Smart Measurements and Analysis for Software Quality Enhancement

Sarah Dahab[1], Stephane Maag[1], Wissam Mallouli[2(✉)], and Ana Cavalli[1]

[1] SAMOVAR, Telecom SudParis, Université Paris-Saclay, Saint-Aubin, France
{sarah.dahab,stephane.maag}@telecom-sudparis.eu
[2] Montimage Research and Development, Paris, France
{wissam.mallouli,ana.cavalli}@montimage.com

Abstract. Requests to improve the quality of software are increasing due to the competition in software industry and the complexity of software development integrating multiple technology domains (e.g., IoT, Big Data, Cloud, Artificial Intelligence, Security Technologies). Measurements collection and analysis is key activity to assess software quality during its development live-cycle. To optimize this activity, our main idea is to periodically select relevant measures to be executed (among a set of possible measures) and automatize their analysis by using a dedicated tool. The proposed solution is integrated in a whole PaaS platform called MEASURE. The tools supporting this activity are Software Metric Suggester tool that recommends metrics of interest according several software development constraints and based on artificial intelligence and MINT tool that correlates collected measurements and provides near real-time recommendations to software development stakeholders (i.e. DevOps team, project manager, human resources manager etc.) to improve the quality of the development process. To illustrate the efficiency of both tools, we created different scenarios on which both approaches are applied. Results show that both tools are complementary and can be used to improve the software development process and thus the final software quality.

Keywords: Software engineering · DevOps team · Metrics combination · Metrics reuse · Metrics suggestion · Metrics correlation · Software quality

1 Introduction

Metrics play a crucial role to improve software quality development process that is becoming more and more complex [1]. To select the right metrics is also of

Supported by the ongoing European project ITEA3-MEASURE started in Dec. 1st, 2015, and the EU HubLinked project started in Jan. 1st, 2017.

M. van Sinderen and L. A. Maciaszek (Eds.): ICSOFT 2018, CCIS 1077, pp. 194–219, 2019.
https://doi.org/10.1007/978-3-030-29157-0_9

prime importance for a successful software development. They have a strong impact on developers actions and decisions [16].

In order to improve the software quality, we need to introduce new metrics with the required detail and automation. Due to the modern development practices, new tools and methods are also necessary being the traditional metrics and evaluation methods not sufficient anymore. Even more, there is a large body of research related to software metrics that aims to help industry while measuring the effectiveness and efficiency of used software engineering processes, tools and techniques to help management in decision-making [4].

To achieve software quality, it is required to integrate new metrics based on constraints combining safety (the system always behaves as it is supposed to) and security (authentication, data protection, confidentiality, ...) and quality of service. Green metrics also become relevant as they contribute to the reduction of energy consumption.

This paper focuses on the combination, reuse, suggestion and correlation of metrics. We have developed two complementary approaches, one based on metrics reuse, combination and suggestion and the other on metrics correlation. They have been implemented in two tools, Metrics Suggester and Metrics Intelligence Tool (MINT). Both approaches contribute to improve software quality development proposing new techniques for metrics application and evaluation.

Regarding the Metrics Suggester approach, it is based on the optimization of the current measurement process which are manual and static and thus very costly. Metrics Suggester proposes an automated analysis and suggestion approach, by using the learning technique Support Vector Machine[1] (SVM), based on AI algorithms. In summary, it consists of suggesting relevant and efficient measurement plans at runtime using a machine learning algorithm.

Regarding the MINT approach, the idea is to identify and design correlations between metrics that contribute to the improvement of the development process and help developers to take decisions about it. The proposed correlations cover all aspects of the system like functional behavior, security, green computing and timing. For instance, we have defined correlations covering different phases of development. Techniques to correlate metrics are provided and recommendations are given as an outcome to the developer and project manager or any other software stakeholder. Recommendations will affect their actions and decisions.

Both techniques are original and introduce innovation with respect to classical methods. Moreover, the application to the combination of metrics regarding software development, security and green computing is a novelty with respect to them.

Both approaches and tools are part of the European ITEA project MEASURE and they have been integrated in the project PaaS platform[2]. Furthermore, in order to reach that result, a close link has been defined between academia and industry for several years strengthened by the EU HubLinked

[1] http://www.statsoft.com/Textbook/Support-Vector-Machines.
[2] https://itea3.org/project/measure.html.

project[3] fostering the U-I relationships (Universities-Industry). In summary, the main contributions of this paper are:

- the design of new complementary approaches to improve software quality development process by introduction of new correlation and suggestion techniques, these lasts based on AI algorithms;
- the development of techniques and tools, Metrics Suggester and MINT, for metrics correlation, reuse, suggestion, and recommendation.
- first functional experimentation of both tools.

This paper is organized as it follows: Sect. 2 presents the related works. Section 3 gives a view of the MEASURE global platform and presents the two approaches and the tools, Metrics Suggester and MINT. Section 4 is devoted to presenting the experiences that are illustrated by experiments and Sect. 5 gives the conclusion and perspectives of our work.

2 Related Works

Many efforts have been done to define metrics for software quality [4,10,21,25]. These works can be associated with standardized quality models such as ISO 9126 quantifying properties with software metrics [5]. Learning techniques are currently arising to effectively refine, detail and improve the used metrics and to target more relevant measurement data. Current works such as [22], [27] and [23] raise that issue by proposing diverse kinds of machine learning approaches for software defect prediction through software metrics. These studies have shown the importance of gathering information on the software engineering process in particular to ensure its quality through metrics and measurements analysis [10]. Thanks to that, standardization institutes worked in that way to propose two well-known norms, ISO/IEC25010 [21] and OMG SMM [4] to guide the measurement plan specification. These two standards have been reviewed by the research and industrial community, and are adapted and applied in many domains [2].

However, even if these techniques have introduced considerable progress to improve the software quality, they have still some limitations. The measurement plan is, in general, manually fixed by the project manager, the implementation of the measures is dependent on the developer and reduce the scalability, maintainability and the interoperability of the measurement process.

For software metrics correlation, there are many works focused on the relations between internal and external software metrics. In [28], the impact of software metrics on software quality is presented and the internal and external attributes of a software product are studied because the relationship between them directly affects its behaviour. The metrics are combination of these attributes. As the number of metrics used in a software project increases, the management and controlling of the project also increases. In [24], the authors

[3] http://www.hublinked.eu/.

investigated the relationship between different internal and external software metrics by analyzing a large collection of C/C++ programs submitted to a programming competition, the Online Judge. In [19], they analyze the links between software reliability and software complexity for evaluating the effectiveness of testing strategies.

These works have been applied mainly to establish correlations between internal and external metrics, and to specific ones. They have been very useful for our work published in [7] and extended in this paper. Even though our approaches are generic and can be applied to any metric, we plan to apply our approaches to evaluate the relation between specific and well selected metrics. Besides, the tools we propose are part of a PaaS open source platform called MEASURE[4] dedicated to host several measuring and analysis tools to enhance software engineering process quality.

