debt), so that we can manage resources for maintainability.
- As a developer, I want to gather data about the impact a code change has on complexity, so that we can identify how rising complexity deteriorates maintainability.

Figure 5 shows relevant metrics for code quality, such as quality rules fulfilment/violation, complexity metrics and technical debt indices from static code analysis (e.g., SonarQube). Important attributes to be saved regarding code quality metrics are shown in Table 10, Table 11, and Table 14 of Annex B. It is important to combine these code quality metrics with the information about commits from the code repositories (e.g., SVN). Important attributes to be saved regarding commits are shown in Table 9.

5.2.2 Product Factor: Testing (Reliability)

The industry partners are interested in monitoring the testing and integration activities during the software development process:

- As a test manager, I want to gather data about the quality and stability level of regression testing, so that regression tests cases are not failed or skipped (ensuring reliability and integratability of the platform)
- As a quality assessor and integrator, I want to gather data about the current state of the integration process, so that we can improve the quality of the code delivered by the developers.

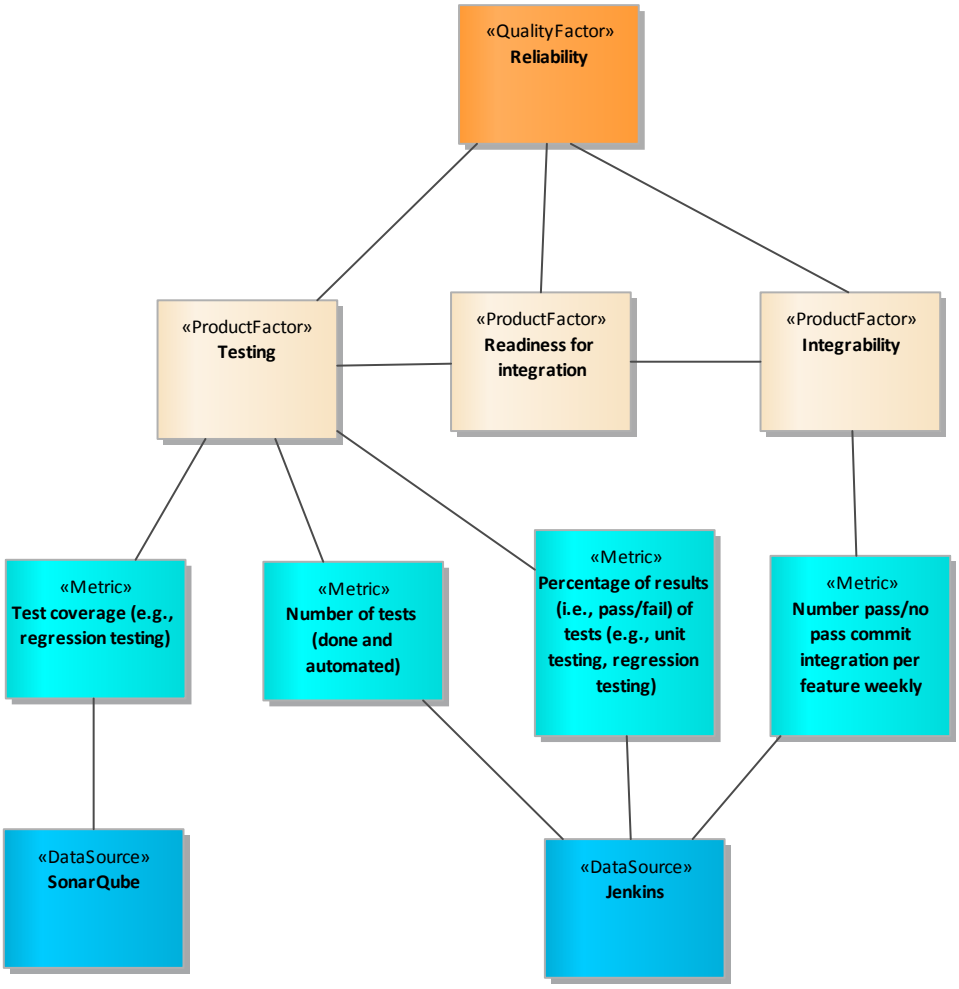


Figure 6. Excerpt of Q-Rapids quality model for Testing (Reliability).

Figure 6 shows the relevant metrics for testing and integration, such as test coverage from static code analysis (e.g., SonarQube), and the results of tests from continuous integration tools (e.g., Jenkins). Important attributes to be saved regarding testing are shown in Table 15 of Annex B.

5.2.3 Process Factor: Time-to-Complete Issues (Productivity)

The industry partners are interested in assessing the accuracy of estimation during planning and time-to-complete issues:

- As a product owner/project manager, I want to gather data about content delivered at Feature Build (FB) compared to planned content on exit, so that we can see the planning capability and accuracy of the team.

Figure 7 shows the relevant metrics for the accuracy of planning tasks and the productivity of closing tickets for tasks and issues, such as estimated time and real invested time for a task from an issue tracking tool (e.g., JIRA). Important attributes to be saved regarding tasks and issues are shown in Table 12 and Table 13 of Annex B. It is important to combine this information with commit information to get data about the effort invested in code changes. For this, it is crucial that developers indicate the id of the task or issue in the commit description.

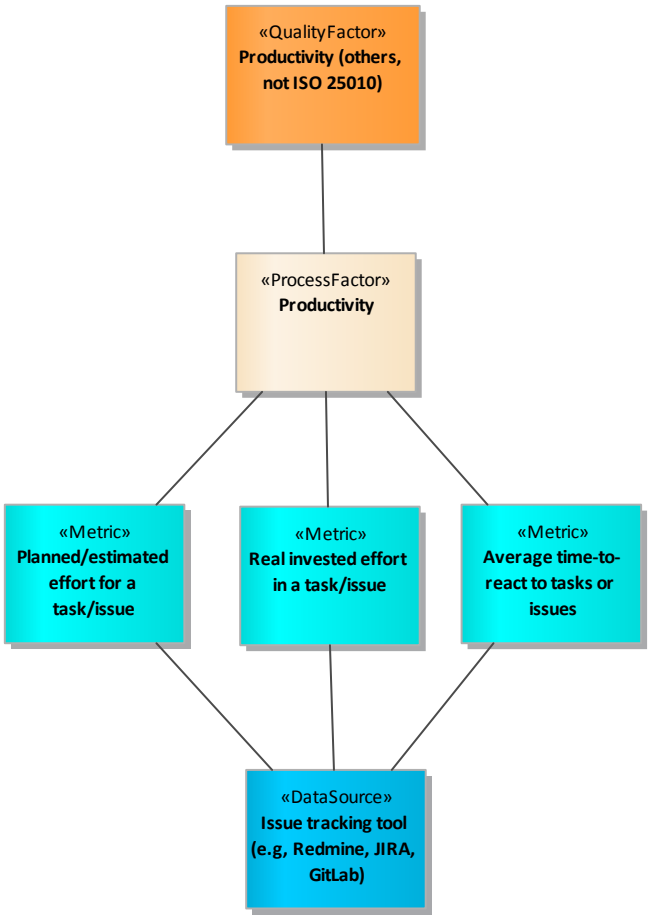


Figure 7. Excerpt of Q-Rapids quality model for Time to Complete Issues (Productivity).

5.2.4 Product Factor: Usage (Functional Suitability)

The industry partners are interested in the main usage of their products (i.e., statistics about the usage of applications):

- As a product director, I want to gather data about the features heavily used by customers, so that it is completely clear how heavily each feature is used by customers and why.
- As a product owner/UX designer, I want to gather data about product usage (e.g., total time spent on functionalities, and most/least used functionalities), so that we can know if an application is used and to what extent.

Figure 8 shows the relevant metrics for the statistics of product usage, such as the number of times a feature is used. This information may come from analysis of log files. Important attributes to be saved regarding usage statistics are shown in Table 16 of Annex B.

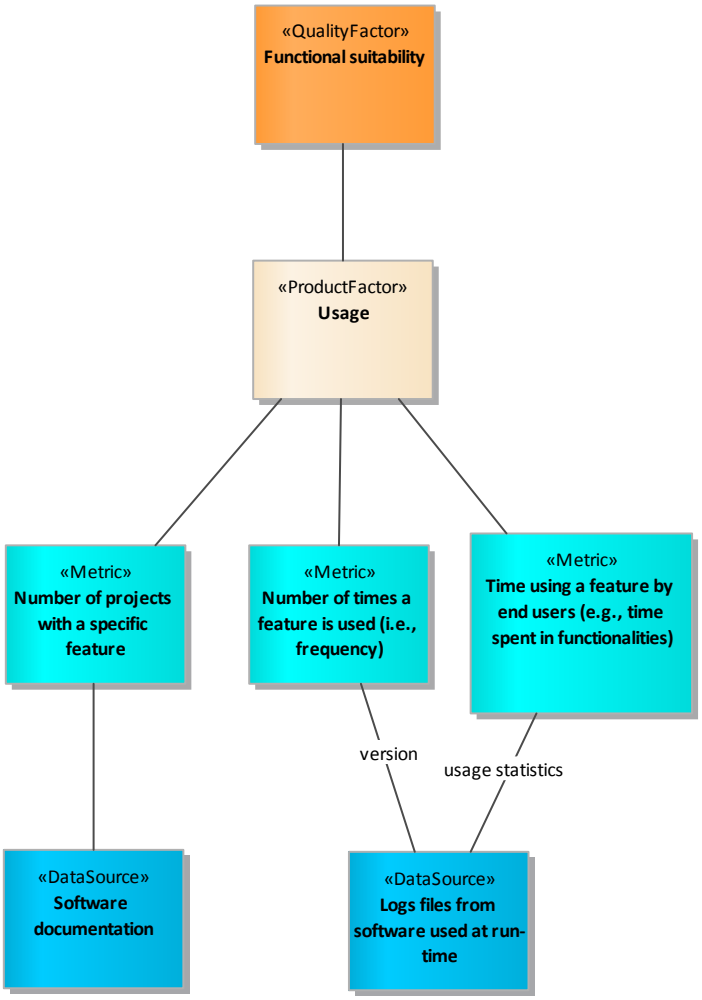


Figure 8. Excerpt of Q-Rapids quality model for Usage (Functional Suitability).

5.2.5 Product Factor: Bugs and Issues (Reliability)

The industry partners are interested in product reliability, the main pain points of products and the discovered bugs:

- As a quality assessor, I want to gather data about the number of defects discovered during validation and in operation, so that the proportion of bugs discovered in validation and in operation is completely clear.
- As a product director, I want to gather data about the most critical issues in operation, so that it is completely clear what the most critical issues are (we need stats over time and over user base).
- As a quality manager, I want to gather product reliability data for KPIs (e.g., MTBF), so that we can maintain efficient service capability/quality/prioritisation.

Figure 9 shows the relevant metrics for crashes at runtime from logs and network monitoring tools (e.g., Kibana). Important attributes to be saved regarding errors and crashes are shown in
Table 17 of Annex B. It is important to combine and compare this data with the issues reported in issue tracking tools (e.g., JIRA, Mantis).

