A Novel Formal Approach to Automatically Suggest Metrics in Software Measurement Plans

Sarah A. Dahab, Juan Jose Hernandez Porras and Stephane Maag
Telecom SudParis, CNRS UMR 5157, Univ. Paris-Saclay, France

Keywords: Software Metrics, Formal Measurement, Software Measurement, Measurement Plan, Formal Software Measurement, SVM, Big Data.

Abstract: The software measurement is an integral part of the software engineering process. With the rise of the software system and their complexity distributed through diverse development phases, the software measurement process has to deal with more management and performance constraints. In fact, the current software measurement process is fixed and manually planned at the beginning of the project and has to manage a huge amount of data resulting from the complexity of the software. Thereby, measuring software becomes costly and heavy. In addition, the implementation of the measures is dependent on the developer and reduce the scalability, maintainability and the interoperability of the measurement process. It becomes expert-dependent and thus more costly. In order to tackle these difficulties, first, we propose in this paper a formal software measurement implementation model based on the standard measurement specification SMM. Then, a software measurement plan suggestion framework based on a learning-based automated analysis.

1 INTRODUCTION

The rise of software systems and their complexity distributed through diverse development phases and projects lead to a huge amount of data to manage, estimate and evaluate.Indeed, considering the quantity of aspects to be measured raising the relevant information to be analyzed and reported become difficult (as concerned by Microsoft Power BI[1]). In this context software measurement becomes then crucial as part of software development projects while the measurement processes become tough. Thus, to ensure a quality and efficient software engineering process, adapted measurement processes are required.

Many works have been done on software measurement to understand and formalize the measurement process in software engineering context. These studies have shown the importance of gathering information on the software engineering processes in particular to ensure its quality through metrics and measurements analysis (Fenton and Bieman, 2014). Thanks to that, standardization institutes worked in that way to propose two well-known norms, ISO/IEC25010 (ISO/IEC, 2010) and OMG SMM (Group, 2012) to guide to the measurement plan specification. These two standards have been reviewed by the research community and industrials, and are adapted and applied in many domains.

[1]https://powerbi.microsoft.com/

Though, an important base is defined and well-known through measurement models which define the measurable software criteria and their associated measures, the manual measurement planning becomes heavy to manage with the quantity of aspects to be measured. It leads to very complex measurement plans, engendering eventual losses of time and performance. These plans are often constrained to: manual planning of the measurements, its serial execution, a lack of structure due to the use of informal metrics, and a lack of flexibility due to fixed measurement plans.

The objectives of this article are: first, to improve the software measurement design phase. The purpose is to use the code generation feature to facilitate a more generic implementation of software measurements.Thereby we propose a software measurement UML model from the formal measurement specification SMM. Secondly, to optimize the manual measurement management by making the metrics use more flexible. Thus, we define a formal data model and design a suggestion process that selects metrics from the current measurement plan and suggests a novel plan. We enable to cover an important

Dahab, S., Porras, J. and Maag, S.
A Novel Formal Approach to Automatically Suggest Metrics in Software Measurement Plans.
DOI: 10.5220/0006710902830290
In *Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2018)*, pages 283-290
ISBN: 978-989-758-300-1

field of measurement by using the Support Vector Machines whose parameters are well chosen (automatically searched).

Finally, our paper is organized as following. The section 2 presents a state of art on software measurements, and learning technique applied to this area. Then, the section 3 introduces fundamental definitions of measurement, the formal measurement specification SMM based on this latter and our design model based on SMM. Our framework is presented in detailed in the section 4. And we apply our approach in Section 5 on a real case study. The experiments are successfully evaluated and discussed. Finally, in Section 6, we conclude and give the perspectives for improvement.

2 RELATED WORKS

The studies on software measurement define the measurement plan as part of the measurement process where a set of metrics is selected according to the property of the measure (Fenton and Bieman, 2014). Several studies propose data models specific to a property. The data models are presented as links between factors, criteria and metrics such as the McCall's model, the standard quality model (Fenton and Bieman, 2014; ISO/IEC, 2010) which bring this selection of metrics to measure the software quality. But the actual implementation and management of all this data is left to the project manager.

While an important set of metrics has been defined for diverse software domains, very few have been used to improve the measurements plans. In (Gao et al., 2011; Wang et al., 2011), the authors propose a prediction model in focusing on the problem of attribute selection in the context of software quality estimation to select metrics.

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 (Laradji et al., 2015; Shepperd et al., 2014; Prasad et al., 2015) raise that issue by proposing diverse kinds of machine learning approaches for SW defect prediction through SW metrics. However, although these papers present interesting results, they are dependent on the targeted measurement scope and somehow do not suggest novel metrics according to the running measured project.

