



An approach to measuring software systems using new combined metrics of complex test

Sarah Dahab

► To cite this version:

Sarah Dahab. An approach to measuring software systems using new combined metrics of complex test. Networking and Internet Architecture [cs.NI]. Université Paris Saclay (COMUE), 2019. English. NNT : 2019SACLL015 . tel-02374756

HAL Id: tel-02374756

<https://theses.hal.science/tel-02374756>

Submitted on 21 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An approach to measuring software systems using new combined metrics of complex test

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom SudParis

Ecole doctorale n°580 Sciences et Technologies de l'information et de la
communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Paris, le 13 septembre 2019, par

SARAH A. DAHAB

Composition du Jury :

Jean-Marc DELOSME Professeur, Université Evry Val d'Essonne (IBISC)	Président
Abdelhamid MELLOUK Professeur, Université Paris Est (TincNET-LiSSi)	Rapporteur
Sébastien SALVA Professeur, Université d'Auvergne (LIMOS)	Rapporteur
Alessandra BAGNATO Chargé de recherche, SOFTEAM Cadextan, R&D	Examineur
Jérôme ROCHETEAU Maître de Conférences, ICAM Nantes (LS2N)	Examineur
Marcos ALMEIDA Chargé de recherche, Invivoo Software	Examineur
Stéphane MAAG Professeur, TélécomSud Paris (SAMOVAR)	Directeur de thèse

Résumé

Les systèmes logiciels sont de plus en plus complexes, ce qui s'explique par la quantité de tâches qu'ils doivent assurer tout en respectant des contraintes strictes dues à leur utilisation quotidienne. Ces exigences imposent une conception de système logiciel de niveau élevé. Assurer une telle conception exige un processus de développement de qualité, et donc une supervision adaptée.

En effet, selon l'ISO/IEC 25010 un logiciel de qualité doit répondre à 8 critères chacun spécifié en plusieurs sous-propriétés. Pour mieux comprendre les enjeux d'un processus de supervision dans un tel contexte, il est important de saisir son fonctionnement. Une supervision se base sur un processus de mesure qui se définit en 3 étapes majeures: l'étape de conception de mesure, qui consiste à déterminer la ou les caractéristique(s) d'un objet à observer et le plan de mesure qui permet de récolter la ou les information(s) souhaitée(s) sur cette(ces) caractéristique(s); l'étape d'exécution des mesures, qui consiste à exécuter les métriques et à récolter les résultats de mesure; et enfin l'analyse des résultats. Pragmatiquement, les processus de mesure sont conçus selon un flux séquentiel : tout d'abord la conception du processus de mesure, puis l'exécution des métriques correspondantes, et enfin l'analyse des résultats de mesure.

Ainsi, dans un tel contexte, ce type de processus est soumis à de nombreux challenges: notamment le coût de gestion de conception, le coût de mesure et enfin la charge de l'analyse. Ces enjeux s'expliquent par différentes problématiques: un manque de base formelle, les définitions et développements de métriques ne suivent pas de standard; le plan de mesure statique, celui-ci est défini au début du processus; l'analyse séquentielle, les résultats de mesure sont analysés les uns après les autres; et le nombre important de quantité de données à analyser et à gérer étant donné la complexité des systèmes évalués.

Afin de traiter ces problématiques, je propose, dans un premier temps, d'améliorer la phase de conception en formalisant un ensemble de métriques selon un standard. Dans un deuxième temps, de réduire le coût d'analyse en utilisant une technique d'apprentissage supervisée. Puis d'avoir un plan flexible qui évolue dans le temps selon les analyses au travers de suggestions de plans de mesure en temps réel. Et enfin, de réduire le coût de gestion de données en utilisant une technique d'apprentissage non supervisée pour générer le modèle d'analyse.

Dans le domaine de la mesure logicielle, la littérature est abondante. En effet, une pléthore de travaux a été menée, notamment concernant les bases d'un processus de mesure logicielle

tel que des méthodologies pour concevoir des plans de mesure et des modèles définissant des propriétés logicielles.

De plus, des recherches ont permis de mettre en lumière certains enjeux, comme la difficulté d'évaluer le coût d'un plan de mesure ou encore la réalité du "Big data" auquel ce domaine doit faire face, etc.

Le premier axe de recherche de cette thèse a pour objectif d'améliorer la phase de conception. Pour cela, je propose de spécifier et de développer formellement des métriques logicielles selon le standard SMM de l'OMG. Ce standard est un métamodèle qui définit tous les éléments nécessaires à la spécification d'une métrique logicielle. Pour atteindre cet objectif, j'ai participé au développement et utilisé un langage graphique basé sur le langage UML qui permet de modéliser graphiquement des métriques logicielles selon le standard SMM. Modéliser des métriques, permet d'avoir une spécification standard et une architecture de code qui suit le standard. Une telle architecture permet de garantir une maintenabilité et une scalabilité des métriques.

Le second axe de recherche consiste à traiter les enjeux liés aux plans de mesure statiques et à l'analyse séquentielle. L'objectif est double: le premier est de suggérer en temps réel des plans de mesure adaptés aux réels besoins du logiciel. Le second est d'automatiser l'analyse de beaucoup de données en utilisant une technique d'apprentissage supervisée. Cela permettra d'avoir des plans de mesure flexibles tout au long du processus de mesure et d'analyser en même temps l'ensemble des métriques composant le plan de mesure en cours d'exécution. Pour cela, je propose une approche basée sur trois procédures:

- l'initialisation, qui définit manuellement les stratégies de mesure et le modèle d'analyse. Cette étape est statique et conçue au début du processus de mesure.
- l'analyse basée sur la technique d'apprentissage supervisée SVM et RFE.
- et enfin la suggestion de plan de mesure qui se base sur le résultat de l'initialisation et d'analyse.

Cette approche a été développée et expérimentée avec des résultats encourageants. Néanmoins, la gestion des données est encore très coûteuse. Notamment pour générer le fichier d'entraînement.

Comme troisième axe de recherche, je propose donc de pallier à cette difficulté en automatisant l'étape d'initialisation du modèle d'analyse. Pour cela, plutôt que de définir manuellement l'interprétation des données, je propose d'utiliser une technique d'apprentissage non-supervisée, X-MEANS, pour apprendre à partir de données brutes du logiciel observé et de générer automatiquement un modèle d'analyse. L'intervention de l'expert ne sera effectuée que pour valider ce modèle.

Cette approche a été expérimentée sur le même cas d'étude que le précédent avec de très bons résultats qui remettent en question la cohérence des modèles de données utilisés avec la réalité, qui est dû à la difficulté de trouver des données réelles interprétables.

Pour conclure, j'ai proposé plusieurs solutions comme contribution à l'amélioration du processus de mesure répondant aux problématiques soulevées en introduction. J'ai spécifié et développé formellement des métriques de différent niveau, architecture, verte et émotionnel selon leur langage standard auquel j'ai participé à l'élaboration. Ces métriques ont été intégrées à une plateforme industrielle, du projet européen MEASURE auquel j'ai participé durant cette thèse, comme contribution à la formalisation de la phase de conception du processus de mesure et comme outils de mesure. De plus, par ce travail d'analyse du standard SMM, j'ai contribué en collaboration avec le laboratoire de recherche Softeam, partenaire du projet MEASURE, à la nouvelle version du standard en remontant quelques incohérences.

Puis j'ai proposé une approche d'analyse automatique et de suggestion de plan de mesure en temps réel afin de dynamiser le processus de mesure et d'améliorer la gestion de grands ensembles de données. Cette approche a été développée et expérimentée avec des résultats assez encourageants et intégrée à la plateforme industrielle comme outil d'analyse.

Enfin, j'ai proposé d'utiliser une technique d'apprentissage non supervisée pour améliorer l'approche précédente. Les résultats sont également encourageants, ils soulèvent tout de même des problématiques sur la pertinence du modèle de données générée et le manque de données réelles et interprétables dans le domaine. En outre, l'approche présente des points à améliorer. Notamment le cycle de mesure qui est statique.

Ces problématiques soulevées et les difficultés auxquelles j'ai été confrontée amorcent quelques perspectives de recherche notamment sur le manque de référence standard dans le domaine. Aujourd'hui, dans le domaine du logiciel, il n'y a pas de référence d'interprétation, cette dernière est à la merci du chef du projet, il est donc intéressant pour moi de contribuer à cette amélioration en travaillant sur la découverte de seuils générique en essayant dans un premier temps de trouver des corrélations entre les données de différents projets et dans un deuxième temps d'y associer des interprétations génériques. Ce qui pourrait peut-être permettre de mettre en place des protocoles de mesure standard à l'instar d'autres domaines scientifiques.

Concernant les cycles de mesure statique, je propose d'utiliser des métriques émotionnelles dans le but d'évaluer un logiciel et de rendre par ce biais les cycles de mesure plus flexible. Pour ce faire, il conviendrait de trouver des corrélations entre les émotions d'utilisateurs et des caractéristiques purement logicielles. Ce travail a commencé durant cette thèse et est en cours de réalisation. Une première étape a été réalisée: la détermination des métriques émotionnelles nécessaires, leurs spécification et développement formel selon le standard SMM. De plus, elles ont été intégrées à la plateforme industrielle du projet MEASURE. Ainsi, les perspectives de ce travail consistent à déterminer les corrélations entre les mesures émotionnelles et les caractéristiques logicielles. Ce travail est en cours de réalisation en collaboration avec l'université JIAOTONG de pékin.

Publications

- Dahab, S. A., Maag, S., Bagnato, A., & da Silva, M. A. A. (2016). A learning based approach for green software measurements. In *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, september 7, 2016*. (Pages 13–22).
- Dahab, S. A., Maag, S., & Che, X. (2017). A software measurement framework guided by support vector machines. In *31st international conference on advanced information networking and applications workshops, AINA 2017 workshops, taipei, taiwan, march 27-29, 2017* (Pages 397–402).
- Bagnato, A., Sadovykh, A., Dahab, S., Maag, S., Cavalli, A. R., Stefanescu, A., . . . & Mallouli, W. (2017). Modeling OMG SMM metrics using the Modelio modeling tool in the MEASURE project. *Génie logiciel*, (120), 46–52.
- Dahab, S. A., Porras, J. J. H., & Maag, S. (2017). A software measurement plan management guided by an automated metrics suggestion framework. In *2017 european conference on electrical engineering and computer science (eecs)* (Pages 9–16).
- Dahab, S., Porras, J. J. H., & Maag, S. (2018). A novel formal approach to automatically suggest metrics in software measurement plans. In *Evaluation of novel approaches to software engineering (enase), 2018 13th international conference on*. IEEE.
- Dahab, S. A., Silva, E., Maag, S., Cavalli, A. R., & Mallouli, W. (2018). 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, july 26-28, 2018*. (Pages 154–165).
- Dahab, S. A., Maag, S., Mallouli, W., & Cavalli, A. R. (2018). Smart measurements and analysis for software quality enhancement. In *ICSOFT (selected papers)* (Volume 1077, Pages 194–219). Communications in Computer and Information Science. Springer.
- Dahab, S. A. & Maag, S. (2019). Suggesting software measurement plans with unsupervised learning data analysis. In *Proceedings of the 14th international conference*

on evaluation of novel approaches to software engineering, ENASE 2019, heraklion, crete, greece, may 4-5, 2019. (Pages 189–197).

Remerciements

Tout d’abord, je tiens à remercier chaleureusement mon directeur de thèse, Stéphane Maag, qui, par ses nombreux conseils, a été une école de formation tout au long de ce travail de recherche. Sans oublier sa relecture finale méticuleuse de chacun des chapitres, me permettant sans aucun doute de préciser mes propos. Je lui dois reconnaissance et lui témoigne toute ma gratitude.

J’adresse mes remerciements également à tous les jurés, pour l’honneur qu’ils me font en acceptant de juger ce travail.

Je remercie évidemment Alessandra Bagnato pour son soutien, sa relecture enrichissante et ses suggestions toujours avisées.

J’exprime toute ma gratitude aux Professeurs Abdelhamid Mellouk et Sebastien Salva d’avoir accepté d’être les rapporteurs de cette thèse.

Toute ma reconnaissance va également à mon père Ahmad Kane-Mbaye ainsi qu’à son maître, qui de loin m’ont inspiré et motivé dans les moments les plus épineux. Indubitablement, cette étude n’aurait pu exister sans leur concours. J’exprime donc les remerciements les plus sincères à leur égard.

Merci à mes collègues Jose Alfredo Alvarez-Aldana et Jorge Lopez pour leur humanisme et les nombreux échanges, ainsi qu’à Juan Jose Hernandez-Porras pour son excellent travail effectué durant son stage et sa disponibilité une fois celui-ci terminé.

Je remercie, bien sûr, les membres du projet avec qui j’ai pu collaborer qui m’ont été d’un grand soutien et d’une grande aide, Wissam Mallouli, Antonin Abherve, Sana Mallouli, Jérôme Rôcheteau et Marcos Almeida.

Sans oublier le professeur Xiaoping Che, pour son accueil au sein de l’université Beijing Xiaotong, et cette enrichissante collaboration qui nous lie.

Un grand merci aux professeurs Alain Finkel et Philippe Hoppenot dont les sagesses venues à point m’ont ressourcées.

Pour finir, je remercie ma famille et mes proches, mon mari dont le soutien a contrebalancé l’intense stress ; Sana Boussetat, qui a su me motiver et m’encourager ; et mes parents pour m’avoir épaulé durant cette odyssée.

Contents

Résumé	iii
Publications	vii
Remerciements	ix
1 Introduction	1
1.1 Software Measurement	2
1.2 Machine learning	4
1.3 ITEA3 MEASURE PROJECT	6
1.4 Our contributions	7
2 State of the art	9
2.1 Software measurement	10
2.2 Software measurement and machine learning	15
2.3 Emotional software metrics	18
3 Formal software measurement	19
3.1 Formal measurement	20
3.2 Standards for software measurement	23
3.3 Experiments	36
3.4 Conclusion & discussion	45
4 Automated software measurement analysis and suggestion	47
4.1 Manual analysis model	49
4.2 Data analysis from measurements	51
4.3 The Suggestion	61
4.4 Application	63
5 Automated initialization phase	75
5.1 The knowledge basis	77
5.2 Automated initialization phase	81

5.3	Experiments	87
6	Emotional metrics as indicator of measurement refinement	91
6.1	Emotional Software metrics	93
6.2	Emotional metrics formal specification	99
6.3	Emotional metrics as measurement plans refinement indicators	103
6.4	Conclusion & discussion	106
7	Conclusion and Future Work	107
7.1	Conclusion	107
7.2	Future works	109

1

Introduction

This thesis entitled "An approach to measuring software by using combined complex metrics test" has as study object the software measurement as an integral part of software engineering systems.

Nowadays, the software systems are more and more complex as well as their issues. Indeed, software systems are integral parts of our lives. From particular usage to professional one, the systems are more efficient and the expectations are more exacting. This is explained by the fact that they are used for many kinds of tasks, from cleaning task to agents assistant as metro driver or car driver, etc. And whatever the task, the system must meet the needs with accuracy and security, which implies the needs to design high standard software systems.

Likewise, the software engineering process becomes more complex due to the complex systems to design. Indeed, in order to design quality software the engineering process must be adapted to meet this need. In order to ensure a quality software engineering process as well as a quality product, it is necessary to have an efficient supervision process. This latter gives crucial information for the management.

Today, the software engineering process is well defined and based on strong background (Wohlin et al., 2012). Each step of the development process is defined and well-known. Usually this process follows a development cycles composed of six phases: the definition of the requirements, their analysis, the design, the implementation, the testing phase and the maintenance one. Each of these phases should be correctly done to ensure a quality product.

However, 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. Considering the quantity of aspects to be measured raising the relevant information

to be analyzed and reported become difficult (as concerned by Microsoft Power BI¹). 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.

1.1 Software Measurement

The measurement is the "process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules" (Fenton & Bieman, 2014). In other words, measuring allows to gather information about an object. This information is theoretical, it is based on a referential which gives meaning to the information. For instance, the number assigned to a distance in meter is based on the meter theory, is a theoretical value and this number has a meaning by the related theory.

Thus, in the measurement context, an object is a set of factors respecting a set of indicators. the factors are the properties which characterize the object. The indicators are the criteria that characterize a property. As example, the pollution of the air is a factor of the air, and there are numerous indicators to express the pollution as the number of specific particles in the air, the humidity etc.

Measuring is a necessary action every day life and for many aspects of our daily: to sell, to purchase, to wear clothes, shoes, to cook etc., and more generally many sciences is based on measurement, economics, physics, biology, computer science etc. Besides, the physicist Lord Kelvin (1824-1904) said: "When you can measure what you are speaking about, and express it into numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science." (T., 1891). Wisdom that has been understanding by the engineering sciences which integrate it as an integral part of their engineering process.

The aim interest of measurement is to make a finding on a real facts in order to interact passively or actively with our world either to understand, to detect, to predict, to improve etc., in short, to supervise (Fenton & Bieman, 2014). In fact, by gathering information about our environment we can optimize and manage the interaction with it.

In our context, the software engineering, the measurement applied to our object, the software systems, makes sense. Indeed, in order to ensure a development quality and quality products, it is important to consider the software measurement aspect. By gathering information on software systems during the engineering process, this latter can be adapted, improved or simply supervised.

Moreover, supervising during a construction process allows to ensure that the requirements are respected throughout the process, and thus avoid the risk to return back to fix the issues.

¹<https://powerbi.microsoft.com/>

As example, in masonry, the angle of the wall is checked at each step of the construction to ensure a straight wall.

Likewise, in software engineering context, supervising the developed software systems during all the engineering process, from the design phase to the production one, allows to ensure a quality of development and products.

In this context, and as explained at the beginning of the section, a software systems or project is a set of properties that meet criteria. The values of these criteria are expressed by metrics. A metric is the computation or the process that assigns a theoretical value to the evaluated criterion. Thus, measuring a software is getting information on a set of properties by evaluating its corresponding criteria through metrics (ISO, 2019).

The software properties, criteria and metrics are well defined in the literature and described in detail in the Chapter 3. The ISO/IEC 25010 (ISO/IEC, 2010a), defines eight software properties, thirty one criteria and more than two hundred metrics.

To sum up, the software measurement is the mean to supervise a software systems through the evaluation of the properties of the concerned software systems. For that, we need to determine the set of metrics necessary to evaluates the properties. This set of metrics is called measurement plan and the evaluation process, from the design to the gather of measurements, is called the measurement process.

Measuring has become a fundamental aspect of software engineering. Accurate measurement is proving to be highly efficient in various domains. For example, measurement is vital in the construction of high quality prediction systems, in the context of the improvement of software development and maintenance projects. Further, and more important, accurate measurement is required for the evaluation to guarantee a system's quality, by highlighting problematic areas, and in the determination of better work practices with the end of assisting practitioners and researchers in their work.

