Journal of Ambient Intelligence and Humanized Computing (2019) 10:1927–1936
https://doi.org/10.1007/s12652-018-0784-5

ORIGINAL RESEARCH

Q-Rapids framework for advanced data analysis to improve rapid software development

Rafał Kozik[1] · Michał Choraś[1,2] · Damian Puchalski[2] · Rafał Renk[2]

Received: 10 November 2017 / Accepted: 30 March 2018 / Published online: 10 April 2018

Abstract
The quality of software, in particular developed rapidly, is quite a challenge for businesses and IT-dependent societies. Therefore, the H2020 Q-Rapids project consortium develops processes and tools to meet this challenge and improve the quality of the software to meet end-users requirements and needs. In this paper, we focus on data analytics that helps software development companies evaluate the quality of the software. In fact, most software development teams use tools such as GitLab, SonarQube or JIRA (among others) to assess the basic characteristics and metrics of the developed software. In this paper, we propose the framework that gathers basic data from the mentioned tools, and processes the data further (e.g. using Apache Kafka, Kibana and Spark) to calculate more advanced metrics, product factors, indicators, and to find correlations between them. In this paper, we present the concept, the technical details, and the initial results of the advanced data analysis methodology. Furthermore, we provide discussion on how to use the system and show the future development directions. We have already implemented the system at software development SME and managed to find interesting characteristics and correlations about the software quality. The results, based on the real data, were interesting to the company product owners and team leaders, and more importantly helped them improve the software quality development process.

1 Introduction

1.1 The context

In the current IT ecosystems, where entities and organizations are highly interconnected and relying on software components, challenges such as optimization of the software code development process, minimization of the risk of software failures and code testing/debugging are critical for business, service providers, and societies.

More and more software is developed worldwide and software development projects are increasingly complex. One of examples of code complexity is popular graphics editor—Photoshop, developed by Adobe. An early version of the tool (v1.0, 1990) included approximately 100 thousands of code lines, while the version from 2012 (CS6) had more than 4 million of code lines (increased by 3730%) (Visual 2015). Another estimation of the code complexity shows that all the online Google services are based on about 2 billion of code lines (Visual 2015). That abovementioned numbers and the observed trends, such as IoT, where software components are present in microdevices, cars, smart home systems shift software source code analysis into a challenge related to big data exploration.

However, the problems of ensuring software quality, its assessment and testing are multidimensional. Software failures happening after the product release impact the product vendors' competitiveness, reputation and market position. Moreover, software flaws generate financial losses. As estimated software bugs can decline product stock price with average of 4–6% (for companies experiencing multiple software failures), and further generate almost 3 billion dollars of market losses (QASymphony 2016). In addition, low quality of code significantly impacts the overall cost of the software development, deployment and further maintenance (Jones and Bonsignour 2011). According to QASymphony data (2016), the process of debugging software during its design phase costs 4–5 times less than fixing bugs after its release. Another dimension of software quality is its relation to the level of security. Software flaws and bugs can impact not only its usability, functional value and user experience,

✉ Rafał Kozik
rkozik@utp.edu.pl

[1] Institute of Telecommunications and Computer Science, UTP University of Science and Technology, Bydgoszcz, Poland

[2] ITTI Sp. z o.o., Poznan, Poland

but also security of users, due to the fact that bugs in the design or implementation phase can be exploited by cyber criminals (Choraś and Kozik 2015). According to the article presented by Tovey (2015), consequences of cyber attacks cost about £18 billion per year to British companies in terms of lost revenues.

One of the mechanisms implemented in the software engineering is software testing, with the objective to detect bugs and flaws in the code, and then to address them before the product deployment. However, the costs of the quality assurance and testing in IT are growing from year to year. Currently, IT organizations spent approximately 1/3 of their budgets on quality assurance with the trend of raising this value to approx. 40% in next 3 years (Jorgensen 2016; Capgemini 2017). Although the process of debugging software during its design phase costs 4–5 times less than fixing bugs after its release (Jones and Bonsignour 2011), it is a non-trivial task that consumes a significant part of budgets and effort. It could be less impactful for a big companies and software houses, however SMEs operating often with limited budgets and resources (Felderer and Ramler 2016) are becoming more and more focused on techniques allowing automation and adequacy of the testing process, to be competitive in relation to big players in the market in terms of software quality, optimization of the development cost and time to market.