3 FROM METROLOGY TO SOFTWARE MEASUREMENT CODE DESIGN

3.1 Measurement Fundamentals

The fundamental mathematics defines a measurable space M as a pair composed of the measured object X and the set of measurable properties A of X such as:

$$M = (X, A) | A \in X \qquad (1)$$

Then, a measure is defined as a function f which assigns a formal value B of a defined formal set $\mathbb{B}$ to a set of properties measurable A of an object X such as:

$$f : A \to B | A \in X, B \in \mathbb{B} \qquad (2)$$

Thus, a measure space MS is defined as the triplet (X, A, f) composed of the measurable space (X, A) and its associated function f , such as f is an application on A as defined below:

$$MS = (X, A, f) | A \in X, f(A) \qquad (3)$$

Finally, two measurable functions f and g such as:

$$f, g : A \to B | A \in X, B \in \mathbb{B} \qquad (4)$$

linked by the relations $R = (+, \times, \div, min, max)$ is a measurable function m, herein called dependent, such as

$$\begin{aligned} m : fRg \to B | B \in \mathbb{B} \\ m : A \to B | B \in \mathbb{B} \end{aligned} \qquad (5)$$

The measurement y is the result of the application f(A) of the function f on the measurable set A such as:

$$y = f(A) | A \in X \qquad (6)$$

3.2 Software Measurement Definitions

In the software engineering context, the software measurement terminologies are defined as below:

Measurand: a measurand is the measured object. In this context, it is a software system, such as software product, in use or software resource. It refers to the element X in a measurable space in the fundamental point of view (FPV).

Software Properties: the software properties are the measurable properties of a software such as complexity or performance. In the FPV, software properties refer to the measurable set A of a MS.

Measurement: a measurement is defined as a direct quantification of a measured property (Fenton and Bieman, 2014). 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. In the fundamental point of view (FPV), the measurement refers to the result of a function of a Measure Space (MS).

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. In the FPV, a measure is the function of a MS.

Direct Measure: is the measure independent of other measures, thus it refers to the simple function in the FPV.

Indirect Measure: is a measure dependent on other measures. It refers to the independent function in the FPV.

Metric: a metric refers to the measure space in the fundamental view. It is the specification of a measurement. The formal definition of a measurement of a property of a computer object. It specifies the measurand, the measure(s) and the property to be measured.

Complex Metric: a complex metric is a metric composed of indirect measure(s).

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.

3.3 Formal Specification and Design Model

In order to optimize the design phase of the implementation of a software measurement, we propose to design a generic UML model from a specification modeling with the OMG's standard SMM (Structured Metrics Meta-model). The purpose is to allow a measurement code generation from a measurement architecture model based on SMM. Indeed, generate a model of the implementation structure from a model of the measurement architecture with SMM allow the use of the code generation feature.

Moreover, this process will allow to have documentation on the measurement architecture with the SMM model, and documentation on the measurement code structure with the UML model and thus, reduce the load of the developer of the manual implementation designing.

In addition, the purpose of having documentation through models and code generation from this latter is to increase the interoperability, scalability and maintainability of the measurement process while decreasing the expert dependency.

SMM: SMM is a standard specification which 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. And a wide range of diversified types of measures is proposed to define the dependency type between dependent measure (as the ratio, binary or grade measure). The SMM specification allows the description in detail of the main following concepts as illustrate in the Fig. 1:

- MeasureLibrary
- Measure
  - Scope
  - Unit of measure
- ObservedMeasure
- Measurement

These concepts refer to the concepts defined in the previous section. The *Scope* is the subset defining the measured property on which the measure is applied. For example, to evaluate the software complexity, we measure the code architecture. Thus, the measurable property is complexity and the measure scope is the code.

The $Unitofmeasure$ allows to specify the type of result.

The *MeasureLibrary* refers to the set of linked measures. And $Measure$ refers to the function of measurement.

Then, $ObservedMeasure$ allows to associate a measure of a libraryMeasure to a measurement. As example, we define a property to be measured which needs the assessment of several dependent measures. $ObservedMeasure$ defines the measure which is associated to the final measurement result, corresponding to the global measurement of the measured property. This information is to help the developer to understand the specification.