Software measurers are, moreover, important tools that assist in the evaluation and institutionalization of Software Process Improvement in the organizations that develop them. Software Measurement is, in fact, a key element in initiatives such as SW-CMM (Capability Maturity Model for Software) (Paulk, Weber, & Chrissis, 2005), ISO/IEC 15504 (SPICE, Software Process Improvement and Capability determination) and CMMI (Capability Maturity Model Integration). The ISO/IEC 90003:2004 standard also highlights the importance of measurement in managing and guaranteeing quality. Or, the ISO/IEC 25000 (ISO, 2005), described in details in the Chapter 3, which defined a complete guide to evaluate the software quality.

Furthermore, software measurement is an empirical science based on experiences. Thus, the complexity of the systems leads to manipulate a large amount of data for a relevant evaluation while the study (Hentschel, Schmietendorf, & Dumke, 2016) highlights that "the consideration of Big Data techniques in software measurement is currently very low", in particular the learning techniques. Thus, machine learning is an interesting stakeholder because of the usefulness that can result from it given the current constraints with which the

software measurement process has to deal.

1.2 Machine learning

“Machine learning is programming computers to optimize a performance criterion using example data or past experience.” (Alpaydin, 2009)

Machine learning is a sub field of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and used by people².

Although, it is a field within computer science, it differs from traditional computational approaches. In traditional computing, algorithms are sets of explicitly programmed instructions used by computers to calculate or problem solve. A learning problem can be defined as the problem of optimizing a performance criterion using past experiences. Using algorithms that learn from data, the learner is able to optimize the performance criterion. The problem can be represented as a model defined up to some parameters and the learning process consist of executing a computer program to optimize the parameters of the model. The model can be predictive when the model is used to make predictions or the model can be descriptive when the model is used to gain knowledge from the data (Jordan & Mitchell, 2015).

This is a method used to devise complex models and algorithms that lend themselves to prediction; in commercial use, this is known as predictive analytics. These analytic models allow researchers, engineers, and analysts to "produce reliable, repeatable decisions and results" and uncover "hidden insights" through learning from historical relationships and trends in the data.

Driven by the growing amount of raw data produced every day and the need for an efficient and automatic way to extract meaning from such dataset, machine learning is a growing field of research. Machine learning methods are data intensive and this is because of the recent explosion in the ability to gather data at a low cost in an online environment. The capability of the algorithms of finding useful information in the large amounts of data have led to the introduction of machine learning to other disciplines including health care, education and manufacturing. The amount of data produced every day and the low cost of storing it has given a huge raise in the application of the available algorithms to solve this problem, now the problem is not to gather data but to find the way to extract information from it. Likewise, for software measurement which aims to manage data to get relevant information to supervise software projects.

Besides, software measurements is currently gaining in popularity and effectiveness due to several breakthrough advancements in data collecting techniques, data analysis, artificial intelligence, human factors, etc. Measurements are used in many research and industrial areas to monitor the involved processes in order to improve the production(s), reduce the costs (time

²<https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html>

and resources), manage the human efforts, etc. (Dumke & Abran, 2013). To tackle these purposes, diverse techniques have been developed and integrated in most of the industrial processes. From measurement plans defined by experts (often the project, software or system managers advised by engineers), these approaches collect an important amount of data that are semi-automatically analyzed. This analysis allows to raise issues, alarms to fix or improve the measured elements.

However, with the rise of the software systems and their complexity distributed through diverse development phases, the software measurement process has to deal with more management and performance constraints. In fact, there is a real need of adapted measurement process. Indeed, the current one suffers of different issues as the lack of formal structure, protocols and dynamism.

Lack of formal structure The software metrics are well defined, as in the ISO/IEC 25000 (ISO/IEC, 2010b) more than two hundred metrics are defined, but there are not formally specified as well as their implementation is dependent on the developer and the entire measurement process. That implies difficulties to manage, exchange and maintain data of the measurement process and thus, reduce its scalability, maintainability and the interoperability. It becomes expert-dependent and thus more costly.

Lack of protocols In the current measurement process there is no measurement strategy, all the metrics are executed during all the measurement process without protocols that prioritize the measurement plans according the real needs. In fact, the expert measures all what he can and not necessarily what he needs. Then a huge amount of data are unnecessarily collected and analyzed. With the amount of data to evaluate due to the complexity of the systems, the cost of the analysis is dramatically increased and the measurement process becomes heavy.

Unlike other engineering systems, more older as medicine or mechanics, etc., their measurement process follows strict protocols. There are initial and general measurement plans, then according to the results, the evaluation is oriented or refine towards others measurements plans more specific. That allows to focus the measurement process to the real needs and to reduce the cost and heaviness of this latter. Likewise, it is unthinkable for us to do all the existing medical exams each time a weird symptom appears.

Lack of Dynamism Consequently, the current software measurement process is fixed and manually planned at the beginning of the project by the project manager. In addition with a serial execution and analysis. Thus, the measurement process is static and dependent to the expert.

Moreover, this work (Shin, Meneely, Williams, & Osborne, 2011b) shows through the use of learning techniques, all metrics are not necessary to predict a behavior and allow to prioritize some metrics.

Thereby, this thesis aims to tackle these issues by considering both relevant research results (Hentschel et al., 2016; Shin et al., 2011b). Thus, our objectives are: first, to improve the software measurement design phase. The purpose is to use a standard language to formally specify our software metrics and generate from this standard specification a standard architecture of the implementation. Secondly, to automate the analysis and the manual measurement management by using machine learning techniques. Thirdly, to add dynamism and flexibility to the measurement process by proposing an approach to suggest measurement plans at runtime according to the results of the analysis. That will allow to orient the measurement process according to the needs, and to get closer to the more experienced engineering systems.

These works are carried out within the context of the European ITEA3 MEASURE project.

1.3 ITEA3 MEASURE PROJECT

The main goal of the ITEA3 Measuring Software Engineering (MEASURE) Project³ is to measure the quality and efficiency of software, as well as to reduce the costs and time-to-market of software engineering in Europe, by implementing a comprehensive set of tools for automated and continuous measurement. Therefore, this project aims at providing a toolset for future projects to properly measure their software quality and efficiency. More importantly, it opens a new field for innovation. The second innovation will be in the advanced analytics of the measurement data enabled by the project.

The MEASURE platform is the research result of the MEASURE project, it offers a comprehensive set of tools for automating continuous measurement over all stages of the software development life cycle (specification, design, development, implementation, testing, and production). Thus, the MEASURE project can develop a body of knowledge that shows software engineers why, how and when to measure quality of process, products and projects.

1.3.1 MEASURE platform

The MEASURE platform⁴ is a tool dedicated to the measurement, analyze, and visualization of the metrics, in order to extract and show information of the software engineering processes. This platform aims to centralize all the services needed for a complete software measurement process, as illustrated in the Figure 1.1. It offers:

- Tools for automatically measure software engineering processes during the whole software life cycle by executing measures defined in SMM standard and extracted from a catalogue of formal and platform-independent measurements.

³<http://measure.softteam-rd.eu/>

⁴www.measure-platform.org

- Methodologies and tools which allow measure tools provider to develop a catalogue of formal and platform-independent measure.
- A Storage solution dedicated to measurements resulting of measure execution in big data context.
- Visualization tools to expose the extracted results in an easy-readable fashion, so allowing a quick understanding of the situation and the possible actions that can be taken to improve the diverse stages of the software life cycle.
- An extension mechanism dedicated to the integration of external analysis tools will provide long terms analysis and predictive evaluations on collected measures.
- An extended API allowing to facilitate the integration on Measure Platform with external tools and services.

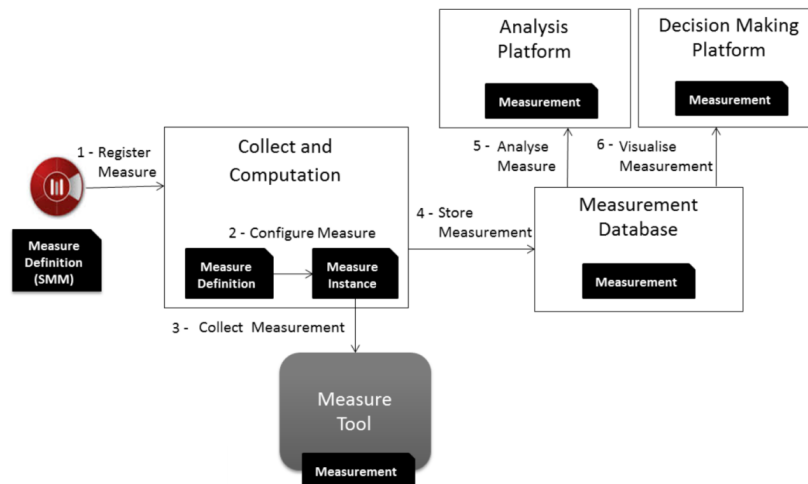


Figure 1.1 – Main process of the MEASURE platform.

1.4 Our contributions

My thesis work aims to contribute on different aspects of the software measurement process:

- the design phase by proposing formal specifications of software metrics through standard language;
- the measurement phase by proposing implementations of metrics according to an architecture based on the chosen standard;
- the analysis phase by proposing an analysis approach, the Metrics Suggester, through the use of learning techniques.

For the first contribution, we formally define and implement three types of metrics:

- "Green" metrics which are usually low level software metrics and used to evaluate all that related to the consumption of energy by the software system;
- Architectural metrics that focus on code aspect of a software project;
- Emotional metrics which evaluate the emotion of users of software application. They are usually used to evaluate the interaction of human-system in order to assess the service quality of the system.

For the second contribution, we propose an analysis approach which aims to improve the current software measurement process by reducing the costs of management and analysis. On one hand, we propose to reduce the expertise charge by automating the analysis through the use of machine learning techniques. On the other hand, we propose to optimize the measurement process performance by reducing its processing load.

In order to reduce the processing load, we built a suggestion algorithm, which generates according to the analysis result a suggestion of a new measurement plans. This allows to reduce the processing load by prioritizing the metrics allowing the execution at each time of the metrics of interest according to the software needs instead of executing all the metrics each time.

This thesis is organized around these contributions as follows: first of all, we introduce in the Chapter 2, the state of the art regarding the different standards for defining and designing the software measurement, the use of machine learning in the analysis of software measurements and the works over emotional metrics in the context of software evaluation. The Chapter 3, introduces the standards used to formally specify and implement our metrics. It presents our contributions in this work. Then, in the Chapter 4, we present our analysis and suggestion approach as contribution to the analysis part of the MEASURE project. This approach aims to automate the analysis and to adapt the measurement process to the needs at runtime. For that we base on manual configuration defined by the experts and the use of supervised learning technique. Finally the Chapters 5 and 6 are improvement works of this approach. The first one, focuses on the configuration phase and aims to automate it by using unsupervised learning technique, while the second one focuses on the measurement cycles during the entire process. It aims to add flexibility to this latter by using emotional metrics as innovative way and to formally specify them.

2

State of the art

Contents

2.1	Software measurement	10
2.1.1	Standard software measurement specification language	11
2.2	Software measurement and machine learning	15
2.2.1	Machine learning	15
2.2.2	Use of machine learning in software measurement context	16
2.3	Emotional software metrics	18

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 & Bieman, 2014). Thanks to that, standardization institutes worked in that way to propose well-known norms as ISO/IEC25010 (ISO/IEC, 2010b) to guide the software measurement process and OMG SMM (Group, 2018) to guide to the measurement plan specification. These standards have been reviewed by the research community and industrials, and are adapted and applied in many domains.

Moreover the literature offers many studies using machine learning for software measurement. Most of them focus on fault prediction for a particular software property. These works are very interesting basis to handle our intention to automate the analysis.

Finally, regarding the emotional metrics for measuring software systems, many studies have been done to evaluate the quality of service of a system by using the user experiences as the well-known (Hassenzahl, 2003). More recently some works that focused on the evaluation of the software systems through the user experiences constitute a relevant basis for our objectives.

In this chapter, we will depict the state of the art regarding these encouraging works according to the following organization: the first section presents the different standards that propose methodologies to define software measurement and languages to specify this latter. From this section, we explain our choice for our work. The both other sections present different works, respectively, that use learning technique in the software measurement context and evaluate software systems via emotional metrics.

2.1 Software measurement

There exist various methods and standards with how to carry out measurements in a precise and systematic manner, of which the most representative are:

- **Goal Question Metric (GQM):** The basic principle of GQM is that the carrying out of the measurement must always be oriented towards an objective. GQM defines an objective, refines that objective into questions and defines measures which attempt to answer those questions.
- **Practical Software Measurement (PSM):** The PSM methodology is based upon the experience obtained from organizations through which the best manner in which to implement a software measurement program with guarantees of success is discovered.
- **IEEE 1992 (Methodology for Software Quality Measures):** according to the IEEE 1992 standard, software quality can be considered as the extent to which the software possesses a clearly defined and desirable combination of quality attributes. This standard

deals with the definition of software quality for a system by using a list of software quality attributes which are required by the system itself.

- ISO/IEC 15939: this international standard identifies the activities and tasks which are necessary to successfully identify, define, select, apply and improve software measurement within a general project or within a business's measurement structure.
- ISO/IEC 25000: provides a guide to the measurement of software quality. It defines a software quality model and a software quality evaluation method.

From these standards, we chose the ISO/IEC 25000 which provides the needed information for our works such as the quality software model and the corresponding sets of metrics. This standard is described in detail in the Chapter 3.

2.1.1 Standard software measurement specification language

The availability of a description language that allows to represent the elements which must be taken into account in the measurement processes might, therefore, be important in decision making and in process improvement. Among these languages, we present:

- GMSL: Goanna Metric Specification Language
- Slammer: Specification LAnguage for Modelling MEasurements and Redesigns
- SMML: Software Measurement Modeling Language
- SMM: Structured Metric Meta-model

GMSL

Goanna (Vogelsang, Fehnker, Huuck, & Reif, 2010) is a tool that performs deep analysis of C/C++ source code using a model checking technique. It verifies the presence or absence of bugs, memory leaks and security vulnerabilities. The tool is fully path-sensitive and inter-procedural; it utilizes additional static analysis techniques such as abstract interpretation. It also provides two specification languages for defining source code verification.

The first language is a tree-query language based on XPath for finding constructs and patterns of interest in the abstract syntax tree (AST), correspondingly, it is called Goanna XPath Specification Language (GXSL).

The second language is based on temporal logic expression over paths in the CFG, it is called Goanna Property Specification Language (GPSL). GPSL allows the embedding of GXSL expression.

The Figure 2.1 shows how these languages fit into the static analysis.

The Goanna Metric Specification Language (GMSL) provides a way to define metrics on an abstract level. A prerequisite for the use of GMSL is a query engine that returns sets of nodes of the AST for which certain syntactic properties hold.

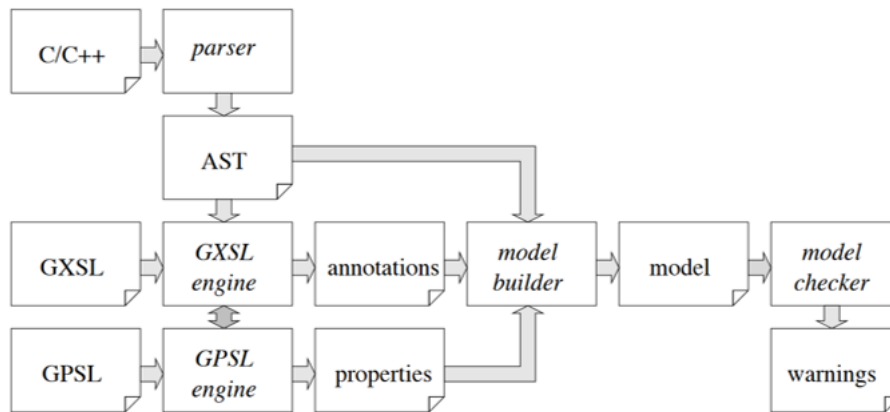


Figure 2.1 – Goanna’s model checking approach for static analysis of C/C++ code

Goanna provides a language to define functions that select certain nodes of the AST of a program. The queries are always evaluated on the entire AST but it is possible to pass parameters to the queries when referring to a particular node (or a sub-tree) in the AST. The result of a GXSL query is a set of AST nodes.

The Goanna GMSL interpreter is an extension to the existing Goanna analyzer. An overview of the extended architecture can be found in Figure 2.2. The metrics interpreter is situated on top of the existing GXSL query engine, i.e., it mostly uses existing library functions for pattern matching constructs of interest, and interprets the metrics’ specification written in GMSL.

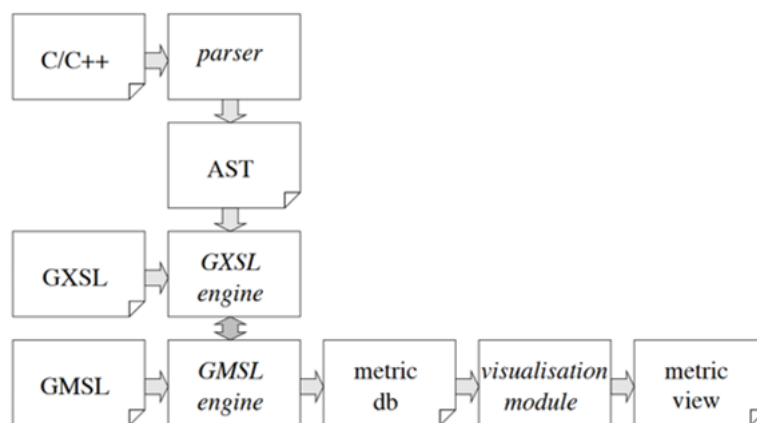


Figure 2.2 – Goanna for computing metrics

The Goanna Metric Specification Language (GMSL) is very interesting for C/C++ code analysis. Nevertheless, in the context of MEASURE, we would like to target other languages (like Java) and also take into account metrics related to different phases of the software life cycle including design, implementation test and operation as well as metrics related to engineering processes. For these reasons, this language is not selected.

SLAMMER

SLAMMER (Guerra, de Lara, & Díaz, 2008) allows the customization of generic measurements and redesigns for a given DSVL (Domain Specific Visual Language). It has been defined through a meta-model that contains generalizations of some of the main types of measurements. These include measurements for global model properties (such as number of cycles and size), single element features (e.g. methods of a class in object oriented languages), features of groups of elements (e.g. their similarity or coupling) and paths (e.g. hierarchies in object oriented languages, navigation paths in web design languages, etc.). SLAMMER allows customizing these generic measurements by using visual patterns, creating new ones (procedurally), and composing them in order to build more complex measurements.

It is also possible to specify threshold values for the measurements that may have an associated action described either procedurally, by customizing a generic action template or by using a graph transformation system. This can be useful if the action performs a redesign that improves the quality of the model or modifies it with respect to the known design patterns.

In order to define this language, related works on the definition of ontology concerning software measurement (Mora, Piattini, Ruiz, & Garcia, 2008; de los Angeles Martin & Olsina, 2003) have been taken into account, as well as the international standard for software quality ISO 15939 (“ISO/IEC/IEEE International Standard - Systems and software engineering—Measurement process,” 2017).

