Cognition, Technology & Work (2019) 21:191–200
https://doi.org/10.1007/s10111-018-0494-y

ORIGINAL ARTICLE

Increasing product owners' cognition and decision-making capabilities by data analysis approach

Michał Choraś[1,2] · Rafał Kozik[1] · Damian Puchalski[2] · Rafał Renk[2,3]

Received: 13 January 2018 / Accepted: 30 May 2018 / Published online: 5 June 2018

Abstract
In this paper, we focus on the innovative advanced data analysis dedicated for product owners in software development teams. The goal of our novel solutions is to increase cognition and decision making in rapid software development process. The major contribution of this work is the methodology and the tool that gathers the input raw data from the tools such as GitLab or SonarQube, and processes the data further (e.g., using Apache Kafka, Kibana, and Spark) to calculate and visualize more advanced metrics, product factors (e.g., in accordance to Quamoco model), and indicators, and to find correlations between them. Then, such high-level data are shown to product owners to increase their cognition, situational awareness, and decision-making capabilities. We have now implemented the proof-of-concept system which was positively validated by product owners using the real data from their projects.

Keywords Software quality · Data analysis · Cognition and decision support · Product owner · Quamoco model · Product factors

1 Rationale

In the current IT ecosystems, that 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.

Indeed, in the rapid software development process, the product owners have to make quick and efficient decisions on a daily basis. Product owners are in between the programmers, technicians, and testers on one side, and management and CTO/CEO on the other side. They need to have the skills to talk to both of those groups, and usually, they need different languages, arguments, and data to talk and report to them. From product owner's perspective, they plan and monitor the sprints, assign tasks, and lead the programmers work. They also need aggregated business and project management level data to report to the managers in organizations. All the time product owners have to rely on many cognitive processes. It is now obvious that increasing cognition capabilities is an important aspect for, e.g., air traffic control (ATC) operators (Friedrich et al. 2018) and critical infrastructure (CI) operators (Amantini et al. 2012) and such is the case also for software product owners.

In the realistic scenario, product owners have raw data to inspect from standard tools like GitLab, Redmine, SonarQube, JIRA, Jenkins, etc. Product owners look at such data and tools and perform cognitive processes. They look for important aspects in the data and for certain indicators to take effective decisions, which is not the easy duty especially in time pressure. Of course, they try to cognitively match some of the data patterns and indicators with their experience and past cases (e.g., of bad, delayed or good, and high-quality software projects). The process is highly cognitive intensive and requires data inspection (sometimes of big volume of data), data analysis, matching, building hypotheses, predicting, guessing, etc.

Unfortunately, there are no tools to support product owner's tasks with higher level data aggregation, analysis, prediction, and decision support.

✉ Michał Choraś
chorasm@utp.edu.pl; mchoras@itti.com.pl

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

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

[3] Faculty of Physics, Adam Mickiewicz University, 61-614 Poznan, Poland

Therefore, in our work, we try to close this gap by proposing data analysis methods and framework. Our work is the part of H2020 Q-Rapids project (*Q-Rapids—Quality-aware rapid software development*, http://www.q-rapids.eu/) and part of our ongoing work on advanced machine learning and data analysis. It should be stressed that the Q-Rapids has a goal to implement quality requirements into the software development processes, and in particular, our research is oriented on application it to agile and lean software developments, as well as rapid software development processes which are the reality especially at SME type of software houses, where projects start and end quickly, but still have to fulfill customers quality requirements.

The remaining of this paper is structured as follows: in Sect. 2, we overview the context and related work. In Sect. 3, we present the architecture and the chosen proposed solutions, as well as information on implementation and initial results. Conclusions are given thereafter.

2 Context and related work

A large quantity of software is developed worldwide and software development projects are becoming increasingly complex. One of the examples of the 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).

The problems of ensuring software quality, its assessment, and testing are multidimensional. Software failures 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), what generates 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). Software flaws and bugs can impact not only its usability, functional value, and user experience, but also security of users, due to the fact that bugs in the design or implementation phase can be exploited by cyber criminals (Kozik et al. 2016). According to study (Telegraph 2015), consequences of cyber attacks cost about £18 billion per year to British companies in terms of lost revenues.