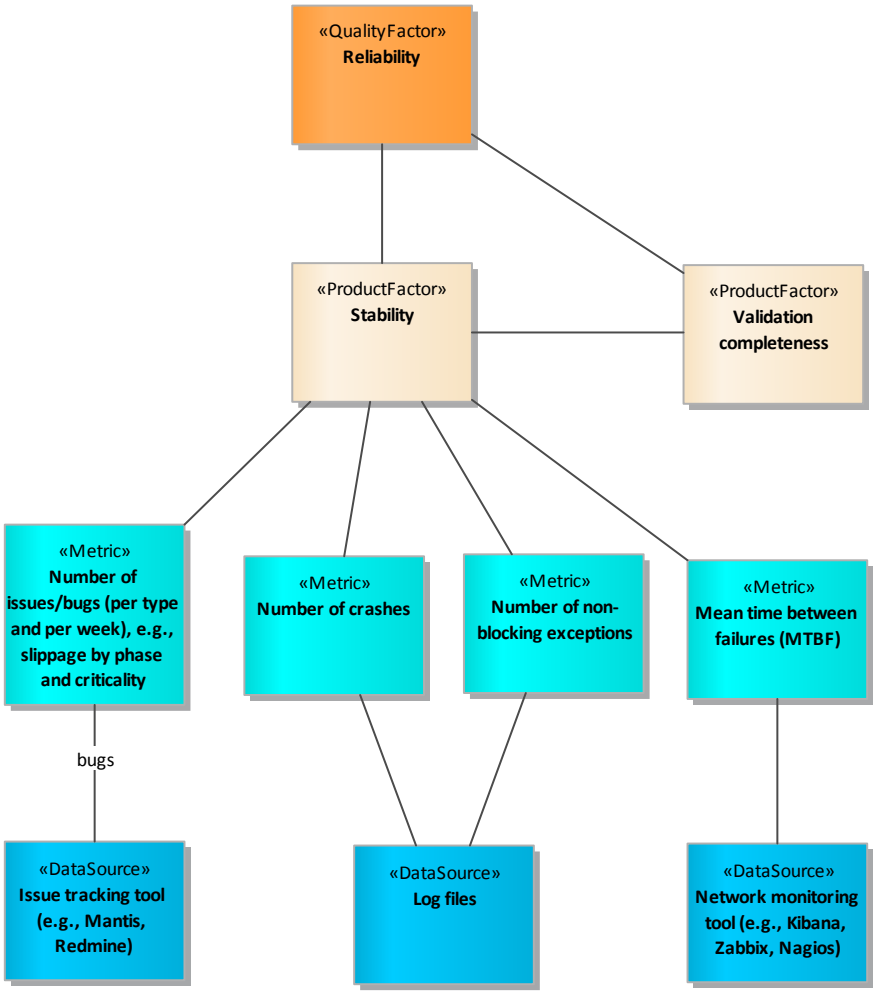


Figure 9. Excerpt of Q-Rapids quality model for Bugs and Issues (Reliability).

5.3 Relevant user stories for data analysis

The relevant user stories for data analysis are also divided by epics:

- Epic 4 (preparation of data for analysis): As a Q-Rapids researcher, I want to prepare the data for data analysis optimally, so that the Q-Rapids data storage can be exploited by Big Data analysis approaches.
- Epic 5 (data analysis approaches): As a Q-Rapids researcher, I want to analyse basic metrics of quality requirements, so that we can create alerts in case of deviations in quality requirements and show their current status.

5.3.1 Preparation of data for analysis

The relevant data identified in Section 5.2 needs to be prepared for data analysis. We identified the following relevant user stories:

- **Computation of basic metrics from possibly diverse data sources**: As a Q-Rapids researcher, I want basic metrics from the Q-Rapids quality model to be computed from the data gathered during data ingestion, so that metrics can be available for data analysis approaches.
    - Motivation: During data ingestion, we are gathering data as provided and structured by the data producers. However, it is necessary to transform such data into

metrics/measures of the Q-Rapids quality model. The metrics of the Q-Rapids quality model will be used by the analysis approaches.

- **Storage of basic metrics**: As a Q-Rapids researcher, I want the computed basic metrics to be stored in a suitable format (either relational or NoSQL, e.g., HDFS or HBase), so that we can later exploit parallelism with Big Data analysis technologies (e.g., Spark or Hadoop).
    - Motivation: There are many ways to store the metrics/measures of the Q-Rapids quality model (see Section 5.2) and its related attributes (see Annex B). In the next phases of the Q-Rapids project, it is necessary to identify the best way to store this information. This depends on two factors: the real data from the use cases, and the analysis approaches selected.
- **Combination of metrics from heterogeneous data sources**: As a Q-Rapids researcher, I want to be able to combine metrics from different data sources, so that we can reason about product factors with more than one data source.
    - Motivation: In software development projects, it is often difficult to combine metrics from different data sources. For instance, it is difficult to know which commits belong to a certain task, or to have in the same place bugs information from both logs and issue tracking systems. Due to the project's goal to integrate heterogeneous data sources, it is important to enable combinability of metrics coming from different sources.

5.3.2 Data analysis approaches

The following user stories summarize the main data analysis approaches:

- **Aggregation of basic metrics into product factors**: As a Q-Rapids researcher, I want the computed basic metrics to be stored in a way that they can be easily grouped or aggregated into product factors (following the Quamoco hierarchy), so that we can build a quality model from the basic measures.
    - Motivation: During analysis, it should be possible to aggregate basic metrics into product factors (e.g., aggregating *cyclomatic complexity* and *comments density* metrics into the *analysability* product factor, see Figure 10). For this purpose, product factors should be defined, and each metric should have a specified weight over the product factor.
- **Aggregation of product factors into quality factors**: As a Q-Rapids researcher, I want product factors to be stored in a way that they can be easily grouped or aggregated into quality factors (following the Quamoco hierarchy), so that we can build a quality model from the product factors.
    - Motivation: During analysis, it should be possible to aggregate product factors into quality factors (for example, aggregating *adaptability* and *analysability* product factors into the *maintainability* quality factor, see Figure 10). For this purpose, quality factors from ISO 25010 should be defined, and each product factor should have a specified weight over the quality factor.

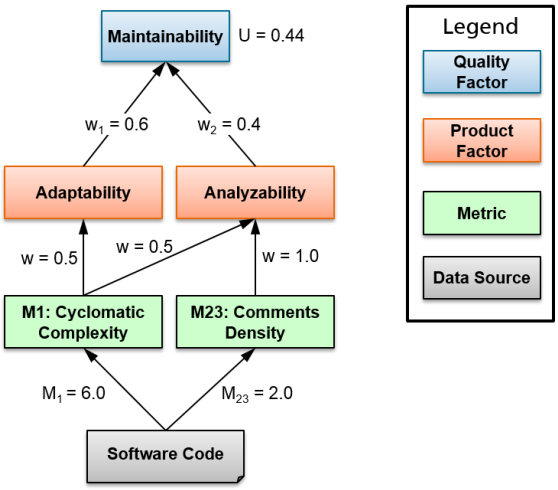


Figure 10. Example of aggregation of metrics into product factors, and product factors into quality factors.

- **Utility functions for proposing candidate quality requirements**: As a Q-Rapids researcher, I want to be able to define utility functions to model the preferences of decision makers with respect to the measures defined for the factors, so that we can analyse when a basic metric has exceeded a defined threshold.
    - Motivation: As defined in Quamoco, the evaluation step requires defining utility functions for modeling the preferences of decision makers with respect to the measures defined for the factors. In Figure 11, we can see examples of utility functions for cyclomatic complexity, comments density and maintainability. Specifically, this is useful for seeing the current state of maintainability (which could be a quality requirement). In a normalised value from 0 to 1, we can see that maintainability has a current value of 0.44. For instance, if our target is to have maintainability at its best value (meaning greater than 0.92), we could create an alert that maintainability of the product should be improved considering the cyclomatic complexity and comments density metrics. This could be an initial example of a candidate quality requirement.

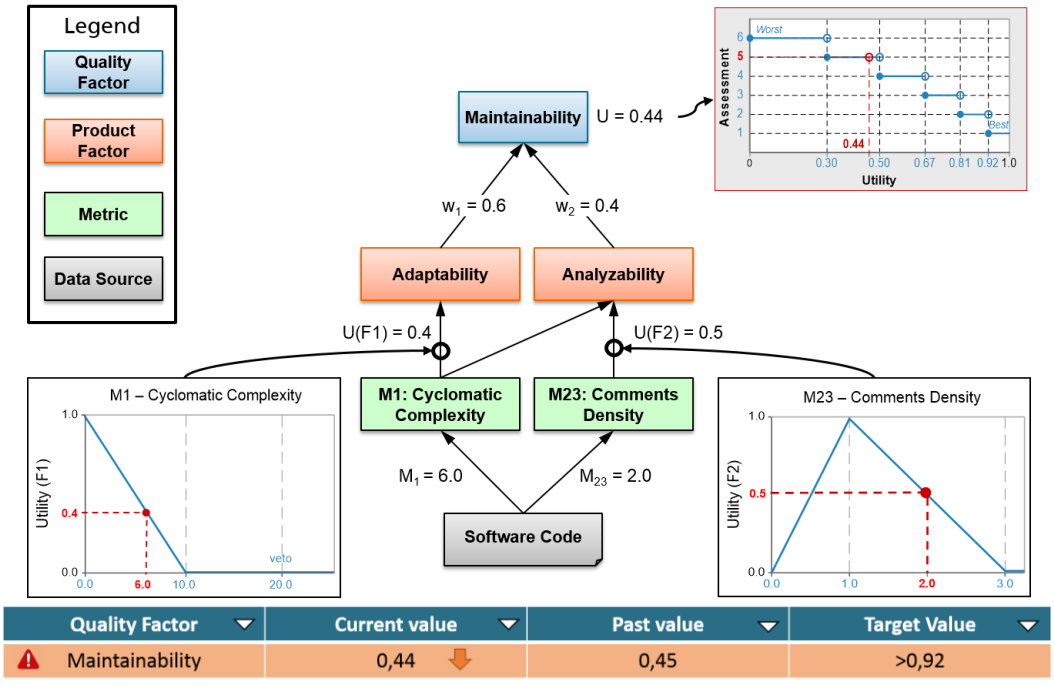


Figure 11. Example of utility functions for assessing the "maintainability" quality factor.

- **Relevant metrics for product/quality factors**: As a Q-Rapids researcher, I want to be able to perform analysis approaches to identify relevant metrics for predicting quality factors, so that we can rely on the most relevant metrics.
    - Motivation: There are two types of quality factor metrics: direct and indirect metrics. On the one hand, direct metrics refer to metrics directly measuring a quality factor, such as the time invested in maintenance tasks, which is a direct metric for maintainability. On the other hand, indirect metrics are correlated to a quality factor but do not directly measure it; for instance, a design structure matrix is a good indirect metric for maintainability (since it expresses an important characteristic of maintainability: modularity). In Q-Rapids, we have plenty of indirect metrics for quality factors. Therefore, it would be useful to apply analysis approaches to see the subset of relevant metrics for quality factors. To do so, we could just focus on a subset of metrics. The following analysis approaches are examples for conducting this type of analysis: correlation analysis (e.g., t-test, p-value); principal component analysis (PCA); F-measure; generalized cross-validation criterion (GCV); multivariate analysis; multiple linear regression (MLR); ANOVA; multilinear regression analysis; Pearson's correlation coefficient (PCC); logistic regression analysis correlations; R2; and Spearman rank correlation.
- **Prediction of future values of basic metrics**: As a Q-Rapids researcher, I want to be able to perform analysis approaches to predict the value of basic metrics in different contexts (i.e., with or without the consideration of candidate quality requirements), so that managers can see the benefits of prioritising a quality requirement in the product backlog.
    - Motivation: In future phases of Q-Rapids in which we will have historical data about the use cases, it will be useful to apply analysis approaches to predict the future value of basic metrics under different actions taken during software development (e.g., adding a new functionality vs. addressing a quality requirement).

6. Results of Data Gathering and Analysis: Overview

This section aims to present the advances on the implementation of the user stories from Section 5. In the D1.2 (Section 3), we reported the progress on the implementation by M15 on the user stories. However, we consider relevant to report here research activities performed about "open" user stories. This activities will guide future implementation activities:

- Combination of metrics from heterogeneous data sources: Section 6.1 and Section 6.2.1.
- Relevant metrics for product/quality factors: Section 6.2.2.
- Prediction of future values of basic metrics: Section 6.2.3.