These approaches have been implemented in the meta-modelling tool AToM (Vangheluwe, DE LARA, & MOSTERMAN, 2002): A Tool for Multi-Formalism Modelling and Meta-Modelling. In this way, the DSVL designer can enrich the DSVL specification with a SLAMMER model specifying a number of measurements and redesigns for it. Starting from this definition, a modelling environment is generated for the language; this environment integrates the measurements and the previously defined actions.

The SLAMMER language looks promising given the MEASURE project requirements. Nevertheless, it provides only support for direct metrics (i.e., atomic metrics). In the context of MEASURE, more research work is planned to target complex and dynamic metrics. For example, metrics defined as the correlation of other metrics and where their definition can change over time (based on a machine learning approach, for instance). For this reason, another language has been chosen for specifying software quality metrics.

SMML

SMML (Mora et al., 2008) is a language which permits software measurement models to be built in a simple and intuitive manner. This language has been produced by using the Software Measurement Meta-model (SMM) as the Domain Definition Meta-model that is also an OMG standard specification (Group, 2018). However, SMML is not a model, it is a language and as any language it has an abstract syntax, a concrete syntax and a semantics.

The SMML syntax is obtained from the Software Measurement Ontology (SMO). This

ontology contains all the elements (concepts and relationships) of the software measurement domain. The Software Measurement Meta-model includes the packages which are alignments with the sub-ontologies of SMO (Basic, Characterization and Objectives, Measures Software, Measurement Approaches and measurement Action).

However, for the development of the Language, all the packages are of interest, in exception of Measurement Action. This package has been excluded as it contains the elements which are related to measurement but not to the problem domain. The following Figures 2.1 and 2.2 show the structure of the packages upon which the SMML language is based.

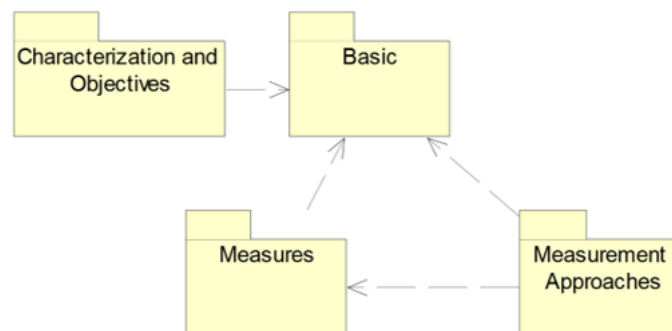


Figure 2.3 – Structure of the package of SMML

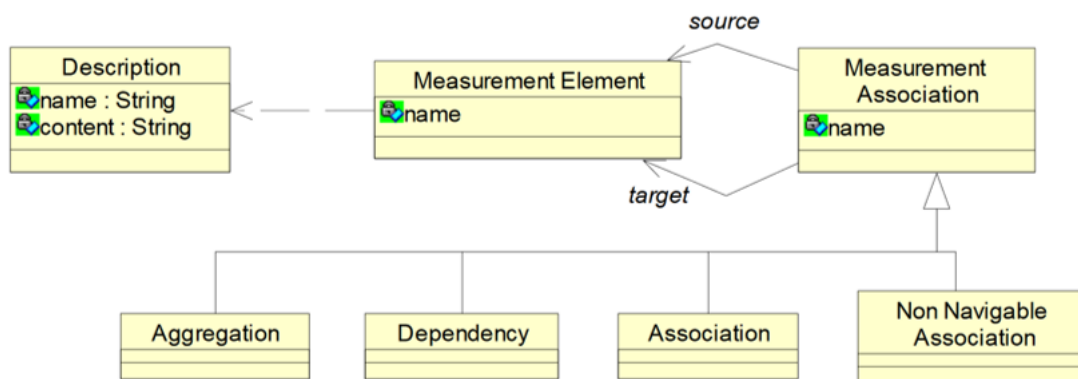


Figure 2.4 – Basic Package of SMML

The SMML language described in this section allowed us to better understand the semantics of the SMM meta-model. Unfortunately, this language, which is the result of research work, there does not exist any maintained tool that support it. For this reason, we decided to adopt SMM natively on one of the consortium tools. This latter is described in details in the Chapter 3.

Conclusion

In conclusion, we have covered some of the most common languages and formats to describe measurements of software and metrics. A summary of their features is provided in the following table 2.5. This analysis lets us decide which language can be used along the

execution of the MEASURE project. The basic idea is to rely on SMM, which is an OMG standard. The language allows targeting direct and complex metrics and can be supported by the MEASURE framework solution.

	Software lifecycle coverage	Metric complexity	Tool	Based on standard
GMSL	(-) Only C/C++ is supported	(-) Only direct metrics	(+) Available	(-) No
SLAMMER	(+) Can covers all SLC	(-) Only direct metrics	(+) Available	(-) No
SMM	(+) Can covers all SLC	(+) Direct and complex metrics	(-) Not maintained	(+) Based on SMM
MEASURE APPROACH	(+) Can covers all SLC	(+) Direct and complex metrics	(+) Based on Modelio	(+) Based on SMM

Figure 2.5 – Comparison of software metric specification languages

2.2 Software measurement and machine learning

2.2.1 Machine learning

Machine learning is a subset of artificial intelligence focused on giving computers the ability to learn through experience. It involves the use of algorithms that allow computers to learn. Over the last few years the use of machine learning has become prominent and useful in daily activities, examples can be found on spam filtering (Joachims, 1998), cyber security (Buczak & Guven, 2015), music recommendations, image processing (Draper, 2000), marketing.

There are different techniques (Toch, Lerner, Ben-Zion, & Ben-Gal, 2019) in machine learning that have been proposed based on the available data of the problem. One approach is supervised learning, the most widely used machine learning technique, which consist of using labeled data to construct a model to be able to do predictions, some examples where supervised learning is used for training are classification and regression problems.

Since labeled data is scarce and difficult to obtain, another approach is unsupervised learning where the data has no labels associated to them and the goal is to organize the data in some way, this means to group data into clusters or give some kind of order to the data.

A third approach is reinforce learning, this approach does not use the training data to determine and output based on an input. For this approach the training data is used to provide an indication whether an action is correct or not. But a state of the art of learning techniques

is not the scope of this chapter but rather the use of learning techniques in the context of software measurement.

For our works, we choose a supervised learning technique, SVM, for automating the analysis and an unsupervised learning technique, X-MEANS, for automating the initialization phase. The reasons of these choices are: first we need to base the training on the experts evaluation in order to have an automated analysis which reproduces the ones of the experts with the minimum of expert interventions; Then, we aim at avoiding the "expert-based" by automating this training.

The descriptions of these algorithms and the choices of the techniques is explained in details in the concerned Chapters 4 and 5.

2.2.2 Use of machine learning in software measurement context

As explained in the introduction, the software measurement is an empirical science, it is based on the experiments on the studied object. Moreover, as shown by the software measurement standards, the evaluation of a software needs to handle a lot of data. That is why, using machine learning techniques is not a nonsense. Besides, this study (Hentschel et al., 2016) shows that software measurements is concerned by the challenges of bigdata and highlights that "the consideration of Big Data techniques in software measurement is currently very low", in particular the learning techniques.

The literature offers relevant works on software measurement using learning techniques as (Shin et al., 2011b) which shows that with learning techniques, all metrics are not necessary to predict a behavior and allow to prioritize some metrics. This allow to determine at runtime the relevant metrics as indicator and to reduce the quantity of metrics necessary to evaluate an aspect of the system.

In the research literature, several works on software metrics selection for software quality have been provided (Gao, Khoshgoftaar, Wang, & Seliya, 2011). Recent techniques based on learning approaches have been proposed. Most of them are dedicated to software defect prediction (Shepperd, Bowes, & Hall, 2014a), (MacDonald, 2018), (Laradji, Alshayeb, & Ghouti, 2015a), metrics selection (Bardsiri & Hashemi, 2017) or even Software testing (Kim, Ryu, Shin, & Song, 2017). However, even if these techniques have introduced considerable progress to improve the software quality, they have still some limitations. The measurement plan is still manually fixed by the project manager or the experts in charge of its definition. Furthermore, the implementation of the measures is dependent on the developer and reduce the scalability, maintainability and the interoperability of the measurement process.

The selection of the metrics in a measurement plan is a crucial process and very few works have been provided. In (Gao 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. We can also cite (Wang, Khoshgoftaar, & Napolitano, 2011) in which the authors study the impact of the metrics on the performance and effectiveness of the software quality

model.

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, Alshayeb, & Ghouti, 2015b; Shepperd, Bowes, & Hall, 2014b; Prasad, Florence, & Arya, 2015) raise that issue by proposing diverse kinds of machine learning approaches for SW defect prediction through SW metrics.

In our work, we do not aim at predicting defects or faults, instead, our purpose is to guide the experts by suggesting, at runtime, specific metrics that could be applied in a measurement plan at a time t . Very few works targeting our objective have been proposed in the literature.

The following papers tackle that idea by presenting interesting ways of metrics selections using learning techniques. In (Kumar, 2012), the author defines Support Vector Machine (SVM) algorithms to identify and classify the reusability of software components. The focus is made on metrics for the structural analysis of different procedures in procedure oriented software systems. Relevant experimental results have been provided and discussed. In (Jin & Liu, 2010), object-oriented metrics have been used to predict the software maintainability and specifically its cost. The authors designed their framework by using SVM and clustering techniques that have been applied on a simple developed software. More recently, (Alves, Fonseca, & Antunes, 2016) surveyed several learning techniques based on software metrics to predict vulnerabilities. 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. In our approach, we intend to classify the measurements through broad domains to select and then suggest metrics to improve the software measurement plan.

In addition, there are some works which use supervised learning algorithms, especially for software defect prediction (Laradji et al., 2015a; Shepperd et al., 2014a) or for prioritize software metrics (Shin, Meneely, Williams, & Osborne, 2011a). Indeed, there are a lot of software metrics, and currently the measurement processes execute all the metrics continuously. This latter shows that we can prioritize the metrics and thus reduce the number of metrics to be executed.

There are also works which propose to use unsupervised learning technique to estimate the quality of software (Zhong, Khoshgoftaar, & Seliya, 2004b) as "expert-based". They also propose to base on clustering techniques to analyze software quality (Zhong, Khoshgoftaar, & Seliya, 2004a). Other works propose to combine supervised and unsupervised learning techniques to predict the maintainability of an Oriented Object software (Jin & Liu, 2010). But all of these works focus on the analysis or prediction of one software property. The aim of our approach is to allow the less of expert dependency to evaluate all the software engineering process, and to suggest flexible measurement plans continuously according to the software need.

The most works on software measurement using learning techniques focus on fault prediction while our objectives by using learning techniques, is at first to predicts behaviors to

automate the analysis and in a second time to determine data models which allow to define protocols of prioritization of measurements according to the situation.

2.3 Emotional software metrics

Regarding the emotional measurement in software engineering context, many research works have been done, more precisely, the literature provides many works on the definition of emotional metrics and also on the use of emotional metrics to measure the usability of software, video games, HMI, etc. Indeed, (Agarwal & Meyer, 2009; Hashemi & Herbert, 2015) studies the use of user emotion to evaluate the user experience. The first one focused on verbal and face emotional expressions while the second one adds to this latter physiological and behavioral emotions.

Moreover many works study use the UX through user's body manifestation to evaluate specific systems as (Rodden, Hutchinson, & Fu, 2010), which use the heartbeat metric to assess the UX in order to evaluate web applications. They are also used to evaluate video games, especially, in (Nacke, Grimshaw, & Lindley, 2010), they study the impact of the sonic stimuli basing on the UX of shooter games users. And this one (Christou, 2013) compares the UX between experienced and inexperienced video gamers. But this latter, do not use the UX to improve the video game, they just analyze the user's body responses in front of user-system interactions.

Finally, (Wang & Che, 2016) use UX through emotional metrics to evaluate mobile games in order to "determine whether the implementation satisfies the requirements", thus the approach is from system point of view and shows that they can exist correlations between software systems and user experience. That is very interesting for our work, but it focused on mobile games, while our study context is software systems.

3

Formal software measurement

Contents

3.1	Formal measurement	20
3.1.1	Measurement Fundamentals	21
3.1.2	Software Measurement terminologies	22
3.2	Standards for software measurement	23
3.2.1	Software quality measurement : ISO 25000	23
3.2.2	Software measurement definition language	26
3.3	Experiments	36
3.3.1	MEASURE Platform	36
3.3.2	Formal software metrics development	39
3.4	Conclusion & discussion	45

The main motivation of this thesis work is to contribute to the improvement of the measurement process in software engineering. This process is well-defined by many works in the area. Likewise the fundamental basis are well specified and defined. Besides, there are some standards in the field as the ISO/IEC 25000 (ISO, 2005) or the OMG standards SMM (Group, 2018). The first one is a guide for developing a quality software product and to design and build a quality measurement process to evaluate the quality software (i.e., Section 3.2.1) and the second one provides a specification format for software metrics (i.e., Section 3.2.2).

Thereby, the literature offers a strong background and well means to design measurement plans but in fact, there are lacks of structure due to the use of informal metrics in the industry. These defects imply difficulties to manage, exchange information on software and to design measurement plans. Indeed, a lot of metrics are defined, as in the ISO/IEC 25000 standard which lists more than 200 metrics or others are implemented but there are not formally defined, and their implementations are just as dependent to the developer, there is not any formal definition. The lack of formal documentations reduces the interoperability, scalability and maintainability of the measurement process.

The first objective of this work is thus to propose formal definitions and implementations of metrics based on the SMM standard (Group, 2018). The purpose is to use the code generation feature to facilitate a more generic implementation of software measurements. We propose a software measurement UML model from the formal measurement specification SMM and thus, to generate from metrology a "formal" measurement code design. Using a graphic format allows to have documentation and thus increase the interoperability of the metrics and the ease to manage it. Moreover, with the code generation from a model feature, we have a safe development and more maintainable since by the model we have a documentation on the code architecture and the requirements (the measurement context).

Finally, this chapter is organized as it follows: first we introduce a formal definition of a measurement concept with the Section 3.1.1. Then, we introduce the software measurement standards, which are the basis of our work, in the Section 3.2 to finish with our contributions of measurement specification and implementation formalization in the Section 3.2.2.

3.1 Formal measurement

The measurement theory defines the fundamental basis of the measurement concept and the terminologies. This is the fundament of the software measurement theory and applications.

Measuring is the action to get information on something. For that, we should know what information is wanted and then how to get it. Thus, before any measurement process it is necessary to define the context: what reality we want to observe? What are the elements of this reality that will give us the information? How to gather this information? The reference theory? And when the information should be gathered? For example, we want to observe the body of a man, more specifically we want information about its height. For that, we need to

observe his height (what), from the top of his head to the bottom of his feet (how), in meter (theory), in standing up position (when).

The theory is necessary as interpretation language of a reality into a theoretical value and to understand the meaning of the value. If: we take my previous example, the meter is the theory used to interpret the height (the reality) into value (175). If we do not define the used theory we can not understand the value 175. And if we do not use the good theory to interpret this value, as the foot unit, we will have a wrong meaning of the information. Indeed, 175 meter does not have the same meaning as 175 foot. Hence the importance of defining the reference theory.

To summarize, to define a metric, it shall define measurement context. The next section, introduces the formal definition of these concepts and their terminologies.

3.1.1 Measurement Fundamentals

The fundamental mathematics define a measurement context as a measure space. This measure space defines the measurable space which defines the observed reality and the elements that provide the desired information about the reality and the function which defines how to obtain the information and the theoretical basis of reference. All of these concepts are so defined as following:

Definition 3.1. A measurable space M is a pair composed of the measured object X and the set of measurable properties A of X such as:

$$M = (X, A) | A \in X \quad (3.1)$$

Definition 3.2. A measure is 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:

$$f : A \rightarrow B | A \in X, B \in \mathbb{B} \quad (3.2)$$

Definition 3.3. A measure space MS is a 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:

$$MS = (X, A, f) | A \in X, f(A) \quad (3.3)$$

Definition 3.4. Two functions f and g such as:

$$f, g : A \rightarrow B | A \in X, B \in \mathbb{B} \quad (3.4)$$

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

as

$$\begin{aligned} m : fRg &\rightarrow B | B \in \mathbb{B} \\ m : A &\rightarrow B | B \in \mathbb{B} \end{aligned} \tag{3.5}$$

Definition 3.5. The measurement y is the result of the application $f(A)$ of the function f on the measurable set A such as:

$$y = f(A) | A \in X \tag{3.6}$$

3.1.2 Software Measurement terminologies

The terminology of these concepts in the software engineering context is defined as below:

Measurand a measurand is the observed reality. In this context, it is the performance or the maintainability of the system. It refers to the element X in a measurable space in the fundamental point of view (FPV).

Software properties the software properties are the software realities to be observed. According to the ISO/IEC standard 25010 (ISO/IEC, 2010a), there are 13 main observable "software reality" (Section 3.2.1).

Scope the scope is the measurable elements of a software such as code, the resources activity or the specification etc. In the FPV, a scope refers to the measurable set A of a MS .

Measurement a measurement is defined as a direct quantification of a measured property (Fenton & Bieman, 2014). This is the value of an evaluation result in a single time. This gives information on the measured property, such as the percentage of the memory used. In the fundamental point of view, 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 an indirect measure is a measure dependent on other measures. It refers to the dependent 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 context of a computer system. It specifies the measurand, the measure(s) and theory and the execution phase.

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.

To summarize, in order to specify a measurement context, it is needed to answers to 5 questions: *What? Who? How? As? When?* and *As?* is for the theory. The Figure 3.1 sums all of these theories.

3.2 Standards for software measurement

In this section, standards on software measurement are introduced. The first is a guide to design a quality measurement process to evaluate the software quality. It provides measurement models, metrics and methodology to manage a measurement process. The second one is a format to define software metrics. It provides a graphic meta-model of software metric.

Both standards are used as basis for proposing formal definitions of software metrics as contributions. And to build analysis model for our experiments described in the next chapter (i.e., Chapter 4).

3.2.1 Software quality measurement : ISO 25000

The standard ISO/IEC 25000 (ISO, 2005) provides a guide to the measurement of software quality. This version is based on the standard ISO/IEC 9126 (Carvallo & Franch, 2006) which defines a software quality model and the standard ISO/IEC 14598 which provides a software quality evaluation method.

The ISO/IEC 25000 is divided in 4 parts. Each division focuses on a specific part of the measurement process:

- the ISO/IEC 25001 focuses on the management of the measurement process: it provides in addition to context and preliminaries about the subject, the requirements and methodology for a quality management.
- the ISO/IEC 25010 provides software and data quality models. These models define the properties responsible of the software quality (as product and in use) and responsible of data quality stored in a structured format.



Figure 3.1 – Measurement context definition summary based on ISO/IEC 25000 and OMG SMM standards.

- the ISO/IEC 25020 provides a guide to apply the measurement, basically it provides the measures (informal) to evaluate each type of measurement (software product, in use, data). It defines more than 200 metrics organized according to the property they are evaluating.
- the ISO/IEC 25030 focuses on the first step of setting up the process by providing a guide to implement the quality requirements.
- the ISO/IEC 25040 provides requirement and guide for the evaluation phase, especially it provides a supervision model for a quality evaluation.