Currently, IT organizations spent approximately 1/3 of their budgets on quality assurance with the trend of raising this value to approximately 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 companies' 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 with big players in the market in terms of software quality, optimization of the development cost, and time-to-market factor.

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. These 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 our concept of measuring quality of software products and processes are not only approaches presented in this field. Evolution of software metrics, aspects of static and dynamic measurements and state-of-the-art analysis of various approaches are presented by Voas and Kuhn (2017). Monitoring of software development processes based on the software metrics is the core of the approach presented by Mäkiaho et al. (2017) to support project managers in reporting, and to ensure programming team members awareness of the project status. On the other hand, 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. Decision making and related concept of situational awareness—with considerations on risk analysis and associated implications of losing the situational awareness are presented in Dekker (2015) and Salmon et al. (2015).

In our previous Q-Rapids related papers (Kozik et al. 2017, 2018), we showed the presented results of software-related data analysis and correlation.

Hereby, in this paper, we present more advanced solutions that allow for synchronization and inspection of data related to software development gathered in GitLab and SonarQube tools. Hereby, we focus on improving cognition

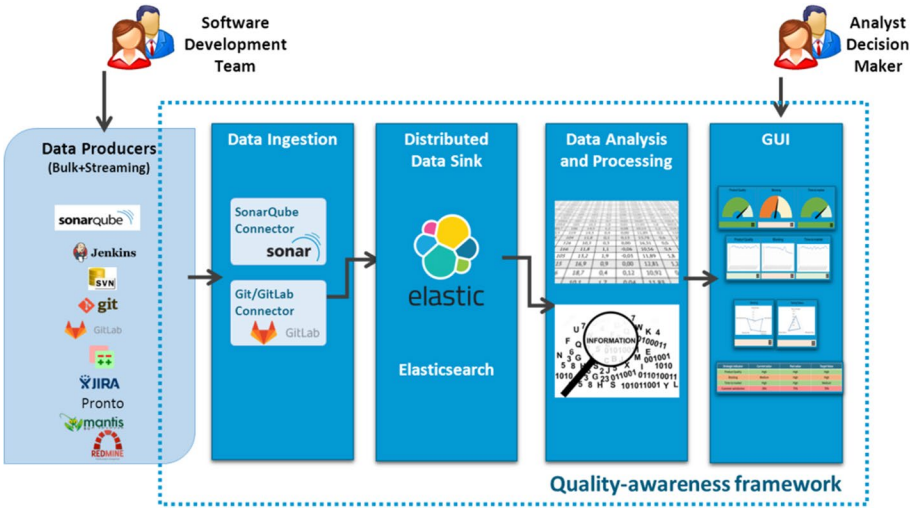

Fig. 1 Conceptual architecture of the proposed solution

of the product owners and decision support by innovative advanced software-related data analysis.

3 Proposed system architecture

In this section, the architecture of the proposed solution is presented. First, we demonstrate the general approach to system architecture, information flow used for the data analysis and the information data sources. Afterwards, we present the subsequent steps of data collection, processing, and presentation to provide Key Strategic Indicators for software engineering product owners and decision makers.

3.1 General approach and data flow

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

We use several data sources to measure the statistics (we call those metrics) related to the software code and software development process. In the current prototype, these metrics are retrieved from GitLab[1] and SonarQube[2] project management tools, taking into account the relevant data protection and privacy regulations and guidelines (Choraś et al. 2015). Git is distributed version control system used to maintain developed code regardless of the development project size. The main feature that characterizes Git is allowing developers to work on multiple local, independent branches that can be merged to the master branch. GitLab is web-based Git repository, offering visualization, issue tracking and project management features, such as task labeling, effort allocation, etc. SonarQube is the tool designed to support continuous inspection of the code quality with features allowing automatic review of the code, bug detection, code security analysis (e.g., vulnerabilities discovery), and static analysis of the code. SonarQube provides its reports based on the number of pre-defined metrics defined to quantify code characteristics, including, e.g., code complexity, testing coverage, etc.