6.1 Q-Rapids Quality Model by M18

Table 7 shows all the elements of the Q-Rapids quality model (M18): quality aspects, product and process factors, assessed metrics, and raw data. Next, we further explain several product and process factors for the maintainability, reliability, functional suitability, and productivity quality aspects. The first three quality aspects are from ISO 25010 and refer to the quality of the software system. The fourth quality aspect refers to the productivity of the software development process.

Table 7. Q-Rapids quality model.

QA[a]	Factor [c]	Assessed Metric [c]	Description	Raw Data	Data Source
maintainability	Code Quality	Non-complex files [b]	Files under the threshold of cyclomatic complexity (10 by default)	*Cyclomatic complexity* per *function* of each file, *total number of files*	SonarQube
		Commented files [b]	Files whose comment density is out of the defined thresholds (by default 10%-30%)	*Density of comments lines* and *lines of code* per each file	SonarQube
		Absence of duplications [b]	Files under the duplicated lines percentage threshold	*Duplicated lines* and *lines of code* per each file	SonarQube
	Blocking Code	Fulfilment of critical/blocker quality rules [b]	Files without critical or blocker quality rules violations	*Number of quality rules violations* per file and their *severity* (blocker, critical, major, minor, info) and *type* (code smell, bug, vulnerability)	SonarQube, Coverity, CodeSonar
		Highly changed files	Unstable files that have been highly changed in the last commits	Per each *commit: files changed*, *lines of code added/modified/deleted*, *author,* and *revision*	SVN, git, Gerrit
reliability	Testing Status	Passed tests [b]	Unit test success density	Number of *unit tests' errors*, *failures*, *skipped*, and *total*	Jenkins, GitLab
		Test Coverage	Tests with an appropriate coverage	*Condition coverage* and *line coverage* per *unit test*	Jenkins (JaCoCo plugin), SonarQube
	Testing Performance	Fast tests' builds [b]	Tests' builds under the threshold of duration	*Duration of unit test execution*	Jenkins, GitLab
	Software Stability	Ratio of open/in progress bugs [b]	Ratio of open issues of type bug with respect to the total number of issues in a customized time frame	Total number of *issues* (a.k.a. tasks) per *status* (e.g., open, done), *type* (e.g., bug, maintenance, feature), and *timeframe* (e.g., current/last month or current/last sprint)	Jira, GitLab, Redmine, Mantis
		Errors at runtime	Occurrence of critical errors at runtime at the end user site	All tracks of logs, including *type of error* (fatal, error, warning, info, debug, trace), *file* and *line* where in occurred, and *error message*	Logs
		Availability Uptime	Percentage of time that the product is accessible	Timestamp in which the system is not available and derived metrics, e.g., *Availability Uptime, Mean time between failures*	Zabbix, Nagios
functional suitability	Software Usage	Features Usage	Appropriateness of the features included in the software product regarding to their usage	Per each *functionality (or feature)*: number of *times used*, *average usage time*	Logs, monitoring plugin
productivity	Issues' velocity	Resolved issues assigned to a date	Resolved issues assigned to a date (simple date, iteration, or release)	Per each *issue* (a.k.a. tasks): *created time, status, updated time, iteration(s), release(s)*.	Jira, GitLab, Redmine, Mantis
		Issues completely specified [b]	Density of incomplete issues in a timeframe	*Fields of each issue* (e.g., description, due date, assignee, estimated time)	Jira, GitLab, Redmine, Mantis

[a] QA (Quality Aspect). [b] The assessed metrics indicated with a 'b' were implemented in the quality model and evaluated in January 2018. The other identified assessed metrics from the workshops are not yet implemented. [c] Each assessed metric can be classified in more than one product or process factor. In the same way, each product and process factor could be classified in more than one QA.

Besides, based on the software quality workshops for process metrics (see D2.2) and a preliminary study on the feasibility to measure those metrics, the following elements of the quality model have been identified and will be implemented. Please note that as process factors are highly context dependent, these process factors are company-specific; that is, not all of them will be implemented in all UCs, but depending on each UC's specific needs (see D2.2 for more details). We are working on validating these assessed metrics with the UCs and updates will be provided in D2.2 at M24.

Table 8. Q-Rapids quality model: further process metrics

QA[a]	Process Factor [c]	Assessed Metric [c]	Description	Raw Data	Data Source
productivity	Issues' Velocity	Issue Throughput	Average time it takes to implement an issue	*Duration for each issue* with *resolved status*, in *timeframe* (e.g. sprint, week, month), and *total issues*	Jira, Mantis, GitLab
		Status-Issue in a timeframe	Density of issues with a particular status within a defined timeframe	*Total number of issues* with *status* (e.g. open, closed, resolved, tested, work in progress), in *timeframe* (sprint, week, month), and *total issues*	Jira, Mantis, GitLab
		Team Throughput	Quantum of story points completed during the last completed sprint	*Total number of issues* (a.k.a. story points) with *status resolved/completed*, in *timeframe of last sprint*, and *total issues*	Jira
	Delivery Performance[b]	Timely release delivery	Percentage of releases delivered on time in a period	*Total on-time releases* delivered in *timeframe* (month, year), and *total releases*	Mantis
		Timely feature delivery	Percentage of features delivered on time during development cycle	*Total on-time features* delivered in *timeframe of a development cycle*, and *total features*	
		Core component commits	Number of commit on core component in a period before the planned release	Number of *commits on "metamodel.bpmn" and "diagram.editor.bpmn" components* in *timeframe* (week, month)	
	Development Speed	Daily build performance	Average daily build pass through time	For every *build*, *build duration* and *automated test duration*, and *total*	Jenkins, Atlassian, Stash
		Automation speed	Average speed of automated test	*Timestamp* for each *commit* when fully tested and when it goes for testing, and *total*	
		Commit review speed	Average commit review speed for a period	*Timestamp* for each *commit* when it is up for review and when it is reviewed, and total	
funcional suitability	External Quality	End user feedback	Feedback frequency from end-users related to issues in a period	*Total number of issues posted* after a support call or a post in forum in a *timeframe* (week, month)	Mantis

[a] QA (Quality Aspect). [b] This process factor was identified as 'on-time delivery' during the GQM workshops (see D2.2) but has been renamed as 'delivery performance'. [c] Each assessed metric can be classified in more than one product or process factor. In the same way, each product and process factor could be classified in more than one QA.

6.2 Results on Data Analysis by M18

6.2.1 Using the Quality Model: Quality Alerts and Raw Data Visualization

As shown in Figure 12, the three most abstract levels (quality aspects, product factors, and assessed metrics) work as "traffic lights", with a normalised value between 0 and 1 and customised thresholds to fulfil. Hence, the users of the Q-Rapids quality model can customise when to raise quality alerts. On the other hand, when a quality alert is raised, stakeholders can explore the raw data looking for the problem and taking a decision.

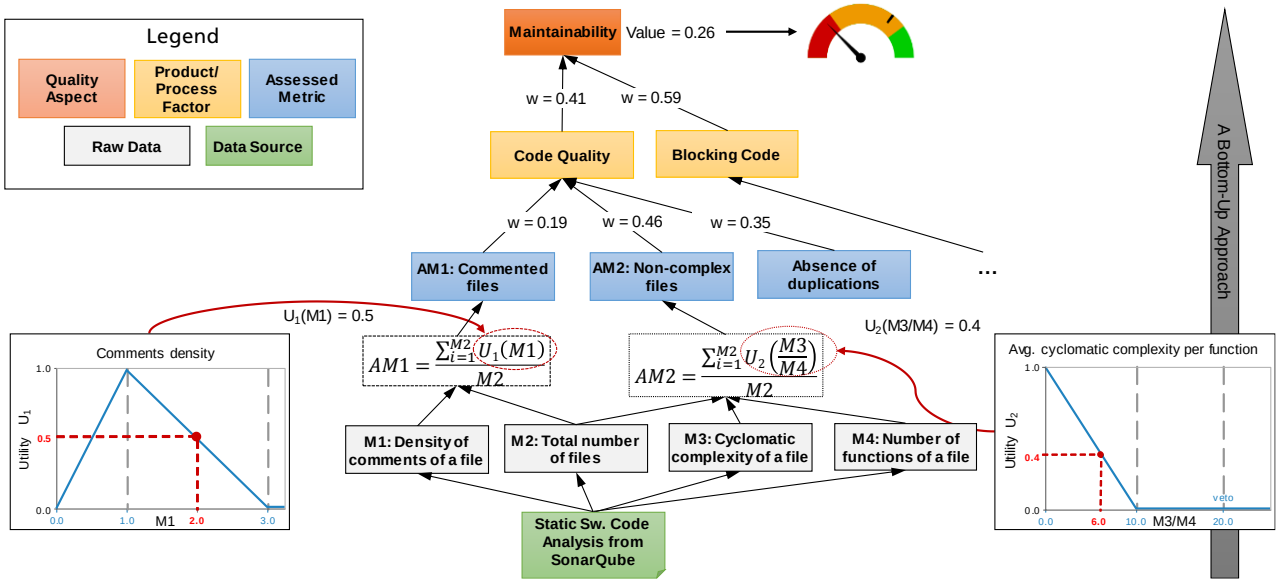


Figure 12. Example of utility functions to assess the "maintainability" quality aspect (adapted from Quamoco).

In this subsection, we give a couple of examples of how to use our quality model with respect to the maintainability and reliability quality factors:

A company has to improve the *maintainability* for one of their high quality products. Taking up the quality model, the quality aspect *maintainability* is composed of two product factors: *code quality* and *blocking code*. In this company, Bob, a quality manager, decides to use the Q-Rapids tool to manage maintainability problems. He installs the tool and the quality model does not raise any alert. However, at the beginning of the next cycle, Bob receives an alert because the *maintainability* bell is ringing. He sees in the tool that the *maintainability* traffic light has moved from green to orange. He calls for a meeting with Jane, a senior developer. Together, they go deeper in the quality model to analyse the situation in depth. Although *code quality* is green, *blocking code* appears as red. They explore further the assessed metrics and raw data of *highly changed files* and *fulfilment of critical/blocker quality rules.* They identify that in the last cycle, the classes of a directory were changed many times by a single developer. Moreover, the classes contained five blocker quality rules violations about code smells. The quality model is offering actionable analytics to refactor the classes of the problematic directory, clearly indicating which classes have been heavily modified and the explanation of the violated quality rules. They take action and add a new issue in the backlog so that the author can solve those problems and not accumulate technical debt.

Figure 13. Example of actionable analytics for the "fulfilment of critical/blocker quality rules" assessed metric. It shows the issues divided by severity and normalized by lines code. Besides, a list of blocker and critical issues that Q-Rapids suggest to take care of.

Another example. At the end of the last cycle, a new release of the product was launched. Unfortunately, during the current cycle the traffic light of *reliability* suddenly go to red and the alarm bell is ringing. Bob's Q-Rapids quality model shows an orange colour for *testing status* and red for *software stability*. He calls for an immediate meeting with Jane. Together, they recognize that the problem is related to *errors at runtime*. Jane further explores the raw data from logs, and realised many critical errors (e.g., 5xx errors) caused by one module. Therefore, immediate action is required to solve the crashes and exceptions identified at the client side. Using further drill-down, Jane is able to identify the exact line of code responsible for the mess and informs her team to work on the issue. Going further, Jane realises that the *test coverage* of such module was worse than average. Therefore, the product owner could identify it and include it in the product backlog with a lower priority. The *errors at runtime* metric is recovered within one hour whereas the *test coverage* gets a stable desired value in the next days.