Our main interest about this standard is the ISO/IEC 25010 (ISO/IEC, 2010a) which provides a quality software model. That means, it defines the software properties which characterize the software quality. As our work aims to cover the measurement of all the software characteristics during all the phases of the software engineering process. This model is then used as basis for our experiments to define our analysis models.

The standards distinguishes two types of system: software as product and in use, so a product in a particular context. The quality of software product is defined by 8 properties:

- Functional Suitability: the quality of the services.
- Performance efficiency: the quality of the resources. activity
- Compatibility: the capacity of working and exchange with other environment, software etc.
- Usability: the ease of use of the software.
- Reliability: the capacity to meet the needs and the fault tolerance capacity.
- Security: the capacity to secure the information and the data.
- Maintainability: the capacity to be maintain.
- Portability: the capacity to be installed and use from different environment.

Each of these properties are divided in sub properties more specific to an aspect of the general property as illustrated in the Figure 3.2.

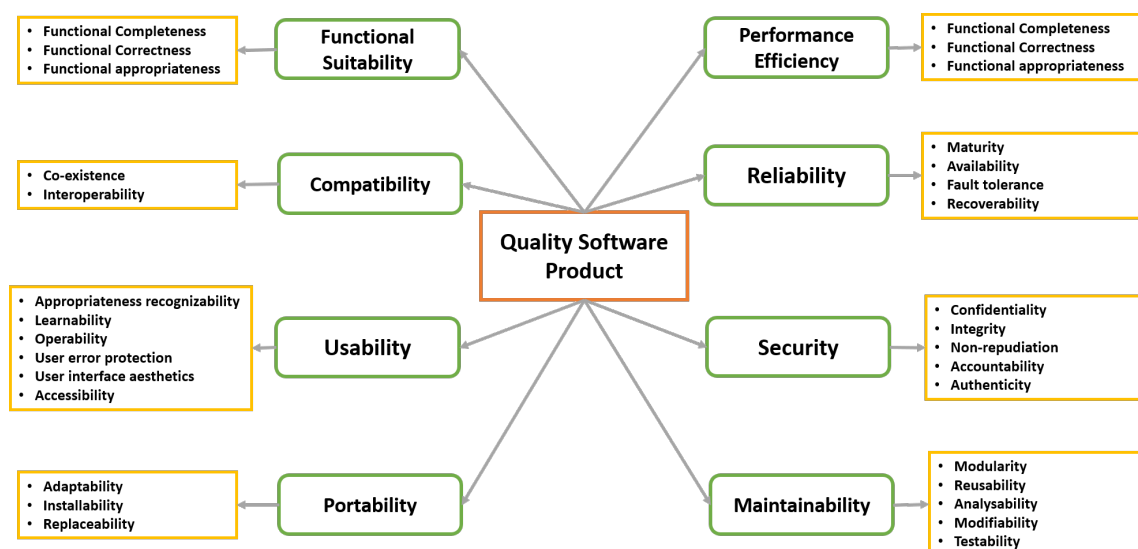


Figure 3.2 – Quality software model.

Regarding the quality of product in use, this is defined in 5 properties: the effectiveness, efficiency, satisfaction, freedom from risk and context coverage. (see (ISO/IEC, 2010a) for more details).

3.2.2 Software measurement definition language

The OMG standard Structured Metric Meta-model (SMM) (Group, 2018), provides a graphic meta-model to specify a measurement context. This language used the OMG standard UML language (UML & MOF, 2018).

The purpose of using this language is to have a formal specification language which will serve as documentation and as architecture deployment model. This model is then used as a skeleton of the implementation and thus, from a formal specification to ensure an implementation that is sustainable, maintainable and easy to handle at each level of the measurement process. Thereby, a specification is accessible and useful at each level of the metric use: by the manager, the developer, and the analyst.

Thus, 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, generating a model of the implementation structure from a model of the measurement architecture with SMM allows the use of the code generation feature.

Moreover, this process permits 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.

This section is organized it follows: first we introduce the SMM meta-model. Then we present the MEASURElanguage, based on UML and designing the SMM meta-model, used to specify and generate a generic model of metric implementation architecture. To finish with our contributions of software metrics specification and implementation.

Structured Metric Meta-model (SMM)

SMM is a standard specification which defines a meta-model to specify a software metric, 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 types between dependent measures (as the ratio, binary or grade measure). The SMM specification allows the description in detail of the main following concepts as illustrated in the Figure. 3.3 that shows the main model of the SMM architecture:

- MeasureLibrary
- Measure
 - Scope

- Unit of measure

- Observation
- Measurement

These concepts refer to the concepts defined in the previous section. The *Scope* is the subset defining the measurable elements. The Unit of measure allows to specify the basis theory.

The *MeasureLibrary* refers to the set of measures whose the *Measure* is dependent while this latter refers to the function of measurement.

Then *Measurement* refers to an evaluation result and *Observation* refers to the application context of the measure.

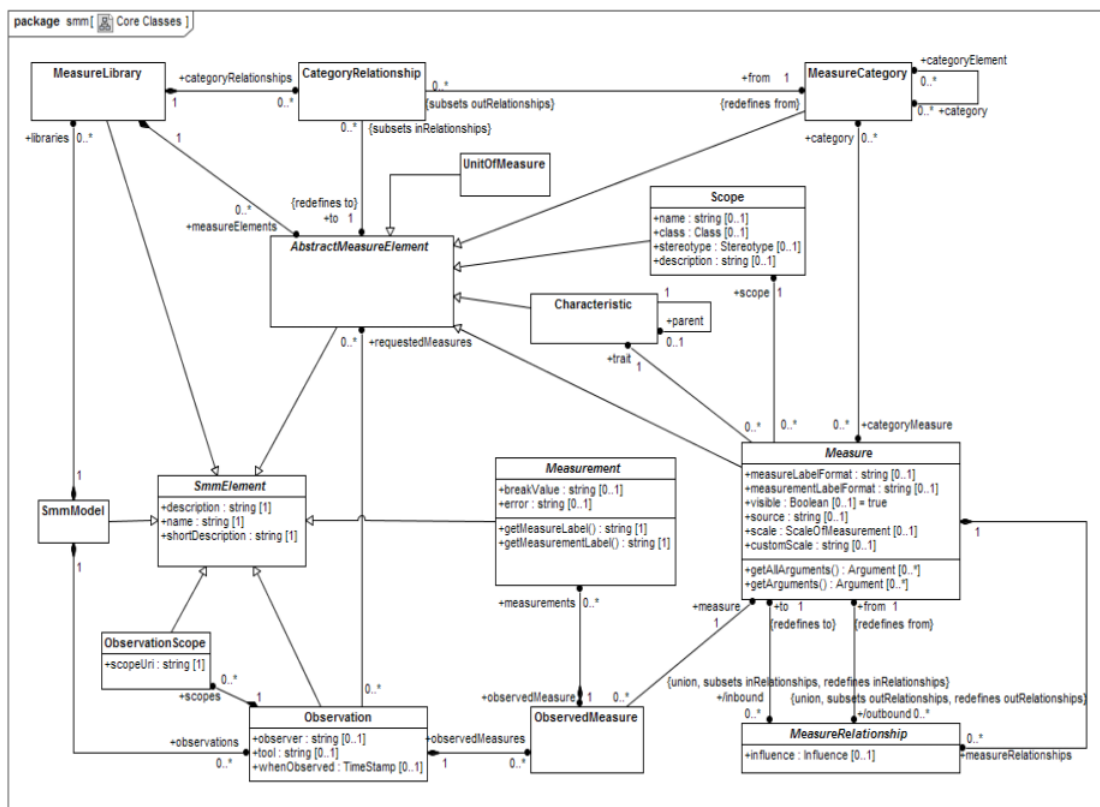


Figure 3.3 – SMM Overview.

There are the key concepts defining a software measure architecture with the SMM specification (Group, 2018), and the links between them. Thus, one can see the parent entity *SmmElement*, which all other elements inherit from. From this entity, the hierarchy link between those concepts is described. More details about such links are given later. We first present the *SmmElement* entity, the *AbstractMeasureElement* and the Measurement elements. Later on, the *SmmRelationship* concepts, and finally, the observation properties are being described.

SmmElement SmmElement is the top abstract class in the SMM specification hierarchy. As shown in the Figure 3.4, it allows to declare a name and a description of entities specified through optional attributes: name, shortDescription, and description.

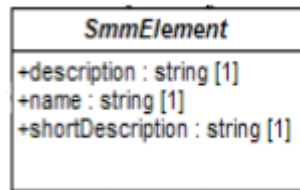


Figure 3.4 – SmmElement.

The entities that directly inherit from SmmElement are:

- SmmModel,
- MeasureLibrary,
- AbstractMeasureElement,
- Measurement
- SmmRelationship,
- Observation,
- ObservationScope,
- ObservedMeasure,
- Argument.

The Figure 3.5 presents the first level hierarchy of the SMM specification. Entities shown in this figure are described below.

SmmModel is an abstract class which defines the entry point into a SmmModel and provides the top-level container for all the elements of the SMM. It is associated with all MeasureLibrary and all observations defined in the model.

MeasureLibrary class represents a set of measure references that are independent of the measurand (SMM term to describe the measured object) and the context. It is associated with a set of AbstractMeasure and a set of category relationships, related to those measures.

The AbstractMeasureElement, Measurement and SmmRelationship are abstract classes which allow to define measures, measurement and relations between entities. These elements will be presented in details in the following subsections.

Observation entity represents the context of the measurement model. It depends on the classes, namely ObservationScope, ObservedMeasure and Argument which allow to define different context aspects. Details about the observation are also given below.

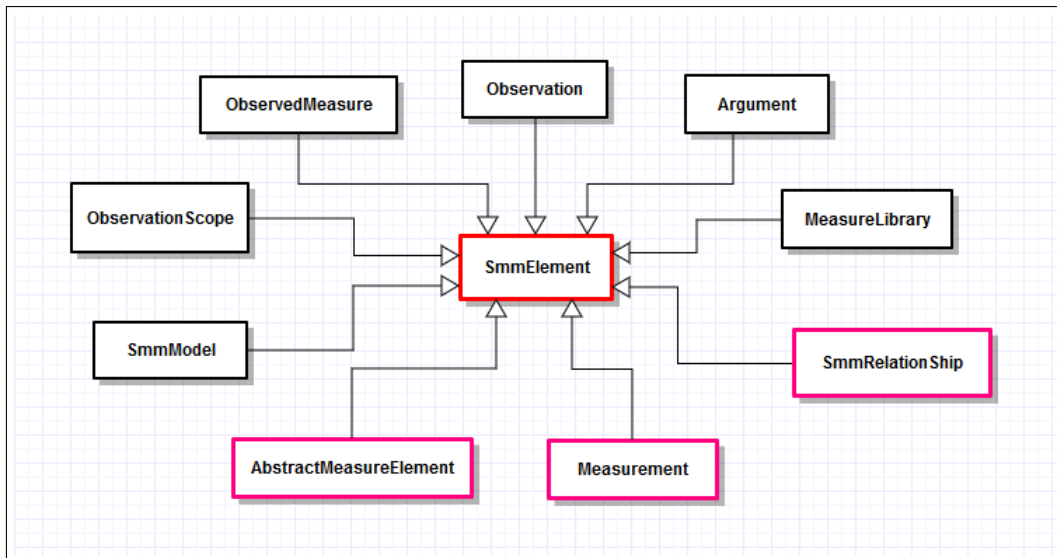


Figure 3.5 – SMM first level hierarchy.

AbstractMeasureElement This abstract class is the top of measure calculation definition. As shown in the Figure 3.6, it can be refined in five different entities:

- Scope,
- Characteristic,
- UnitOfMeasure,
- Operation,
- OclOperation,
- Measure.

The Scope, Characteristic and UnitOfMeasure classes allow to specify the Measure details. The scope class specify the measurable element, the Characteristic class allows to specify the observed reality, while the UnitOfMeasure class provides a representation for the units of measure definition of the Measure class.

Operation class defines the procedure to be executed by a Measure with the needed parameters. OCLOperation class defines OCL (Object Constraint Language) helper methods for having more information about the parsing of measures.

The abstract class Measure allows to define the different types of calculations (see Figure 3.7). The abstract class Measure can be refined in two concepts: GradeMeasure and DimensionalMeasure.

The GradeMeasure class defines calculations that return a classification of a dimensional measure.

The DimensionalMeasure abstract class allows to specify measures that assign numeric values. Dimensional measures have units of measure and their values belong to a dimension.

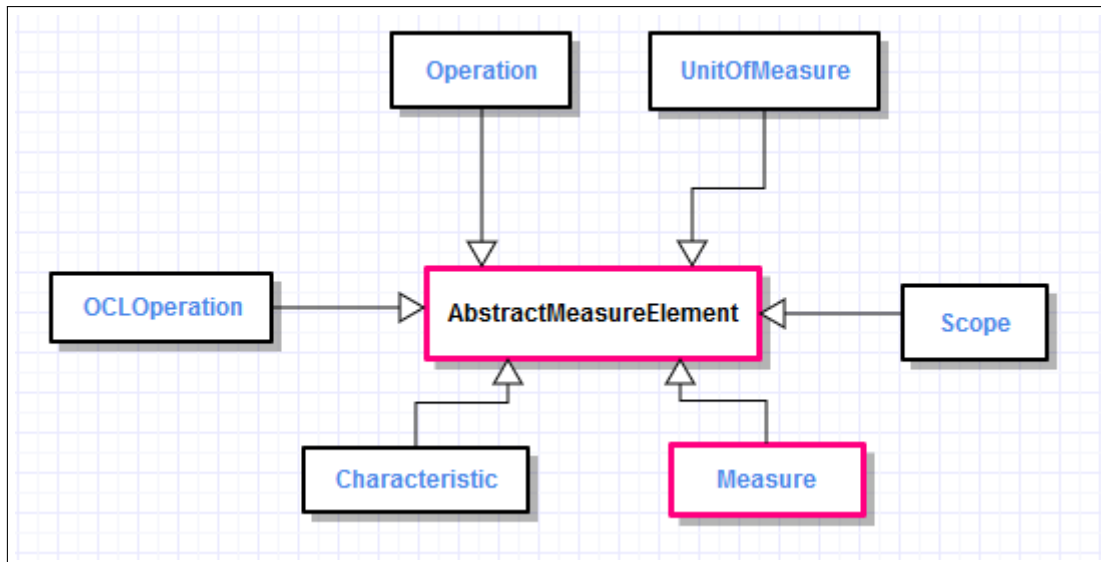


Figure 3.6 – Measure level hierarchy (1).

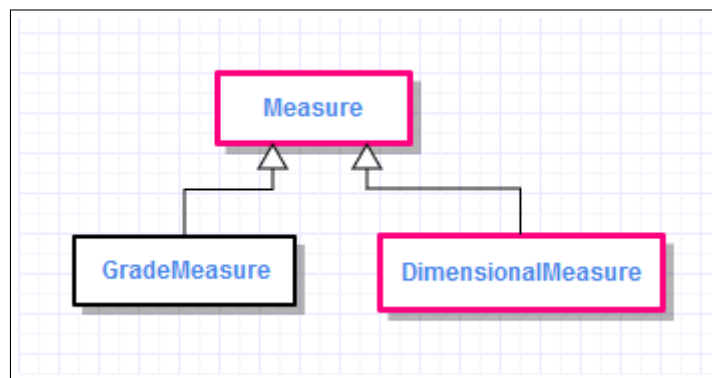


Figure 3.7 – Measure level hierarchy (2).

Figure 3.8 depicts the measure hierarchy from DimensionalMeasure. This entity can be refined in:

- DirectMeasure,
- CountingMeasure,
- RankingMeasure,
- CollectiveMeasure,
- BinaryMeasure,
- RatioMeasure,
- NamedMeasure,
- RescaledMeasure.

There are two types of measures, the direct measure, directly applied to the measurand and the rest, which are applied to the result of direct measure, called derived measure.

DirectMeasure and CountingMeasure classes define the measures directly applied to the measured software. The first one is associated with an Operation entity that specifies the calculation operation to be executed. The second refines the first entity and it represents a DirectMeasure which operation returns the value of 0 or 1. Those measures are known as direct measures.

Regarding the derived measures, there exists the RankingMeasure class, which defines measures that classify the results of a DirectMeasure. On the other hand, the CollectiveMeasure class represents measures that collect measurements of a given measuring element and entities similarly related to this element; it has an attribute Accumulator which declares the accumulation function. In the same way, the BinaryMeasure class represents a measure that collects measurements of two entities related to the measuring entity. It defines a BinaryFunctor attribute which refers to the binary function that combines the two base results.

The RatioMeasurement class represents measures which return the ratio of two other measure results.

Furthermore, the NamedMeasure defines well-known measures and its name is sufficient to identify the needed measure. Meanwhile, the RescaleMeasure class refers to the calculations that compute a measurement already defined in a unit of measure to another unit of measure.

All those measures return a result that can be modelled by the SMM specification, referred to as Measurement.

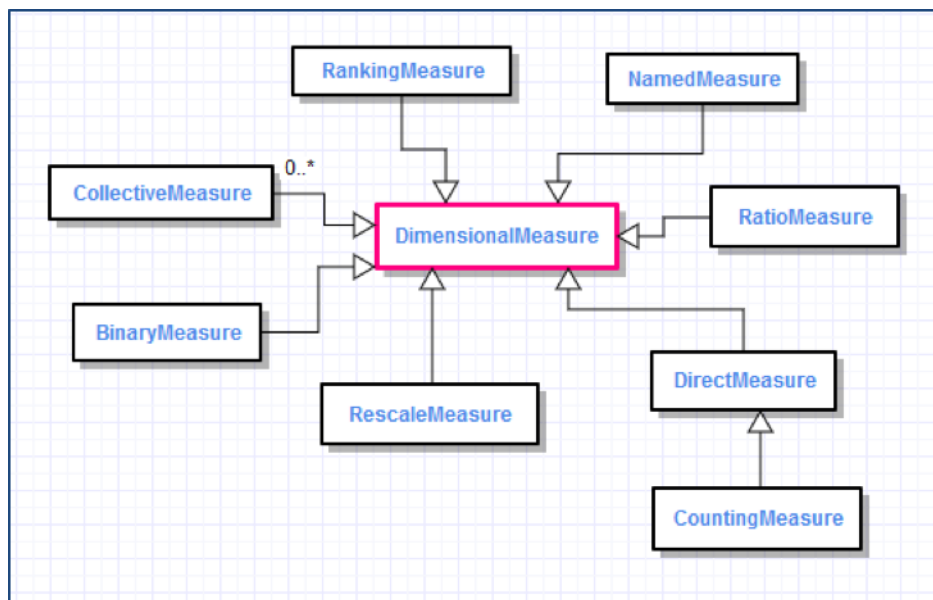


Figure 3.8 – Measure level hierarchy (3).