In the future, we plan to extend our solution and not limit data sources to those two data producers, since in different organizations and software houses, different tools are used (e.g., some teams/organizations may use Jenkins and some GitLab CI instead).

It will require the task of implementing the right connector between a project management tool and our prototype. Obviously, interfacing with third-party tools for software project management is not a trivial task. In our case, we have used tools that have advanced and well-documented web interfaces allowing us to retrieve the data in a form of JSON documents. However, it may not be the case for other tools. In particular situations, additional effort will be required (e.g., HTML scrapping) to retrieve the relevant data. In addition, using unofficial, self-written, and customized data crawlers requires additional effort when the format of source data changes.

As shown in the figure above, the raw data maintained in the GitLab and SonarQube tools feed distributed data sink and data analysis and processing modules. To achieve this, we implemented the GitLab and SonarQube connectors focused on the acquisition of the data produced by software development tools. The synchronized data are preliminarily processed, filtered, and anonymized. Synchronization can be done with different time intervals (e.g., once a day) both automatically and manually (please refer to Sect. 3.3). GitLab-based data are additionally preprocessed due to the fact that the structure of such data, scheme for labeling, etc. is highly dependent on the software development methodologies, and definition of development processes used in

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

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

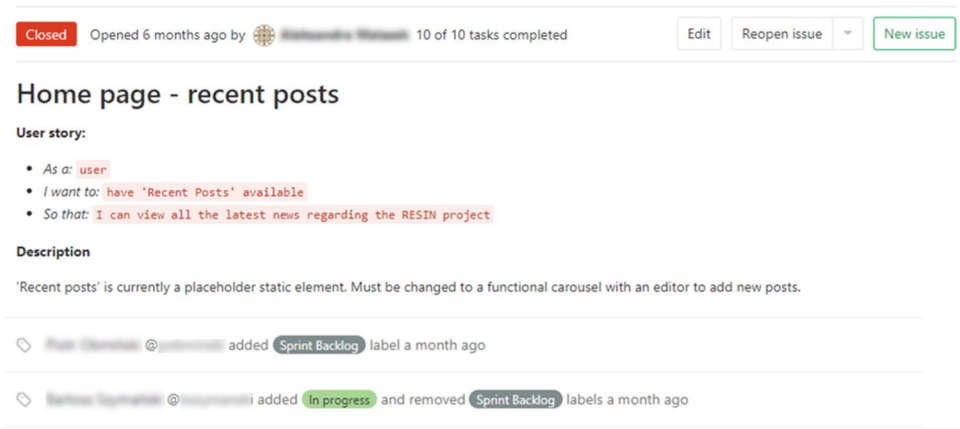

Fig. 2 Example of user story tracking in GitLab

the organization that can be different for different product owners, project managers, or projects even within the same company (e.g., we found out that even teams within the same organization use different label conventions).

3.2 Data producers

For the purpose of Q-Rapids project, we plan to feed our system with three categories of the source data:

- Code-related, including code repository statistics and information from tools used to static code analysis and quality review (e.g., SonarQube). Those can include for example data based on the SonarQube review reports, covering such aspects as code complexity, duplication, maintainability, reliability, security, size, and testing status.
- Development process-oriented, including information from management tools related to software development projects effort allocation, scheduling, issue tracking, etc. Those data can include complete backlog information, in particular time-stamped activities of developers categorized using the labels, backlog size and content (e.g., planned and executed tests, finalized product functionalities), and overall progress of the development.
- Software behavior (data collected at runtime during testing)—based on the usage of already developed and deployed products, including, e.g., aspects of usability, functionalities usage/failures, user-friendliness, etc.

However, at the current stage in this paper, we focus only on the implemented connectors to code-related and process-related data, obtained from GitLab and SonarQube tools.

As presented in Fig. 2, all the issues (e.g., user stories, bugs, etc.) maintained in the GitLab allow for extraction of time stamp for each modification, e.g., adding/removing labels, closing/opening/reopening issues, etc.