1.2 Related work

In the H2020 Q-Rapids project (Q-Rapids 2017; Franch et al. 2017; Guzmán et al. 2017), the concept of quality-aware decision making based on key strategic indicators is proposed.

The overall goal of the project is to support strategic decision-making processes by providing strategic indicators in the context of quality requirements in agile and rapid software development. For the purposes of the project, a strategic indicator is defined as a specific aspect that a software development company has to consider as crucial for the decision making process during the software development. Aspects such as e.g. time-to-market, maintenance cost, customer satisfaction, etc. can be considered as strategic indicators depending on the context. Those strategic indicators are built on top of the measurements and factors calculated on the basis of the software development related data, stored in the management tools such as GitLab or SonarQube.

The Q-Rapids project and the concept of measuring quality of software products and processes are not the only approaches in this field. Software metrics, their evolution, distinction between static and dynamic measurements and retrospective analysis of various approaches have been gathered by Voas and Kuhn (2017). Another approach to monitor software development with the use of metrics has been presented by Mäkiaho et al. (2017). Their tool, called MMT was developed to observe and visualize project metrics, to help project managers in reporting, and to ensure awareness of the project status among the team members. Vytovtov and Markov (2017) have presented their approach to source code quality estimation based on software metrics with the use of LLVM compiler, which can assess the code at compile time to provide a programmer information about its current quality.

In addition, there are other related works partially related to our approach, for example where the analysed software code is limited to a specific programming language. Singh et al. (2013), considered the generation of quality metrics for C# code, while Winter et al. (2013) presented the tool for specification and visualization of Java-based source code. Moreover, the authors of several articles focused only on the selected group or type of software metrics, limiting considerations to e.g. analysis of the software maintainability (Hira and Boehm 2016), or software cost estimation (Menzies et al. 2017).

Moreover, numerous works present results and approaches for some background aspects of software development quality, such as study on metrics correlation (Mamun et al. 2017; Kozik et al. 2017) or the aspects of software metrics fluctuation and instability (Arvanitou et al. 2016; Mauša and Grbac 2017).

Yet another aspect is the visualisation of the metrics, where a number of approaches from domains other than software engineering can be employed. For example analysis and visualization of meteorological data (including its historical evolution, similarly to our approach) with the goal to detect and predict anomalies that can lead to critical situations has been proposed by Cipolla et al. (2017).

In this paper we present a proof-of-the-concept of the solution that allows for synchronization and inspection of data related to software development gathered in GitLab and SonarQube tools. Moreover, in the paper we presented preliminary results of tests of our solution in one of the SME companies that run software development projects and commercially develops tools for e.g. healthcare domain.

The rest of paper is structured as follows: in Sect. 2 the proposed system architecture is presented and system use cases, actors and information flow are discussed; Sect. 3 presents an overview on our experiments; Sect. 4 presents experimental results—including code metrics and more advanced correlation results. The applicability of the results and the future work is also presented in Sect. 4. Section 5 concludes the paper and discusses the results.

2 The proposed system architecture

In this section, the architecture of the proposed system is presented. First, we demonstrate the concept of the information flow used for the data analysis. Afterwards, the