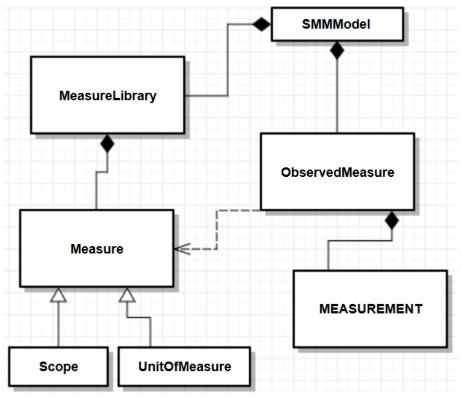

Figure 1: SMM Overview.

In this paper, we base on SMM to model a generic software measurement implementation design as a code structure to guide the implementation of measures.

SMM-based Software Measurement Code Structure Model: From the meta-model SMM, we propose a generic UML model defining the code structure of a software metric. As illustrated in the Fig. 2, the model is made of the principal class Measure which composed of two objects:

- Scope which consists of the definition of software structure to be measured such as the file code or a repository path.
- Measurement represents a measurement result and its unit of measure.

And specified in two types of measures:

- DirectMeasure which refers to the independent measure. This class is free from any other measures.
- IndirectMeasure which has dependencies with other measures.

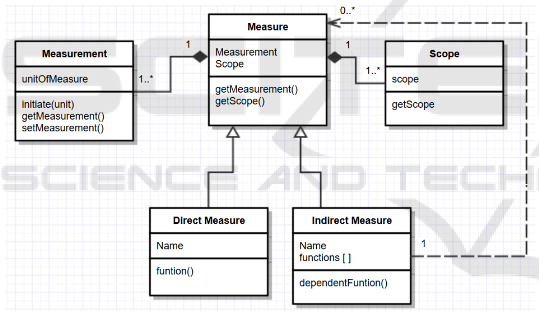

Figure 2: UML model designing a Software Measurement Code architecture.

In the next section, we propose an approach of a software measurement analysis and interpretation as contribution to the improvement of the measurement execution phases.

4 OUR AUTOMATED DYNAMIC MEASUREMENT PLAN MANAGEMENT FRAMEWORK

In order to optimize the current measurement process which are manual and static and thus very costly, we propose an automated analysis and suggestion as an approach, by using the learning technique SVM.

4.1 Automated Software Measurements Analysis

SVM: A support vector machine (SVM) (Vapnik and Vapnik, 1998) 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 $(x_1, y_1), \ldots, (x_l, y_l)$, where each vector $x_i \in R^n$ 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 (7) defines a separating hyperplane

$$f(x) = w^T x + b = 0 \tag{7}$$

where $w \in R^n$ and $b \in R$ are parameters that control the function. Function f gives the signed distance between a point x and the separating hyperplane. A point x is assigned to the positive class if $f(x) \geq 0$, and otherwise to the negative class.

Features & Classes: In this paper, we use the SVM in order to classify a set of measurements. We define this set of measurements 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.

Software Measurement Classification: Our framework is based on the SVM classification results to orient the measurement plan. It aims at highlighting the main observed software characteristics during the measurements. We name them as the *properties of interest*. The principle is to classify a vector in the class corresponding to the property whose values of the vector show this type of interest. Then, the class with the most number of classified vectors, called *Biggest*, corresponds to the property(ies) of interest, while the others with less number of classified vectors, called *Others*, correspond to the properties that do not show much interests. The measurement plan should insist on the first property, unlike the others. Thus, a specific procedure is defined according to the class put forward to guide the measurement plan by suggesting the measurements on the concerned property. For that purpose, the mapping system is used.

The algorithm defining this procedure is called *Analysis*. It takes as input a set of vectors of float called *vectors*. And as output, it returns the name of the class with the most number of classified vectors such as describe by the pseudo-code below:

```
Procedure : Analysis
begin
input : vectors float[X] | X number of measures
classification (vectors)
output : string className
end
```

4.2 Measurement Plan Suggestion

The Mapping System: This mapping system aims to allows to determine which features should be gathered for the vectors analysis and for which gathering should be stopped. This decision is based on the classification result and enables to improve the measurement process at this level. In other words, by regulating the measurements gathering as needed, this ensures to improve the performance of the data management for the analysis.

We map classes with metrics, and metrics with features. 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: The learning classes refer to the different measured properties of a software. The metrics defined for the measurement process are those which allow to measure these 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. Thus, the metrics associated to a class are the needed measurements to measure the property represented by the class.

Features-Metrics: The features of a vector refer to the results of the measures making up the metrics defined for the software measurement. They are used to classify the vectors. Their 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. 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.

We define the mapping system as a data structures. The first is a two dimensions list to link each class with the list of related metrics, called *classMetrics* . And the second a two dimensions list to link each feature of the list of related metrics, called *featMetrics*. The pseudo-code is defined below:

```
DataStructure :
begin
        classMetrics string[X][]
        featMetrics string[X][]
end
```

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 purpose is to have at each cycle a dynamic selection of the mandatory features.