On the basis of such data, we can track the course of the given project in terms of work intensification and then combine and/or compare the data to the code-related metrics, e.g., extracted from the SonarQube tool, such as code complexity, number of duplicated code lines in relation with total number of lines, etc.

3.3 Data ingestion

As mentioned in the previous sub-section, the data collection from the currently connected tools (GitLab, SonarQube) can by synchronized and stored for further processing and analysis both automatically with scheduled intervals, as well as manually if needed. Therefore, historical data will be available for the analysis or re-analysis even after the project finalization, e.g., to compare different projects or to add a new data gathered from the other sources. In addition, the evolution of the code-related metrics can be built based on the code changes, tracked in the code repository (e.g., Git repository).

Both GitLab and SonarQube provide WebAPIs for external accessing of the data maintained in the tools. For example GitLab default API[3] allows for gathering issue-related data using JSON format GET requests. More information about SonarQube WebAPI can be found in the official technical documentation.[4] In addition, both tools provide plugin libraries for most of the popular programming languages. However, for our purposes to fully control the analyzed data, and to be able to extend the solution to other tools, we had to develop our own connectors.

[3] https://docs.gitlab.com/ee/api/README.html.

[4] https://docs.SonarQube.org/pages/viewpage.action?pageId=2392172.

3.4 Data modeling, analysis and processing

In the Q-Rapids project, we use the hierarchical model of the software (code) quality proposed as Quamoco approach (Wagner et al. 2012, 2015; Ferenc et al. 2014). The starting point for the Quamoco was two-tier modeling of the software quality applied in the ISO/IEC 25010 standard, where product quality and quality in use is differentiated. In the Quamoco, authors implemented three levels of abstraction, from bottom to up: (1) measures that are quantified into (2) product factors that impact (3) top-level entity—quality aspects. The latter two levels of the hierarchy are quantified factors, i.e., product factors and quality factors, that in this chain indicate on how specific measures (retrieved from the source code), impact product quality, either positively or negatively. Finally, the idea behind the Quamoco was to build a technology-independent quality model, allowing for limiting effort and complexity of building models for specific domains or for specific programming language.

In the Q-Rapids, we also assumed three-level hierarchy (from the low level to the high-level of the model):

1. Software quality metrics, derived directly from the source code/data sources;
2. Quality factors, calculated based on the gathered metrics with the defined weights
3. Key strategic indicators, calculated based on the aggregated and interpreted quality factors.

Based on this hierarchy and available data producers, namely GitLab and SonarQube tools connected to our prototype system, we derived two key strategic indicators supporting decision makers in the task of product development process. Those indicators are "Product Quality" indicator and "Blocking" indicator. According to our quality model, those two indicators are quantified based on quality factors, i.e., "Code Quality" and "Testing Status" constitute "Product Quality" indicator, while factor named "Blocking Code" has a direct impact on "Blocking" strategic indicator. As explained earlier, quality factors are computed based on the metrics obtained from the data producers. Relations between the two bottom concepts of our model are as follows:

- "Code Quality" factor is computed based on three SonarQube metrics (directly related to the code), namely ratio of commented code lines, complexity of the code and density of the code duplication
- "Blocking Code" factor is computed based on the SonarQube metric called "Non-blocking files"
- "Testing Status" factor is computed based on two metrics obtained using the GitLab tool, namely density of non-bugged code, and the percentage of passed tests.

```
--------------------
METRICS INDEX MAPPING
--------------------

PUT poc.metrics
{
   "mappings": {
      "metrics": {
        "properties": {
          "metric": {
            "type": "keyword"
          },
          "name": {
            "type": "keyword"
          },
          "description" : {
            "type" : "text"
          },
          "evaluationDate": {
            "type": "date"
          },
          "value": {
            "type": "float"
          },
          "factors" : {
            "type": "keyword"
          },
          "datasource" : {
            "type" : "keyword"
          }
        }
      }
    }
  }
```

Fig. 3 Example of quality metric index

3.4.1 Data indexing

The next step after the collection of data is the data indexing. In the Distributed Data Sink we use the ElastickSearch cluster. We defined four indexes for three different classes of metrics—metrics are indexed as the raw data and as the normalized data.

