Overview of an Automated Framework to Measure and Track the Quality Level of a Product

Mariana Falco
LIDTUA/CONICET
Facultad de Ingeniería, Universidad Austral
Pilar, Buenos Aires, Argentina
mfalco@austral.edu.ar

Ezequiel Scott
Institute of Computer Science
University of Tartu
Tartu, Estonia
ezequiel.scott@ut.ee

Gabriela Robiolo
LIDTUA
Facultad de Ingeniería, Universidad Austral
Pilar, Buenos Aires, Argentina
grobiolo@austral.edu.ar

***Abstract*—Product owners need to comprehend the product quality level, in a synthetic and intuitive way to facilitate the decision of accepting or rejecting the iteration. This article presents the basis of an automated framework to measure and monitor the quality level of a software product, within each iteration. This framework is based on the Product Quality Evaluation Method (PQEM), which was designed by the authors and it allows the evaluation of the quality characteristics of a software product, using the Goal-Question-Metric approach, the ISO/IEC 25010, ISO/IEC 25023, the extension made of test coverage concept to quality coverage applied to each quality characteristic, and technical debt and waste. Within the automated framework, the measurement is semi-automatic which is shown in the illustrative example. The development of the framework will begin shortly, and it is expected to carry on new measurements on new iterations of an application.**

***Index Terms*—quality attributes, quality characteristics, quality measurement, quality management, PQEM, automated framework.**

I. Introduction

Modern software development is straight away. Nowadays, companies deploy new code into production weekly and even daily. For example, Amazon unfolds new software every 11.7 seconds [1]. High-performing IT businesses deploy software 30 times more frequently with 200 times shorter lead times [2]. With such fast release times and more frequent releases, it is almost impossible not to see that the quality of software can be impacted by pressure on the speed of releases, increasing the number of defects in production.

Although some authors [2] point out that high-performing companies tend to have fewer amounts of failure, there is a need to understand that it is not correct to accelerate the development while neglecting the fact that quality must be in keeping with the objectives. The adverse impact of low-level quality is much more significant in this changing world, which means that measurement plays a fundamental role in the development of effective and efficient software [3].

Quality plays a vital role in the software life cycle process, and for that after each iteration, it is necessary to analyze the quality level of that iteration before proceeding to the next [4]; a task that it is not easy today. This is because project managers and practitioners doesn't have a synthetic and intuitive way to quickly visualize the quality level of each iteration, and the quality level of each product. Also, the only real way to measure quality in a way that is systematic but also economically feasible is to automate it [4].

Even though there are some manual and automatic ways of analysing quality like SonarQube, there are no fully equipped tools to measure a set of quality characteristics. Based on these challenges, we present a framework which is the automated version of Product Quality Evaluation Method (PQEM) [5], which is a five-step method per iteration, whose main goal is to perform a thoughtful quality assessment of the different iterations within a software product, and that produces a single value between 0 and 1 as the final outcome that represents the product quality level.

This framework, as well as PQEM, is structured around the Goal-Question-Metric approach [6], the set of quality characteristics defined by the standard ISO/IEC 25010 [7], the set of measures defined by the standard ISO/IEC 25023 [8], the extension of coverage testing made by some of the authors to achieve the measurement, and the definition of acceptance criteria related to the expected quality level per each iteration.

Likewise, it includes technical debt and waste. In this context, project managers and quality leaders will be able to visualize the historical progresses for each iteration of each product; as well as establishing a community that set, define and create goals, questions and metrics. The framework embedded the idea that a clear visual presentation often improves understanding and increases the effectiveness of the metrics [3]. It is worth mentioning that the framework has not been developed yet, but this is what some of the authors are carrying out nowadays.

The present article is structured as follows: in Section II, the related work will be addressed, while Section III will describe and characterize each of the steps within the Product Quality Evaluation Method (PQEM). Section IV will delineate the characteristics of the automation as a part of the framework, while Section V will approach the illustrative example of the application of PQEM to the second iteration of a mobile and web. Section VI will address the discussion and threats to validity, and finally, Section VII will describe the conclusions and future work.