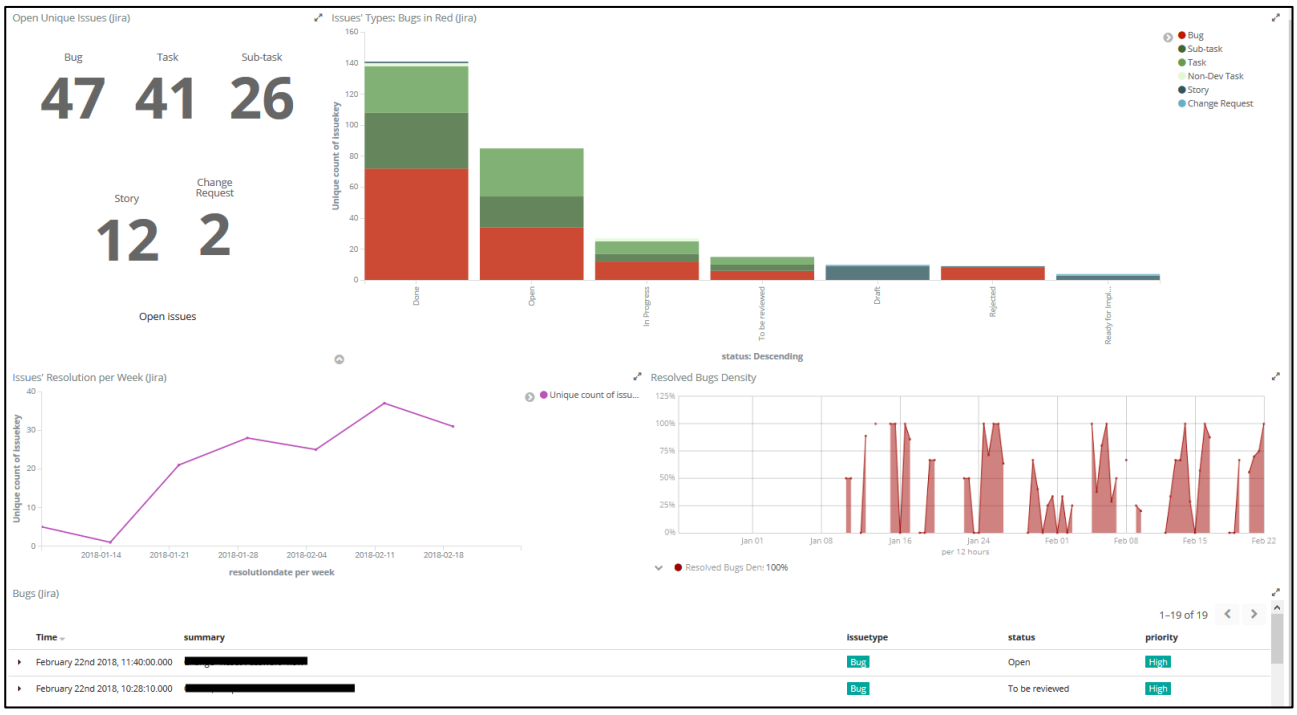


Figure 14. Example of actionable analytics for the "ratio of open/in progress bugs" assessed metric. It shows the ratio of bugs with respect other issues types, the percentage of resolved issues that are bugs, and a list of open high&highest priority bugs that Q-Rapids suggest to take care of.

6.2.2 Data Correlation

We say that two sets of values are correlated with each other whenever these proportionally decrease or increase together. The correlation could be negative as well. This means that one value will increase whenever the other one will decrease (and vice versa).

For the sake of a simplicity, let us call these sets of values as signals. Moreover, let us assume that the order of values in that sets matters. The most common way to measure the correlation level between two signals y_1 and y_2 is to use following formula:

$$corr(y_1, y_2) = \frac{cov(y_1, y_2)}{\sigma_1 \sigma_2}$$

where σ_1 and σ_2 indicate standard deviation of y_1 and y_2 signal respectively and $cov(y_1, y_2)$ indicates the covariance between y_1 and y_2 and can be calculated used the following formula:

$$cov(y_1, y_2) = \frac{1}{N} \sum_{i=1}^{N} (y_1^{(i)} - \mu_1)\,(y_2^{(i)} - \mu_2)$$

where *N* indicates the number of samples, $\mu_1$and μ_2 the mean values of signal $y_1$and y_2. Finally, the standard deviations and the means can be obtained using the following formulas:

$$\sigma_k = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - \mu_k)^2}$$

$$\mu_k = \frac{1}{N} \sum_{i=1}^{N} y_k^{(i)}$$

Usually, it makes more sense to correlate two signals that are shifted with respect to each other's. This allows investigating the correlation of current signal's values with historical ones. More precisely, we can introduce additional parameter *s* in the *corr* function formula:

$$corr(y_1, y_2, s) = \frac{\frac{1}{N} \sum_{i=1}^{N} (y_1^{(i)} - \mu_1)\,(y_2^{(i-s)} - \mu_2)}{\sigma_1 \sigma_2}$$

Using the presented above approach we can investigate whenever pairwise correlations appear in the collected data. Some examples of hypotheses could be as follows:

1. Increasing size of sprint backlog (number of development tasks planned for a sprint) correlates with the increasing cognitive complexity and amount of duplicated lines (e.g. as the result of work under the time pressure),
2. Increasing amount of duplicated code correlates with the increasing number of defects (bugs).
3. Increasing complexity of code correlates with the increasing number of defects (bugs).
4. Increasing size of sprint backlog (number of development tasks planed for a sprint) correlates with the number of delayed tasks.

In Fig. 1, we have presented the correlograms indicating how backlog size correlates with the number of duplicated lines (upper left diagram) and number of delayed tasks (upper right). The lag on x-axes indicates the shift parameters (development days). From the figures below we can observe that there are some bars that are above the statistical significance thresholds. In other words, approx. 7 days after (~ 1 sprint) the

team was overloaded (with tasks in the backlog) we will observe the degradation of code quality. In that case the increased number of duplicated lines increased.

A bit strange-looking could be the negative correlation of backlog size vs. duplicated lines at lag 0. But it must be noticed that it is the consequence of falling number of tasks in the backlog after the significant peak on Feb-10.

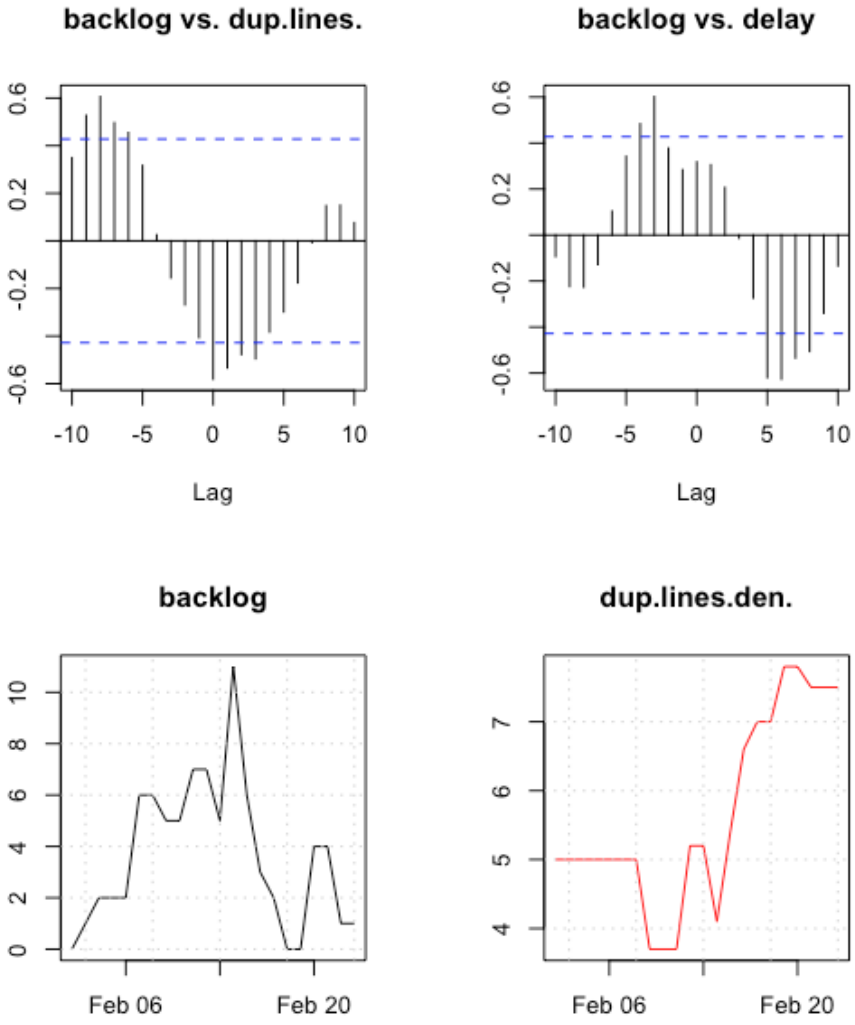


Figure 15 Correlograms backlog size vs. the number of duplicated lines (upper row) and the backlog size vs number of delayed tasks. The blue dashed lines indicate statistical significance thresholds. Bottom row shows the correlated series (backlog size on the left and duplicated lines density on the right).

We have consulted this with the use case provider. We have learnt that this situation was quite specific to the analyzed project. First of all, the team was adapting a new technology framework, and as a result sometimes it was difficult for them to precisely estimate the time needed to deliver planed tasks.

Future perspective on integration

The presented statistical tool is basic and important while finding relations between two variables. We do not foresee major obstacles that will impede current approach adapted for data gathering. We can consider two options for integration (to be checked in the future):

- Implementing correlation using scripted fields, so that the correlation for a given lag would be returned within a single Elasticsearch query.
- Using similar data processing pattern that is used by QR-EVAL for quality factors calculations.

6.2.3 Time Series Analysis and Prediction

Time Series Analysis

Time series analysis can be defined as the decomposition of a given $y^{(t)}$ signal into a trend $T^{(t)}$ a seasonal component $S^{(t)}$ and the residual component $e^{(t)}$. An example is shown in Fig. 2.

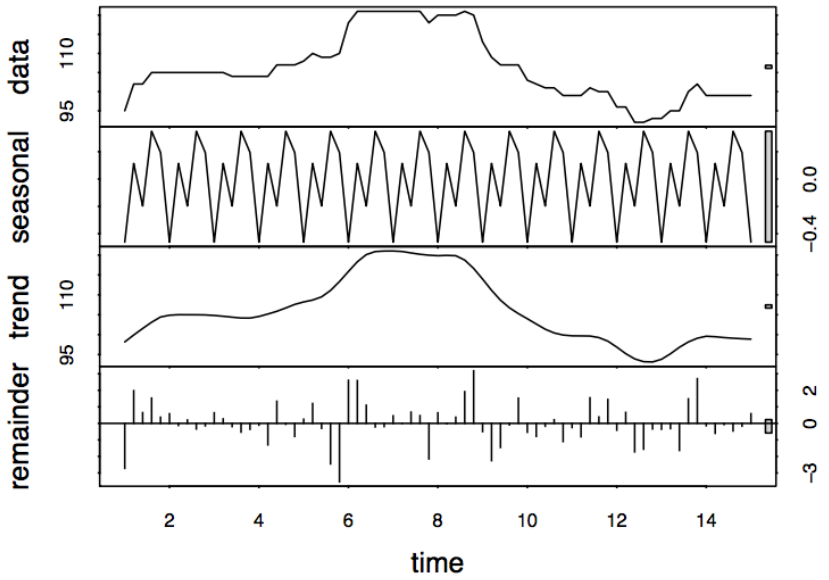


Figure 16 Example of signal $y^{(t)}$ (data) decomposition into seasonal fluctuation, trend, and the reminder.