As presented in Fig. 3, the metrics index includes such properties as:

- metric: unique identifier of metric used in Kibana/Elasticsearch,
- name: defined name of the metric,
- description: way for the metric calculation,
- evaluationDate: time of calculation,
- value: current value (for normalized metrics can take on values in 0–1 range),
- factors: denotes quality factors constituted from given metric,
- datasource: address of the given index (measure) storage.

Such convention allows us for efficient indexing and calculation of the relevant concepts.

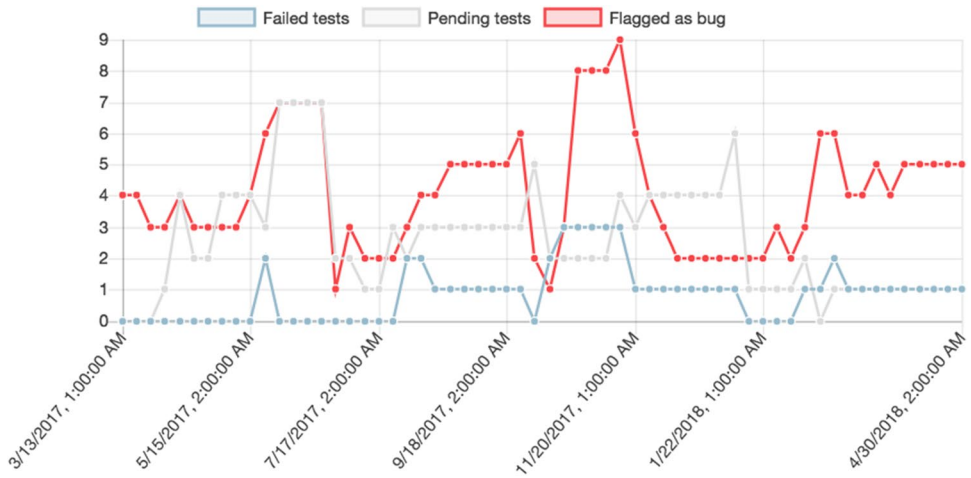

Fig. 4 Example of testing-related statistics visualization (number failed tests, pending tests and test flagged as bug in the GitLab)

3.4.2 Metrics calculation

Basic metrics are retrieved directly from the connected tools, as explained in section presenting data sources. Currently, in practice for real software projects we can collect the following metrics for the further aggregation:

- SonarQube-based
  - Comments ratio,
  - Complexity,
  - Duplication density,
  - Non-blocking files,
- GitLab-based
  - Percentage of passed tests,
  - Non-bug density,

However, a number of basic quality metrics that are not used for calculation of quality factors and quality indicators can be accessed using our connectors and can be visualized to provide added value for decision makers. Those include for example statistics calculated based on the GitLab data, related to programming sprints (e.g., presenting actual work in progress), to sprint backlog content (e.g., presenting ratio of opened/closed/delayed issues in the backlog) and to testing status.

Figure 4 shows the visualization of the project evolution based on the testing performance statistics.

Moreover, the huge number of the SonarQube ratings is accessible using our connectors, such as project files and directory size, maintainability and reliability scores, etc.

3.4.3 Calculation of quality factors and indicators

As explained in the Sect. 3.4, quality factors are calculated based on the metrics that are aggregated using the predefined weights.

Currently we calculate following quality factors:

- Code Quality—based on:
  - comments density/ratio—percentage of commented/annotated code lines in relation with total number of code lines;
  - code complexity scoring—based on number of linearly independent paths within the code, that depends on the number of functions, loops, conditional expressions, etc.;
  - duplicated code density—percentage of duplicated code blocks in relation with the whole code;
- Blocking Code—based on non-blocking files metric, and
- Testing Status—based on:
  - non-bug density—percentage of non-bugged (or debugged) code blocks in relation with the whole analyzed code;
  - percentage of passed tests.