3 Measurement Approaches and Tools

3.1 The MEASURE PaaS Platform

The MEASURE platform provides services to (1) host, configure and collect measures, (2) store measurements, present and visualize them and (3) analyze them

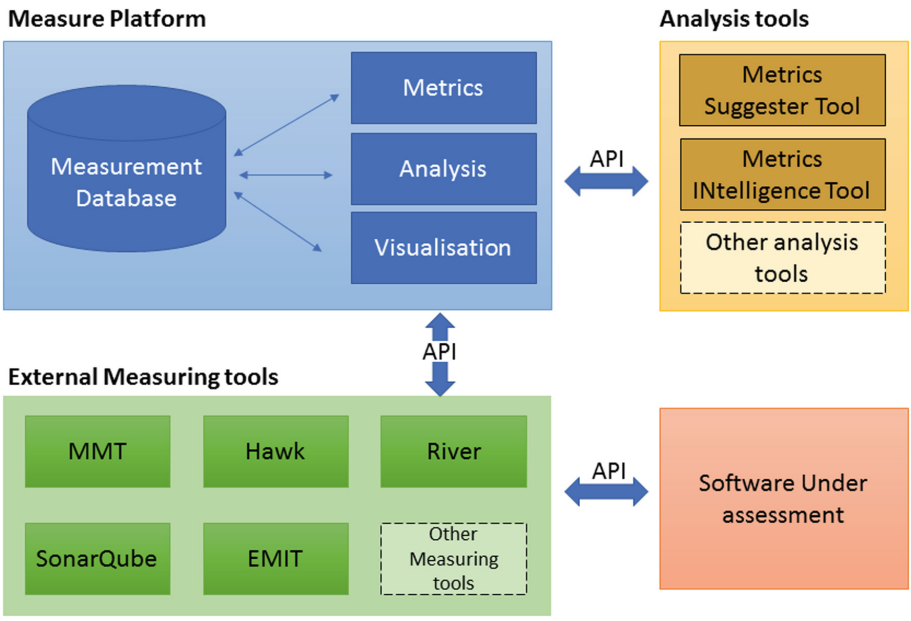

Fig. 1. The MEASURE PaaS platform.

[4] https://github.com/ITEA3-Measure/.

and provide recommendations. These measures are first defined in SMM (Structured Metrics Meta-model) standard[5] using the extension of Modelio modelling tool[6] dedicated to SMM modelling. The MEASURE platform is able to collect measurements (data resulting of the execution of an instantiated measure) thanks to external measuring tools (e.g., Hawk [11] for design and modelling related measurements, SonarQube [12] for testing related measurements, MMT[7] for operation related measurements, EMIT [3] for energy consumption related measurements, etc.) (Fig. 1).

Direct measures collect data in physical world while the derived (complex or composed) measures are calculated using previously collected measurements as input. Collected measurements are stored on a NoSQL database designed to be able to process a very large amount of data. To collect measurements, the direct measures can delegate the gathering work to existing measuring tools integrated with the MEASURE PaaS platform.

The measurements can also be processed by analysis tools to present consolidated results. The analysis platform is composed of a set of tools that allow combining and correlating measurements in a meaningful way in order to provide suggestions and recommendations for the software developers and managers.

Finally, stored measurements and recommendations are presented directly to the end user following a business structured way by the Decision-making platform, with a web front-end that allows organizing measures based on projects/software development phases and displays its under various forms of charts.

In order to study and improve the software quality processes and ease the tasks of project engineers and managers, we defined a methodology based on two modules: Metrics Suggester and Metrics Intelligence. The used terminology, the formal modelling language and our two techniques are described in the following.

3.2 A Formal Software Measurement Context

Several concepts are commonly used in the software engineering context. We provide some measurement terminologies in the following [15,17].

Terminology

Measurand: a measurand is the measured object. In this context, it is a software system, such as software product, in use or software resource.

Software Properties: the software properties are the measurable properties of a software such as, for instance, complexity or performance.

[5] https://www.omg.org/spec/SMM/About-SMM/.
[6] https://www.modelio.org/.
[7] http://www.montimage.com/products.html.

Measurement: a measurement is defined as a direct quantification of a measured property [9]. This is the value of an evaluation result in a single time. This is information on the measured property, such as the percentage of the memory used.

Measure: a measure is the definition of a concrete calculation to evaluate a property, such as the calculation of the number of lines of code.

Metric: a metric is a measure space, in other words, the specification of a measurement. This is the formal definition of a measurement of a property of a computer object by specifying the measurand, the measure(s) and the software property to be measured.

Measurement Plan: a measurement plan is an ordered set of metrics (simple or complex). They are all expected to be executed at a specific time t or during a well-defined duration and according to an ordered metrics sequence. They can be run sequentially or in parallel.

The OMG Structured Metrics Meta-model. Our methodology is based on the OMG SMM (Structured Metrics Meta-model) standard to formally model our metrics in terms of measure, scope (subset of measured properties) and measurement but also in order to easily generate the corresponding Java code [6]. Our main purpose is to have a standard documentation on the measurement architecture with the SMM model, which will also optimize the design phase of the implementation of a software measurement. Indeed, this process will enable measurement code generation from a measurement architecture model based on SMM. This will reduce the developer's burden of manual implementation.

SMM is a standard specification that defines a meta-model to specify a software measurement architecture, in other words to specify a *Measure Space* applied to a computer system. It defines the meta-models to express all necessary concepts to specify a measurement context. A wide range of diversified types of measures is proposed to define the dependency type between dependent measures (as the ratio, binary or grade measure). The language allows to define direct/indirect measures and complex metrics:

- Direct Measure: is the measure independent of other measures, thus it refers to the simple evaluation function.
- Indirect Measure: is a measure dependent on other measures.
- Complex metric: a complex metric is a metric composed of indirect measure(s).

As an example, the Fig. 2 represents the model of the computational energy cost metric in SMM with the Modelio tool. This complex metric (represented by 3 stack levels) depends on three other metrics, two of them are direct metrics (represented by a microscope): the memory access count and I/O usage metrics, and the third one is also a complex metric denoted CPU energy model. It returns

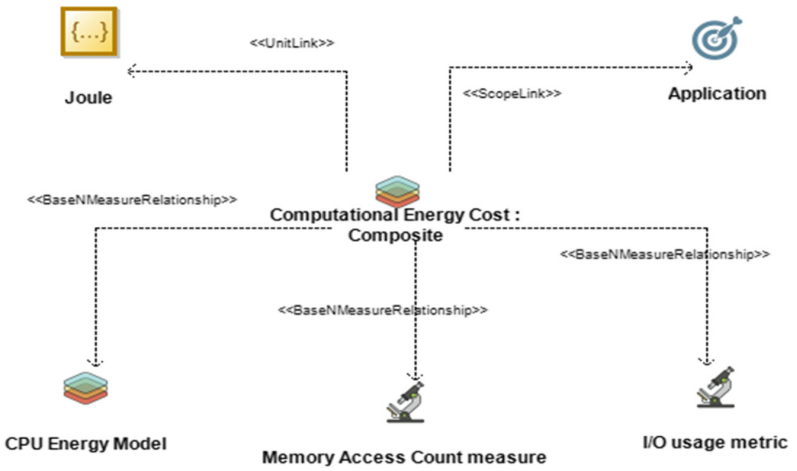

Fig. 2. The computational energy cost metric model in SMM. (Color figure online)

a numerical value in Joule. A low energy cost means a better software. Thus, it is. Then, the unit of measure of the computational energy cost is a Joule and represented in the figure by the yellow symbol "{...}". Finally, this metric is applied on an application, which is represented by the blue target in the model. Each component is modeled as a UML class allowing the code generation from a SMM metric model.

We describe in the following the two approaches and tools composing our methodology.