One of the most commonly used ways to obtain the trend of a signal $y^{(t)}$ is to use the linear filter such as moving average:

$$T^{(t)} = \frac{1}{a+1}\sum_{i=0}^{a} y^{(t-i)}$$

An example of two moving averages has been shown in Fig. 3. In general, the bigger the value *a*, the smoother the filtered signal is.

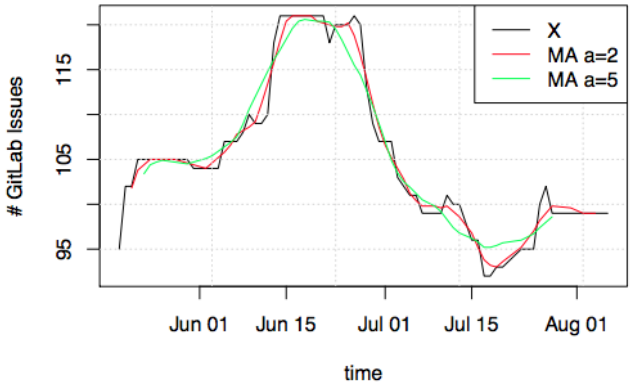


Figure 17 Examples of moving averages (blue and green) for a signal X.

Having the trend, we can subtract it from the original signal to obtain the residuals (remainder and the seasonal component). The seasonal component could be problematic as it may obscure the signal that we want to model and forecast. Usually, the seasonality is the phenomenon that repeats over time (e.g. daily, monthly, yearly). In software development, the seasonality may be caused by the plan of releases of new

versions of software or milestones of the project. Once we identify the seasonality we can model it. Afterward, the model of seasonality can be removed from the analyzed signal (it is often called Seasonal Adjustment or Deseasonalizing). An example of seasonality can be observed in Fig.4, which indicates the average number of completed tasks. Here we observed seasonality at the level of sprints. In the analyzed example the developers usually completed the task at the end of a sprint. Therefore, this can explain the periodic peaks.

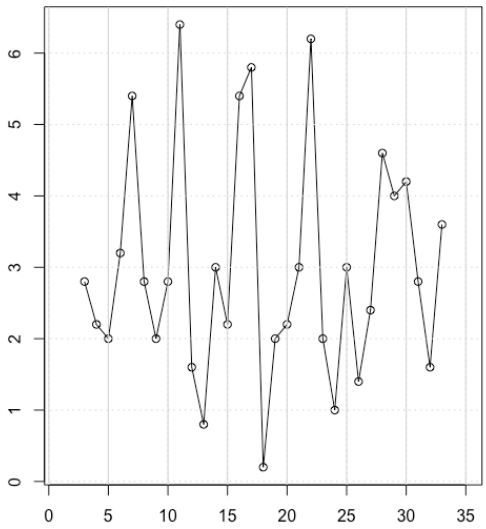


Figure 18 Seasonality in the average (per developer) number of completed tasks.

In order to accurately model the time series, the seasonality has to be subtracted. It can be achieved using a variety of approaches. The curve fitting or differentiating are some of them. As these procedures are usually seamlessly embedded in the frameworks for time series modeling we will avoid describing them in detail here.

Time Series Prediction

The time series are modelled in order to use that model for prediction purposes. In other words, the model is built using the training data in order to produce some forecast on unknown data. The general idea is shown in Fig.5. The training data is indicated by green colour.

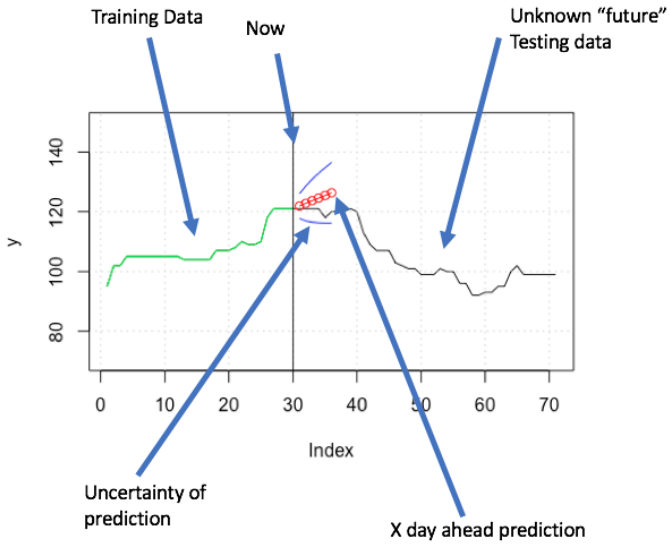


Figure 19 Example of prediction idea

The data is used to obtain the model parameters. Having the measurement for today (indicated as Now) we can produce a forecast for upcoming days (red circles indicated as 'x day ahead prediction'). Usually the prediction is subjected to some prediction error. Therefore, we need to define some uncertainty levels for prediction (the blue lines). In fact, for the presented example the blue lines indicate the 95% confidence thresholds, meaning that we have 95% chances that the predicted value will fall between the blue lines. As we can see the confidence levels are getting wider as we want to look further into the future.

As for the modelling, there are a variety of approaches. The most well-known are: exponential smoothing, Holt-Winters, and ARIMA (Autoregressive Integrated Moving Average). The exponential smoothing predicts the next value of a given time series using a geometric weighted sum of a past data samples.

$$y^{(r+1)} = \alpha y^{(r)} + \alpha(1-\alpha)y^{(r-1)} + \alpha(1-\alpha)^2 y^{(r-2)} + \cdots$$

However, the exponential smoothing model should only be used with signals that have not systematic trend and seasonal components. These can be subtracted using the approaches mentioned above. However, to avoid doing this manually the Holt-Winters model can be used directly. The model has three soothing parameters, namely α (for the constant level value), β (for the trend), and γ (for seasonal variations). Before we introduce some results of prediction, we would like to shortly introduce the third kind of model, namely the ARIMA.

The ARIMA states for Autoregressive Integrated Moving Average. The model is commonly denoted as *ARIMA(p,d,q)* where parameters *p*, *d*, and *q* are non-negative integers, that control three main components of the model: (i) the order (number of time lags) of the autoregressive model, (ii) the degree of differencing, and (iii) the order of the moving-average. The ARIMA models are defined for stationary time series (those which statistical properties such as mean, variance, and autocorrelation are constant over time). In order to obtain it, the non-stationary time series have to be differentiated until non-stationarity is eliminated. The number of differencing iterations corresponds to ***d*** parameter of the model. In order to choose other two parameters (*p* and *q*) we must analyze autocorrelation diagram (ACF) and partial autocorrelation diagram (PACF) of the analyzed signal. From the diagram shown in Fig.6, we may conclude that *p* will be 0, because *ACF* function drops below significance threshold (dashed line) after lag 0. Similarly, we can estimate *q=0* value using PACF diagram.

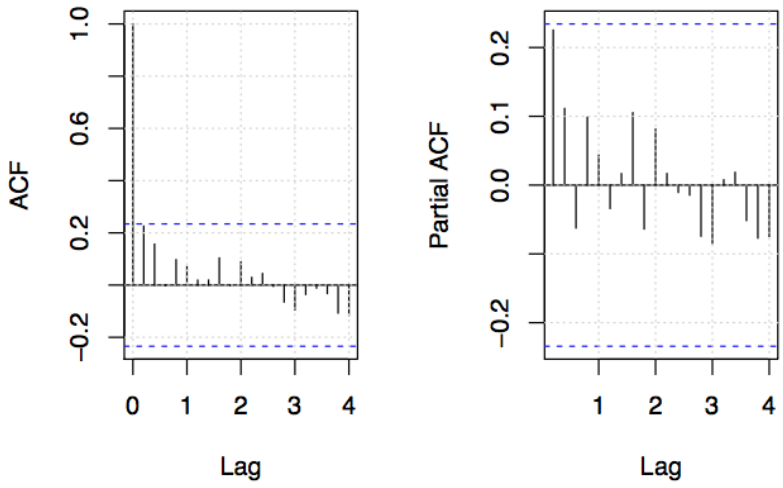


Figure 20 ACF and PACF diagrams.

The predictions using different models can be slightly different in terms of the forecasts values and the confidence thresholds (example is shown in Fig. 7).

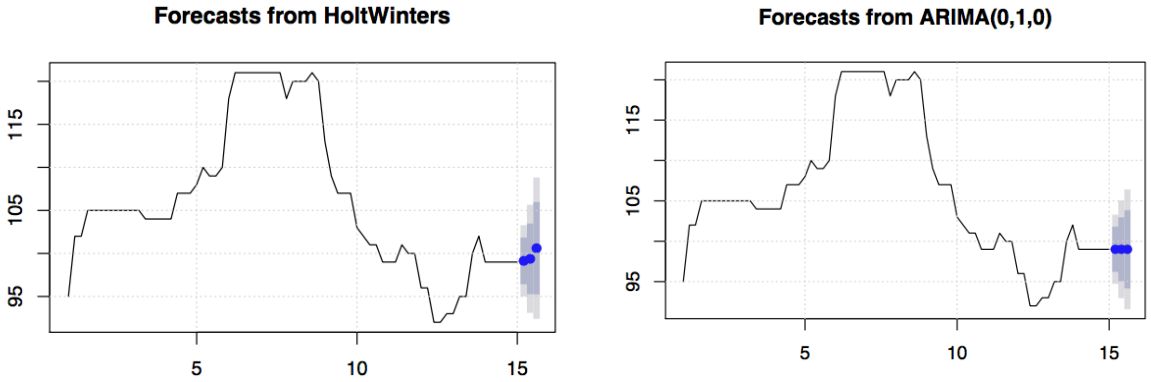


Figure 21 Forecasts and confidence thresholds for Holt-Winters and ARIMA models.

Nonetheless, these the accuracy of predictions can be assessed using the evaluation data. An example of prediction drawn on top of the true signal and the absolute values of differences are shown in Fig. 8.

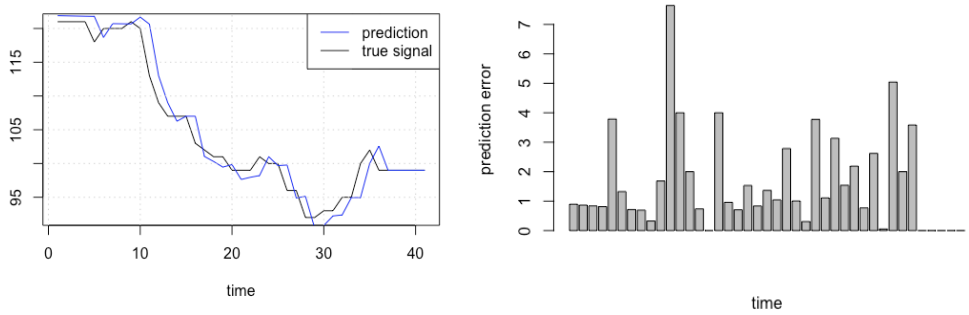


Figure 22. Prediction vs. true signal (left) and the absolute differences (right).

Looking at the distribution of errors (Fig. 9), it can be noticed that for the analyzed example (prediction of the number of issues for the next day) the majority of errors are less than 2.

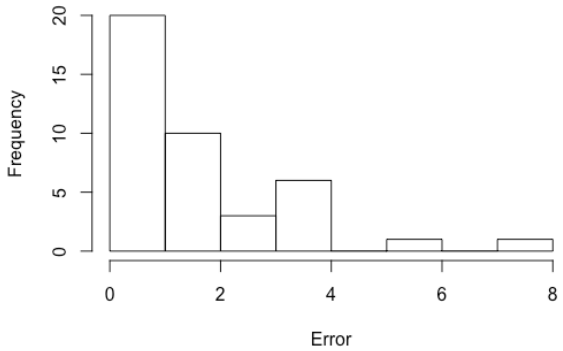


Figure 23 Distribution of errors.