978-1-7281-5957-7/20/$31.00 ©2020 IEEE

II. Related work

Quality plays a vital role in the software life cycle process, and for that after each iteration, it is necessary to analyze the result of that iteration before proceeding to the next [4]. PQEM is based on this idea, since it allows evaluating the quality of the product in each iteration, comparing the quality value obtained against the acceptance criteria defined for it. It structures its base of quality characteristics through Goal-Question-Metric, and allows summarizing by characteristic and by iteration by extending the concept of testing coverage [5]. In the same way, it is based on the standards ISO/IEC 25010 and ISO/IEC 25023. PQEM can be applied manually, but in this article its extension is defined as an automatic quality evaluation tool.

A manual approach was presented by Wingkvist et al. [9], who seek to assess and assure the quality of technical documentation, building on the Goal-Question-Metric paradigm and suggesting a metrics-based quality model. Quantitative metrics are collected throughout production and use of the technical documentation. Löwe et al. used VizzAnalyzer [10], which is a software comprehension framework to asses the quality of the documents. Apel et al. [11] presented the quality assessment of the microservices, DevOps, and containerization architectures by defining a large number of metrics. As in PQEM, the approach address the following quality characteristics: Portability, Maintainability, Performance Efficiency, Functional Suitability, Reliability, Compatibility, and Security.

Mostly, automation is easily identifiable in testing as it significantly reduces testing time and avoids repetitive tasks, aiding in improving the quality of the software [12]. For example, SonarQube[1] allows for carrying out a continuous code inspection to detect bugs and vulnerabilities in the branches and pull requests of the project.

In this line, Shchramme et al. [13] have proposed a solution to analyze and measure usability metrics during the implementation phase. Specifically, they have developed a framework featuring code annotations that provides a systematic evaluation of the usability throughout the source code. Their framework can be applied under a role-based approach, where different technical users can use the annotations to inspect quality issues related to usability. The annotation processor is in charge of automatically processing the annotations in the code and calculate the metrics at compile time (i.e., with no need to execute the code), providing numerical results for the usability metrics, in terms of percentage values, according to the calculations and optimal values specified in ISO/IEC 25023 [8].

Also, the literature addresses perspectives related to low level measurements (product and process metrics) which are used to predict and control higher level quality attributes. Schrettner et al. [14] present an approach for modeling, collecting, storing and evaluating such software measurements, which can deal with all types of metrics collected at any stage of the life cycle. They have developed the Unified Quality Monitoring (UQM) application, which integrates the collection of quality data, high level query management and reporting features.

Mayr and others [15] have defined a quality model for embedded systems, an approach for iteratively developing a quality model for specific types of software, and support for performing largely automated quality assessments of ES source code. In this way, they addressed the operationalization of the quality model regarding (semi-) automatic quality assessments to get interpretative quality statements on higher abstraction levels. This assessment model enables collecting required measurement data, evaluating the collected data, and aggregates these results according to the developed quality model. Additionally, they adapted the open-source quality analysis toolkit ConQAT [16] to cope with the requirements that emerged from their assessment model.

It is worth mentioning that the Continuous Quality Assessment Toolkit (ConQAT) provides the tool-support required to enact continuous quality control in practice, by supporting the rapid development of quality dashboards that integrate diverse quality analysis methods and tools. Through advanced aggregation and visualization mechanisms, these dashboards enable developers and project managers to track key quality aspects of software projects in an efficient and timely manner.

The dashboards are accompanied by a set of interactive tools that support the in-depth inspection of identified quality defects and help to prevent the introduction of further deficiencies. Also, ConQAT is not limited to the analysis of source code but takes into account various types of other development artifacts like models or textual specifications [16]. Finally, Tsuda et al. [17] have establish a SQuaRE-based comprehensive software quality evaluation framework, called Waseda Software Quality Framework (WSQF), which implements many product quality and quality-in-use measurement methods originally defined in the SQuaRE series [18].

While automation tools exist, not all of them address a comprehensive analysis of all quality characteristics or associated metrics (whether they are defined by ISO/IEC 25023 [8] as well as new ones), nor do they calculate quality level in a multidimensional aggregated number as it does PQEM. Therefore, the automation of this method will serve as a means of interface with what exists or what will be developed in the future, complemented by the standards, in order to achieve a comprehensive tool that will allow decisions to be made in each iteration of a software product.