3.3 The Software Metrics Suggester

As previously mentioned, one of our approaches consists on suggesting relevant and efficient software measurement plans at runtime using a machine learning algorithm. In order to detail our methodology, we first introduce some concepts in the following.

Basics. In our previous paper [7], we developed a supervised learning technique based on SVM with training datasets. These datasets contain vectors labeled by experts. In an industrial context, the labeling process can be complex, time and resource consuming [13]. In this paper, our main objective is to automatically generate our measurement plans from totally unlabeled data. Our goal being to define an unsupervised learning methodology. To do so, we propose an algorithm (Algorithm 1) based on a clustering technique. This latter allows to identify in an automatic way the software classes of interests from unlabeled data that are themselves automatically labeled with dummy classes.

Finally, each obtained cluster will be classified and vectors of measurements automatically labeled to be fed as inputs to our SVM approach. In the following,

we formally describe the detailed procedures along with a generalized classifier for the suggestion of measurements plan.

X-means Clustering. First, while measuring a system, we have a continuous stream S of n measurements. These measurements can be considered as events. The concept of event is interesting since it defines a formal link between the two methods proposed in our approach, Metric Suggester and MINT. Each event can be represented as a data point in a space x_i and can be expressed as:

$$\{(x_i)\},\ x_i \in \mathbb{R}^d, i \in \{1, 2, ..., n\} \tag{1}$$

where d is the dimension number of the input space or attributes (a_i), and n is the number of samples.

Generally, we can associated low-level events with high-level or complex events $y_i \in \mathbb{R}$ by a prediction function $f(x_i)$ (Eq. (4)). However, because no labeled event data is assumed, we decided to apply a clustering technique that could categorize the data into classes of metrics. One famous technique commonly applied is the K-means algorithm [18]. Though it is very efficient in many areas, it requires to know the value of K. In our paper, we herein suppose that we do not know its value, that depends on the software metrics in use and the collected data. Therefore, the X-means clustering algorithm is proposed [26]. X-means will allow us to split the input data (1) into K clusters without the need to define the expected number of them at the first stage. The best K subsets are chosen such that all points in a given subset "belong" to some center c_j, $j \in (1, 2, ..., k)$ with a low inter-cluster similarity. Basically, the algorithm aims at minimizing the following distance objective function:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} |D_{MH}(x_i^j, c_j)|, \tag{2}$$

where $|D_{MH}(x_i^j, c_j)|$ is the Mahalanobis distance measure between a event data point and a cluster center [8]. Later, we also use this distance measure to define the boundaries of each rule attribute. By using the Eq. (2), we can assign the events data points x_i to the cluster whose distance from the cluster center c_j is lower of all the cluster centers and which satisfies the Bayesian information criterion (BIC). After that, each cluster center is updated by taking the weighted average value of event points in that cluster (3) for better clustering results.

$$C_{j_u pdate} = \frac{1}{|c_j|} \sum_{i=1}^{c_j} x_i \tag{3}$$

Finally, class labels y_i can be assigned for each event cluster automatically by our system. Then, once this assignation is performed, the vectors are labelled and the SVM process can be executed at runtime for beginning the measurement plans suggestion.

Algorithm 1. Event Clustering.

Input: Unlabeled event data-set $\{x_i\}_{i=1}^{n} \in \mathbb{R}^d$
Output: Labeled event data-set $\{(x_i, y_i)\}_{i=1}^{n} \in \mathbb{R}^d$,
Clusters centers $C_i^j \in y_i$

1 Initialize an empty stack $\varphi \leftarrow 0$
2 Define initial number of clusters $K_0 \leftarrow 2$
3 Divide unlabeled event data-set into $C_1, C_2, ..., C_{k_0}$ clusters using k-means with setting $k \leftarrow k_0$.
4 **repeat**
5 Divide each cluster C_i into $C_i^{k_0}$ sub-clusters using k-means with $k \leftarrow k_0$.
6 Calculate $BIC(C_i)$
7 Calculate $BIC, MNDL(C_i^{k_0})$
8 **if** $BIC(C_i) > BIC^{'}(MNDL(C_i^{k_0}) > MNDL^{'}(C_i^{k_0}))$ **then**
9 The two-divided model is preferred, and the division is continued with $C_i \leftarrow C_i^1$.
`// push event data into the stack`
10 $x_i \rightarrow \varphi$
11 $C_i^{k_0} \rightarrow \varphi$
12 $BIC(C_i) \rightarrow \varphi$
13 **return** step 5
14 **end**
15 **if** $BIC(C_i) < BIC^{'}(MNDL(C_i^{k_0}) < MNDL^{'}(C_i^{k_0}))$ **then**
16 Clusters C_i are no longer divided and set $C_i \leftarrow C_i^{k_0}$.
17 **if** $\varphi \rightarrow 0$ **then**
18 **goto** step 26
19 **else**
`// Extract all the stacked data`
20 $\varphi \rightarrow x_i$
21 $\varphi \rightarrow C_i^{k_0}$
22 $\varphi \rightarrow BIC(C_i)$
23 **return** step 5
24 **end**
25 **end**
`//` C_i `cluster identification becomes unique.`
26 $C_i \leftarrow C_i^*$
`//` Initial k_0 `divided clusters become unique.`
27 $C_j \leftarrow C_j^*$
28 **until** $i \leq k_0$;

Support Vector Machine. A support vector machine (SVM) [29] is a linear classifier defined by a separating hyperplane that determines the decision surface for the classification. Given a training set (supervised learning), the SVM algorithm finds a hyperplane to classify new data. Consider a binary classification problem, with a training dataset composed of pairs $(\boldsymbol{x_1}, y_1), \ldots, (\boldsymbol{x_l}, y_l)$, where each vector $\boldsymbol{x_i} \in R^n$ and $y_i \in \{-1, +1\}$. The SVM classifier model is a hyperplane

that separates the training data in two sets corresponding to the desired classes. Equation (4) defines a separating hyperplane (Source [7]):

$$f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b = 0 \tag{4}$$

where $\boldsymbol{w} \in R^n$ and $b \in R$ are parameters that control the function. Function f gives the signed distance between a point $\boldsymbol{x}$ and the separating hyperplane. A point $\boldsymbol{x}$ is assigned to the positive class if $f(\boldsymbol{x}) \geq 0$, and otherwise to the negative class. The SVM algorithm computes a hyperplane that maximizes the distance between the data points on either side, this distance is called *margin*. SVMs can be modeled as the solution of the optimization problem given by (5), this problem maximizes the margin between training points (Source: [7]).

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2} \| \boldsymbol{w} \|^2 \\ \text{subject to:} \quad & y_i(\boldsymbol{w}^T \boldsymbol{x_i} + b) \geq 1, i = 1, \ldots, l \end{aligned} \tag{5}$$

All training examples labeled -1 are on one side of the hyperplane and all training examples label 1 are on the other side. Not all the samples of the training data are used to the determine the hyperplane, only a subset of the training samples contribute to the definition of the classifier. The data points used in the algorithm to maximize the margin are called *support vectors*.

Features and Classes. The set of measurements that is classified using SVM is defined as a *vector* of *features*. Each feature is a field of a vector and a measurement of one specific measure. Each field is unique. So a feature is a measurement composing a vector for our classification. Further, the vectors are classified into *classes* according to the feature values. Each class refers to a measured software property, such as the maintainability or reliability. The features composing a vector are the measurements which give information on the classes. Some of them can give information on several classes or only one. The features are chosen according to the metrics defined in the starting measurement plan.