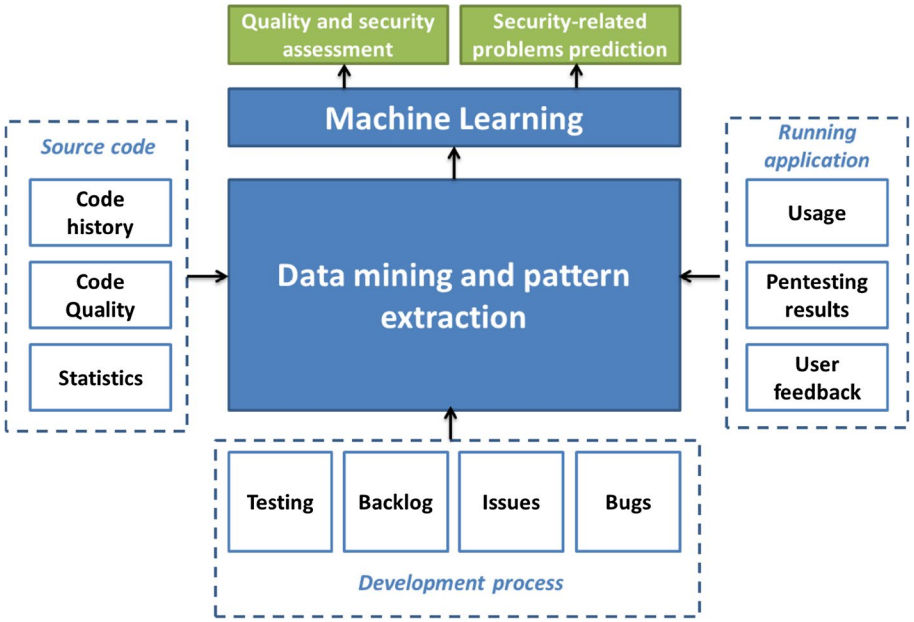

Fig. 1 The conceptual architecture of the proposed solution. Currently, we use the following types of the data sources for our analysis: source code, development process data, and data collected from running application

information data sources (also called data producers) and key system elements are briefly presented.

2.1 The system use cases and actors

The presented system could be devoted to middle-level roles in the organization, such as product owners, SW team leaders, scrum masters etc.

To keep this paper self-containing we introduce following terms that are further used in the paper, namely:

- Issue—as an unit of programming work needed to accomplish some defined progress of software development (e.g. performing test, implementing given feature, fixing bug, etc.).
- Task—is the small (undividable) portion of programmer's work that leads to solve certain development problem.
- Sprint—is the period of time in which programmer's team has to complete specific tasks, after the sprint completion the results have to be ready to review. Each sprint starts with the sprint planning.
- Code quality metrics—are measurable values referred to the certain aspect of developed code (e.g. code complexity, code repetition).
- Strategic indicators—are aggregated information for the decision makers estimated based on the quality metrics and related to the quality requirements.

2.2 The information flow

The conceptual architecture of the system is presented in Fig. 1.

We use several data sources (data producers) to measure the statistics (we call those metrics) related to the project.

Those metrics are retrieved from GitLab[1] and SonarQube[2] project management tools. In the future we plan to extend this list, since in different organizations and software houses, different tools are used (e.g. some teams/organizations may use Jenkins and some GitLab CI instead). However, in many cases it is just a matter of having the right connector between a project management tool and our prototype. Some examples of the used metrics are included in Table 1.

It must be mentioned that the first and the second category of metrics can be obtained from GitLab and SonarQube (from where we currently gather the data). The third category is the plan for the near future.

The collected data is stored in our system for further processing. For example, the project manager has the ability to access a variety of data in one place and use additional functionalities such as visualization, correlation analysis (both presented in the Sect. 3), and prediction of quality metrics.

However, in this paper we particularly focused on a correlation analysis to find relevant relations between the metrics and prove some of our research hypotheses.

[1] https://gitlab.com/.

[2] https://www.sonarqube.org/.

Table 1 Categorization of the metrics used for the data analysis

Category	Type	Metric or information
Source code	Code history	Commits—number, date, detailed description
		Developers—number, names, and involvement
		Branches—names, commits, changes history
	Code quality	Complexity of classes, functions, and files
		Number of duplicated lines and its density
		Comments density and its number
Development process	Testing	Testing time—maximum, minimum, and average
		Passed failed tests indicated for specific functionality, feature or improvement
	Backlog	Features and task planned for sprint backlog
		Time spent to implement specific feature or functionality
	Issues	Number of issues currently opened and recently closed
		Open issues—these bugs, tasks and features that remain unsolved
		Re-open issues—those which has be opened back due to some improvements of modifications
	Bugs	Number of bugs reported to specific functionality or feature
		Bugs criticality
		Time to fix a specific bug
Running application	Usage	Time spent on using particular features (average, max, and min)
		Used features—list of most frequently used functionalities
	Security	Vulnerabilities indicated in the code
		Exploits that have been reported
		Criticality of security flaws
	User feedback	Rates given by users to evaluate usefulness of the application

2.3 The architecture of the system

The current architecture and deployment model is presented in Fig. 2.

There are three key elements namely: our proxy, Apache Kafka,[3] and Apache Spark.[4] Of course, other platforms, e.g. such as Kibana[5] can be also used.