III. Product Quality Evaluation Method

Product Quality Evaluation Method (PQEM) [5] is a five-step method per iteration, whose main goal is to analyze, study, measure and assess the quality level of the different iterations within a software product, and that produces a single value between 0 and 1 as the final outcome that represents the product quality level. This is basically the degree to which the software product fulfills its quality attribute requirements [19].

[1]SonarQube web site – https://www.sonarqube.org/

PQEM is structured by the standard ISO/IEC 25010 [7], which provides a basis to analyze a software product with respect to the following set of quality characteristics: Functional Suitability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability; and around the standard ISO/IEC 25023 [8] which defines quality measures for quantitatively evaluating system and software product quality in terms of the previous characteristics. The five steps of PQEM are described in Table I.

TABLE I
THE FIVE STEPS OF PQEM

Step	Description
1	Product setup
2	Elicitation of Quality Attributes Requirements
2 (a)	Select quality characteristics and sub-characteristics
2 (b)	Specify quality attributes requirements
2 (c)	Define metrics of each quality attribute requirement
2 (d)	Define acceptance criteria of each quality attribute requirement
3	Measure and test each quality attribute requirements
4	Collect and synthesize results
5	Assessment of the product quality level
5 (a)	Collect measurement

The *product setup* is where the stakeholder defines the amount of expected iterations of the software product, and the definition of an acceptance criteria for the expected quality level per iteration. PQEM is based on the Goal-Question-Metric approach [6], a set of requirements, quality characteristics and metrics to measure their fulfillment are elicited for later aggregation.

The *elicitation* process of functional and non-functional requirements which are identified as a Quality Attribute Requirement (QAR), is followed by: a) the measurement itself, b) the collection and synthesizing of the results that include the implementation of the extension of the testing coverage [20] as a quality coverage, and c) the final assessment of the product quality level obtained. The synthesis of the results is done by the equations shown in Table II.

As such, in Table II the labels are as follows: q identified each quality characteristic, i identifies each iteration, n is the number of quality characteristics defined, $OCqi$ is the obtained coverage per quality characteristic for each iteration, $NpQARqi$ is the number of passed QARs per quality characteristic for each iteration, $NQARqi$ is the number of QARs per quality characteristic for each iteration, $ECqi$ is the expected coverage per quality characteristic per iteration, $TNQARi$ is the total number of QARs per iteration, $OvCqi$ is the overall coverage per quality characteristic per iteration, $TECi$ is the total expected coverage per iteration, and $TOCi$ is the total obtained coverage of QARs per iteration.

It is worth mentioning that the five steps described in Table I are repeated for each iteration within the product life cycle. Also, the PQEM method is semi-automated due to the fact that each step can be done manually, while the measurement itself can be done through the aid of software in order to reach the coverage calculations and the quality level. This characteristic makes the method suitable to be applied in any software development method that defines iterations, like most of the agile methods. The model can also be applied to different contexts within academia and industry, in which there is a need to monitor and control the quality level of a product per iteration.

TABLE II
EQUATIONS FOR THE QUALITY ASSESSMENT IN PQEM

Equation	Calculation	
OCqi	$\frac{\text{number-of-passed-QARqi}}{\text{number-of-QARqi}} = \frac{NpQARqi}{NQARqi}$	(1)
ECqi	$\frac{\text{number-of-QARqi}}{\text{total-number-of-QARi}} = \frac{NQARqi}{TNQARi}$	(2)
OvCqi	$\frac{\text{number-of-passed-QARqi}}{\text{total-number-of-QARqi}} = \frac{NpQARqi}{TNQARi}$	(3)
TECi	$\sum_{q=1}^{n} ECqi = 1$	(4)
TOCi	$\sum_{q=1}^{n} OvCqi$	(5)

IV. AUTOMATED QUALITY MEASUREMENT FRAMEWORK

The framework provides the stakeholders the possibility of understanding and visualizing the evolution of the product quality level through the life cycle of the product. The components to be defined in PQEM, such as the metrics or the catalogue selection, are for the user's view, as if they were a check list of what or not included for the quality analysis of the project. One point must be clear: the framework is automated and the measurement is semi-automated.