Future perspective on integration

The presented tools for time series modelling and predictions can be easily integrated with the current version of data gathering and the analysis framework. There plenty of JAVA libraries[4][5] that can be integrated with the Elasticsearch in the same manner as QR-EVAL component.

[4] https://github.com/signaflo/java-timeseries
[5] https://github.com/Workday/timeseries-forecast

7. Specification of the Data Gathering and Analysis Architecture

7.1 The initial specification (M6)

This section describes the high-level data gathering and analysis architecture to be realized in Q-Rapids. The architecture shall provide flexible and extensible means:

- To collect all necessary information from various data sources such as software systems, log files and integration frameworks.
- To realize the analysis functionality needed to support quality-aware strategic decision making.

The architecture (see Figure 24) will take up the idea of the lambda architecture approach used for Big Data solutions. This is also in line with the integration of Q-Rapids, as reported in Deliverable 4.2. The data gathering and analysis architecture of Q-Rapids will be composed of three main layers:

1. The data ingestion layer will be responsible for integrating all necessary data and information from the outside (i.e., external data producers) into the Q-Rapids tool;
2. the data storage layer will be responsible for data persistence, and,
3. the data processing layer will transform and analyse data (batch and streaming).

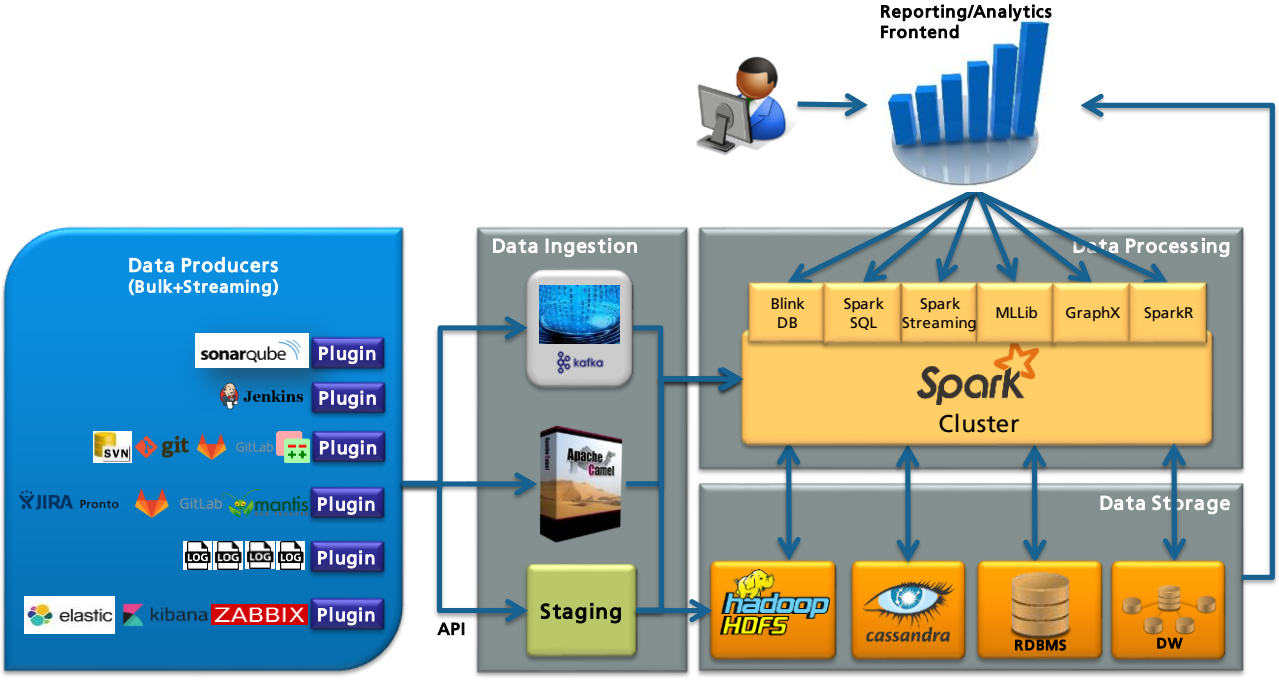


Figure 24. Q-Rapids adjusted lambda architecture.

For all three layers, a number of frameworks might be used to ease the development of functionalities related to Q-Rapids. For instance, Apache Kafka and Camel can be used for data ingestion. For data storage, different approaches could be used depending on the analysis approaches used, such as Hadoop HDFS or relations databases. In the case of data processing/analysis, Spark could be used. Within each layer, different frameworks might even be used in parallel if specific requirements demand specialised frameworks. However, it is intended to rely only on a small set of these tools to reduce the overall number of dependencies on software libraries and to be able to deploy the Q-Rapids platform even on commodity hardware. Finally, the reporting/analytics frontend of Figure 24 refers to the dashboard of WP3.

The final selection and specification of the frameworks to be used in the data storage and analysis layers will be performed in the next phases of Q-Rapids. The reason is that we will see the real data from the use cases in the proof-of-concept phase. Section 7.1.1 describes the ingestion layer and its possible frameworks.

7.1.1 The ingestion layer

To make use of data from various software systems, the ingestion layer is the communication channel to the Q-Rapids platform (Figure 25). It has to be able to deal with a probably large volume of data (e.g., by capturing runtime information with high velocity) as well as with a large variety of different data structures and formats produced by the diverse data sources.

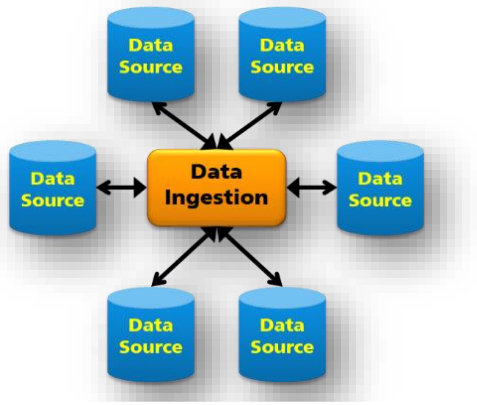


Figure 25: Abstract view on data ingestion.

Therefore, the main goal of the ingestion layer is to connect the software systems (data flow sources) and the Q-Rapids platform (data flow target). According to software engineering best practices, the connecting of systems should be decoupled as much as possible in order to keep the involved systems and their components autonomous from each other. If we rely only on low-level API calls, the maintainability of the overall system would severely suffer over time and the technical debt would increase. For instance, this might be the case if new requirements emerge regarding the data to be collected, if API calls change with new releases, or if new software systems to be connected to the platform demand changes in the existing API on the Q-Rapids platform (and then probably also in several data sources).

The proposed ingestion layer for the Q-Rapids platform, therefore proposes making use of existing publish-subscribe messaging frameworks to connect the data sources to the platform in a decoupled fashion. Each data source (software system) locally realises the required data extraction functionality (e.g., by realising a plugin for the system) and sends the extracted data to an appropriate topic handled by the messaging system. A component which is interested in that data (e.g., components in the data storage layer or in the analysis layer) subscribes to the topic and gets informed whenever new data is available. The same publish-subscribe concept can be used to realise a back channel from the Q-Rapids platform back to the software systems (e.g., to ask for more details upon demand, to provide valuable quality information back to a developer, etc.).

Independent of this publish-subscribe approach, the ingestion layer might still use direct API calls whenever needed or more appropriate. For instance, it would not make much sense to load all data from a software repository into the platform (i.e., to make a shadow drive within the storage layer) just to have all data within the platform. Instead, when quality-related data flowing from a repository to the platform (e.g., the number of classes changed, the LOC changed in a class) is not sufficient and the analysis or recommendation components need more information (e.g. information about severe control flow changes within the changed classes), then the data gathering could still use direct API calls to extract that kind of additional information from the repository directly.

Because the ingestion layer is likely to have to deal with large amounts of data and/or with data generated with great velocity, a publish-subscribe framework like Apache Kafka™ capable of handling high quantities of data is needed. Apache Kafka[6]™ is a well-known framework used for building real-time data pipelines and streaming apps. It is horizontally scalable, fault-tolerant and very fast, and is already used by many companies. Kafka is run as a cluster on one or more servers to store streams of records in categories called topics. Each record consists of a key, a value and a timestamp.

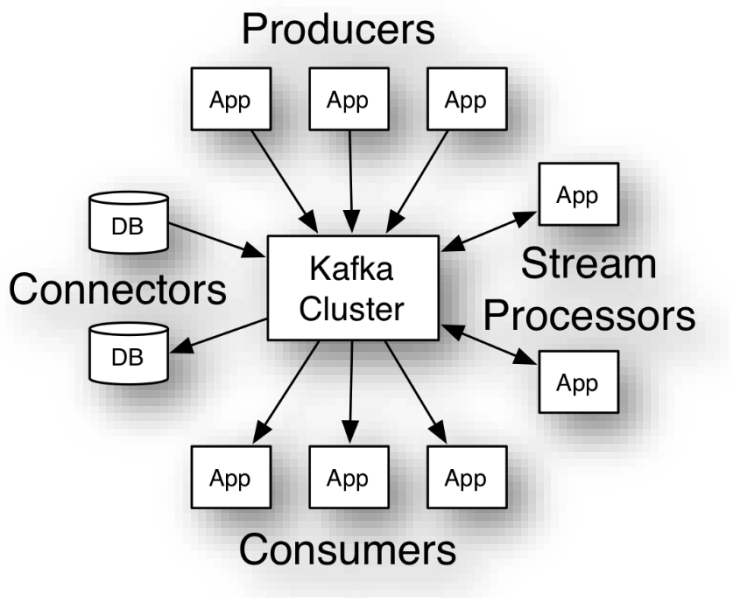


Figure 26: Main parts of Apache Kafka.

Kafka has four core APIs (see Figure 26): The Producer API[7] allows an application to publish a stream of data objects to several Kafka topics. The Consumer API[8] allows an application to subscribe to these topics to get informed about new data arriving in this topic in order to process that stream of data objects. The Streams API[9] allows an application to act as a processor for streams of data objects, i.e. it consumes an input stream from topics and produces output streams to other topics, effectively transforming the input streams into output streams. So, to realise necessary data transformations when data is flowing from the data sources into Q-Rapids components of the data storage or data analysis layer, the Streams API could be used to delegate the transformations to the ingestion layer if appropriate. Still, each plugin at a data source might perform such transformations on its own before sending it to Kafka. However, using the Streams API would be more efficient if there are different data sources with common transformation requirements. The Connector API[10] allows building and running reusable producers or consumers that connect Kafka topics to existing applications or data systems. For example, a connector to a relational database might capture every change to a table.

Kafka Connect (a component of open-source Apache Kafka) is a framework for scalable and reliable Kafka connections with external systems such as databases, key-value stores, search indexes and file systems. It makes it easy to quickly define connectors that move large data sets into and out of Kafka. Source Connectors import data from another system (e.g., a relational database into Kafka) and Sink Connectors export data (e.g., the contents of a Kafka topic to an HDFS file). So, Kafka Connect can ingest entire databases or collect metrics from application servers into Kafka topics, making the data available for stream processing with low latency. An export connector can deliver data from Kafka topics to external systems like the Q-Rapids data storage layer (e.g., into HDFS, MySQL, etc.) or the Q-Rapids analysis layer (e.g., into Apache Spark for analysis, or into Solr or Elasticsearch to realise fast search capabilities). There exist Kafka Connectors[11] for databases

[6] https://kafka.apache.org/documentation.html
[7] https://kafka.apache.org/documentation.html#producerapi
[8] https://kafka.apache.org/documentation.html#consumerapi
[9] https://kafka.apache.org/documentation/streams
[10] https://kafka.apache.org/documentation.html#connect
[11] https://www.confluent.io/product/connectors/

(e.g., using JDBC-Connector) as well as other systems like Jenkins[12] for collecting information about build and deployment processes.