Measurement SMM defines the different types of results with the Measurement entity. For each type of Measure there is the same type of Measurement. The Measurement hierarchy is the same as that of Measure. Measurement is an abstract class and can be specified in:

- GradeMeasurement,
- RankingMeasurement,
- DimensionalMeasurement.

GradeMeasurement and RankingMeasurement classes inherit directly from the Measurement abstract class. The first entity represents the result of the GradeMeasure and the second of the RankingMeasure.

The DimensionalMeasurement abstract class defines the result of DimensionalMeasure and can be refined in:

- DirectMeasurement,
- CountingMeasurement
- CollectiveMeasurement,
- BinaryMeasurement,
- RatioMeasurement,
- NamedMeasurement,
- RescaledMeasurement.

As explained in the previous paragraph, the DirectMeasurement class defines the result of the DirectMeasure entity and CountingMeasurement is a subclass of DirectMeasurement. It represents the CountingMeasure result.

For the rest, the definition is similar, each one defines the result of the associated Measure, namely CollectiveMeasure class for CollectiveMeasure, BinaryMeasurement for BinaryMeasure etc.

The SMM specification defines the SmmRelationship entity to model the link between the derived measures and measures on which they are based. This allows to associate the derived measurements with its respective direct measurement.

SmmRelationship SmmRelationship is an abstract class and a subclass of the SmmElement. SmmRelationship is divided into two types of association: MeasureRelationship and MeasurementRelationship. The first one is used to define the links between Measures, and the second - for the links between the Measurements. Each type of SmmRelationship is defined as a subclass of it.

MeasureRelationship is refined by:

- EquivalentMeasureRelationship,
- RefinementMeasureRelationship,

- GradeMeasureRelationship,
- RescaledMeasureRelationship,
- BaseMeasureRelationship.

The EquivalentMeasureRelationship class allows to define two measures, which are equivalent. The RefinementMeasureRelationship class is used to declare that a Measure M1 is refined by a Measure M2. The GradeMeasureRelationship class represents the link, which associates the DimensionalMeasure graded by a GradeMeasure. Furthermore, the RescaledMeasureRelationship is used to link the DimensionalMeasure rescaled by a RescaledMeasure. The BaseMeasureRelationship is an abstract class to define the links of hierarchy between a derived Measure and its base Measures. This link can be refined in:

- RankingMeasureRelationship,
- BaseNMeasureRelationship,
- Base1MeasureRelationship,
- Base2MeasureRelationship.

The RankingMeasureRelationship class defines the association between a DimensionalMeasure and a RankingMeasure which ranks the first one.

The BaseNMeasureRelationship class represents the link between a CollectiveMeasure and a DimensionalMeasure, which the CollectiveMeasure is based on. The Base1MeasureRelationship and Base2MeasureRelationship classes define the relationships between a BinaryMeasure and its two base DimensionalMeasure. The Base1MeasureRelationship for the first DimensionalMeasure and Base2MeasureRelationship for the second.

The relationship hierarchy of Measurements is the same as that of Measures. The link classes are subclasses of MeasurementRelationship, and they are:

- EquivalentMeasurementRelationship,
- RefinementMeasurementRelationship,
- GradeMeasurementRelationship,
- RescaledMeasurementRelationship,
- RankingMeasurementRelationship,
- BaseMeasurementRelationship,
 - BaseNMeasureRelationship,
 - Base1MeasureRelationship,

– Base2MeasureRelationship.

Thereby, via the UML language, SMM allows to define, an architecture of software measurement, as well as the context of the concrete measurement (observation).

Observations In an SMM specification, the different aspects of a measurement environment are defined through four elements: The Observation class, which represents the contextual information as date of measurement, the tools being used and the measurer through three attributes. Tools and observer are string attributes. The date attribute is a Timestamp entity defined by SMM as a primitive type, which represents a point in time. The ObservationScope class allows to describe the subject of the related measurement through a string attribute referencing the model or a part of model measured. The ObservedMeasure class represents the link which associates Measures and observations. The Argument class defines the parameters or variable arguments that are passed to the measures of Operations, which use parameters.

In the following sections, we describe our contributions to reach our objectives.

SMM-based software metrics graphic language

In order to reach the previous purpose, that is to say, to generate a "standard" code architecture from the SMM model, we use the Modelio modelling tool¹ of our MEASURE Project partner and its extension dedicated to SMM modelling. This extension provides the *MEASURE language* to model a metric in SMM and supports the code generation feature.

Modelio is an open source modeling tool based on UML. It supports several OMG's standard as SysML, MARTE and etc. By using its extendable module SMM which provides the MEASURE language, we can model a SMM-based metric and generate the corresponding code architecture.

The MEASURE language The MEASURE language defined a UML model composed of 5 classes. The main class is the *SMMMeasure* which refers to the MEASURE element in SMM. It specifies according to its type, so there are as many classes measure as of type of measures defined in the SMM.

Then, the *measure* class is linked to the *ScopeProperty* class. This one defines the elements on which the assessment will be done. For example, to measure the complexity of a code, we put the path of the code.

Then, there is the *Unit* class by which the type of results is defined. In other words, the unit of measure, as example for a direct measure the unit could be a number or for a ratio measure the unit could be a percentage.

Finally, the *MeasureReference* class are the measure classes whose the main measure is dependent.

¹<https://www.modelio.org/>

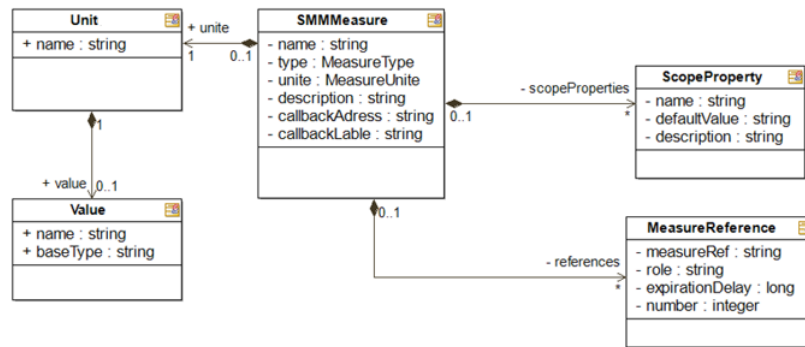


Figure 3.9 – MEASURE Language.

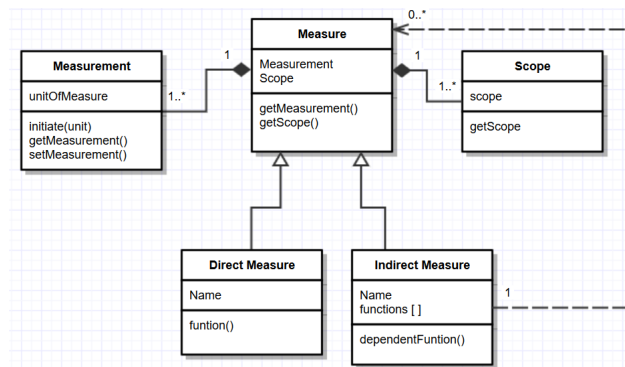


Figure 3.10 – UML model designing a Software Measurement Code architecture.

Code architecture generated from SMM model The metric code architecture based on SMM is a generated as generic UML model defining the code structure of a software metric. It is so, the interpretation of the SMM model in OO architecture. As illustrated in the Figure 3.10, the model is made of the principal class *Measure* which is composed of two objects:

- the *Scope* for the definition of software structure to be measured such as the file code or a repository path.
- The *Measurement* which refers to the *Unit* class in the MEASURE language and defines the type of the measurement result.

And it is specified in two types of measures:

- *DirectMeasure*, if it is an independent measure.
- *IndirectMeasure*, if it is dependent on other measures.

SMM-based metrics implementation Finally, from this "standard" architecture and by using the code generation feature, a skeleton of the metric is generated. As illustrated in the Figure 3.11, the code is structured into 3 files:

- The scope file through XML file,

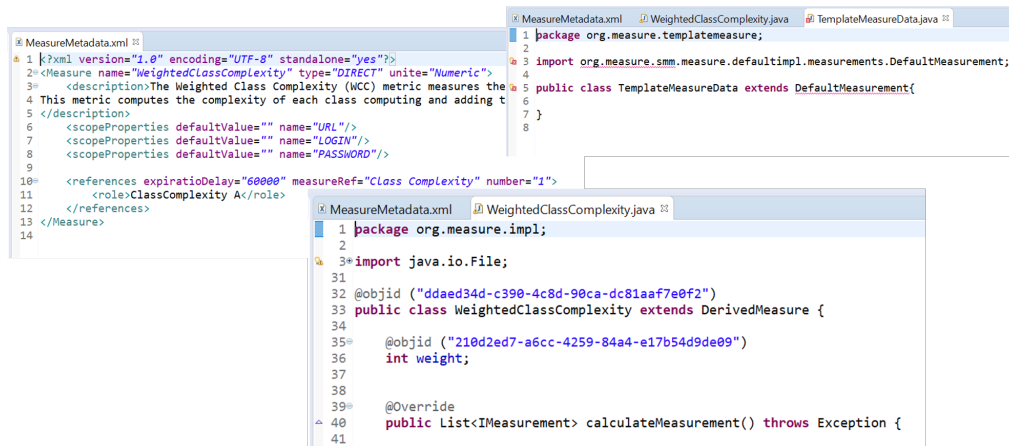


Figure 3.11 – SMM-Based Metric code implementation.

- The Measure file through a java class,
- The Measurement file through a java class.

The XML file contains all the information about the measure, its name, its type, its type of result. This information is stored in the "Measure" tag. Then, there is the information about the measurable elements with the "scopeProperty" tag, then the references to the dependence metrics by the "references" tag.

Regarding the Java class Measure extends the type of measure `DirectMeasure` or `DerivedMeasure`, if it is an indirect measure. It provides the method `calculateMeasurement()` where the assessment is implemented.

Finally, the Java class Measurement extends `DefaultMeasurement` and this class implements the type of returned result if this latter is not a number (integer).

3.3 Experiments

In this section, we presents our contributions we propose specifications, implementations through SMM of low and medium level software metrics. These metrics have been integrated in the MEASURE platform, described below, as formal measuring approaches.

3.3.1 MEASURE Platform

The MEASURE platform provides services to host, configure and collect measures, storing measurement, present and analyze them. These measures are first defined in SMM standard using the Modelio modeling tool² and its extension dedicated to SMM modeling. They are packaged under an executable format as Measure Definition.

Next, measures are registered and stored on Measure platform using the dedicated REST service or the Web user interface. In order to initiate the collect of measurement, the next step

²<https://www.modelio.org/>

consists on defining instance of measure based on measure definitions. A measure represents a generic data collection algorithm that has to be instantiated and configured to be applied on a specific context. For example, a measure which collects data related to an SVN repository must be configured by the URL of this repository.

The MEASURE platform can collect measurements (data resulting of the execution of an instantiated measure). Direct measures collect data in physical world while the Derived Measures are calculated using previously collected measurement as input. Collected measurements are stored on a NoSQL designed to be able to process a very large amount of data. To collect measurements, the direct measures can delegate the collect work to existing measurement tools.

Finally, stored measurements are presented directly to the end user following a business structured way by the Decision-making platform, a web application which allows organizing measures based on projects / software development phases and display its under various forms of charts. The measurements can also be processed by analysis tools to present consolidated results.

Platform architecture

The Measure Platform architecture is organized around three main deployment units: The Measure Platform which manages the collection and computation of measurement and the presentation of this measurement, the analysis platform which regroups measurement analysis tools and finally the Measure Front which regroups measure tools and Platform Agents. As described in the Figure 3.12, there is the main Measure Platform which enables the communication between the different stores and the analysis tool, the measurement tools and the agent platform. The measurement tools allow to use other external measurement tools. And the agent platform allows a local execution of the metrics with a storage of results in the platform.

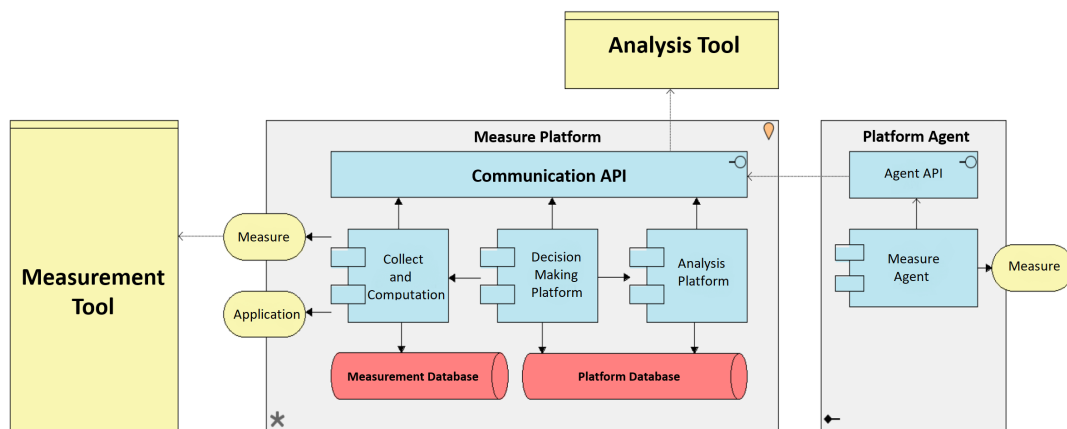


Figure 3.12 – MEASURE platform

Measure Platform component

This is the central component, the Measure Platform provides services related to data collection, analysis and display. It is composed of six sub-components:

- The Communication API is a remote API which allows the Analysis Platform and Agent to communicate with the Measure Platform. It provides various services like an access to measurements data, the configuration of measures or the ability to manage the scheduling of measure execution.
- The Collect and Compute Component is in charge of the storage of Measure Definition, the instantiating of Measure, the scheduling of measure execution and organizes the measurements collection and processing.
- Decision Making Component represents the web application which assures the presentation of measures and collected measure to the end user.
- The Platform Database is a standard SQL database which manages data in relation with the collect, organization and presentation of measures.
- The Measurement Database is a separate NoSQL database which ensures the measurement persistence in an efficient and scalable way.
- The Analysis Platform ensures the integration of external analysis tools into the platform. As the Measure Platform focuses its activity around collecting and calculating measurement in real time, the Analysis Platform will provide long terms analysis and predictive evaluations by working on the history of measures stored in measurement database.

PlatformAgent component

This component manages the measure tools on client side which collects data. The executable of the measure provides a way to collect data in the physical world. likewise a measure can be executed on the Platform side and collects physically this data through an existing measure tool, a measure can be directly executed on the client side by the intermediary of a Platform Agent.

Measurement Tool component

This component manages the measurement tools. This latter are external tools which collects or calculates measurements from a specific source. In the context of the project, 5 Measurements tools have been developed and lots of exiting tools in the market, as Sonar Cube for code quality metrics, has been integrated in the platform as measurement tools.

Analysis Tool component

This component manages the analysis tools. These latter are sets of external services which work on the historical measures values in order to provide advanced and valuable analysis functions to the platform. In order to support a large set of analyzed services and do not limit them to a specific technology, the analysis tools are external processes. The analysis tool is integrated to the platform using a specific API. This integration includes embedded visualization provided by the analysis services into the platform.

The REST API allows an analysis tool to register on the platform, to receive notifications from the platform and access to information related to the project defined and the measures collected by the platform. On its side, the analysis tool provides web pages which are embedded into the platform web application.

Measure component

The Measure component manages the developed measures. A measure is a small and autonomous java program based on the SMM specification which allows to collect measurements. A measure can be direct (collect of measurements in the physical world), a proxy (it ensures communication between a measurement tool and the Platform) or derived (calculated by the aggregation of existing measures).

Application component

This component manages the applications. An application is a set of measures aggregated together in order to address functional requirements. The application is associated with a visual dashboard which is directly integrated into the Decision Making platform when the application is deployed on a project.

The platform activity is organized around its ability to collect measurement by executing measures defined by the SMM standard. SMM measures are auto-executable component, implemented externally, which can be executed by the platform to collect measurements.

3.3.2 Formal software metrics development

In this section, we present our contributions to the formalization of metrics, the implementation of the assessment and their integration in the MEASURE platform. A metric is integrated in the platform as a measuring tool, executable locally or directly from the platform (i.e., Section 3.3.1).

This section is organized as follows: for each metric herein introduced, first we define it, then we present its modeling, after, we describe the implemented calculus of the metric and its integration in the platform.

The models are presented by an image corresponding to the class type: the unit of measure is represented by yellow braces, the scope by a target white and blue and the measure by its type. A direct measure is represented by a microscope and a collective measure by a cube of different colors etc.

JVMCpuUsage metric

Definition JVMCpuUsage computes the CPU usage by the execution of the JVM and by the whole system that executes the JVM.

SMM model The metric is modeling as a direct measure since it is independent. Its scope is the JVM where the metric is executed and its unit of measure is a set of numbers, that is why the unit is called JVMCpuUsageUnite As shown in the Figure 3.13.

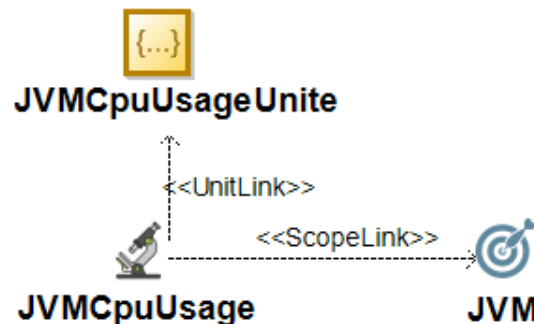


Figure 3.13 – SMM modeling from formal definition of JVMCpuUsage metric.

Implementation The implementation uses the library `OperatingSystemMxBean` to return the CPU usages of the JVM execution and of the entire system executing the JVM. It returns two `Double` corresponding to the both types of CPU usage computed. Those results are either between 0.0 and 1.0 or a negative value if the CPU usage is not available. The GitHub of the measure is available at: ³

Integration This metric is integrated in the platform. If it used directly from the platform, it will assess the JVM usage of the platform. Otherwise, if it is used locally through an agent it will assess the local jvm usage.

The result of the `CpuUsage` measure is represented by two numbers associated to the type of CPU usage computed (the JVM and the System).

The figure 3.21 shows its integration and the Figure 3.14 shows the result of its execution from the platform.

³<https://github.com/ITEA3-Measure/Measures/tree/master/CpuUsageMeasure>

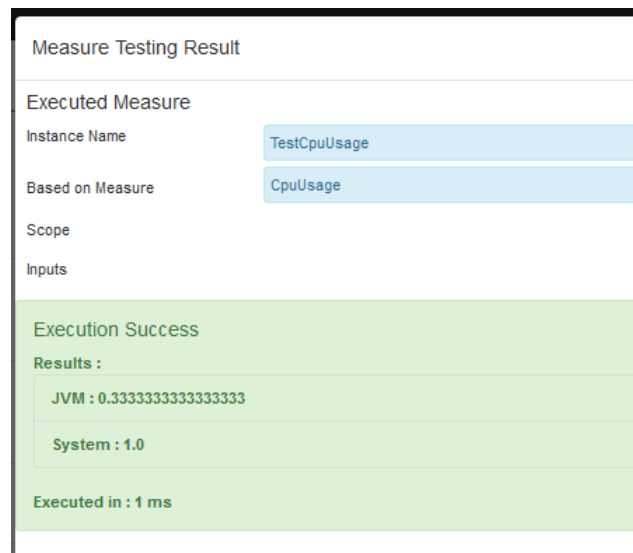


Figure 3.14 – Result of the JVMCpuUsage metric execution from the platform.