A. Standards: ISO/IEC 25010 and ISO/IEC 25023

The ISO/IEC 25000 known as SQuaRE (System and Software Quality Requirements and Evaluation) is a family of standards that aims to create a common framework for evaluating the quality of software products. This family is made up of five divisions: quality management, quality model, quality measurement, quality requirements, and quality assessment [18]. Within the quality model division, we used the ISO/IEC 25010 [7] which describes the quality model for the software product and for the quality in use.

This standard presents the characteristics and sub-characteristics of quality against which to evaluate the software product. And also, the ISO/IEC 25023 [8] specifically defines the metrics for measuring the quality of software products and systems. ISO/IEC 25023:2016 defines quality measures for quantitatively evaluating system and software product quality in terms of characteristics and sub-characteristics defined in

ISO/IEC 25010 and is intended to be used together with ISO/IEC 25010.

B. Technical debt and waste

Technical debt (TD) is defined as a metaphor that reflects technical compromises that can yield short-term benefit but may hurt the long-term health of a software system [21]. There is a need of more empirical studies on the application of specific technical debt management (TDM) approaches in industrial settings. Moreover, dedicated TDM tools are needed for managing various types of technical debt in the whole TDM process. Consequently, the PQEM tool incorporates a level of technical debt management in order to analyze for example, the following types of TD: requirements TD, architectural TD, design TD, code TD, test TD, build TD, documentation TD, infrastructure TD, versioning TD, and defect TD. It is worth mentioning that this will be possible by classifying each QRA that you obtain as a result of the evaluation as "not passed".

Later on, and deriving from Lean development, it is feasible to approach the concept of waste understanding it as that which does not add value [22], and waste types can be interpreted as those additional functions, changes of tasks, additional processes, functionalities partially realized, movement, defects and creativity of unused employees [23]. As such, waste is measured from a "trash can" where will be thrown away all artifacts that are discarded per iteration, and within the same trend graph it will be displayed the technical debt and waste values.

C. API

The tool is able to integrate different automated tools that measure quality attributes in order to obtain a full quality analysis of each software product analyzed. For example, it is possible to connect to SonarQube. This will lead to obtain an interconnected quality platform, instead of individual ways of measuring quality.

D. Catalogue

The PQEM tool will provide a catalogue of metrics applied in the industry, in order to bring a complete list of ready-to-apply metrics: containing a catalogue of metrics defined for the quality characteristics that constitutes the ISO/IEC 25010, a catalogue with the set of metrics defined by the ISO/IEC 25023 [8], a catalogue of goals, questions and metrics defined for quality analysis and applied to industry, which were found in a literature survey done by the authors but not yet published. Also, it is possible to build a personal catalogue reusing measures defined in other projects, and the others catalogues described earlier.

E. Community and traceability

In this way, and with the creation of specific catalogues or metrics, it will be possible to promote the generation of a community that identifies the metrics used, the most common and those necessary to add. Also, PQEM as a tool provides the ability to visualize and compare the progress of each product of a company, also making it possible to carry out historical analyzes between products from the same company, giving a particular perspective for developers by identifying points for improvement and a global perspective for project leaders and managers.

V. Illustrative example and results

This section summarizes how the implementation of PQEM resulted in the second iteration of a web and mobile application, called HeartCare embedded in the health field. In this way, HeartCare [5] ensure that the physical recovery of cardiac patients who are part of a cardiac rehabilitation program can take place in an environment outside hospitals.

PQEM gave a total quality level for the second iteration of HeartCare a value of 0.90. It was previously defined with stakeholders that the quality acceptance criterion for said iteration was 0.70 (completing *Step 1* in Table I). Within *Step 2* the quality characteristics (as quality attributes requirements or QARs), questions, metrics and acceptance criteria as well as the results are stored in a structured artifact (spreadsheet), as shown in Table III. The *Result column* contains the result of the measurement made per row for all QARs (1 passed, 0 failed).

Based on the stakeholder needs, different quality characteristics from ISO/IEC 25010 and ISO/IEC 25023 have been selected. In order to fill the Result column in Fig. 1 (Table I, *Step 3*), it was carried out an analysis of the HeartCare implementation in order to indicate whether the question was implemented or not, allowing later to explicit define a passed or failed result. Each question may need to carry out tests or counts to achieve an answer, for example to fill the row with ID 17 shown in Table III, the incompatibility errors within the web version, the mobile version and the sensor were summarized. Only one compatibility error were found, and so this QAR was set as passed. This same procedure was performed for all the QARs.