Quality Indicators that are the highest level of the adopted quality model that are currently calculated are:

- Product Quality—based on code quality and testing status;
- Blocking—based on blocking code factor.

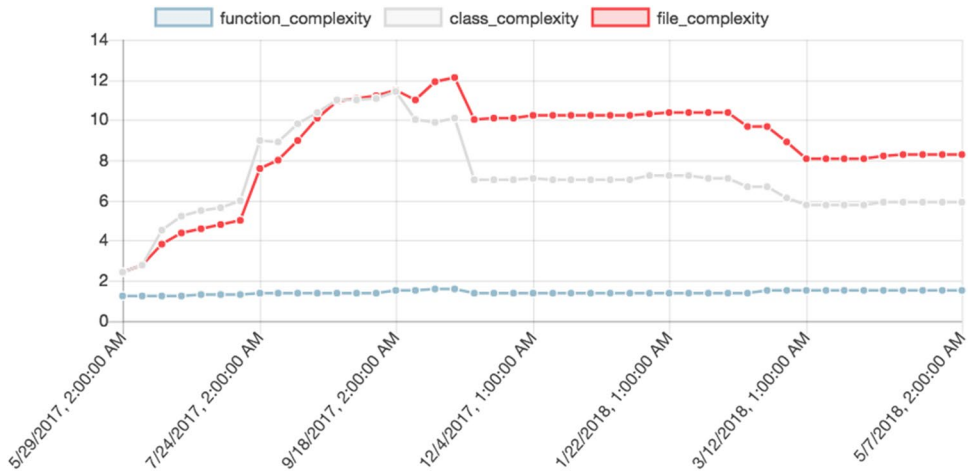

Fig. 5 Visualization of quality metrics (complexity of functions, classes, and file) used to compute the quality factor

3.5 Data presentation/visualization (GUI)

For the visualization purposes in the current version of the system, we decided to use Kibana solutions as well as our own charts. Kibana is an open source for Elasticsearch providing data visualization functionalities. As a part of Elasticsearch platform, Kibana allows to visualize data indexed through Elasticsearch cluster. User has wide capabilities to create and configure different types of charts (such as bar, line, pie charts, etc.) on top of large volumes of data processed by the Elasticsearch. The most common combination of Elasticsearch platform, Kibana visualization tool, and Logstash engine used to data collection and log-parsing is called Elastic Stack.

In the current version of the proposed solution, we provide visualized aggregated data to the end-user through the web-based GUI. Calculated code characteristics (from metrics to strategic indicators) can be displayed in two different modes, including textual presentation of the data, and the data projected on the charts. In addition, the visualization module allows user to display those data calculated for the current point of the project, as well as charts presenting evolution of the metrics, factors, and indicators over the time, from the beginning of the project. This also allows for analysis of historical data (if provided) to the data gathering module. Another configurable property of the data visualization is possibility to adjust grid of the time-based charts to the current needs and present evolution of data with granularity from days up to months.

In Fig. 5, metrics related to the code quality (code complexity, code duplication, and comments density) are presented using time-based chart.

Furthermore, the quality factors can be visualized, including evolution charts and radar charts allowing decomposition of quality factors into more generic metrics and separate analysis of particular quality factor components.

Finally, the most general, top view of the quality analysis is quality indicators allowing decision maker for quick evaluation of the quality-related aspects in particular software development project. This will allow the product owner (or other decision maker) to assess the code quality versus the quality requirements, specific in given company or organization.

3.6 Verification and validation

In this section, we present some results of the preliminary experiments conducted in the real environment. For the experiments, we have used the real data collected while developing real commercial products for customers at software development companies.

The companies 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, SonarQube tool has been used. The data from GitLab and SonarQube tools were collected incrementally as received.

The goal of our experiments was to validate the correctness of the architectural assumptions and to assess the usefulness of the provided functionalities (e.g., to visualize calculated and aggregated data).

The results have been presented at the validation sessions to product owners involved in the development project in their companies. The validation was following our hierarchical approach to build key strategic indicators, i.e., first direct metrics based on the GitLab and SonarQube data were presented, and then calculated quality factors and indicators were discussed.