The Mapping System. In order to suggest relevant and effective measurement plans, a mapping system is defined between classes and metrics, and between metrics and features. It aims at allowing an automate suggestion procedure. This mapping is performed by the experts of the measured system. According to the type of interest (in terms of numbers of vector contained) of the classes highlighted by the SVM classification, some metrics will be added or removed from the measurement plan. Thus, new features will be gathered and others will no longer be.

Classes-Metrics. A relationship between a class and some metrics is needed to measure specific targeted software properties. The classes are used for the classification of the vectors according to their features values. As above mentioned, our classification method is to classify a vector in the class corresponding to the property whose the values of the vector show a type of interest.

Features-Metrics. The features values inform about the properties (classes) of interest. There are features which give information on only one property and others which can give information on several different properties (complex metrics). Some of the measures can be used by different metrics. Thus, the features associated with a metric are the features corresponding to the measures which composed the metric. In order to ensure the sustainability of measurement cycles by having at each cycle an information on all measured properties, a set of metrics should always be gathered. This set is called mandatory features. To select the mandatory features, we use the RFE technique, explained below, based on SVM.

The Feature Selection. The goal of the Feature Selection (FS) process is to select the relevant features of the raised classes. Its objective is to determine a subset of features that collectively have good predictive power. With FS, we aim at highlighting the features that are important for classification process. The feature selection method is Recursive Feature Elimination (RFE) [20]. RFE performs backward elimination that consists of starting with all the features and test the elimination of each variable until no more features can be eliminated. RFE begins with a classifier that was trained with all the features that are weighted. Then, the feature with the absolute smallest weight is eliminated from the feature set. This process is done recursively until the desired number of features is achieved. The number of features is determined by using RFE and cross validation together. In this process each subset of features is evaluated with trained classifier to obtain the best number of features. The result of the process is a classifier trained with a subset of features that achieve the best score in the cross validation. The classifier used during the RFE process is the classifier used during the classification process.

Measurement Plan Suggestion. Based on the classification, matching and FS, two sets of classes are notified: the one with the most vectors called *Biggest* and the other set constituted of all the other classes called *Others*. The Biggest means that the corresponding property is the most interested element while the Others means that the corresponding properties are not the elements of interest. Thereby, our *Suggestion* procedure is applied for the property corresponding to the Biggest. Indeed, the Biggest property needs a further measurement, while the Others one no longer need it. Basically, based on the procedures *Analysis* and *Selection*, we raise unnecessary features for the classification that should be removed from the measurement plan. Through this method, the measurement load is increased only on needs and decreasing due to less interested properties. This suggestion approach allows to reach a lighter, complete and relevant measurement plan at each cycle of the software project management.

3.4 MINT- Metrics Intelligence Tool

As mentioned in our paper [7], MINT is a software solution designed to correlate metrics from different software development life cycle in order to provide

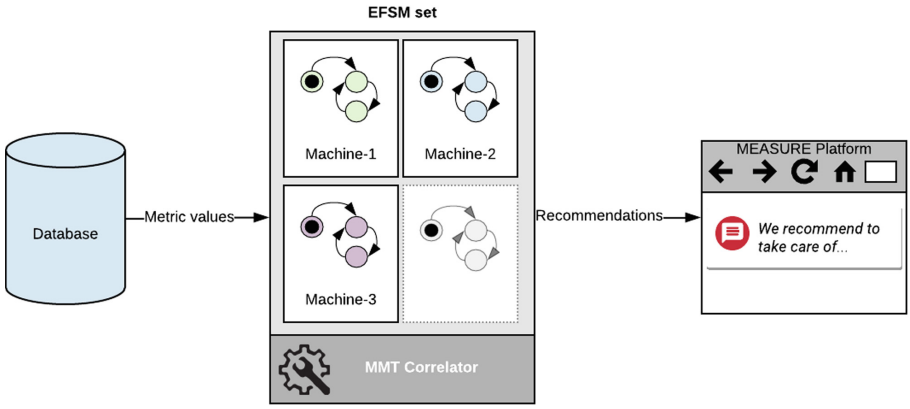

Fig. 3. MINT approach overview.

valuable recommendations to different actors impacting the software development process. MINT considers the different measurements collected by the MEASURE platform as events occurring at runtime. The correlation is designed as extended finite state machines (EFSMs) allowing to perform Complex Event Processing (CEP) in order to determine the possible actions that can be taken to improve the diverse stages of the software life cycle and thus the global software quality and cost (Fig. 3).

Background

Metrics Correlation. The correlation can be defined as a mutual relationship or association between metrics (or the values of its application). Metrics correlation can be the basis for the reuse of metrics; it can help to predict one value from another; it can indicate a causal relation between metrics and can establish relations between different metrics and increase the ability to measure. Examples of correlation are: to correlate two metrics from the same development phase; to correlate the same metric at different times; to correlate a metric (a set of metrics) from phase X regarding metrics of phase Y. As an outcome, recommendations and a selection of metrics will be proposed to the developer to improve the software development. MINT is based on correlation techniques.

Complex Events Processing. Complex event processing (CEP) [14] technology addresses exactly the need of matching continuously incoming events against a pattern. Input events from data streams are processed immediately and if an event sequence is matching a pattern, the result is emitted straight away. CEP works very efficiently and in real-time, as there are no overheads for data storing. CEP is used in many areas that include for instance manufacturing processes, ICT security, etc. and is adapted in this paper for software quality assessment process.

Extended Finite State Machine. In order to formally model the correlation process, the Extended Finite State Machine (EFSM) formalism is used. This formal description allows to represent the correlation between metrics as well as the constraints and computations needed to retrieve a meaningful recommendation related to software quality assessment.

Definition 1. An Extended Finite State Machine M is a 6-tuple M $= <$ $S, s_0, I, O, \vec{x}, Tr >$ where S is a finite set of states, s_0 is the initial state, I is a finite set of input symbols (eventually with parameters), O is a finite set of output symbols (eventually with parameters), $\vec{x}$ is a vector denoting a finite set of variables, and Tr is a finite set of transitions. A transition tr is a 6-tuple tr $=< s_i, s_f, i, o, P, A >$ where s_i and s_f are the initial and final state of the transition, i and o are the input and the output, P is the predicate (a boolean expression), and A is an ordered set (sequence) of actions.

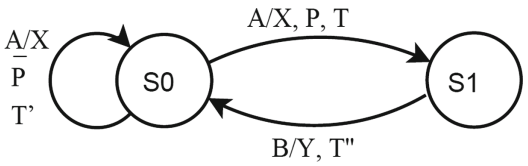

Fig. 4. Example of a simple EFSM with two states (Source [7]).

We illustrate the notion of EFSM through a simple example described in Fig. 4. The ESFM is composed of two states S_0, S_1 and three transitions that are labeled with two inputs A and B, two outputs X and Y, one predicate P and three tasks T, T', and T''. The EFSM operates as follows: starting from state S_0, when the input A occurs, the predicate P is tested. If the condition holds, the machine performs the task T, triggers the output X and passes to state S_1. If P is not satisfied, the same output X is triggered but the action T' is performed and the state loops on itself. Once the machine is in state S_1, it can come back to state S_0 if receiving input B. If so, task T'' is performed and output Y is triggered.