Apache Kafka™ is a scalable, reliable and fast framework for realising the ingestion layer of the Q-Rapids platform and making use of data collected from different software systems. But the number of existing Kafka Connectors is limited. So, before spending effort on realising a new Kafka Connector to connect a software system, one might use Apache Camel™[13] connectors instead. Camel is less scalable and efficient than Kafka and should not be used for large and/or fast data flows, but it provides a lot of out-of-the box components to connect software systems with each other, and it already has a connection component for connecting to Kafka.

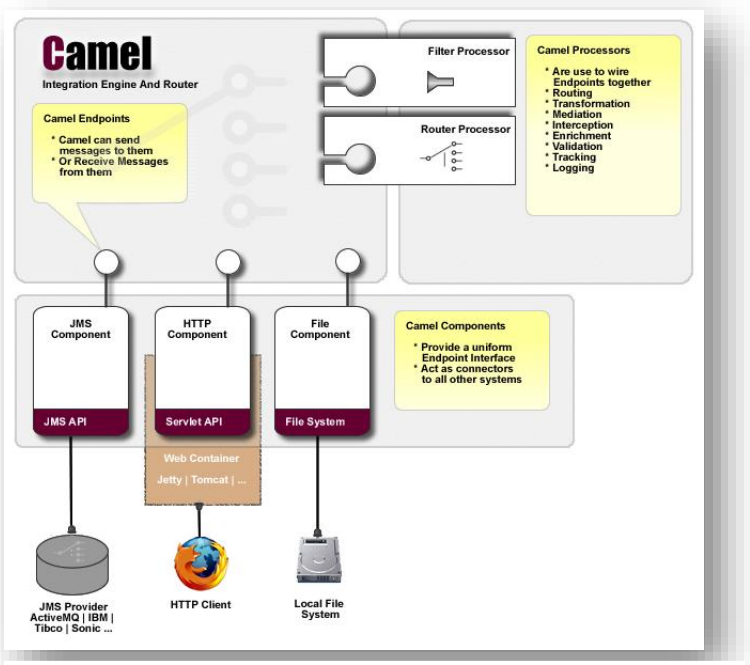


Figure 27: Apache Camel architecture.

Camel realises many Enterprise Integration Patterns (EIP)[14] based on design patterns from asynchronous messaging systems. This is realised by defining data flow routes between the connection endpoints of Camel components. Camel Components provide uniform Connection Endpoints for the data flow routes, and the data flow can be controlled using Camel Processors just like using the Producer and Consumer API of Kafka (e.g., to perform data transformation, to apply data anonymization, etc.). The principle architecture of Apache Camel is shown in Figure 27. There are Camel Components for HDFS, AWS, Hbase, Cassandra, Spark, File, LDAP, JDBC, (S)FTP, FTPS,HTTP, Rest, ActiveMQ, Kafka, Netty, RMI, JMS, DOCKER, Servlet, DropBox, Gmail, GIT, Github, JIRA, GoogleDrive and others.

7.1.2 Initial tools to be integrated into Q-Rapids for the proof-of-concept.

We will select a subset of the following data sources for the proof-of-concept phase (for the most relevant data sources used by the industry partners, see the data producers in Figure 24). The remaining data sources will be handled in future phases of the project.

The common data sources used by all partners are SonarQube, Jenkins, the code repositories of the partner (e.g., SVN, Git, GitLab) and the issue tracking tool of the partner (e.g., Redmine, GitLab, JIRA, Mantis). Besides, there are optional data sources (in some cases such sources exist, but there are confidentiality issues), such as log files from software used at runtime, network monitoring tools (e.g., Kibana, Zabbios, Nagios) and other continuous integration tools of the partner in addition to Jenkins.

[12] https://github.com/yaravind/kafka-connect-jenkins
[13] http://camel.apache.org/
[14] http://camel.apache.org/enterprise-integration-patterns.html

7.2 Lessons learned and current data gathering and analysis specification (M18)

From M6, the architecture specification presented in Figure 24 has become more mature. Further decisions for the data ingestion and storage layers have been performed and deployed in the use cases. Lessons learned are reported in D1.2. We briefly summarize next the current specification of the Q-Rapids architecture with respect to WP1. Figure 28 shows a current high-level architecture view depicting the modules of the Q-Rapids tool and the data flow. It takes over the idea of the lambda architecture approach used for Big Data solutions. The red part, composed of four layers, is related to the Q-Rapids quality model.

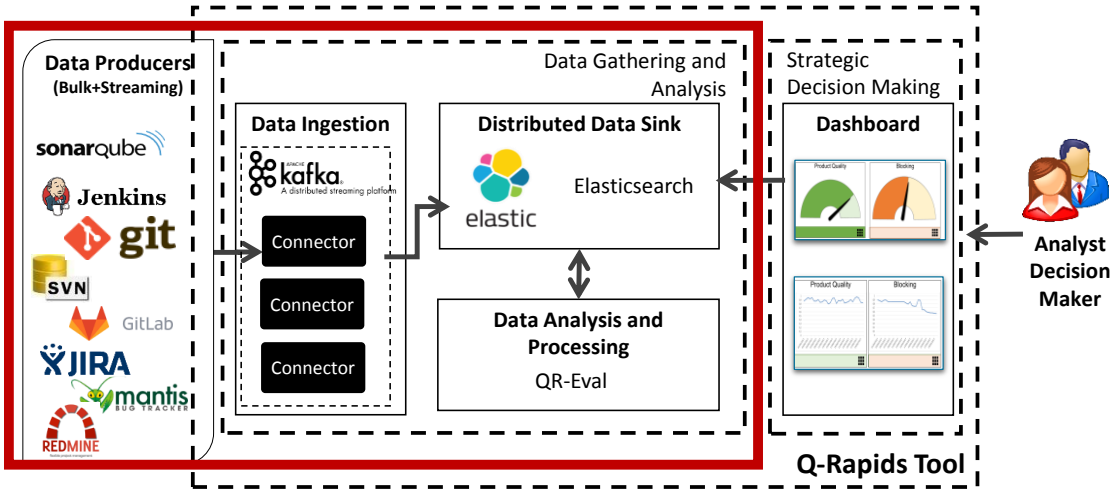


Figure 28. Q-Rapids adjusted lambda architecture at M18. See evolution with respect to Figure 24.

First, the data producers layer consists of external heterogeneous data sources with information about software quality. Currently, the Q-Rapids tool supports data gathering from static code analysis (e.g., SonarQube), executed tests during development (e.g., Jenkins), code repositories (e.g., SVN, Git, GitLab), and issue tracking tools (e.g., Redmine, GitLab, JIRA, Mantis).

Second, the data ingestion layer consists of several Apache Kafka connectors to gather data from data producers. These connectors query the API of data producers to ingest the data into Kafka. For instance, the Jira connector reads all the features from each issue (e.g., description, assignee, due date) from the JSON document got from the Jira API. Apache Kafka is a Big Data technology serving as primary ingestion layer and messaging platform, and offering scalability via clusters capabilities.

Third, the data distributed sink layer is used for data storage, indexing and analysis purposes. Both the raw data (i.e., collected data of different RSD cycles) and the quality model assessment (i.e., aggregations) are stored in a search engine, called Elasticsearch. This allows to define four types of indexes, three for the most abstract elements of the quality model (quality aspects, product and process factors, and metrics), and the fourth for the raw data. As Apache Kafka, it offers scalability via cluster capabilities, which is required when storing huge amounts of data. Besides, we have selected the Elastic stack, due to its capability to quickly perform aggregations, which becomes fundamental for the different levels of the quality model.

Fourth, the data analysis and processing layer performs the quality model assessment. The execution of the quality model assessment is performed by querying the distributed data sink, and applying the utility functions in properties files. This is highly customizable for each use case needs. For instance, the Q-Rapids tool users can set up the quality model (utility functions, weight of product factors, and so on), and the frequency in which the quality model assessment is executed (e.g., daily, hourly).

Inside the Q-Rapids project, WP3 is working on an additional layer with a dashboard, which is relevant for the Q-Rapids tool, but out-of-scope in this document.

Conclusion

This deliverable defines:

1. The current "as-is" scenario of the use cases: the identification of available data sources.
2. The "to-be" scenario: data gathering and analysis user stories to make the generic Q-Rapids quality model operational for the use cases.
3. An architecture including tools and frameworks being used (and that could be used) to collect and analyse data.

By M18, we have implemented most of the elicited user stories. By means of implementing the proposed architecture, we enabled gathering and analysing the data of the Q-Rapids quality model during the previous phase ("proof-of-concept" from month 7 to month 15).

References

[1] M. Galster, P. Avgeriou, Empirically-grounded reference architectures: a proposal, in: Qual. Softw. Archit., ACM, New York, NY, USA, 2011: pp. 153–157. doi:10.1145/2000259.2000285.

[2] E.Y. Nakagawa, M. Guessi, J.C. Maldonado, D. Feitosa, F. Oquendo, Consolidating a Process for the Design, Representation, and Evaluation of Reference Architectures, in: IEEE/IFIP Conf. Softw. Archit., IEEE, 2014: pp. 143–152. doi:10.1109/WICSA.2014.25.

[3] V. Basili, A. Trendowicz, M. Kowalczyk, J. Heidrich, C. Seaman, J. Münch, D. Rombach, Aligning Organizations Through Measurement - The GQM+Strategies Approach, (2014) 205. doi:10.1007/978-3-319-05047-8.

[4] S. Wagner, A. Goeb, L. Heinemann, M. Kläs, C. Lampasona, K. Lochmann, A. Mayr, R. Plösch, A. Seidl, J. Streit, A. Trendowicz, Operationalised product quality models and assessment: The Quamoco approach, Inf. Softw. Technol. 62 (2015) 101–123. doi:10.1016/j.infsof.2015.02.009.

[5] V.R. Basili, G. Caldiera, H.D. Rombach, The Goal Question Metric Approach, in: Encycl. Softw. Eng., Wiley, 1994.

[6] D. Zhang, S. Han, Y. Dang, J.G. Lou, H. Zhang, T. Xie, Software analytics in practice, IEEE Softw. 30 (2013) 30–37. doi:10.1109/MS.2013.94.