In general, usefulness of the system has been assessed positively; in particular, aggregated and visualized data related to backlog tracking (as presented in Fig. 6) were evaluated as an added value in relation with the standard capabilities of the GitLab, allowing product owner to verify

Fig. 6 Proposed charts for increasing cognition and decision making related to sprints planning and assessment

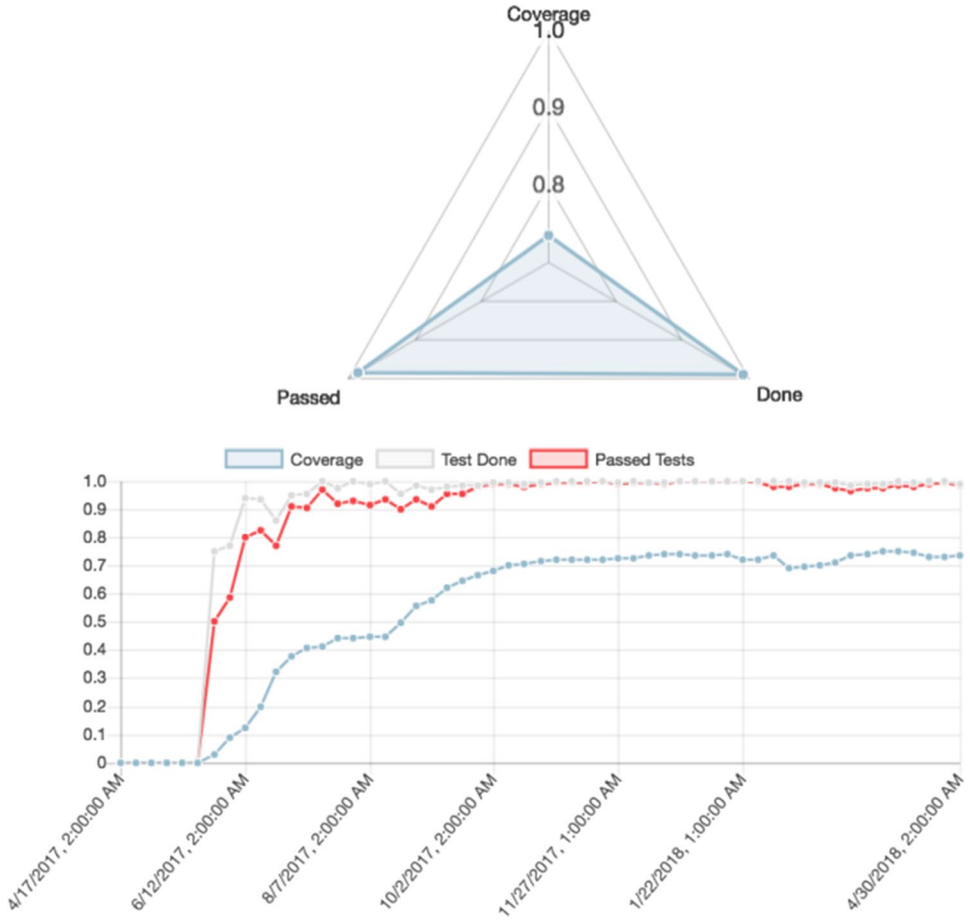

Fig. 7 Example of the current visualization solution for quality indicators

development team productivity and/or correctness of sprints planning.

As for quality factors (e.g., Fig. 7), the visualization was also assessed as useful; however, the product owners expect the capabilities for customization of particular metric weights constituting the factor. These weights can depend on the current project and different priorities defined for this project. In addition, product owners reported that time-based charts could be scaled using "sprint" unit (alongside days and months) with the possibility to define custom duration of the sprint, to better adjust chart grid to the actual milestones defined in the project.

The current implementation allowed to meet the need expressed by senior staff and product owners to plan and assign tasks and sprints in rapid software development processes. 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.

4 Conclusions

In this paper, we described our work on increasing cognition and decision-making capabilities of product owners at software development companies and organizations. We presented the architecture and the advanced data analysis methodology to close the gap between the current software development support tools and the real needs of product owners in rapid software development environment.