Writing Correlation Processes

Correlation Process Inputs and Outputs. The basic idea behind MINT approach is to specify a set of correlation rules based on the knowledge of an expert of the software development process. These rules can rely on one or different sets of metrics (seen as inputs) and allow different recommendations to be provided (seen as outputs) to different kinds of actors:

- Actors from the DevOps team: Analysts, designers, modellers, architects, developers, tester, operators, security experts, etc.
- Actors from the management plan: product manager, project manager, responsible of human resources, responsible of financial issues etc.

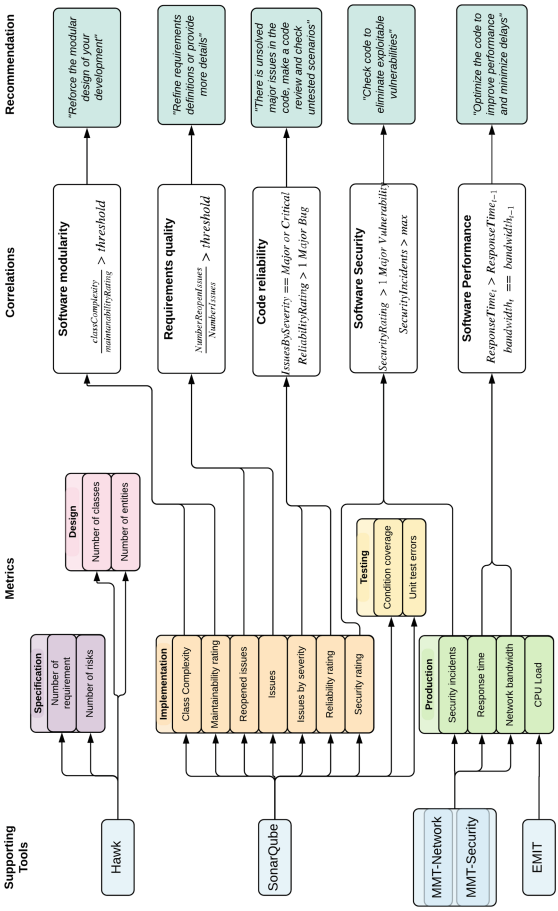

Fig. 5. Example of correlation processes (Source [7]).

The automatic generation of such rules or their continuous refinement based on some artificial intelligence techniques is an ongoing work and out of the paper scope.

Example of Correlation Processes. The correlation processes rely on different measurements that are computed and collected by external tools. Some examples of correlations are presented in the Fig. 5.

Software Modularity. The assessment of the software modularity relies on two metrics provided by the SonarQube tool that are the class complexity and the maintainability rating. The class complexity measure (also called cognitive complexity) computes the cognitive weight of a Java Architecture. The cognitive

weight represents the complexity of a code architecture in terms of maintainability and code understanding. The maintainability rating is the ratio of time (according to the total time to develop the software) needed to update or modify the software. Based on these definitions, and considering that a modular code can be more understandable and maintainable, we can correlate the two metrics and compute the ratio $R = class$ complexity/maintainability rating. If this ratio is more than a specific threshold set by an expert, the recommendation "Reinforce the modular design of your development" will be provided to the software architect and developers.

In the initial state, we can either receive the input related the class complexity denote *cc* or the maintainability rating denoted *mr*. The process accesses respectively to the states "cc received" or "mr received". If we receive the same measurement related to the same metric, we update its value and loop on the state. Otherwise, if we receive the complementary metric, we compute the ratio $R = class$ complexity/maintainability rating. If this ratio is less than the defined threshold, we come back to the initial state otherwise, we raise the recommendation. Timers are used to come back to the initial state if the measurements are too old. For sake of place, only this EFSM is presented in Fig. 7. All the others follow the same principles (Fig. 6).

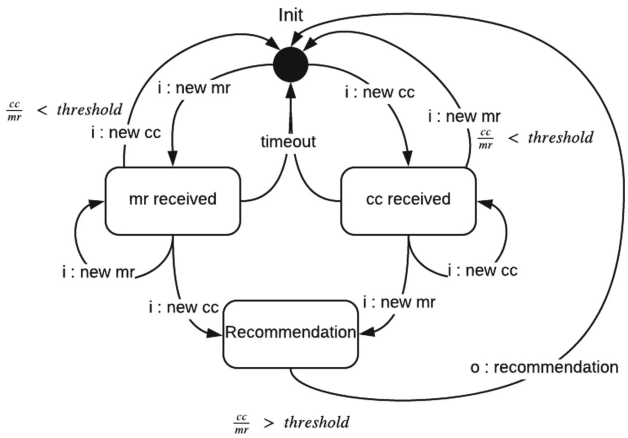

Fig. 6. Software modularity correlation processes (Source [7]).

Requirements Quality. The assessment of the requirements quality can rely on two metrics provided by the SonarQube tool that are the total number of issues and the total number of reopened issues. These numbers are collected during the implementation phase and we can consider that the fact that we reopen an issue many times during the development process can be related to an ambiguous definition of the requirement that needs to be implemented. If we have a

ratio R = number of reopened issues/number of issues that is more than a specific threshold, we can consider that the requirements are not well defined and that the development needs more refinement about them. The recommendation "Refine requirement definitions or provide more details" will be provided to the requirements analyst.

Code Reliability. The assessment of the code reliability relies on two metrics provided by the SonarQube tool that are the number of issues categorized by severity and the reliability rating. The issues in SonarQube are presented with severity being blocker, critical, major, minor or info and the reliability rating are from A to E: A is to say that the software is 100% reliable and E is to say that there is at least a blocker bug that needs to be fixed. Based on these definitions and considering that a reliable code should be at last free of major or critical issues, we can check that there is no major, critical nor blocker issues and the reliability rating is < C corresponding to 1 major bug. If this condition is not satisfied, the recommendation "There is unsolved major issues in the code, make a code review and check untested scenarios" will be provided to the software developers and testers.

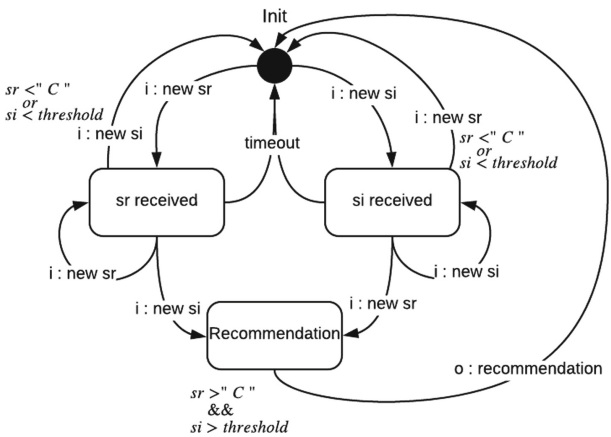

Fig. 7. Software security correlation processes.

Software Security. The assessment of the software security relies on two metrics, one provided by the SonarQube tool that is the security rating (denoted sr in Fig. 7) and the other is provided by MMT that is the number of security incidents (denoted si in Fig. 7). The security rating in SonarQube provide an insight of the detected vulnerabilities in the code and are presented with severity being blocker, critical, major, minor or no vulnerability. The number of the security incidents provided by MMT reports on successful attacks during operation. The evaluation of security demonstrates that if an attack is successful this means that

the vulnerability in the code was at least major because an attacker was able to exploit it to perform its malicious activity. Based on these definitions, and considering that a reliable code should be at last free of major vulnerabilities, we can check if there is a major vulnerability and that the number of attacks at runtime are more than a threshold. If this condition is satisfied, the recommendation "Check code to eliminate exploitable vulnerabilities" will be provided to the software developers ans security experts.