The Feature Selection: The Feature Selection (FS) process goal is to find the relevant features for the classes found in the classification process by determining a subset of features that collectively have good predictive power. With feature selection, our objective is to highlight the features that are important for classification process. This leads to the possibility of not considering all the features but only the ones corresponding to the types of interest.

The feature selection method used is Recursive Feature Elimination (RFE) (Khalid et al., 2014). RFE process is optimized for the classifier to be used. RFE performs backward elimination. Backward eliminations consist of starting with all the features and test the elimination of each variable until no more features can be eliminated. RFE starts with a classifier that was trained will all the features and each feature is assigned a weight. 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. 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.

We define the algorithm *Selection* which takes as input of float called *vectors* and return as output a list of features *selectedfeatures*. And apply the RFE technique on the *vectors* such as described below:

```
Procedure : Selection
begin
```

```
input : vectors float[X] | X number of measures
        RFE (vectors)
output : selectedFeatures string [Y] | Y in X
end
```

Measurement Plan Suggestion Algorithm: Following the classification, two sets of classes are highlighted: 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* algorithm, described below, 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.

This procedure is based on the data structure and both procedures *Analysis* and *Selection* presented above. It takes as input a set of vectors *vectors* and it returns as output a set of metrics as new suggested measurement plan.

```
Procedure : Suggestion
begin
map DataStructure
biggest string
selectedFeats string[Y]
input : vectors float[X] | X number of measures
  biggest = Analysis(vectors)
  selectedFeats = Selection(vectors)
selectMs string []
  for each i in selectedFeats :
        feats=selectedFeats[i]
        selectMs[i]=map.featMetrics[feats]
  end for
output :
map.classMetrics[biggest] + selectMs
end
```

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.

5 EXPERIMENTS

Our methodology has been implemented and integrated in a tool. We assess our suggestion process by analyzing the new measurement plan based on the results of the classification process and using them in the feature selection process and to identify the class of interest. The objective is to denote the effects of using the suggested measurement plan and its impact on the classification of new data and the amount of data gathered by that plan.

The case study of this experiment is an in use Oriented Object platform of the European project MEASURE[2]. The measurement data used are the measurement result applied on this platform.

5.1 Setup

We herein considered the following measurement plan which is determined by the expert. A plan with 15 features, 15 metrics and 4 software quality properties. Each metric is composed of only one feature and the mapping between metrics and classes is the following:

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

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 were processed. This data set was divided into 32 subsets each containing 500,000 vectors. For each period of the suggestion process, only one subset was used as input.

The initial measurement plan use during the experiment consisted of the following 5 metrics: Maintainability Index, Response Time, Running Time, Usability, Computational Cost. These metrics where selected by the expert as an example of a measurement plan with a small number of metrics that has links to all the software quality properties.

During the suggestion process each metric is assigned a number, in our experiments the number each metric was assigned is shown in Table 1.

5.2 Results

During the suggestion process 15 metrics (Table 1) were available to suggest new plans. With these metrics 15 unique measurement plans, 15 different sets of metrics, were used in the suggestion process. Table 2 lists the plans and in which cycle they were used.

[2]https://itea3.org/project/measure.html

Table 1: Each metric and its assigned index during the suggestion process.

Index	Metric
1	Cognitive Complexity
2	Maintainability Index
3	Code Size
4	Bugs
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

The progress of the number of metrics in the measurement plan can be seen in Fig. 3. The number of metrics move between only 4 metrics and the maximum a plan with all the metrics.

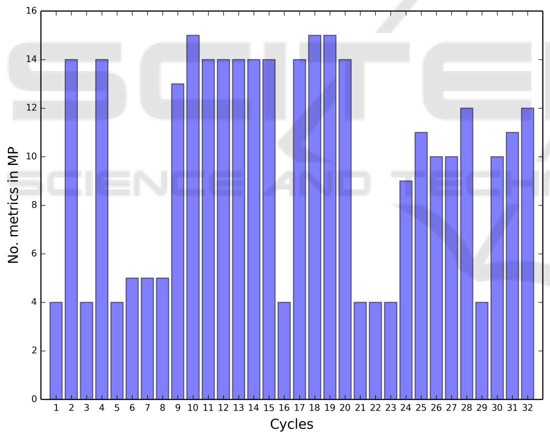

Figure 3: Number of metric of the suggested plan in each cycle.

Fig. 4 show how the classification of the vectors was distributed during the cycles, the percentage of the vectors assigned to each class.