The proxy implements the interface to the external project management tools (currently GitLab and SonarQube). It is responsible for connecting (e.g. via VPN), downloading the data, preliminary pre-processing, and removing sensitive data. The proxy also provides graphical user interface (GUI) for data visualization and system configuration.

The Apache Kafka publish-subscribe system is used to communicate with Apache Spark. In particular, the preprocessed data is published at a specific Kafka topic and further consumed by the Apache Spark framework, where complex and more sophisticated data processing patterns can be used.

Such approach allows for big data processing that will be useful and required to analyze large projects at large organizations.

The proxy design pattern for data gathering has been used in the proposed architecture due to the privacy reasons. When we consulted this architecture with our possible

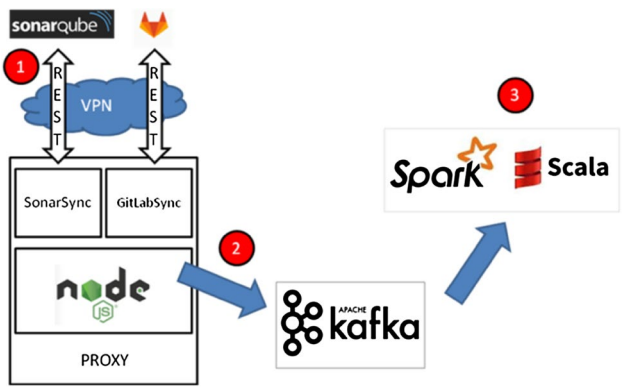

Fig. 2 The conceptual architecture of the proposed solution. The project-related data is collected via VPN (1) at the proxy instance (2) where basic data preprocessing, analysis and anonymisation take place. Computationally expensive operations, data mining and machine learning take place in the Apache Spark cluster (3)

[3] https://kafka.apache.org/.

[4] https://spark.apache.org/.

[5] https://www.elastic.co/products/kibana.

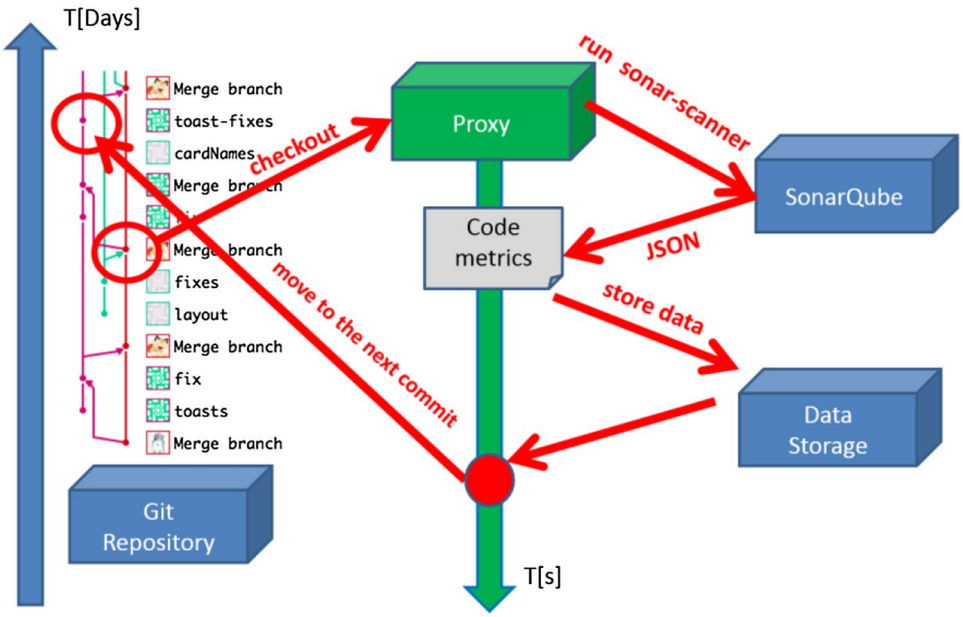

Fig. 3 Procedure of the data acquisition in the process of code metrics calculation

end-users, we learned that customers usually want to have control over the data pushed to a cluster or a cloud.

The described architecture was deployed and used in the experiments (based on real-life data from software development company) described in Sect. 3.