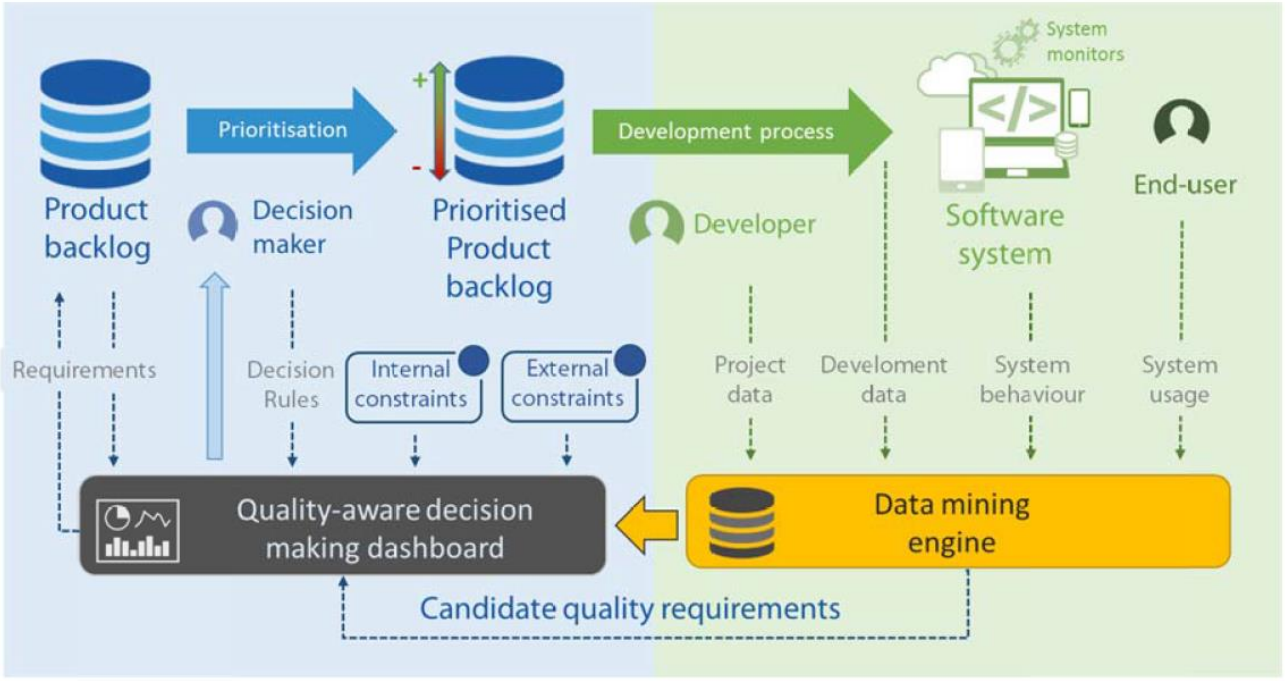
Annex A – Interview scripts for WP1

Less than 5 Minutes *Elapsed* 0

WARM-UP INTRODUCTION

- ✓ *Goal of the Interview*

To collect data for the data gathering and analysis of the Q-Rapids project (focusing on the green/right part).



- ✓ *Structure, related themes, and duration of the interview*

This interview has five parts, with three sections with questions about your use case:

- Introduction (5 minutes).
- Section 1: As-is scenario of your use case (20 minutes).
- Section 2: To-be scenario of your use case (15 minutes).
- Section 3: Acceptance and impact of Q-Rapids (15 minutes).
- Wrapping up (5 minutes).

The complete duration is about 1 hour.

- ✓ *Confidentiality and Sharing the Results*

The information that we collect from this interview will be kept confidential. Information about you that will be collected during the research will be put away and no-one but the Q-Rapids researchers will be able to see it. Any information about you will have a number on it instead of your name. Only the researchers will know what your number is. It will not be shared with or given to anyone except the research team introduced at the beginning of the present document.

Nothing that you tell us will be shared with anybody outside the Q-Rapids project, and nothing will be attributed to you by name. The knowledge that we get from this research will be shared with you and the other participants before it is made widely available to the public. Each participant will receive a summary of the results. The knowledge that we get from doing this research will be published as part of the Deliverables of the Q-Rapids project. In addition, we will publish the results in order that other interested people may learn from our research.

- ✓ *Recording the Interview*

The interview will be recorded (in audio) for later transcription.

- ✓ *Interviewee information*

It is required that the person interviewed has had the role of manager or developer in the use case (or who has the knowledge to respond as such).

Demographic information	
Respondent name	
Respondent email	
Respondent telephone	
Job position in the company	
Role in the selected project (i.e., Q-Rapids use case)	
Years of experience in rapid software development	

- ✓ *Open question*

Do you have any question or comment before starting the interview?

Around
20
Minutes
Elapsed
5

1. As-Is Scenario

We aim to gather information about the current situation of your use case.

As-Is Scenario: Description of the current situation

1.1. Which different data sources and their corresponding (repeatable) technologies do you have at the company for:
- **1.1.1.** Projects (e.g., planning and effort sheets)?
- **1.1.2.** Development (e.g., code repositories, bug reports)?
- **1.1.3.** System behaviour (e.g., system logs)?
- **1.1.4.** Software usage (e.g., profiling data, performance measures)?
- **1.1.5.** User feedback (e.g. questionnaire, complains, maintainability/change requests)?
- **1.1.6.** Any other important source (not to lose any data)?

1.2. Who is the contact person to get access to these data sources and the corresponding documentation? (Note: we are looking for IT of the software company).
** Note: this question was done at the end of the interview not to record personal data.*

1.3. Do you gather information about the software quality of your systems (at run-time)? If so,
- **1.3.1.** What tools do you use for gathering it?

1.4. Do you analyse the software quality of your systems (at run-time)? If so,
- **1.4.1.** What information (metrics) do you use for analysing quality requirements?
- **1.4.2.** What methods or tools do you use for the analysis of quality requirements?

1.5. To analyse the software quality of your systems, do you integrate heterogeneous data sources at your company that complement each other? For example, considering several data sources in your decision making (what tools are used).

As-Is Scenario: Critical analysis of the situation

1.6. What are the main problems/concerns that you have when analysing quality requirements nowadays?

1.7. What is the impact of these problems on the project and business goals?

Around
15
Minutes
Elapsed
25

2. To-Be Scenario: Future with Q-Rapids

To-Be Scenario: Q-Rapids in your use case

Now, we want to discuss with you your expectations on the Q-Rapids framework for managing quality and functional requirements in an integrative manner.

Imagine that the Q-Rapids tool is already implemented and helps you assessing the level of software quality during development and at runtime.

Please, reply only considering your use case.

2.1. What information about actual quality issues (of the software system(s) of the use case) would Q-Rapids provide you?
2.2. What software metrics (about the software system(s) of the use case) would Q-Rapids provide you?
2.3. Which criteria would you use to evaluate the success of the data gathering and analysis of Q-Rapids?

To-Be Scenario: Examples in your use case

2.4. Please, give an specific example of your expectation on how Q-Rapids would help to your use case.

Around
15
Minutes
Elapsed
40

3. Acceptance and Impact of Q-Rapids

The impact of Q-Rapids in your use case

We want to identify possible risks of Q-Rapids adoption

3.1. What quality characteristics should have the Q-Rapids tool to be accepted and used in your company? For example: integration with existing systems (what are those), integration with existing processes (how do they look like), easy to use, do not need effort from the developers, and acceptable by workers committee.

3.2. What can hinder the acceptance and use of Q-Rapids?

The impact of Q-Rapids in practice

3.3. Besides your company, do you think that Q-Rapids could be used in other firms?

3.4. In your opinion, what could be the target community of Q-Rapids?

Less than
5
Minutes
Elapsed
55

WRAPPING-UP

- ✓ Next steps:
  - The responses will be validated by the interviewee, who may complete it or even change the parts that are considered inaccurate or erroneous.
  - During the analysis process, we may need to contact the respondent again to ask for more details on a specific issue.
- ✓ Data access questions: How to get documentation of and access to these data sources?
  - We are going to contact the people from Question 1.2 to get access to these data sources and the corresponding documentation. Next steps with these people:
    A. Questionnaire:
      - *Is there any access, privacy, security issue, constraint about the data sources?*
      - *Is there any documentation about their data structure (e.g., data schema per data source)?*
    B. Telephone interview (if necessary).

Annex B – Data storage: list of valuable attributes

The data storage of the Q-Rapids tool needs to include basic metrics from the Q-Rapids quality model. The tables below show valuable attributes that need to be saved. It is important to note that the tables below report the state at M6. However, in the second phase of the project ("proof-of-concept", months 7-15), we have defined the data structure based on both the real data from each use case and the selected analysis approaches. Updated information can be found in the Annex B of the deliverable D1.2.

The tables below indicate relevant attributes, their data type and a description of different topics:

- Version control data, see Table 9.
- Metrics data (vector version), see Table 10.
- Metrics data (single metric version), see Table 11.
- Task data, see Table 12.
- Issue data, see Table 13.
- Duplication data, see Table 14.
- Testing data, see Table 15.
- Usage data, see Table 16.

Crashes and errors data, see

- Table 17.

Table 9. Version Control Data (e.g. SVN, git)

Attribute	Datatype	Description
systemid	STRING	System versioned
repository_url	STRING	Address of repository
repository_type	STRING	Type of repository (e.g., SVN, git)
revisionkey	STRING (or LONG)	commit id
logmessage	STRING	commit message
Author	STRING	commit author
timestamp	TIMESTAMP	datetime of commit
filename	STRING	changed file
Action	STRING	add, delete, modify
lines_added	INT	size of change [add, modify]
lines_deleted	INT	size of change [delete, modify]
lines_modified	INT	size of change [modify]

Table 10. Metrics Data (Vector Version, example metrics)

Attribute	Datatype	Description
systemid	STRING	Unique id of the analyzed system
Tooled	STRING	Metric Tool
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement
Scope	STRING	The metric scope: system, file, class/interface, method, …
filename	STRING	pathname of the analyzed file
classname	STRING	Class or Interface name, if applicable
methodname	STRING	method name, if applicable

LOC	INT	Lines of Code
LCOM	INT	Lines of Comment
STMT	INT	Number of Statements
MCC	INT	Cyclomatic Complexity
WMC	INT	Weighted Methods per Class
DIT	INT	Depth of Inheritance Tree
NOC	INT	Number of Children
CBO	INT	Coupling between Object Classes
RFC	INT	Response for Class
LCOM	FLOAT	Lack of Cohesion of Methods
Any other relevant code metrics could be added	…	…

Table 11. Metrics Data (Single Metric Version). For each metric e.g., DIT, LCOM

Attribute	Datatype	Description
Systemid	STRING	Unique id of the analyzed system
Tooled	STRING	Metric Tool
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement
Scope	STRING	The metric scope: system, file, class/interface, method …
Filename	STRING	pathname of the analyzed file
classname	STRING	Class or Interface name, if applicable
methodname	STRING	method name, if applicable
metricname	STRING	Name of the Metric
Value	NUMERIC	Value of the Metric

Table 12. Task Data

Attribute	Datatype	Description
task_id	STRING	Unique task id
person_id	STRING	Person responsible
system_id	STRING	System the task is related to
issue_id	STRING	Issue the task is related to, if applicable
Created	TIMESTAMP	Time the task was created
Started	TIMESTAMP	Time the task was started
Finished	TIMESTAMP	Time the task was finished
effort_planned	FLOAT	Effort planned for the task
effort_used	FLOAT	Effort used for the task
effort_remain	FLOAT	Effort remaining for the task
Status	STRING	tbd.

Table 13. Issue Data (useful if they write issue_id during commit info). Maybe to be merged with task

Attribute	Datatype	Description
issue_id	STRING	Unique issue id
system_id	STRING	System the task is related to

Type	STRING	e.g. Bug, Vulnerability, Smell, …
Severity	STRING	e.g. Blocker, Critical, Major, …
reporter_id	STRING	Person that reported the issue
person_id	STRING	Person responsible for resolving the issue
Created	TIMESTAMP	Time the issue was created
resolution	FLOAT	e.g. unresolved, False Positive, Removed, Fixed, No Fix

Table 14. Duplication Data (Software Clones)

Attribute	Datatype	Description
system_id	STRING	Unique task id
tool_id	STRING	Tool used to measure duplication
timestamp	TIMESTAMP	Detection Runtime
duplication_id	STRING	Id of duplication group
filename	STRING	File containing duplicated Lines
startline	INTEGER	Line within file the duplication starts
endline	INTEGER	Line within file the duplication ends
length	INTEGER	Length of duplicated block

Table 15. Testing data.

Attribute	Datatype	Description
Environment_id	STRING	System versioned
system_id	STRING	Unique task id
Result_test	INT	Result code
Coverage	FLOAT	Test coverage percetage

Table 16. Usage data.

Attribute	Datatype	Description
system_id	STRING	Unique task id
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement
line_number	LONG	Line of code
Feature_id	STRING	Unique feature id
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement

Table 17. Crashes and errors data.

Attribute	Datatype	Description
system_id	STRING	Unique task id
timestamp	TIMESTAMP (e.g. ISO8601)	time of measurement
line_number	INT	Line of code