Software Performance. The assessment of the software performance relies on two metrics provided by the MMT tool that are the response time and the bandwidth usage. The response time denotes the delay that can be caused by the software, hardware or networking part that is computed during operation. This delay is in general the same for a constant bandwidth (an equivalent number of users and concurrent sessions). Based on this finding, we can correlate the two metrics and compute that the response time is not increasing for during time for the same bandwidth usage. If this response time is increasing, the recommendation "Optimize the code to improve performance and minimize delays" will be provided.

Programmer Code Quality. The assessment of a programmer code quality can rely on three metrics (1) number of lines of codes pushed by each developer and provided by Git or SVN repository API, (2) the complexity of the code computed by SonarQube and (3) the number of bugs detected in this specific code provided by SonarQube also. This assessment can be done each time a new code is pushed on Git or SVN (which constitutes a fourth event in the FSM machine that specifies the correlation rule). The recommendation for developers pushing bad code (resulting to a lot of bugs) is to have training regarding good practices in coding or to a specific technology or library used in the development or/and can provide a hint the project manager about the quality of developers skills.

Project Management and Fulfillment of Deadlines. The assessment of project management quality is generally performed by checking if the project is advancing according to the initial plans. This assessment can be done by checking the percentage of fulfilled requirements and correlating this to the timing plan. If the project is late a recommendation can be to add more developers in the project or to change priorities in the development strategy, if the project is advancing more than expected, reallocation of human resources on other projects can be an option.

4 Experiments

Fifteen software metrics have been selected by experts of the MEASURE platform[8] (mainly its administrator, the project manager and tools engineers). The list of metrics is depicted in the Table 1.

[8] http://194.2.241.244/measure/.

Table 1. Each metric and its assigned index during the experiments (Source [7]).

Index	Metric
1	Cognitive Complexity
2	Maintainability Index
3	Code Size
4	Number of issues
5	Response Time
6	Running Time
7	Usability
8	Computational Cost
9	Infrastructure Cost
10	Communication Cost
11	Tasks
12	I/O Errors
13	Precision
14	Stability Response Time
15	Illegal Operations

Then measurements corresponding to these metrics are collected. Our approach is based on the classification of the collected vectors into well-defined classes. However, one of the novelties in that new paper compared to [7] is that the training data set is automatically obtained using our X-means clustering algorithm. It means that our classes are obtained from the results of the algorithm. This is what we depict in the first subsection below. After that, we apply our two techniques and tools on the data collected through the MEASURE platform and detail the results.

4.1 The Training Data Set and the Classification Process

In order to obtain our clusters and then provide our classes, we have run our X-means algorithm on a collection of 1000 vectors containing, each of them, the measurements for the 15 metrics. As this can be noted, we here considered metrics defined from one single metric. Due to the management of the MEASURE project and the dates allowing to collect some data, the schedule when these data have been collected and the data for suggesting the measurement plans had to be tuned. Indeed, the data corresponding to the training data set and the ones collected for the plans suggestion were not matching exactly; and the results when using SVM was not efficient. For these reasons, most of the data used within the procedure of training data set has been manually changed to fit with the platform in use during the learning approach, that is the measurement plans suggestion process.

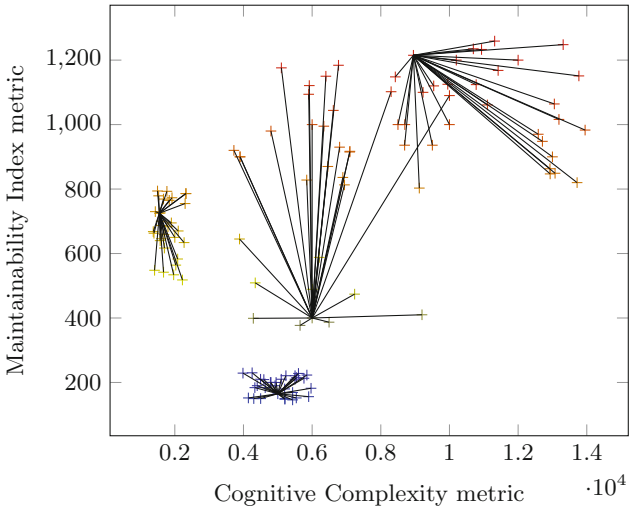

Fig. 8. A visualization of our X-means clustering results.

Based on that data, our X-means approach has been successfully applied. For that purpose, the pyclustering library has been used and configured for our methodology[9]. Therefore, the tool provided four main clusters defined by four centers. We illustrate these results in the Fig. 8. In this figure, we made the choice to consider the two first features, i.e., 'Cognitive Complexity' and 'Maintainability Index' as the two axis. For a sake of visualization clarity, the other axis are not illustrated.

As previously mentioned, our objective is the categorize these clusters in terms of set of metrics. Then the above mentioned experts have analyzed the results of our approach to finally extract the four following classes that basically correspond to software class properties:

- **Maintainability (Class 1):** Cognitive Complexity, Maintainability Index, Code Size, Number of issues.
- **System Performance (Class 2):** Computational Cost, Infrastructure Cost, Communication Cost and Tasks.
- **Performance (Class 3):** Response Time, Running Time and I/O Errors.
- **Functionality (Class 4):** Usability, Precision, Stability Response Time and Illegal Operations.

These classes and the obtained training data set is therefore used for our learning based suggestion approach as described in the following.

[9] https://github.com/annoviko/pyclustering.

4.2 Suggester Experiments

The suggestion process is evaluated by analyzing the new measurement plans (MP) based on the results of the classification process. These results are used in the feature selection process in order to identify the class of interest. The objective is to highlight the effects of using the proposed measurement plans and its impact on the classification of new data and on the amount of data collected by this plan.

The used and analyzed measurement data are the measurement results provided by our industrial MEASURE platform. Data are collected at runtime from selected features/metrics.

Setup. We herein considered the following measurement plan determined by our experts. An initial MP can be defined by 15 features, 15 metrics and 4 software quality properties. As previously said, each metric is composed of only one feature and the mapping between metrics and classes has been provided by the previous step with the clustering approach.

Using the previously described plan, we considered the class with the most predicted instances during each cycle. A huge set of 16,000,000 unclassified vectors (unlabelled) were collected and processed (representing a collection of diverse data during a long period of time). This data set was divided into 32

Table 2. Measurement plans used during the suggestion process and the cycles where they were used. Metrics of the plans are represented by the indexes described in Table 1 (Source [7]).

	Metrics	Cycles
MP1	2, 5, 6, 7, 8	1
MP2	4, 5, 6, 12	2, 4, 17, 22, 23, 24
MP3	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 14, 15	3, 5, 18
MP4	8, 9, 10, 11	6, 30
MP5	7, 8, 9, 10, 11	7, 8, 9
MP6	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 13, 14, 15	10
MP7	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	11, 19, 20
MP8	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15	12, 21
MP9	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14	13, 14, 15, 16
MP10	3, 4, 5, 6, 8, 9, 10, 11, 12	25
MP11	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	26, 32
MP12	1, 2, 3, 4, 5, 6, 8, 9, 10, 11	27
MP13	1, 3, 4, 5, 6, 8, 9, 10, 11, 12	28
MP14	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12	29
MP15	1, 2, 3, 4, 5, 6, 7, 8, 9, 10	31

subsets each containing 500,000 vectors. For each period of the suggestion process, only one subset was used as input.