Moreover, in the paper, we addressed the validation of our solutions, e.g., in one of the SME companies that run software development projects and commercially develops tools for, e.g., healthcare domain.

Our future work is devoted to enlarging the set of the analyzed data sources and the calculation of more metrics, product factors, and strategic indicators.

Moreover, we currently work on predictive modeling and analysis based on software-related data using the advanced machine learning techniques (Choraś and Kozik 2015). In fact, the prediction of the metrics related to software development process (e.g., number of bugs, time) is currently one of the needs of the product owners. Currently, we work on the time series based approach for such prediction, and we use models such as Holt–Winters, Exponential Smoothing and ARIMA as well as deep neural networks (Andrysiak et al. 2014; Viji et al. 2018).

We also work on finding correlations between various metrics and on the lifelong learning intelligent system approach to use data learnt on past projects and still use the for predictions. The future work is also devoted, for example, to alerting, generation of quality requirements, and what-if analysis that could be used to simulate and show the different courses of actions taken by product owners [similarly as for example in the critical infrastructures protection services (Kozik et al. 2015)].

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

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

Amantini A, Choraś M, D'Antonio S, Egozcue E, Germanus D, Hutter R (2012) The human role in tools for improving robustness and resilience of critical infrastructures. Cogn Technol Work 14(2):143–155

Andrysiak T, Saganowski Ł, Choraś M, Kozik R (2014). Network traffic prediction and anomaly detection based on ARFIMA model. In: International joint conference SOCO'14-CISIS'14-ICEUTE'14, Springer, Cham, pp 545–554

Capgemini (2017) World quality report 2016-17, 8th ed. 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. Logic J IGPL 23(1):45–56

Choraś M, Kozik R, Renk R, Hołubowicz W (2015) A practical framework and guidelines to enhance cyber security and privacy. In: Herrero Á, Baruque B, Sedano J, Quintián H, Corchado E (eds) International joint conference CISIS'15 and ICEUTE'15. Springer, Cham, pp 485–496

Dekker SW (2015) The danger of losing situation awareness. Cogn Technol Work 17(2):159–161

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

Ferenc R, Hegedűs P, Gyimóthy T (2014) Software product quality models. In: Mens T, Serebrenik A, Cleve A (eds) Evolving software systems. Springer, Berlin, Heidelberg, pp 65–100

Franch X, Ayala C, López L, Martínez-Fernández S, Rodríguez P, Gómez C, Rytivaara V (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

Friedrich M, Biermann M, Gontar P, Biella M, Bengler K (2018) The influence of task load on situation awareness and control strategy in the ATC tower environment. Cogn Technol Work 20:205. https://doi.org/10.1007/s10111-018-0464-4

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: REFSQ2017, pp 167–173

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

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, Renk R, Hołubowicz W (2016) Cyber security of the application layer of mission critical industrial systems. In: IFIP international conference on computer information systems and industrial management, Springer, Cham, pp 342–351

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

Kozik R, Choraś M, Puchalski D, Renk R (2018) Q-Rapids framework for advanced data analysis to improve rapid software development. J Ambient Intell Hum Comput. https://doi.org/10.1007/s12652-018-0784-5

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

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

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

Salmon PM, Walker GH, Stanton NA (2015) Broken components versus broken systems: why it is systems not people that lose situation awareness. Cogn Technol Work 17(2):179–183

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

Viji C, Rajkumar N, Duraisamy S (2018) Prediction of software fault-prone classes using an unsupervised hybrid SOM algorithm. Cluster Comput. https://doi.org/10.1007/s10586-018-1923-7

Visual (2015) http://www.visualcapitalist.com/millions-lines-of-code/. 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: Open innovations association (FRUCT), 2017 20th conference of, IEEE, pp 505–511

Wagner S, Lochmann K, Heinemann L, Kläs M, Trendowicz A, Plösch R et al (2012) The quamoco product quality modeling and assessment approach. In: Proceedings of the 34th international conference on software engineering, IEEE Press, pp 1133–1142

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