3 Experiments

To execute the experiments, firstly we have gathered, used and analyzed the real-life data collected while developing the real commercial products for customers at SME company that develops software and shared their data with us.

The developed product is a dedicated web-based system for company resources management. The considered development process took almost 5 months and is divided roughly into ten sprints.

The company used GitLab tool to manage the project-related data, namely issues (backlog, user stories, features, tasks, and bugs), source code repository, and continuous integration (CI). To control the quality of the produced code the company has used SonarQube.

The goal of our experiments was to validate the correctness of the architectural assumptions of the Q-Rapids framework, to verify the research hypotheses related to correlation of software quality metrics with the process management characteristics, and to assess the usefulness of the provided functionalities. Also we provided results obtained during the validation of our system to the company representatives.

The data from GitLab and SonarQube tools was collected incrementally as received. The details on the collected data and tentative results have been presented in the next section.

The procedure of data acquisition is presented in the Fig. 3.

As depicted above, the real-project data was obtained from the Git via our proxy and then, SonarQube tool was fed with the data to calculate the code quality metrics.

4 Results

In this section we have presented the results obtained for the proposed system. First, we demonstrate the measurements of two selected metrics—data complexity and a number of code comments.

Afterwards, we present tentative results of the advanced metrics correlation. Finally, we presented the impact of the measured metrics on strategic indicators affecting high-level decision making.

4.1 Code metrics

For the purpose of our use-case and the validation of our solution we focused on two groups of metrics: (a) related to the code complexity (Figs. 4, 5) and (b) related to the code comments density (Fig. 6).

The total number of lines of code in the inspected project is presented in the Fig. 7.

As presented in the figures above, cognitive complexity and duplication of lines of code increase with the increase of a total number of code lines in the project. In the same time, the overall ratio of commented lines to a total number of lines decreases with the progress of development, and reaches only about 3% in the end.

Fig. 4 Cognitive complexity metric

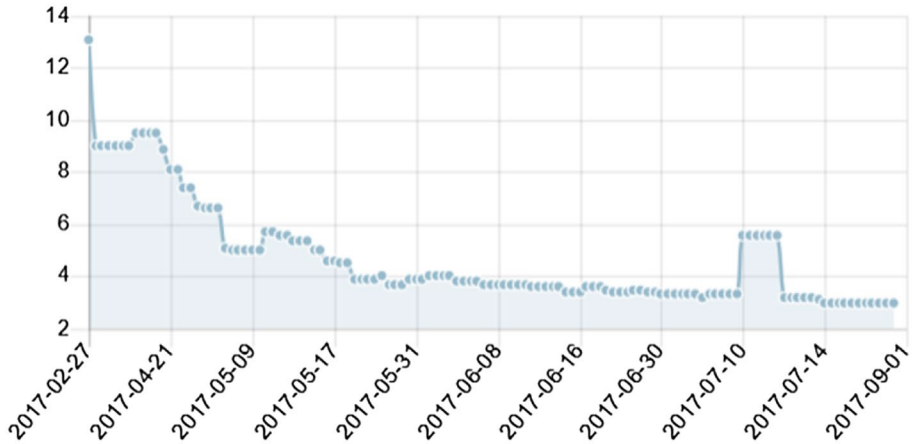

Fig. 5 Total number of duplicated lines in the code

Fig. 6 Density of comments in the code (percentage)

Comparing increment of total number of code lines and comment lines density at the same phases of project, it can be noticed that absolute number of commented lines increased slowly, when code complexity and duplication of lines were significantly raising.

The lessons learnt for this use case is that it could be a serious problem for the further code maintenance and its possible reuse.