The initial measurement plan used during the experiment consisted of the following 5 metrics: Maintainability Index, Response Time, Running Time, Usability, Computational Cost. These metrics where selected by the experts as an example of a measurement plan with a small number of metrics that have links to all software quality properties. During the suggestion process a number was assigned to each metric as depicted in Table 1.

Results. During the suggestion process, 15 metrics (Table 1) were available to suggest new MP. Figure 9 shows how the classification of the vectors was distributed during the cycles and the percentage of the vectors assigned to each class. From these metrics, 15 unique measurement plans were used in the suggestion process. Table 2 lists the plans and in which cycle they were used.

MP1 was only used at the beginning of the process, this was the plan suggested by the expert. We note that MP2 was the most used plan during the process (6 times). This plan is composed by the metrics linked to the Performance property and was suggested when the classification of vector to class 3 overwhelmed the other classes. This tells us that if we focus on the Performance property then the metrics in MP2 are sufficient.

MP3 was suggested when the four classes were present in the classification results and class 4 was the class of interest. The tool suggests to take into consideration more than the linked metrics to the class, it seems that these features help to the classification of class 4.

MP4 was suggested when the input vectors were only classified to class 2, this MP2 consists of the metrics linked to that class. This happens when the input vectors are classified to only one class, the same can be observed in cycle

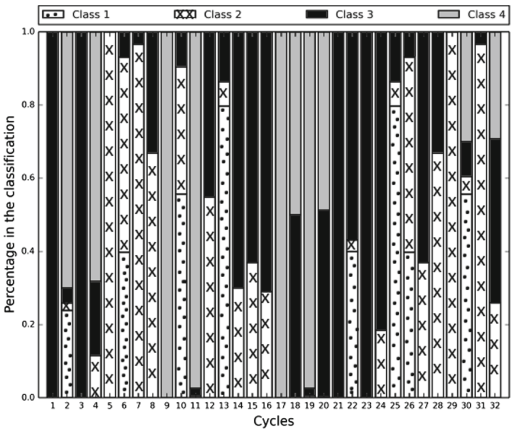

Fig. 9. Classification results of each cycle. The results show the percentage in the predictions of each cycles for the 4 classes (Source [7]).

1 but with class 3. MP5 has only one more metric than MP4, Usability. It is also a MP focused on System Performance property. MP11 was also suggested when class 2 overwhelmed the number of classifications during the classification phase.

MP7, MP8 and MP9 are very similar measurement plans. These plans have the highest number of metrics, MP7 15 metrics and MP8&9 14 metrics. These plans are suggested when the classification results usually have more than 2 classes. This is because the classes do not share any metric between them. A measurement plan with the majority of the metrics is expected to classify well the majority of the classes. MP10, MP12, MP13, MP14 and MP15 where suggested in the same case as the previously mentioned plans but these plans where only suggested one time during the process.

4.3 MINT Experiments

To test the efficiency of the MINT tool, we created 30 scripts enabling to generate different values for the fifteen metrics that are relevant for the correlation processes defined in the Fig. 5. For each correlation, we created 2 scripts: one that meets the condition that satisfies the recommendation and another that does not satisfy it. The 10/30 scripts are summarized in Table 3.

Table 3. Experiments scripts (Source [7]).

Correlation	Script	Metrics constraint
Code modularity	1	Class complexity/maintainability rating > threshold
Code modularity	2	Class complexity/maintainability rating < threshold
Specification quality	3	Number of reopened issues/number of issues > threshold
Specification	4	Number of reopened issues/number of issues < threshold
Management quality	5	Issues by severity = Major or Critical Reliability rating > 1 Major bug
Management	6	Issues by severity ≠ Major and ≠ Critical or Reliability rating < 1 Major bug
Security	7	Security vulnerability > Major vulnerability Security incident > threshold
Security	8	Security vulnerability < Major vulnerability or Security incident < threshold
Performance	9	Response time$_t$ > response time$_{t-1}$ bandwidth$_t$ = bandwidth$_{t-1}$
Performance	10	Response time$_t$ <= response time$_{t-1}$ or bandwidth$_t$ > bandwidth$_{t-1}$

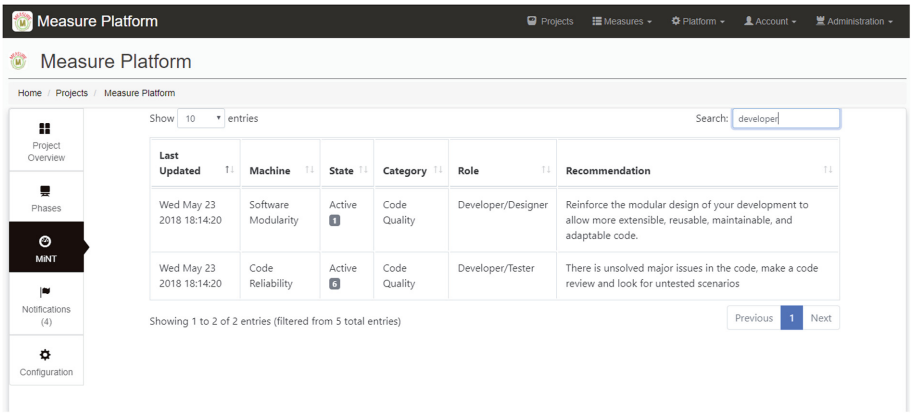

Fig. 10. Recommendation triggered by script 1.

Each script pushes the metric values into an event bus that feeds the 5 correlation processes defined in Sect. 3.4. The results correspond to the desired recommendations and the Fig. 10 displays an example of recommendation provided by the MINT tool for a software developer.

This experiment showed the efficiency of the tool. More work is planned to apply this tool to real datasets provided by real users in the context of the software development process.

5 Conclusion and Perspectives

This paper present an innovative approach to enhance software quality based on the analysis of a large amount of measurements generated during the software development process. The analysis is performed at different phases from the design to the operation and using different measuring tools (e.g., Hawk, SonarQube and MMT). The approach is implemented using two tools: Metric Suggester and MINT tools.

The Metrics Suggester tool is very valuable to reduce the energy and cost in gathering the metrics from different software life cycle phases and allows to reduce the number of the collected metrics according to the needs defined as profiles or clusters. It uses the support vector machine (SVM) that allows to build different classifications and provide the relevant measuring profile, the MP. The algorithm used in the tool as well some experiments demonstrate the efficiency of the tool to focus on relevant metrics depending the engineering process needs.

MINT is a rule based engine that relies on the ESFM formalism. It acts as a complex event processor that corrects the occurrence of measurements on time and provides a near real-time recommendation for the software developers and managers. The tool already integrates a set of default correlation rules that are able to provide valuable recommendations during the software development

and operation. The tool has been experimented using different scenarios and demonstrates an interesting added value

The data analysis platform of the MEASURE solution integrates the two tools and implements analytic algorithms (SVM and CEP) to correlate the different phases of software development and perform the tracking of metrics and their value. Correlations cover all aspects of the system like modularity, maintainability, security, timing, etc. and evaluate the global quality of the software development process and define actions (suggestions and recommendations) for improvements. The paper present the innovation of these tools and extended experiment according to the research paper published in [7]. More experiments are planned in the context of MEASURE ITEA-3 project with real use cases provided by industrial partner. We believe that these experimentation will allow to facilitate the exploitation of the tools in industrial contexts.