Step 4 in Table I defines a set of equations that represent an extension of the testing coverage, to calculate the coverage of each QAR, which are defined in Table II.

The selected quality characteristics (abbreviated as QC on Table IV) by the stakeholders were Availability, Fault-Tolerance, Recoverability, Functional Suitability, Interoperability, Modifiability, Performance Efficiency, Security, Usability, and Portability. Following the Equations described in Table II, we calculate the coverage value for Recoverability as $OC_{q2} = \frac{NpQAR_{q2}}{NQAR_{q2}} = \frac{5}{7} = 0.71$. Since the acceptance criteria for Recoverability was defined as 0.70; and according to Table IV, the quality level TOC for the second iteration was 0.90. So, we concluded that the product iteration is accepted as TOC is bigger that the quality level previously defined.

When compared with the TOC value obtained for the previous iteration, the application reached the acceptance criteria without an outstanding difference (only with 0.075) and with a bigger technical debt (0.225) [21]. It is worth mentioning

TABLE III
ARTIFACT TO STORE DATA. EXAMPLE FOR FAULT-TOLERANCE.

ID	Quality Characteristic	Question	Quality Measure	Acceptance criteria	Result
13	Fault-Tolerance	Are the amount of crashes under control?	Number of crashes	Number <10	Passed
14	Fault-Tolerance	Are the amount of hangs under control?	Number of hangs	Number <10	Passed
15	Fault-Tolerance	Are the amount of incorrect functional responses under control?	Number of incorrect responses	Number <15	Passed
16	Fault-Tolerance	Are the amount of updates requiring restart under control?	Number of updates requiring restart under control	Number <4	Passed
17	Fault-Tolerance	Are the amount of incompatibility errors under control?	Number of incompatibility errors	Number <4	Passed

TABLE IV
SUMMARY OF RESULTS FROM THE APPLICATION OF PQEM TO THE SECOND ITERATION OF HEARTCARE

QC	$NQAR_{q2}$	$NpQAR_{q2}$	OC_{q2}	EC_{q2}	OvC_{q2}
Availability	12	10	1	0.086	0.086
Fault-Tolerance	5	5	1	0.02	0.02
Recoverability	7	5	0.71	0.03	0.02
Funct. Suitability	59	56	0.95	0.23	0.22
Interoperability	6	4	0.67	0.02	0.02
Modifiability	59	59	1.00	0.23	0.23
Performance Eff.	17	15	0.88	0.07	0.06
Security	19	13	0.68	0.07	0.05
Usability	64	58	0.91	0.25	0.22
Portability	10	8	0.80	0.04	0.03
Total	258	233		1.00	0.90

that in PQEM, TD is calculated as 1-TOC and for this second iteration of HeartCare the TD value is 0.1.

The above describes the PQEM application in the second iteration of a web and mobile application. The results obtained serve as a means of testing not only for the method, but also for understanding the points that need to be improved for the next iteration of the HeartCare application.

VI. DISCUSSION

Implementing PQEM as a semi-automatic tool could benefit developers in their everyday's activities, especially those that are related to software quality control. However, the implementation of PQEM introduces several challenges. In the following, we discuss what the main challenges are and a set of proposals that aim to address those challenges.

The first steps of PQEM (see Table I) involve a set of decisions, which are the most challenging to be automated. In Step 2, the selection of the quality characteristics and sub-characteristics, the definition of the Quality Attribute Requirements (QARs), and the definition of the metrics along with the QARs acceptance criteria are hard to automate since these steps are subject to many constraints such as the application domain, the business rules, the target users, and even the technology used. In consequence, those decisions should be made by stakeholders and be specified by them as inputs of the tool. Although this intervention makes the approach semi-automatic, a possible way to make easier these decisions is to offer a set of predefined quality characteristics, QARs, and acceptance criteria according to the characteristics of the product or application domain. For example, one of the predefined guidelines is formed by the set of quality characteristics and metrics defined by the ISO/IEC 25013 [8].

Another challenge of adding automation to PQEM is the definition of test cases that will be related to the QARs. Test cases play an important role in PQEM since their results impact directly on the obtained coverage (see Table I, Step 3). In PQEM, test cases should be associated with the QARs, and it can be possible to have test cases associated with more than one QARs. Although the links between QARs and test cases should be also defined by the stakeholders, several techniques and tools can help in the automatic execution of the test cases as well as the further collection, calculation, and synthesis of the metrics.