4.2 Results of correlation of quality metrics

In our experiments we used gathered data from ongoing software development process to verify the following hypotheses:

1. increasing size of sprint backlog correlates with the increasing cognitive complexity and amount of dupli-

Fig. 7 Total number of code lines

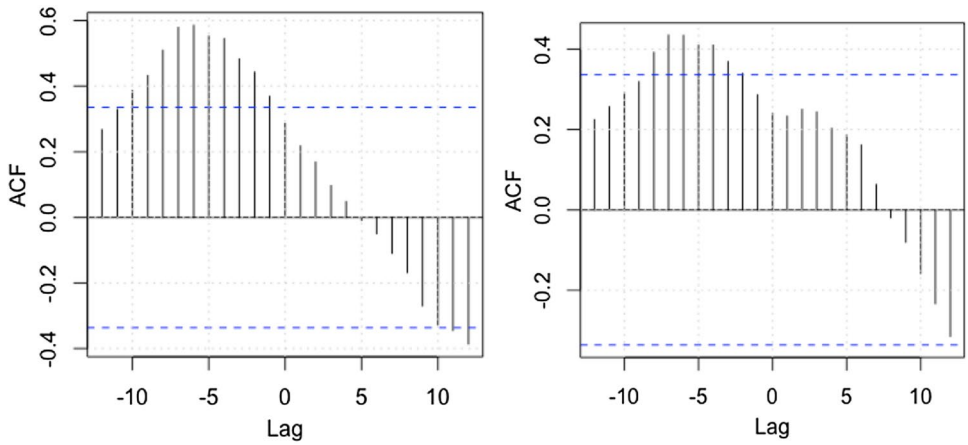

Fig. 8 The correlation (ACF) between the backlog size and the cognitive complexity for varying lag (left) and the backlog size and the number of duplicated lines (right). The dashed lines represent an approximate confidence interval (95%)

cated 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), and

To verify those hypotheses, we have used estimates obtained from cross-correlation function of two time series.

The results verifying the two first hypotheses are shown in Fig. 8.

It can be noted that there exist statistically significant correlation between the size of the sprint backlog and the cognitive complexity of code as well as the number of the duplicated lines. For both of these metrics we have observed higher correlations for negative lags.

It means that the decreased code quality in the future could be the future result of the currently potentially oversized backlog. This turned out to be the valuable information for the product manager/owner, which can help plan the future work.

The estimates of cross-correlation function for the third hypothesis are included in Fig. 9.

It can be noted that these values are high and statistically significant.

However, this relation is not surprising, since usually complex code is prone to errors, in particular when time pressure is presented in the development process.

4.3 Discussion of the results and their context

Adopting the Q-Rapids project approach and Quamoco approach (Wagner et al. 2015) to the problem of modeling of code quality, we assumed the following hierarchy (from the low level to the high-level of the model):

1. Software quality metrics, derived directly from the source code.
2. Product factors, calculated based on the gathered metrics with the defined weights.
3. Quality factors, calculated based on the aggregated and interpreted product factors.

This approach is shown in the Fig. 10.

On the basis of those assumptions, we concluded that:

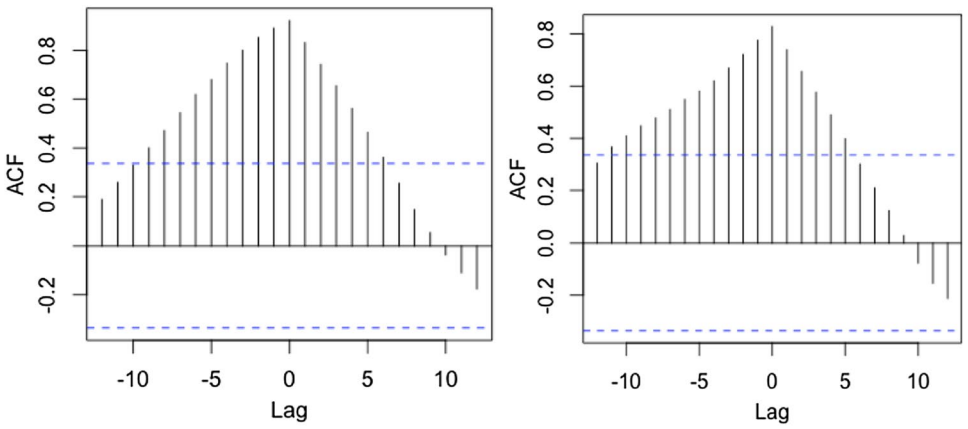

Fig. 9 The correlation (ACF) between the cognitive complexity and number of bugs (left) and the number of duplicated lines with number of bugs (right). The dashed lines represent an approximate confidence interval (95%)