JVMMemoryUsage metric

Definition This measure computes the memory usage of the heap, memory reserved for the objects allocation by the JVM and the memory usage used by the JVM for its execution (the heap memory excluded).

SMM model The metric is modeling as a direct measure since it is independent. Its scope is the JVM where the metric is executed and its unit of measure is a set of numbers, that is why the unit is called JVMMemoryUsageUnite As shown in the Figure 3.15.

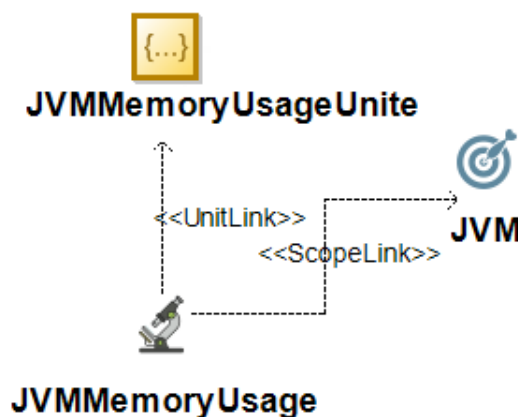


Figure 3.15 – SMM modeling from formal definition of JVMMemoryUsage metric.

Implementation The implementation of this measure uses the library MemoryMxBean to return the memories usages for the JVM execution. It returns two Long corresponding to the amount of the different memories used in bytes. The GitHub of the measure is available at: ⁴

⁴<https://github.com/ITEA3-Measure/Measures/tree/master/MemoryUsageMeasure>.

Integration As JVMCpuUsage metric, this one is integrated in the platform. If it is used directly from the platform, it will assess the JVM memory usage of the platform. Otherwise, if it is used locally through an agent it will assess the local jvm memory usage.

The result of the MemoryUsage measure is represented by two numbers associated of the type of memory usage computed (the Heap and the JVM). This is illustrated in the Figures 3.21 and 3.16.

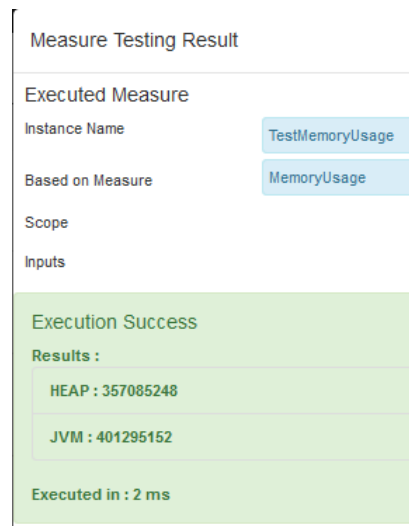


Figure 3.16 – Result of the JVMMemoryUsage metric execution from the platform.

ClassComplexity metric

Definition This measure 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 scope of this metric is represented by three elements: the URL of a SVN repository containing the measured Java project, the login and the password necessary to connect to this repository.

SMM model The metric is modeling as a direct measure since it is independent. Its scope is an OO code and its unit of measure is weight, more precisely a cognitive weight. The Figure 3.17 illustrates the modeling.

Implementation The implementation of this measure computes the cognitive weight of a Java project by adding the cognitive weight of each method of each class. The GitHub of the measure is available at:⁵ In order to compute the cognitive weight of a method, we focus on the nested level of conditional statements/blocks (if-then-else) and the for and foreach loops. As shown in the pseudo-code below, each level is associated with a weight (a number) and

⁵<https://github.com/ITEA3-Measure/Measures/tree/master/CognitiveComplexityMeasure>

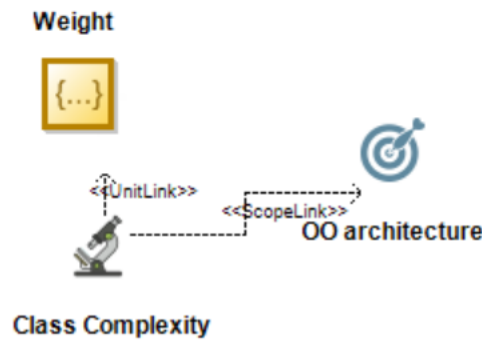


Figure 3.17 – SMM modeling from formal definition of ClassComplexity metric.

thus all these weights are summed. Therefore, we add the weights of all methods to have the cognitive weight of a class as shown in the pseudo-code below.

```
CognitiveWeight(statements block, integer weight, integer level)
    For each statement S of block
        If S is conditional or for or foreach statement:
            level2=level+1
            weight2=weight+level
            S1 = block1 of S
            CognitiveWeight(S1,weight2,level2)
        End If
    End For
End
```

Integration The metric is integrated in the measure platform as shown in the Figure 3.21. To execute it, it is mandatory to fill the path of the project to be evaluated.

The result of the cognitive measure is the number representing the cognitive weight of the project, in other terms, it returns the sum of the weight of each method of each class of the project provided as the scope parameter. This is illustrated in the Figure 3.18

WeightedClassComplexity metric

Definition The Weighted Class Complexity (WCC) metric measures the quality of code design and its maintainability. This metric is based on the ClassComplexity (CC) metric and adds the number of attributes of each class to compute the cognitive weight of a class. Then to compute the full code weight, WCC adds the weight of each class if there are in the same level or multiply if there is derivation link

SMM model from the informal and formal definition (see Figure 3.19 we model the metric as a collective measure as it depends on the ClassComplexity metric. Its scope is an OO code and its unit of measure is weight, more precisely a cognitive weight.

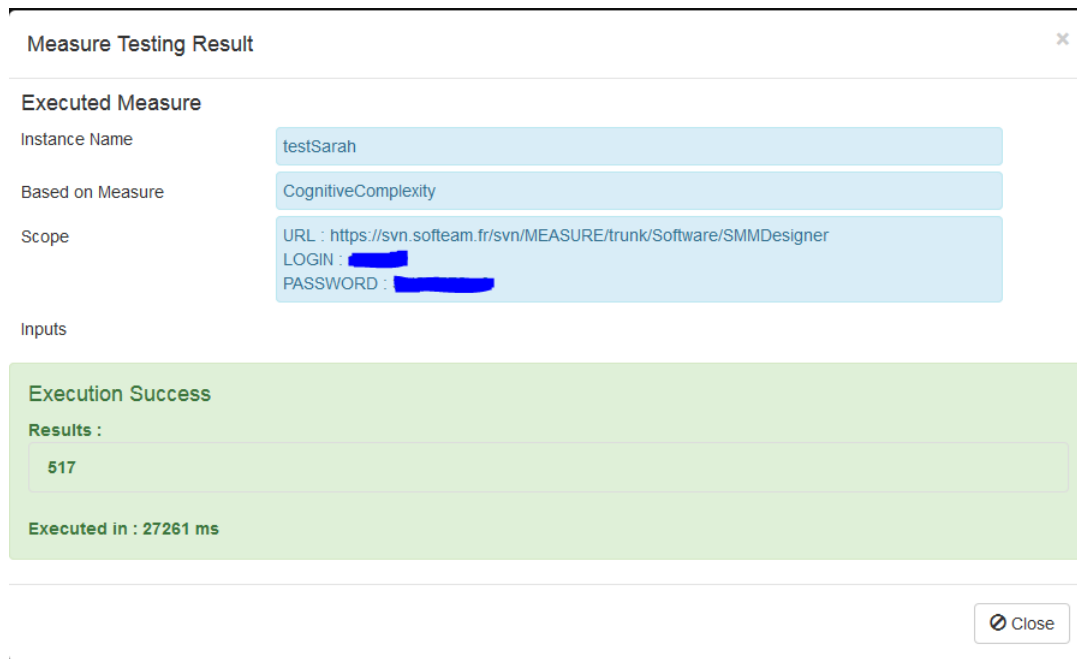


Figure 3.18 – Result of the ClassComplexity metric execution from the platform.

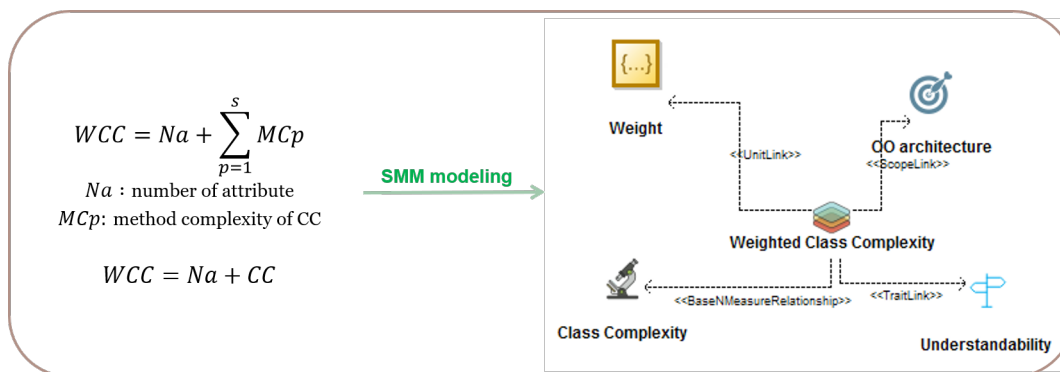


Figure 3.19 – SMM modeling from formal definition of WCC metric.

Implementation The implementation of this measure computes the cognitive weight of a Java project by adding the sum of the cognitive weight of each attribute of each class to the cognitive weight of the CC measure result (sum of the cognitive weight of each method of each class). In order to compute the cognitive weight of an attribute, we focus on the type of attributes. As shown in the pseudo-code below, each type is associated with a weight (a number) and thus all these weights are summed. Therefore, we add the weights of all attributes to have the attributes cognitive weight of a class. Finally, the attributes weight is summed with the cognitive weight of the methods computing by the ClassComplexity measure.

```
AttributeCognitiveWeight(attributes attSet, integer weight)
```

```
  For each attribute A of attSet
```

```
    If A is primitive type:
```

```
      weight=1
```

```
    else If A is List<primitive> type:
```

```
      weight=weight+2
```

```

        else If A is object type:
            weight=weight+3
        else If A is List<object> type:
            weight=weight+5
        else If A is other type:
            weight=weight+4
        else If A is List<other> type:
            weight=weight+6

    End If
End For
End

```

Integration The metric is integrated in the measure platform. To execute it, it is mandatory to fill the path of the project to be evaluated.

The result of the WeightedClassComplexity measure is the number representing the cognitive weight of the project, in other terms, it returns the sum of the cognitive weight of each attribute and each method of each class of the project provided as the scope parameter. This is illustrated in the Figure 3.20

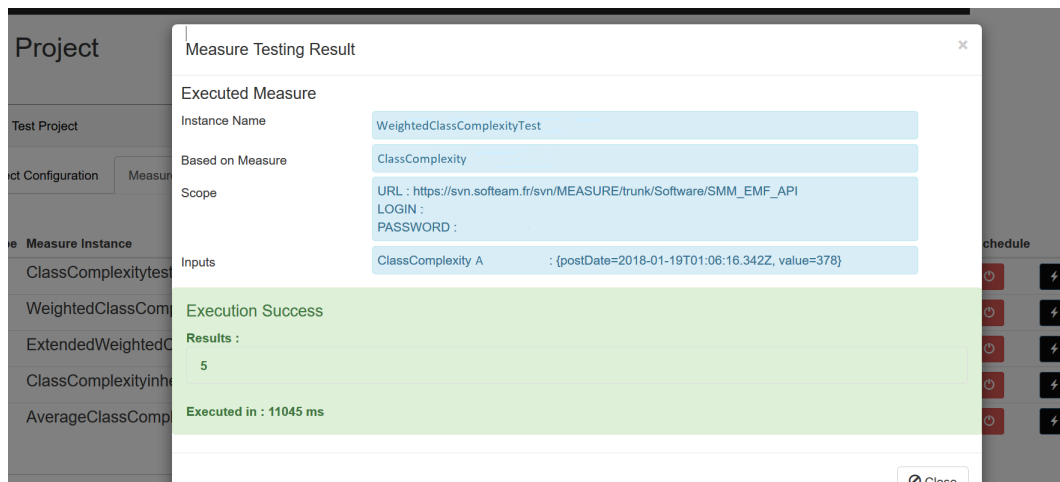
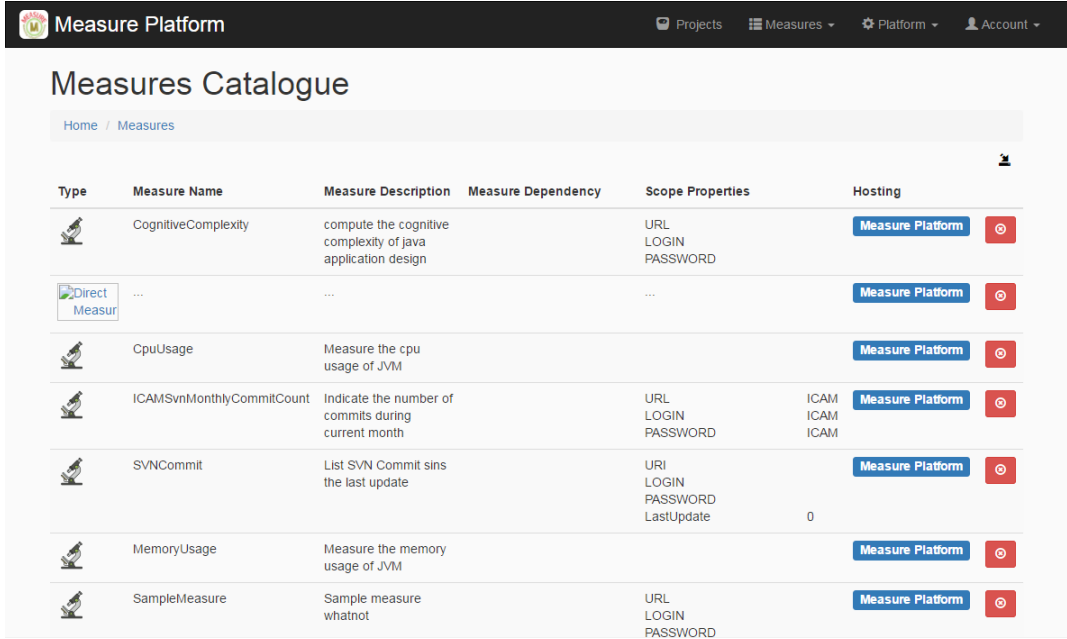


Figure 3.20 – Result of the WeightedClassComplexity metric execution from the platform.

3.4 Conclusion & discussion

To conclude, in order to contribute in the lack of formalization in the field of software measurement, we deeply studied the OMG's standard specification SMM which proposes a meta-model for the specification of software metrics. Through this task, we proposed a formal design approach of software measurement based on this standard. This approach aims at facilitating the use of SMM and also to extend this formalization feature to the implementation,



Type	Measure Name	Measure Description	Measure Dependency	Scope Properties	Hosting
	CognitiveComplexity	compute the cognitive complexity of java application design		URL LOGIN PASSWORD	Measure Platform
	...	...	...		Measure Platform
	CpuUsage	Measure the cpu usage of JVM			Measure Platform
	ICAMSVnMonthlyCommitCount	Indicate the number of commits during current month		URL LOGIN PASSWORD	ICAM ICAM ICAM Measure Platform
	SVNCommit	List SVN Commit sins the last update		URI LOGIN PASSWORD LastUpdate	0 Measure Platform
	MemoryUsage	Measure the memory usage of JVM			Measure Platform
	SampleMeasure	Sample measure whatnot		URL LOGIN PASSWORD	Measure Platform

Figure 3.21 – Metrics integration in the platform.

that allows to have not only all the design part of the measurement process formalized but also documented due to the proposed graphic approach and an implementation of software metrics according to an architecture standard.

Until this work, the software metrics were specified in natural language with the ISO/IEC standard but not formally. Also, several tools offer some implementations of metrics but they are not formalized and their implementations are dependent to the developer and do not follow a standard architecture.

Furthermore, by this work carried out in an industrial European project MEASURE context and in collaboration with SOfTEAM CADEXTAN, we contribute to the SMM release 1.2 of 2018 (Group, 2018) by sharing the founded issues and ambiguities that have been fixed.

The goal of all this work is to have a solid formal basis to improve the software measurement design phase, in order to increase the interoperability, scalability and maintainability of the measurement process.

Finally, we contribute in the MEASURE project, by the integration of metrics specified and implemented through the SMM standard into the MEASURE platform as measuring tool and accessible from this latter. That allows to increase the functionality of the platform and its interest.

4

Automated software measurement analysis and suggestion

Contents

4.1	Manual analysis model	49
4.1.1	Initial measurement plan	49
4.1.2	The mandatory metrics	50
4.1.3	The mapping system	50
4.2	Data analysis from measurements	51
4.2.1	Classification	52
4.2.2	Supervised learning	55
4.2.3	Classification technique	55
4.2.4	The Support Vector Machine	57
4.2.5	The dynamic mandatory metrics selection	60
4.3	The Suggestion	61
4.3.1	Analysis results	62
4.3.2	Use of the analysis model	62
4.3.3	The suggestion algorithm	62
4.4	Application	63
4.4.1	Metrics Suggester tool architecture	64

4.4.2	Industrial integration	67
4.4.3	Experiments	70
4.4.4	Setup	70

In this chapter we present our approach to handle the following issues: the static measurement plan and the sequential analysis of a lot of data. By improving this latter, we try to reduce the management load while ensuring an efficient evaluation continuously.

The first objective is to have dynamic measurement plans that evolve according to the software project needs. That means that the evaluation will be guided to the interest points of the software project. Indeed, as these measurements allow to gather global and/or more in details information on the state of the evaluated software at time t , hence based on this information, we can determine the interest points and so the next evaluation. For that, we suggest a measurement plan specific to the interest points at time t . This step corresponds to the suggestion phase of our approach.

The second objective is to perform a simultaneous analysis of the whole measurements. Instead of analyzing each measurement one by one, we propose to analyze all the measurements gathered at the same time t . From a global continuous evaluation of the system during a period of time t , a specific analysis on the state of the software is returned. This step corresponds to the analysis phase of our approach.

In order to reach both objectives by considering the context challenges which are: the amount of data to supervise due to the complexity of the current systems; the need of an analysis and a suggestion at runtime to have a supervision in accordance with the software needs; and thus a real need of an adapted engineering process supervision, we propose an approach using supervised learning technique. The advantages of using learning technique is that allows to build a tool able to analyze a big amount of data and that automatically and at runtime. Indeed, we just have to train our analysis tool on a supervised or manual analysis model corresponding to the needs. Moreover, the involvement of the project expert is thereby reduce to the initialization of the measurement context which reduces the management cost.