Our proposal to (semi) automate PQEM is based on a tool chain that includes several existing tools. The tool chain could support the definition of test cases using different techniques, in the context of the quality characteristics defined by ISO/IEC 25013 [8]. Figure 1 shows how a set of existing tools can work together. First, developers (or any other stakeholder) have to define the QARs and record them in a project management tool such as JIRA[2]. In this sense, QARs can be treated as any other issue in the issue-tracking system. It is worth mentioning that QARs are a key part from the PQEM method, and so the framework will endure the idea of providing an API to integrate different tools as shown in the tool chain (see Figure 1) that measure those QARs in order to obtain a full quality analysis of each software product analyzed.

In the next step, developers complete the implementation of the QARs supported by development environments such as Eclipse[3] or IntelliJ IDEA[4]. Once the implementation phase is completed, they send the updated source code by using recommended configuration management practices that include version control (Git[5]) and build-automation (Gradle[6]) along with a continuous integration server (Jenkins[7]).

These practices are already well-known software engineering practices that most of the companies apply nowadays and they are essential to provide automation with PQEM. The CI

[2]Attlassian web site – https://www.atlassian.com/software/jira
[3]Eclipse web site – https://www.eclipse.org/
[4]Intellij IDEA web site – https://www.jetbrains.com/es-es/idea/
[5]Git web site – https://git-scm.com/
[6]Gradle web site – https://gradle.org/
[7]Jenkins web site– https://jenkins.io/

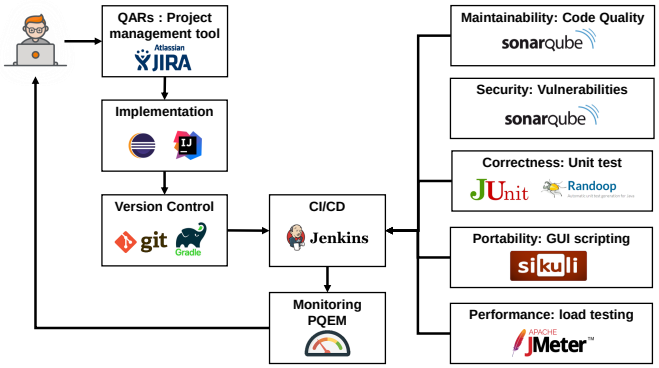

Fig. 1. Tool chain to support PQEM integration.

platform will trigger the execution of the test cases related to the QARs and it will also trigger the calculation of the metrics. During the monitoring phase, the PQEM tool will collect all the reports generated by the tool chain, calculate the metrics, and synthesize the results (Table I, Step 4).

There are eight quality characteristics defined by IEEE: Availability, Interoperability, Performance Efficiency, Security, Usability, Modifiability, and Functional Suitability. From an automatizing perspective, Functional Suitability is perhaps the easiest quality characteristic to address; many tools have been used for years to verify the correctness of the systems, and these tools have shown to be stable and became well-known by developers. Junit[8] for Java, testunit[9] for python are just few examples of frameworks that support unit testing. When these frameworks are used to verify the correctness of a software project, the collection of metrics aligned to the Functional Suitability results straightforward. In addition, the correct setup and project configuration with tools such as Gradle could make also simple the application of PQEM in existing projects.

Although developers are usually responsible for designing and writing test cases, there are tools that can automate this process up to a certain degree. For example, Randoop[10] is a tool that can automatically generate hundreds of test cases by applying feedback-directed random testing [26]. Little effort is required by developers to create an adequate configuration for testing the product with Randoop, making this tool suitable to be integrated to the proposal.

In addition to the Functional Suitability, Maintainability could also be partially automated by integrating tools such as SonarQube[11]. For example, a set of predefined QARs related to Maintainability could be automatically verified by the static analysis performed by SonarQube. This tool gives flexibility to define test rules that verify the maintainability of the code and the presence of code smells. Similarly, SonarQube provide a set of rules that determine whether the system under test has security vulnerabilities or not. This feature will allow developers to also check another quality characteristic: Security. These features make SonarQube a tool worth to be used along with PQEM. Following this approach, tools like Sikuli[12] JMeter[13] could be also used to cover QARs related to quality characteristics such as Portability and Performance.