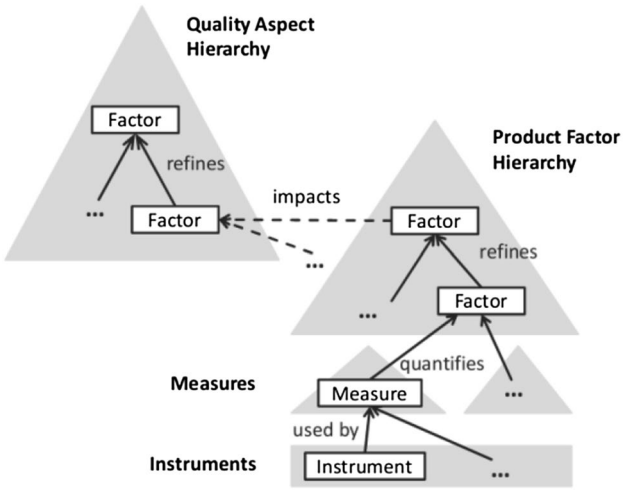

Fig. 10 The quality model concept (Wagner et al. 2015)

- Cognitive complexity and a number of duplicated code lines (code metrics) impact analyzability of code (product factor).
- Cognitive complexity (code metric) impacts adaptability of code (product factor).

and that:

- Adaptability of code and analyzability of code (product factors) impacts maintainability of code, what is important quality factor for the company involved in our use case.

On the basis of the measured code metrics, coming from the real data sources used in the commercial software development project, we concluded that the source code is characterized by the relatively high level of duplication and cognitive complexity (as shown in the Sect. 4.1).

On the other hand, the number of commented lines is low (below 4% in relation to the total number of code lines).

Those metrics negatively impact (decreases) product factors, namely code analyzability and adaptability, what can be significant limitation in case of further reuse of code, decreasing the level of code maintainability. In this case, the maintainability was expressed as quality factor, that determines high-level strategic decision making in the inspected project.

Results of our experiment have been provided to the product owner and top management of the company responsible for the software development.

Therefore, after deploying our solution and our initial findings, the corrective actions were taken in relation to the software development processes in the company.

Our concept and implementation now evolves towards the advanced system that will use the calculated metrics in order to help in meeting quality requirements, better plan tasks and issues, better assign tasks to particular developers, and finally, meet strategic indicators of the product and organization (such as time to market, etc.).

The challenge and the need expressed by senior staff and product owners is the ability to plan and assign tasks and sprints. Indeed, looking at the proposed and calculated metrics allows for taking such decisions with more knowledge and situational awareness, also showing the hidden correlations between various aspects in the projects.

It is worth to mention, that some of the metrics should have high values (in other words high values are desirable), while some other metrics should have lower values (in other words lower values or zeros are desirable) and the proposed solution takes it into account and normalizes the results or adjusts visualization.

Furthermore, the consortium works on the module for generation the quality requirements and matching them to the project status expressed in the calculated metrics. Those automatically generated requirements (quality requirements

for the code) will support the decisions taken by product owners.

Moreover, the what-if analysis could be used to simulate and show the different courses of taken actions [similarly as for example in the critical infrastructures protection services (Kozik et al. 2015)].

5 Conclusions

In this paper, we presented the Q-Rapids framework architecture for advanced analysis of the software quality related data. We presented the concept, architecture, practical deployment at external software development company as well as the initial results. The goal is to support rapid software development process to meet quality requirements and customer needs.

Our initial results prove that after such data analysis the valuable lessons learnt and software development metrics are provided to relevant roles in the company. Our future work is devoted to enlarging the set of the analyzed data sources (e.g. JIRA etc.), calculation of more metrics, product factors and strategic indicators, using the logs from the running products, and last but not least, to provide mechanism prediction based on advanced machine learning techniques (Andrysiak et al. 2014).

Of course, we are continuously in the process of validating our solution at more use-cases internal and external to the project (large and small software development companies).

Acknowledgements This work has received funding from the European Union's Horizon 2020 research and innovation programme under Grant agreement no. 732253. We would like to thank all the members of the Q-Rapids H2020 project consortium.