This approach is built into three procedures : the manual elaboration of the analysis model; the automated and dynamic analysis based on the analysis model; and the suggestion of measurement plan based on both latter. And in two main stages: the initialization phase which consists to define the measurement context and the interpretations of the measurements; and the computation phase which includes the measurement analysis and the suggestion of metrics. The first phase is unique and manually fixed at the beginning of the measurement process while the second one is dynamic and in continuous. This latter follows up a temporal cycle: at each period of time t_i of the full measurement process, the analysis procedure is proceed on the measurements gathered during the interval of time $[t_{i-1}, t_i]$ followed by the suggestion. Then, the analysis and suggestion proceed at the next period of time t_{i+1} on the measurements gathered on the next interval of time $[t_i, t_{i+1}]$ and so on... until the last period of time t of the measurement process. Thus, we have the initialization fixed at the time t_0 and

the other computations in continuous and according to a temporal cycle t_i where $i \in \{1, ..n\}$ and n the number of period of time composing the supervision time t .

The following sections describe the different procedures in details.

4.1 Manual analysis model

The analysis model is the basis of our suggestion algorithm. The model is manually elaborated by the experts and defines the measurement context : what is observed and how it is observed by determining the software properties, the corresponding set of metrics and the mandatory metrics, the ones that must always be in the suggested measurement plans. It is defined at the beginning of the measurement process as input to our approach.

4.1.1 Initial measurement plan

The initial measurement plan is the definition of the measurement context by identifying the software properties to be analyzed and the set of metrics necessary to evaluate these properties.

Software properties

The software properties correspond to what should be observed during all the measurement process. They correspond to the software characteristics we want to evaluate. The measurement plans are oriented according to the state of these properties at this time t_i . At each new measurement cycle, one property (the one of interest) will be evaluated more in details than the others as this latter does not show any interest at this moment t_i .

We formally define the set of software properties y as:

Definition 4.1.

$$y = \{y_1, ...y_l\} \quad (4.1)$$

where l is the number of considered properties and y_k a unique property.

Thus, these properties are the basis of the organization of the model in a mapping system which defines the correlations between metrics and properties (see the Figure 4.1). The model is so built over these properties.

Set of metrics

The set of metrics groups all the metrics that could be computed during all the measurement process. These are the ones for measuring all the considered software properties. Thus, the suggestions will be subsets of this set of metrics.

This set is cut into subsets of metrics and each subset groups a list of metrics that give information on the same property. That means that there is the same number of subsets of metrics as properties. Each property is associated to a subset of metrics.

These subsets of metrics are determined for the suggestion phase. They define how to orient the measurement process to the property of interest.

We formally define the ordered set m such as:

Definition 4.2.

$$m = \{m_1, \dots, m_n\} \quad (4.2)$$

where n is the number of considered metrics and m_i a unique metric.

4.1.2 The mandatory metrics

The mandatory metrics are the metrics that should be always in the measurement plans. There are determined by the experts by considering the fact that the measurement gathering of the metrics not suggested are stopped. As the next measurement is reduced to the property of interest. It is necessary to avoid the risk of losing all information about the other properties by ensuring a continuous evaluation of all properties.

In order to avoid any loss of critical information necessary for a continuous and relevant measurement process, we define a set of mandatory metrics that cannot be removed from the next suggested measurement plan and thus the gathering of the corresponding information should be continuously performed. This set of metrics is fixed and it cannot be modified during the full measurement process.

These metrics are chosen so that give at least one essential information on each property. In order to have always the relevant minimal information on each property.

4.1.3 The mapping system

The mapping system consists to define the links between set of metrics and property. This mapping is one of the basis for the suggestion. The expert defines how a property will be evaluated more in details. Indeed, the set of metrics linked to the property of interest will be added to the next measurement plan. And the set of metrics related to the other properties will be removed from the next measurement plan except the mandatory ones.

The mapping aims at allowing an automation of the suggestion procedure by regulating the measurements gathering and at suggesting relevant and effective measurement plans.

We formally define the mapping system C as:

Definition 4.3.

$$C = (m_i, y_k) \quad (4.3)$$

where m_i is the metric i and y_k the property k . Some metrics are common to several properties while some metrics are specific to a single one. These correlations are used by the suggestion step (cf. Section 4.1.3).

To summarize, the analysis model is manually performed and only once as basis of the analysis and the suggestion procedures: the properties, the general set of metrics and the

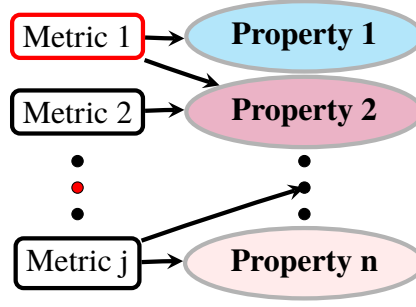


Figure 4.1 – Our mapping system

mandatory metrics are used for the analysis and the mapping system is used by the suggestion. It defines the measurement context during all the process : what parts of the system should be monitored (properties); how they should be monitored (mandatory metrics); and how each of them should be explored (mapping).

As illustrated in the Figure 4.1, there is some number of different properties and metrics, and each metric is associated to one or more properties. The property are not disjointed, they can share sub-characteristics. Thereby, they can share several metrics which give the information on these sub-characteristics. And, some metrics are defined as mandatory, herein encircled in red. These metrics are well chosen to have from a minimal information an indicator on the state of all properties. For instance, we will prefer atomic metrics rather than complex metrics and/or one metric which gives information on several metrics. In this example, we choose the metric 1 which is associated to two properties etc.

Finally, we formally define the analysis model mp as a set of metrics m , properties y and their correlations c :

Definition 4.4.

$$\begin{cases} m = \{m_1, \dots, m_n\} \\ y = \{y_1, \dots, y_l\} \\ c = (m_i, y_k) \end{cases} \quad m_i \in m \text{ and } y_k \in y \quad (4.4)$$

Once, our measurement basis is well defined with the analysis model, we can now initiate the explanation of the analysis procedure.

4.2 Data analysis from measurements

The analysis procedure aims to analyze simultaneously several measurements gathered during a period of time t and to dynamically determine the metrics of interest at time t . Herein, that consists to analyze simultaneously all the metrics defined in the initial measurement plan of the analysis model and to prioritize the measurements that show an interest at the time t of the supervision.

The main principle consists to analyze a set of metrics as a single data. That means, the set of metrics of the measurement plan are analyzed as one single vector of measurements, all

gathered at the same time t . The result of the analysis is thus returned as a single information on the system.

This information is the interpretation of the measurements as an interest indicator about one property defined in the previous model. This interpretation is built through a classification process: according to the set of values in the vector, this one is classified in one category. This process of classification tends to imitate the interpretation of the expert. In fact, it is based on the ones of the expert via a file simulating the different types of possible analysis.

This file, called training file, aims to train an SVM classifier used as an analysis tool. As SVM is a supervised learning technique, so it needs a basis model in order to learn on this latter. Learning on a basis allows to have an independent (automated) analysis tool while being in accordance with our needs.

Our analysis process is based on the classification of set of vectors of measurements and on the dynamic selection of interest metrics. This latter is based on the classification result and on the trained classifier to determine which metric(s) has/have the most impact on the classification. For that, we use the RFE algorithm, described above. The results of both are used by the suggestion procedure.

4.2.1 Classification

The classification is the first process of the analysis procedure, it aims to classify a set of data into categories then to return the category with the most data classified. These data to be classified are in the form of vectors. Each data is a multidimensional vector and its size is defined at beginning by the analysis model.

We formally define a vector $\vec{v}$ as:

Definition 4.5. A vector $\vec{v}$ is the analyzed data model. This is a set of values of different metrics. Each field of the vector is a value x_i of a specific metric executed at time t . Thus, a vector contains a set of information on a software at time t and it is defined as:

$$\vec{v} = \{x_1, \dots, x_n\} \quad (4.5)$$

Where n is the number of metrics in the analysis model.

These metrics give information on one or several software properties. This correlation between software properties and metrics is herein used for the suggestion of measurement plans.

Features & Metrics

The vector $\vec{v}$ is composed of the results of the metrics defined previously. Each field of the vector corresponds to the result of one metric. Moreover, each field corresponds to the result of a single one metric and it is always the same but gathered at different moment.

Thus, we formally define a field of a vector of measurements called feature as :

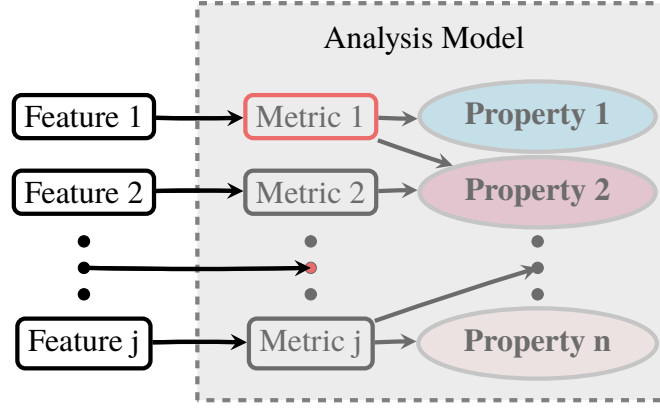


Figure 4.2 – Correlation between features and analysis model

Definition 4.6. A *feature* is a field of a vector. It refers to the metric i associated to the field i of a vector $\vec{v}$. A feature is unique. The place of metrics in the vector is fixed. So a feature is a value x_i of the metric i in a vector $\vec{v}$.

That means that each feature corresponds to a result of one metric and the ordered place of this metric in the vector is always the same. All the vectors analyzed will have the same organization. The difference is the value x of the fields which correspond to the results of their reference metrics at a time t .

As illustrated in the Figure 4.2, there are as many features as there are metrics and each feature refers to one metric in the vector.

Property & Classes

As explained previously, the classification consists to classify vectors into categories. These categories are herein called classes and formally defined as :

Definition 4.7. A *class* is a cluster of vectors. It refers to a group of vectors with close values. In a broader sense, it refers to a group of vectors providing the same information type on the software and is defined as below:

$$class = \{\vec{v}_1, \dots, \vec{v}_p\} \quad (4.6)$$

Where p is the number of vectors classified in the class.

These classes are determined by the analysis model. In fact, each category (class) corresponds to one property. That means that there are as many classes as there are defined properties and each class refers to one property as illustrated in the Figure 4.3.

To sum up, the feature and classes are concepts in the analysis point of view. A feature refers to the result of a metric at a time t . A class refers to a property in the sense that all the vectors giving interesting information about the same property are classified in the same class.

Interpretation

The interpretation consists to classify a set of vectors $\vec{v}$ on a class as illustrated in the

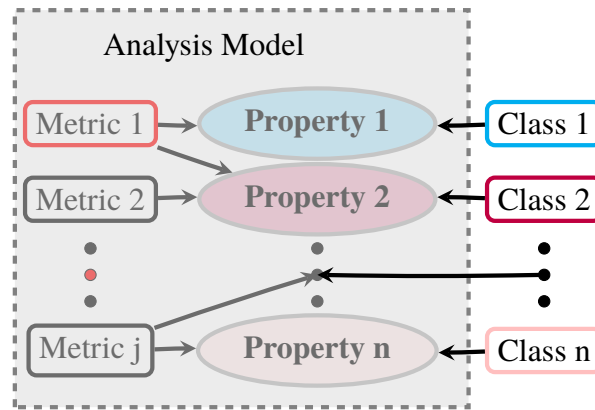


Figure 4.3 – Correlation between classes and analysis model.

Figure 4.4. The distribution of the vectors in the different classes is based on the values of the vector. In fact, the vector is a carrier of information on the supervised system. And according to the values of the vector field, the information is different.

As the features of a vector refer to the results of the metrics defined for measuring the software, their values inform about the properties. If the features corresponding to one property show an interest while the others features does not show any interest, then the vector will be classified in the class corresponding to this property and so on for each vector.

Each vector is a set of information on the software at time t_i . That means the first vector $\vec{v}_1$ is composed of values gathered at the time t_1 and the second $\vec{v}_2$ at the time t_2 and so on, and the last one $\vec{v}_n$ at the time t_n such as the set $\{t_1, \dots, t_n\}$ are period of times during the entire measurement process. The classification result is then a distribution of set of vectors of measurements gathered during a period of times t_i .

Finally, this distribution will be used to determine the property which needs more interest to be studied and thus to refine a little more the monitoring of this property while obviously keeping the fundamentals information on the other properties.

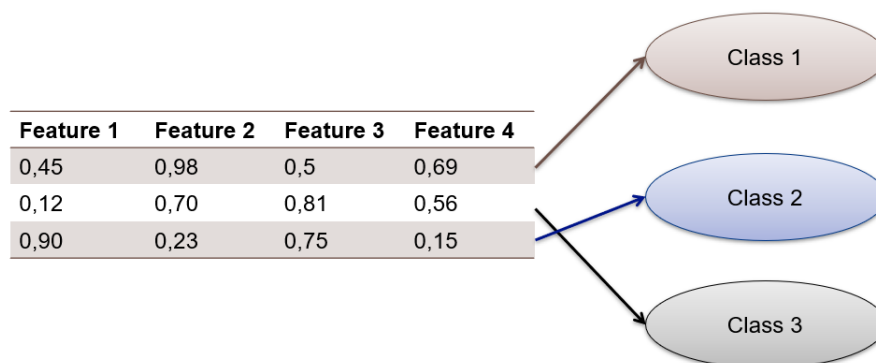


Figure 4.4 – Example of classification

Actually, the classification consists to determine the interesting property from a set of information (set of vectors) by distributing the data in categories. But, as this task is very costly in time and it needs the intervention of the experts of the system, we want to automate this task. For that, we propose to use a supervised learning technique SVM which will learn

on a classification of set of data done manually by an expert. Then from this training, we will have an automated classifier specific to our context which can imitate the experts and will allow to automate this analysis procedure and thus significantly reduce the management cost.

In the next section, we describe the supervised learning technique SVM used to automated this procedure. There are many supervised techniques, but for this work we focus on the classification one as it corresponds to our need: classify a set of measurements according to their values in order to orient the measurement process. Furthermore, among the set of classification algorithm existing, our choice fell on the Support Vector Machine (SVM) one. The reasons of this choice are that it supports a large dimensional data while keeping its performance and it does not need a lot of data for the training (Kotsiantis, Zaharakis, & Pintelas, 2007).

4.2.2 Supervised learning

Many algorithms of machine learning are focus on finding a function that is able to reproduce an approximation of the problem. The problem is to find the function that has the best accuracy when it is used to make a prediction. Supervised learning techniques are based on a training data, that it is a collection of vectors and labels and using these pairs to find a function that can predict new inputs. The algorithms that use a training set to infer the approximation function are considered supervised learning because the algorithm uses the training data to make predictions and this data guides the algorithm to a function with an acceptable level of accuracy based on the training data. In supervised learning systems we can find spam classifiers, face recognition systems and medical diagnosis systems.

4.2.3 Classification technique

On machine learning, classification is the problem of identifying the category to which a new observation belongs to. The categories are determined by a training set with observations whose category is known. A classifier, an algorithm that optimizes the classification based on the training set, is used to predict the category of a new instance. In machine learning, the observations are usually refereed as instances and the categories as classes, each class is associated with a specific label. The objective of a classifier is to be able to use an instance's characteristics to determine the class it belongs to. Classification is one of the problems where machine learning has been successfully applied.

Linear classification

A linear classifier makes a classification decision based on the value of a linear combination of an instance's characteristics to determine the class the instance belongs to. A linear classifier functional form is a hyperplane that can separate the training data in the desired classes. A

separating hyperplane is defined by a set of points $\mathbf{x}$ that satisfy:

$$\mathbf{w}^T \mathbf{x} + b = 0 \quad (4.7)$$

where $\mathbf{w}$ and b are parameters that control the function. For example, suppose a binary classifier, this classifier separates the training data in two classes with the following labels $\{-1, +1\}$, the classification rule for an instance x_i is to assign -1 if $\mathbf{w}^T \mathbf{x}_i + b < 0$ and $+1$ if $\mathbf{w}^T \mathbf{x}_i + b > 0$. The Figure 4.5 is an example of a binary classifier, the separating hyperplane splits the space and is the decision boundary for classifying instances into one of the two classes.

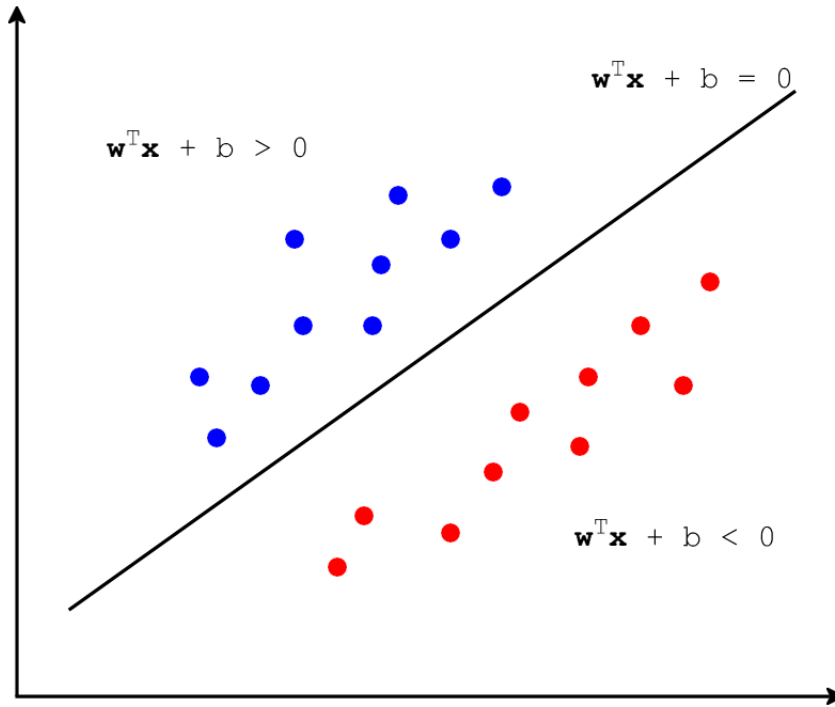


Figure 4.5 – A separating hyperplane controlled by $(\mathbf{w}, b)$ for a two-dimensional training set.

Non-linear classification

In real world applications linear separation of the data may not be possible, these cases require more complex solutions to find an appropriate separator. To solve this problem non-linear separators are required. But one strategy used by machine learning, when linear separation is not possible in the original space, is to change the representation of the data. This is done by mapping the input data in space X to a new space F :

$$\phi : X \rightarrow F \quad (4.8)$$

The objective of mapping data to a different space is to find a space where the linear separation of the data is possible and use the linear methods for classification in the new space as illustrated in the Figure 4.6).