Despite being possible to automate the execution and collection of metrics related to Functional Suitability, Maintainability, Security, Portability, and Performance of the product, providing automation for the rest of the quality characteristics might be more challenging. Usability is particularly hard to automate since it is mainly human-centric. Usability testing requires to define first the type of users that the product is targeting, and then, the usability scenarios that those users will probably perform. Although several techniques can be used for this purpose, nowadays there is a lack of tools that allow for automatic usability testing. PQEM allows to calculate the quality level of an Usability test by unifying the result per each respondent into a single value per question, and a final value for Usability.

Energy efficiency is important to make more sustainable software; for this reason, researchers have studied the topic from several perspectives. For example, CPU-intensive processing in mobile devices has been studied by Rodriguez et al. [24]. Anwar et al. [25] also studied energy efficiency in Android apps and found that code smell refactorings positively impact the energy consumption. In another study, the authors pointed out the need to develop automated software analytics tools that allow software practitioners to understand the relationship between energy usage and other quality attributes [27]. Although the definition of metrics and acceptance criteria related to energy efficiency is a challenging task, we believe that the proposed framework could include this support in the near future. Furthermore, Interoperability and Modifiability are also quality characteristics that could be challenging to measure and more research is needed in this regard.

VII. Conclusion

In this article, we have presented an automatic framework that provides a space for quality management of a software product. It is based on PQEM [5], which allows measuring the quality level of each iteration of a product. It should be noted that the framework is automatic and the measurement is semi-automatic, and in this context it is an automated tool that will be developed throughout this year. But, it is viable to mention that applications and validations of the PQEM method have already been made.

In the same way, the framework allows to obtain a clear idea of a quality management environment. So, in addition to the development of the framework, the foundations are being laid for making new measurements in the new iterations of HeartCare in order to achieve having examples in several iterations of the same product. Understanding what the framework

[8]JUnit framework website – https://junit.org/junit5/
[9]Test unit – https://github.com/test-unit/test-unit
[10]Randoop website – https://randoop.github.io/randoop/
[11]SonarQube web site – https://www.sonarqube.org/
[12]Sikuli web site – http://sikulix.com//
[13]JMeter web site – https://jmeter.apache.org/

allows, we intend to integrate it with existing tools to provide the user with a unified means of quality measurement.

In this context, it is expected to develop new research and tools, in order to have a robust and complete framework useful for any project leader, technical leaders and quality managers. Also, the energy efficiency for CPU-intensive processing is an important concern that is studied from several perspectives. For example, they showed that using energy-aware job stealing increases the energy efficiency of mobile computational Grids because it increases the number of jobs that can be executed using the same amount of energy [24].

ACKNOWLEDGMENT

Mariana Falco is supported by the National Scientific and Technical Research Council (CONICET). Ezequiel Scott is supported by the Estonian Center of Excellence in ICT research (EXCITE), ERF project TK148 IT. Gabriela Robiolo and Mariana Falco are supported by Engineering School, Universidad Austral.

REFERENCES

[1] Altexsoft, (03 Nov, 2017), What Software Quality (Really) Is and the Metrics You Can Use to Measure It, https://www.altexsoft.com/blog/engineering/what-software-quality-really-is-and-the-metrics-you-can-use-to-measure-it/

[2] Puppets Lab. State of DevOps report 2015. https://media.webteam.puppet.com/uploads/2019/11/2015-state-of-devops-report.pdf?_ga=2.207590612.619038807.1584477951-277285324.1584477951

[3] S.H. Kan, Metrics and models in software quality engineering. Addison-Wesley Longman Publishing Co., Inc, 2002.

[4] H.M. Sneed A. Merey, Automated Software Quality Assurance. IEEE Transactions on Software Engineering, SE-11(9), 1985, pp. 909–916. doi:10.1109/tse.1985.232548

[5] M. Falco and G. Robiolo, A Unique Value that Synthesizes the Quality Level of a Product Architecture: Outcome of a Quality Attributes Requirements Evaluation Method. In: Franch X., Männistö T., Martínez-Fernández S. (eds) Product-Focused Software Process Improvement. PROFES 2019. Lecture Notes in Computer Science, vol 11915. 2019, Springer, Cham

[6] V. R. Basili, Software modeling and measurement: the Goal/Question/Metric paradigm, 1992.