Acknowledgment. This work is partially funded by the ongoing European project ITEA3-MEASURE started in Dec. 1st, 2015, and the EU HubLinked project started in Jan. 1st, 2017.

References

1. Akbar, M.A., et al.: Improving the quality of software development process by introducing a new methodology-AZ-model. IEEE Access **6**, 4811–4823 (2018). https://doi.org/10.1109/ACCESS.2017.2787981
2. Alshayeb, M., Shaaban, Y., AlGhamdi, J.: SPMDL: software product metrics definition language. J. Data Inf. Qual. **9**(4), 20:1–20:30 (2018). https://doi.org/10.1145/3185049
3. Bagnato, A., da Silva, M.A.A., Abherve, A., Rocheteau, J., Pihery, C., Mabit, P.: Measuring green software engineering in the MEASURE ITEA 3 project. In: Condori-Fernández, N., Procaccianti, G., Calero, C., Bagnato, A. (eds.) Proceedings of the 3rd International Workshop on Measurement and Metrics for Green and Sustainable Software Systems, MeGSuS 2016, Co-Located with 10th International Symposium on Empirical Software Engineering and Measurement (ESEM 2016), Ciudad Real, Spain, 7 September 2016. CEUR Workshop Proceedings, vol. 1708, pp. 33–42. CEUR-WS.org (2016). http://ceur-ws.org/Vol-1708/paper-06.pdf
4. Bouwers, E., van Deursen, A., Visser, J.: Evaluating usefulness of software metrics: an industrial experience report. In: Notkin, D., Cheng, B.H.C., Pohl, K. (eds.) 35th International Conference on Software Engineering, ICSE 2013, San Francisco, CA, USA, 18–26 May 2013, pp. 921–930. IEEE Computer Society (2013). https://doi.org/10.1109/ICSE.2013.6606641
5. Carvallo, J.P., Franch, X.: Extending the ISO/IEC 9126-1 quality model with non-technical factors for cots components selection. In: Proceedings of the 2006 International Workshop on Software Quality, WoSQ 2006, pp. 9–14. ACM, New York (2006). https://doi.org/10.1145/1137702.1137706. http://doi.acm.org/10.1145/1137702.1137706
6. Dahab, S.A., Maag, S., Hernandez Porras, J.J.: A novel formal approach to automatically suggest metrics in software measurement plans. In: 2018 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE). IEEE (2018)

7. Dahab, S.A., Silva, E., Maag, S., Cavalli, A.R., Mallouli, W.: Enhancing software development process quality based on metrics correlation and suggestion. In: Proceedings of the 13th International Conference on Software Technologies, ICSOFT 2018, Porto, Portugal, 26–28 July 2018, pp. 154–165 (2018)
8. De Maesschalck, R., Jouan-Rimbaud, D., Massart, D.L.: The Mahalanobis distance. Chemometr. Intell. Lab. Syst. **50**(1), 1–18 (2000)
9. Fenton, N., Bieman, J.: Software Metrics: A Rigorous and Practical Approach. CRC Press, Boca Raton (2014)
10. Fenton, N.E., Pfleeger, S.L.: Software Metrics - A Practical and Rigorous Approach, 2nd edn. International Thomson, Boston (1996)
11. García-Domínguez, A., Barmpis, K., Kolovos, D.S., da Silva, M.A.A., Abherve, A., Bagnato, A.: Integration of a graph-based model indexer in commercial modelling tools. In: Baudry, B., Combemale, B. (eds.) Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, 2–7 October 2016, pp. 340–350. ACM (2016). https://doi.org/10.1145/2976767. http://dl.acm.org/citation.cfm?id=2976809
12. García-Munoz, J., García-Valls, M., Escribano-Barreno, J.: Improved metrics handling in SonarQube for software quality monitoring. In: Omatu, S., et al. (eds.) Distributed Computing and Artificial Intelligence, 13th International Conference. AISC, vol. 474, pp. 463–470. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40162-1_50
13. Ge, Z., Song, Z., Ding, S.X., Huang, B.: Data mining and analytics in the process industry: the role of machine learning. IEEE Access **5**, 20590–20616 (2017)
14. Grez, A., Riveros, C., Ugarte, M.: Foundations of complex event processing. CoRR abs/1709.05369 (2017). http://arxiv.org/abs/1709.05369
15. Group, O.M.: Structured Metrics Metamodel (SMM) (October), pp. 1–110 (2012)
16. Hauser, J., Katz, G.: Metrics: you are what you measure!. Eur. Manag. J. **16**, 517–528 (1998)
17. ISO/IEC: ISO/IEC 25010 - systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Technical report (2010)
18. Jain, A.K.: Data clustering: 50 years beyond k-means. Pattern Recogn. Lett. **31**(8), 651–666 (2010)
19. Kevrekidis, K., et al.: Software complexity and testing effectiveness: an empirical study. In: 2009 Annual Reliability and Maintainability Symposium, RAMS 2009. IEEE (2009)
20. Khalid, S., Khalil, T., Nasreen, S.: A survey of feature selection and feature extraction techniques in machine learning. In: Science and Information Conference (SAI), pp. 372–378. IEEE (2014)
21. Kitchenham, B.A.: What's up with software metrics? - A preliminary mapping study. J. Syst. Softw. **83**(1), 37–51 (2010). https://doi.org/10.1016/j.jss.2009.06.041
22. Laradji, I.H., Alshayeb, M., Ghouti, L.: Software defect prediction using ensemble learning on selected features. Inf. Softw. Technol. **58**, 388–402 (2015). https://doi.org/10.1016/j.infsof.2014.07.005
23. Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. **27**(C), 504–518 (2015). https://doi.org/10.1016/j.asoc.2014.11.023

24. van der Meulen, M., Revilla, M.A.: Correlations between internal software metrics and software dependability in a large population of small C/C++ programs. In: ISSRE 2007, The 18th IEEE International Symposium on Software Reliability, Trollhättan, Sweden, 5–9 November 2007, pp. 203–208 (2007)
25. Papadopoulos, L., Marantos, C., Digkas, G., Ampatzoglou, A., Chatzigeorgiou, A., Soudris, D.: Interrelations between software quality metrics, performance and energy consumption in embedded applications. In: Stuijk, S. (ed.) Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, SCOPES 2018, Sankt Goar, Germany, 28–30 May 2018, pp. 62–65. ACM (2018). https://doi.org/10.1145/3207719.3207736
26. Pelleg, D., Moore, A.W., et al.: X-means: extending k-means with efficient estimation of the number of clusters. In: ICML, vol. 1, pp. 727–734 (2000)
27. Shepperd, M.J., Bowes, D., Hall, T.: Researcher bias: the use of machine learning in software defect prediction. IEEE Trans. Software Eng. **40**(6), 603–616 (2014). https://doi.org/10.1109/TSE.2014.2322358
28. Shweta, S.S., Singh, R.: Analysis of correlation between software complexity metrics. IJISET Int. J. Innovative Sci. Eng. Technol. **2**(8), 902–905 (2015)
29. Vapnik, V.N., Vapnik, V.: Statistical Learning Theory, vol. 1. Wiley, New York (1998)