References

Andrysiak T, Saganowski L, Choraś M, Kozik R (2014) Network traffic prediction and anomaly detection based on ARFIMA model. In: Proceedings of SOCO-CISIS-ICEUTE conference. Springer, pp 545–554

Arvanitou EM, Ampatzoglou A, Chatzigeorgiou A, Avgeriou P (2016) Software metrics fluctuation: a property for assisting the metric selection process. Inf Softw Technol 72:110–124

Capgemini (2017) World quality report 2016–17, 8th edition. https://www.capgemini.com/world-quality-report-2016-17/. Accessed 9 Oct 2017

Choraś M, Kozik R (2015) Machine learning techniques applied to detect cyber attacks on web applications. Log J IGPL 23(1):45–56

Cipolla E, Maniscalco U, Rizzo R, Stabile D, Vella F (2017) Analysis and visualization of meteorological emergencies. J Ambient Intell Humaniz Comput 8(1):57–68

Felderer M, Ramler R (2016) Risk orientation in software testing processes of small and medium enterprises: an exploratory and comparative study. Softw Qual J 24(3):519–548

Franch X, Raty T, Rytivaara V, Ayala C, Lopez L, Martinez-Fernandez S, Partanen J (2017) Data-driven requirements engineering in agile projects: the Q-Rapids approach. In: 2017 IEEE 25th international requirements engineering conference workshops (REW). IEEE, pp 411–414

Guzmán L, Oriol M, Rodríguez P, Franch X, Jedlitschka A, Oivo M (2017) How can quality awareness support rapid software development? A research preview. In: International working conference on requirements engineering: foundation for software quality. Springer, Cham, pp 167–173

Hira A, Boehm B (2016) Function point analysis for software maintenance. In: Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement. ACM, p 48

Jones C, Bonsignour O (2011) The economics of software quality. Addison-Wesley Professional, Reading

Jorgensen PC (2016) Software testing: a craftsman's approach. CRC Press, Boca Raton

Kozik R, Choraś M, Flizikowski A, Theocharidou M, Rosato V, Rome E (2015) Advanced services for critical infrastructures protection. J Ambient Intell Hum Comput 6(6):783–795

Kozik R, Choraś M, Puchalski D, Renk R (2017) Data analysis tool supporting software analysis process. In: Proceedings of 14th IEEE international scientific conference on informatics. IEEE, pp 179–184

Mäkiaho P, Vartiainen K, Poranen T (2017) MMT: a tool for observing metrics in software projects. Int J Hum Cap Inf Technol Prof (IJHCITP) 8(4):27–37

Mamun MAA, Berger C, Hansson J (2017) Correlations of software code metrics: an empirical study. In: Proceedings of the 27th international workshop on software measurement and 12th international conference on software process and product measurement. ACM, pp 255–266

Mauša G, Grbac TG (2017) The stability of threshold values for software metrics in software defect prediction. In: International conference on model and data engineering. Springer, Cham, pp 81–95

Menzies T, Yang Y, Mathew G, Boehm B, Hihn J (2017) Negative results for software effort estimation. Empir Softw Eng 22(5):2658–2683

QASymphony (2016) The cost of poor software quality https://www.qasymphony.com/blog/cost-poor-software-quality/. Accessed 9 Oct 2017

Q-Rapids (2017) EU H2020 project. http://www.q-rapids.eu/. Accessed 9 Oct 2017

Singh P, Singh S, Kaur J (2013) Tool for generating code metrics for C# source code using abstract syntax tree technique. ACM SIGSOFT Softw Eng Notes 38(5):1–6

Tovey A (2015) Cyber attacks cost British industry £34bn a year. http://www.telegraph.co.uk/finance/newsbysector/industry/defence/11663761/Cyber-attacks-cost-British-industry-34bn-a-year.html. Accessed 9 Oct 2017

Voas J, Kuhn R (2017) What happened to software metrics? Computer 50(5):88

Vytovtov P, Markov E (2017) Source code quality classification based on software metrics. In: 2017 20th conference of open innovations association (FRUCT). IEEE, pp 505–511

Wagner S, Goeb A, Heinemann L, Kläs M, Lampasona C, Lochmann K, Trendowicz A (2015) Operationalised product quality models and assessment: the Quamoco approach. Inf Softw Technol 62:101–123

Winter V, Reinke C, Guerrero J (2013) Sextant: a tool to specify and visualize software metrics for Java source-code. In: 2013 4th international workshop on emerging trends in software metrics (WETSoM). IEEE, pp 49–55