[7] ISO/IEC 25010, (s.f) https://iso25000.com/index.php/en/iso-25000-standards/iso-25010.

[8] Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE), ISO/IEC 25023:2016, https://www.iso.org/standard/35747.html

[9] A. Wingkvist, M. Ericsson, R. Lincke and W. Lowe, W. A metrics-based approach to technical documentation quality. In 2010 Seventh International Conference on the Quality of Information and Communications Technology, 2010, pp. 476-481, IEEE

[10] W. Löwe, M. Ericsson, J. Lundberg, T. Panas N. Pettersson, Vizzanalyzer-a software comprehension framework. In Third Conference on Software Engineering Research and Practise in Sweden, Lund University, Sweden, 2003.

[11] S. Apel, F. Hertrampf and S. Späthe, Towards a Metrics-Based Software Quality Rating for a Microservice Architecture. In International Conference on Innovations for Community Services, 2019, pp. 205-220, Springer, Cham.

[12] D. R. Mohammad, S. Al-Momani, Y.M. Tashtoush M. AlsmiratA. Comparative Analysis of Quality Assurance Automated Testing Tools for Windows Mobile Applications. 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019, doi:10.1109/ccwc.2019.8666463

[13] M. Schramme J.A. Macías, Analysis and measurement of internal usability metrics through code annotations. Software Quality Journal, 27(4), 2019, pp. 1505-1530.

[14] L. Schrettner, L.J. Fülöp, A. Kiss T. Gyimóthy. Software quality model and framework with applications in industrial context. In 2012 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 453-456, IEEE.

[15] A. Mayr, R. Plösch, M. Kläs, C. Lampasona M. Saft, A comprehensive code-based quality model for embedded systems: systematic development and validation by industrial projects. In 2012 IEEE 23rd International Symposium on Software Reliability Engineering, 2012, pp. 281-290, IEEE.

[16] ConnQAT User Guide 2013.10, https://www.cqse.eu/download/conqat/conqat-book-2013.10.pdf

[17] Tsuda, N., Washizaki, H., Honda, K., Nakai, H., Fukazawa, Y., Azuma, M., Komiyama, T., Nakano, T., Suzuki, H., Morita, S. and Kojima, K., 2019, May. Wsqf: Comprehensive software quality evaluation framework and benchmark based on square. In 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) (pp. 312-321). IEEE.

[18] The ISO/IEC 25000 series of standards, https://iso25000.com/index.php/en/iso-25000-standards

[19] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice, 3rd ed. Addison-Wesley Professional, 2012.

[20] J.J. Chilenski and S.P. Miller. Applicability of modified condition/decision coverage to software testing. Software Engineering Journal, 9(5), 1994, pp. 193-200.

[21] Z. Li, P. Avgeriou P. Liang, A systematic mapping study on technical debt and its management. Journal of Systems and Software, 101, 2015, pp. 193-220.

[22] N. Oza, M. Ikonen, P. Kettunen and P. Abrahamsson, Exploring the sources of waste in kanban software development projects. In Proceedings of the 36th Conference on Software Engineering and Advanced Applications (EUROMICRO '10). 2010, pp. 376–381.

[23] K. Conboy X. Wang and O. Cawley.leagile software development: An experience report analysis of the application of lean approaches in agile software development. Journal of Systems and Software 85 (2012), pp. 1287–1299.

[24] J.M. Rodriguez, C. Mateos and A. Zunino, "Energy-efficient job stealing for CPU-intensive processing in mobile devices", Computing 96.2, 2014, pp. 87–117, https://doi.org/10.1007/s00607-012-0245-5

[25] Anwar, Hina, Dietmar Pfahl, and Satish N. Srirama. "Evaluating the Impact of Code Smell Refactoring on the Energy Consumption of Android Applications." 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2019.

[26] C. Pacheco and M.D. Ernst. Randoop: feedback-directed random testing for Java. In Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, pp. 815-816, 2007.

[27] Anwar, Hina, and Dietmar Pfahl. "Towards greener software engineering using software analytics: A systematic mapping." 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA). IEEE, 2017.