Starting with MP1, this plan was only used during the start of the process, this was the plan suggested by the expert. Then MP2, this was the most used plan during the process (6 times), this plan is form by the metrics linked to the Performance property and was suggested when the classification of vector to class 3 overwhelm the other classes. This tells us that to focus on the Performance property the metrics in MP2 are sufficient.

MP3 was suggested when the four classes where

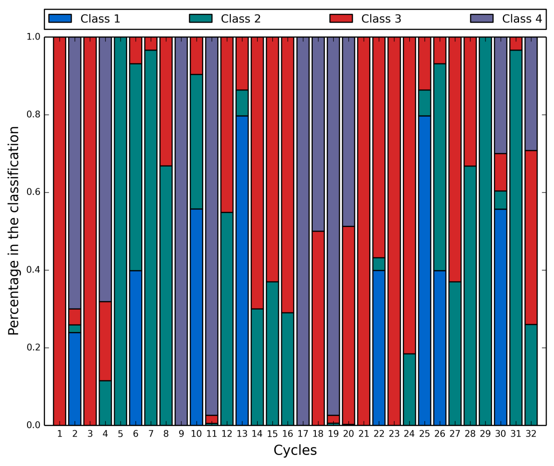

Figure 4: Classification results of each cycle. The results show the percentage in the predictions of each cycles for the 4 classes.

present in the classification results and class 4 was the class of interest. The tool is suggesting to take into consideration more than the linked metrics to the class, it seems that this features help to the classification of class 4.

MP4 was suggested when the input vectors were only classified to class 2, this MP2 consist 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 1 but with class 3. MP5 only has one more metric that MP4, Usability, and also is a measurement plan focus in System Performance property. And MP11 also was suggested when class 2 overwhelm the number of classifications during the classification.

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. This 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 must of the metrics is expected to classified well the majority of the classes. MP10, MP12, MP13, MP14 and MP15 where suggested in the same case as the previously mention plans but this plans where only suggested one time during the process.

6 CONCLUSION AND PERSPECTIVES

To conclude, in this article we propose a formal design approach of software measurement by basing on the OMG's standard specification SMM. The goal is

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 describe in Table 1.

	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

to have a solid formal basis to improve the software measurement implementation phase, in order to increase the interoperability, scalability and maintainability of the measurement process.

Furthermore, we design an automated measurement plan suggestion framework to add flexibility, and expert-independent analysis. The learning technique SVM is used combined with the RFE algorithm and allow to manage a huge amount of data.

Our methodology has been implemented and successfully experimented on a real use case. The tool was able to manage a large data set, the tool managed 16 million unclassified vectors.

As future works, we plan to integrate the formal design model to an industrial modeling tool of our partner of the European project MEASURE.

Regarding the suggestion tool, we plan to validate our approach by comparing our results with results of the actual processes. To support a bigger amount of data by increasing the number of unclassified instances and using a training file with more samples with a larger vector. Then, to improve the suggestion we plan to add the possibility to generate at runtime a novel combined metric by basing on the analysis. In addition, we expect to improve the analysis visualization and reporting to a better readability.

Finally, we project to define innovative metrics as emotional one-which measure the user emotions-for measuring the quality of video games, or the quality of user experience for VoD use as example. In other words, to measure usability of an industrial system.

REFERENCES

Fenton, N. and Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC Press.

Gao, K., Khoshgoftaar, T. M., Wang, H., and Seliya, N. (2011). Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience*, 41(5):579–606.

Group, O. M. (2012). Structured metrics metamodel (smm). (October):1–110.

ISO/IEC (2010). Iso/iec 25010 - systems and software engineering - systems and software quality requirements and evaluation (square) - system and software quality models. Technical report.

Khalid, S., Khalil, T., and Nasreen, S. (2014). A survey of feature selection and feature extraction techniques in machine learning. In *Science and Information Conference (SAI), 2014*, pages 372–378. IEEE.

Laradji, I. H., Alshayeb, M., and Ghouti, L. (2015). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58:388–402.

Prasad, M. C., Florence, L., and Arya, A. (2015). A study on software metrics based software defect prediction using data mining and machine learning techniques. *International Journal of Database Theory and Application*, 8(3):179–190.

Shepperd, M., Bowes, D., and Hall, T. (2014). Researcher bias: The use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6):603–616.

Vapnik, V. N. and Vapnik, V. (1998). *Statistical learning theory*, volume 1. Wiley New York.

Wang, H., Khoshgoftaar, T. M., and Napolitano, A. (2011). An empirical study of software metrics selection using support vector machine. In *The 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE)*, pages 83–88.