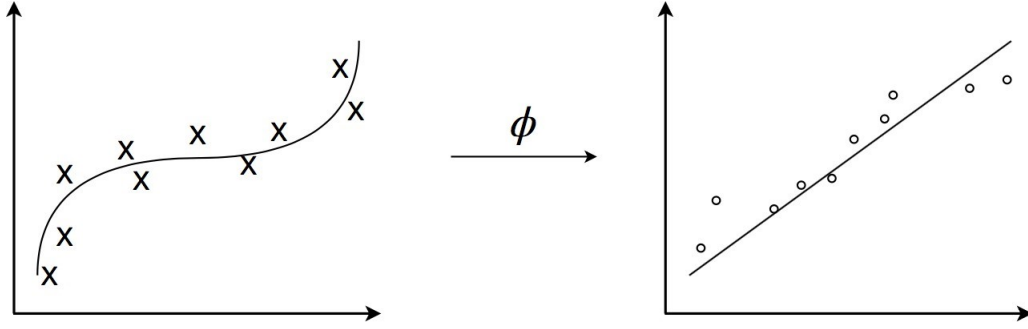


Figure 4.6 – Mapping the input data in space X to a new space F .

4.2.4 The Support Vector Machine

Support Vector Machine (SVM) is a supervised learning method used for classification, SVMs are well established and studied classifiers. Consider SVM as a binary classifier. A SVM classifier objective is to find a hyperplane that can separate the the training data in two sets corresponding to the classes, but there are often multiple hyperplanes that can separate the training set (Figure 4.7). The SVM algorithm picks the hyperplane that maximizes the distance between the hyperplane and data point on both sides, this distance is called margin. The objective of the algorithm is to maximize the margin. The points closest to the SVM are called support vectors and the classifier is called a support vector machine because the hyperplane only depends on the support vectors (see the Figure 4.8).

The aim of SVM is not to replicate the classification of the training data, the aim is to be able to predict the class for new instances. How well a model predicts the right class for new instances is called generalization. For classification problems the objective is to find a model that has a good generalization.

A support vector machine (SVM) (Vapnik & Vapnik, 1998) is a linear classifier defined by a separating hyperplane that determines the decision surface for the classification. Given a training set (supervised leaning), the SVM algorithm finds a hyperplane to classify new data. Consider a binary classification problem, with a training dataset composed of pairs $(x_1, y_1), \dots, (x_l, y_l)$, 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 4.9 defines a separating hyperplane

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0 \quad (4.9)$$

where $\mathbf{w} \in R^n$ and $b \in R$ are parameters that control the function. Function f gives the signed distance between a point $\mathbf{x}$ and the separating hyperplane. A point $\mathbf{x}$ is assigned to the positive class if $f(\mathbf{x}) \geq 0$ 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 the equation 4.10 (Betancourt, Morerio, Marcenaro, Rauterberg, & Regazzoni, 2015), this

problem maximizes the margin between training points.

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1, i = 1, \dots, l \end{aligned} \quad (4.10)$$

All training examples label -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 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*.

In real world problems, the data usually can only be separated using a non linear decision surface. In these cases the input data is mapped to a higher space where the linear separation of the data is possible. Kernels allow to work in a higher space without explicitly mapping the data to the new space. A commonly used kernel is the radial basis function (RBF) kernel (Vapnik & Vapnik, 1998). This kernel is very popular and often delivers a good performance. The RBF kernel introduces additional parameters that influence the performance of the classifier.

Choosing the appropriate values for parameters is important for the performance of the classifier. These values are usually found with grid search and cross validation. Grid search performs an exhaustive search over specified parameter values for a classifier. With grid search, the optimal parameters can be obtained (Yang, Li, Zhang, & Wang, 2016). Each combination of parameter values is used to train a classifier and then the classifier is evaluated to determine the best combination of parameter values. The evaluation of the classifiers is done with k -fold cross-validation. k -fold cross-validation (Li, Salman, Test, Strack, & Kecman, 2013) is a technique to evaluate the accuracy of a classification model. In cross-validation, the training set is divided into k -folds and the classifier is refitted k -times. Each time the classifier is refitted, a fold is excluded from the training set and used as a test set to evaluate the accuracy of the classifier. During the evaluations the misclassifications are accumulated to produce an estimate of the prediction performance of the classifier. With grid search and cross validation the parameters values are determined for the SVM.

Training

The main interest of the classifier is the classification of unseen data. To evaluate the generalization of the model, a performance measure is needed. We define the performance of the model as the percentage of samples that were correctly classified in a known data set. Because of this, two sets are needed when training a SVM, a training set and a test set. The training set is used for training and the test set is used to calculate the performance of the trained model.

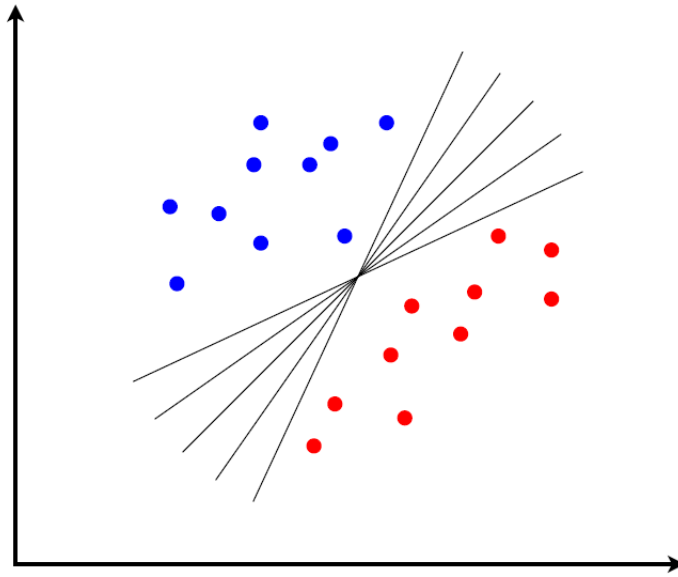


Figure 4.7 – Multiple hyperplanes.

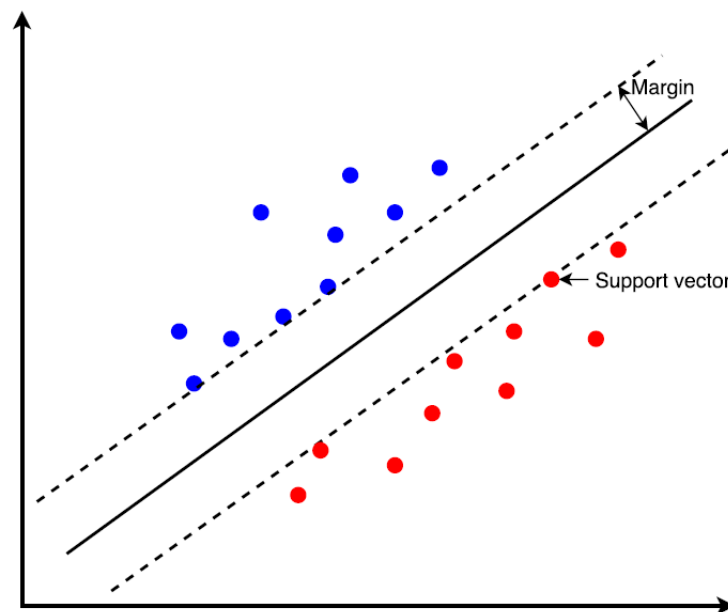


Figure 4.8 – Linear support vector machine.

Multi Class classifier

Until there, we considered SVMs as binary classifiers, but some problems require classifying instances into one of the more than two classes. Using SVM there are two approaches to construct a multi-class classifier: one-against-the-rest and one-against-one. Suppose a classification problem with k classes.

The first approach One-against-the-Rest consists of training k classifiers, one SVM per class, each SVM is trained to distinguish the instances from one class, from the instances of all the remaining classes. On the other hand, One-against-One consists of training $\frac{k(k-1)}{2}$ classifiers, one SVM for each pair of classes, each SVM is trained to distinguish the instances from one class from the instances of another class.

The One-against-the-Rest approach shows a performance that is good in comparison with One-against-One approach, but the second method has shorter training times, making it more suitable for practical use. In One-against-One, the training set for each classifier is usually smaller compared with the training set of a classifier in One-against-the-Rest, that is why usually the training time decreases using One-against-One approach.

Gaussian kernel SVM is a linear classifier, but in real world applications, linear separation of the data may not be possible. We know that we can map the input data to a new space where a linear method can be used to find a separator, but we need to know the mapping function. This can be solved using a Kernel method, with a kernel there is no need to know the mapping function, a kernel allows to work in the new space without computing the representation of the input data in the new space. This approach is known as the kernel trick. The kernel trick has been successfully applied in SVM, this has allowed the use of SVM to tackle problems where linear separation is not possible in the original space. Several kernels are used with SVM. For example linear kernel, polynomial kernel and Radial Basis Function (RBF) kernel. We are going to focus on RBF kernel, this kernel has proven to be a good choice to solve non linear problems (Deng, Tian, & Zhang, 2012). Another reason is that this kernel only has two hyper-parameters that influence the model, C and γ . When the SVM is implemented with RBF kernel, the two hyper-parameters determine the generalization capabilities of the classifier.

There are two things that need to be avoided when training a model. First is to avoid over-fitting, this occurs when the model identifies relationships in the training data and these relationships do not hold in general. The second thing to avoid is under-fitting, this occurs when model is too simple for the training data, all the training instances are classified into the majority class.

If γ is too large over-fitting occurs and when γ is very small under-fitting occurs. A small value for C will cause under-fitting, on the contrary when C is too large over-fitting occurs. The appropriate value of C and γ determine the robustness of the model.

4.2.5 The dynamic mandatory metrics selection

The second process of the analysis is the dynamic selection of the mandatory features. This process aims to determine the features which are necessary for the classification process. It is based on the classifier and the result of the classification of all the data. Thus, this allows to increase the reliability of the measurement process by ensuring the minimum loss of significant information and the sustainability of measurement cycles by having at each cycle an information on all measured properties, a set of metrics should always be gathered.

In order to proceed the selection, we use the Recursive Feature Elimination (RFE) algorithm described below.

RFE

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 the classification process.

The feature selection method herein used is the Recursive Feature Elimination (RFE) (Khalid, Khalil, & Nasreen, 2014). 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.

To sum up, the analysis procedure aims to detect the property which needs more attention than other during all the supervision process and to highlight the metrics, that give noticeable information.

This procedure is based on the previous one and on the training file for the initialization of the classification process and both are defined by the expert at the beginning of the measurement process while the analysis (the classification and the selection) are continuously processed and at runtime.

The results of this procedure are basis of the suggestion one which is described below.

4.3 The Suggestion

This procedure aims to dynamically generate a measurement plan adapted to the software needs at the period of time ti . The objective through this process is to have a flexible measurement plan during all the measurement process and thus focus the measurement on the real needs and reduce the measurement cost that does not show any interest at this time.

This measurement plan only consists of metrics defined in the analysis model. Its composition varies with the results of the analysis. In fact, according to this latter a subset of metrics is suggested. The suggestion is based on the analysis results: the classification and the selection. From the classification the link with analysis model is used to determine a part of the suggestion while the rest is determined by the selection

4.3.1 Analysis results

The analysis return as results (i.e., Section 4.2: a classification of vectors into classes and a set of metrics selected as the metrics that influenced the classification result.

The result of the classification is then a number of clusters with a numbers of classified vectors. These vectors have been classified into these categories according to the highlighted property by the values composing these vectors. And each category refers to a measured property. Thus, we use the result of the classification to determine which property shows an interest at the time t_i . For that we consider that the category with the most number of classified vectors represents the property of interest. And thus, the next measurement plan will be built in order to focus the next measurement cycle on this property.

In addition to the classification result, the analysis return a result of the selection of features. This result is composed of the set of features which influenced the classification. In other words, the metrics that have been necessary to have this type of classification. That means, that the values of these metrics have impacted the classification. And herein, we use this information to prioritize the metrics: the selected features have significant values at time t_i which will orient the interpretation of the vectors. Thus, we consider these values as indicators of interest on what the corresponding metrics evaluate. Thereby, these metrics are kept in the new measurement plan.

Finally, from the analysis, we get the property on which focuses the next measurement cycle and the metrics with "critical" values which shows an interest to be monitored.

4.3.2 Use of the analysis model

From the analysis results, we get the information on the software at the time t_i . The last step consists to generate a measurement plan adapted to this state, in order to orient the next measurement cycle on the software needs at this time without loss of any other critical information.

For that, we use the analysis model which defines the set of mandatory metrics, that should always be gathered to avoid the loss of information. And the mapping system between properties and set of metrics to determine the subset of metrics that will allow to focus the measurement on the property of interest, the one which the associated category is the one with the largest number of vectors classified. And the other subset of metrics, the ones associated to the other properties.

4.3.3 The suggestion algorithm

Once all the interesting elements are highlighted by the previous procedure, the new measurement plan can be generated as a subset of metrics which allows to gather more information on these highlighted elements. This makes it possible to orient the next measurement cycle on specific parts of the measured system at runtime and dynamically. In other words, the

measurement process is no longer static and fixed on the same elements all the time t but flexible according to the needs at each period of time t_i . The measurement plan mp_i is so performed by generating a subset composed of the set of metrics associated to the property highlighted as the one of interest by the analysis, the selected metrics by the RFE algorithm and the mandatory metrics defined in the analysis model. If all the vectors are classified in the same class, we thus suggest all the metrics defined in the analysis model.

This procedure is formally described below, by the Algorithm 1. It takes as input the initial measurement plan, herein called mp , the trained classifier f and the set of vectors to be analyzed $\{\vec{v}\}_i$, gathered during the interval $[t_{i-1} - t_i]$. And where mp_i is the suggested measurement plan, mm the defined mandatory metrics and fs the feature selection algorithm (RFE).

Algorithm 1 Metrics Suggestion

```

Input  $mp, f$ , unlabeled  $\{\vec{v}\}_i$ 
1: Output  $mp_i$ 
2:  $y \leftarrow \{\}$ 
3:  $mf \leftarrow \{\}$ 
4: for each  $\vec{v}^i$  in  $\{\vec{v}\}_i$  do
5:    $y \leftarrow f(\vec{v}^i)$ 
6: end for
7: if  $y == 0$  or  $y$  without duplicate == 1 then
8:   return  $mp$ 
9: else
10:   $mf \leftarrow fs(f, \{\vec{v}\}_i)$ 
11:   $mp_i \leftarrow mf + mp[most\_common(y)] + mp[mm]$ 
12:  return  $mp_i$ 
13: end if
  
```

To summarize, we propose an analysis of measurements and a suggestion of metrics approach, called the metric suggerter tool, build in 3 procedures: the configuration phase, manually done, initialize the analysis model and training file; the analysis phase, composed of the classification and features selection processes; and the suggestion phase that suggests a new measurement plan based on the analysis results and the mapping system of the analysis model. See Figure 4.9

Through this method, the measurement load is increased only on needs and decreased on less interesting properties. This suggestion approach allows to reach a lighter, complete and relevant measurement during the entire supervision period of the software project.

4.4 Application

This approach has been implemented and the corresponding framework was the object of some experiments. Moreover, it was integrated in an industrial platform (i.e., Section 3.3.1), the one of the European MEASURE project, as an analysis tool. The tool is implemented as a web application and its industrial integration is done through calls to the MEASURE platform

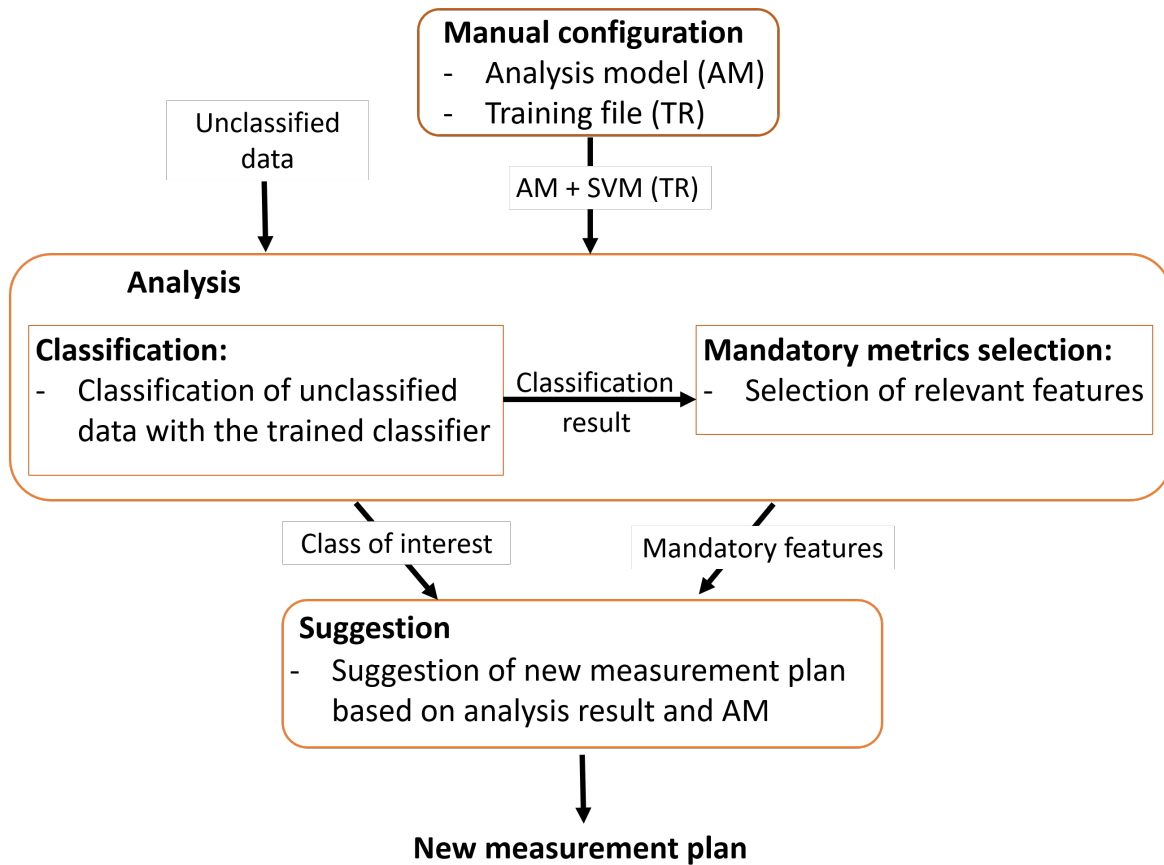


Figure 4.9 – Metrics Suggester process overview.

API. All the details of the architecture, the integration and the experiments are described in the following sections

4.4.1 Metrics Suggester tool architecture

our analysis and suggestion approach, called Metrics Suggester, is built as a web application. The architecture, as illustrated in the Figure4.10, is organized around the machine learning unit (ML tool), which regroups the classification and feature selection algorithms used for the analysis procedure.

Implementation

Regarding the implementation, it is done with the following frameworks, libraries and applications:

- The library used to develop the learning algorithms SVM is Scikit-learn (Pedregosa et al., 2011) and their implementations of classifiers.

Our interest is in two types of SVMs provided by the library. First is SVC, a classifier that supports various kernels (linear, RBF for example) and multi-class support. SVC is the classifier used to predict the classes of unclassified data. The second type is

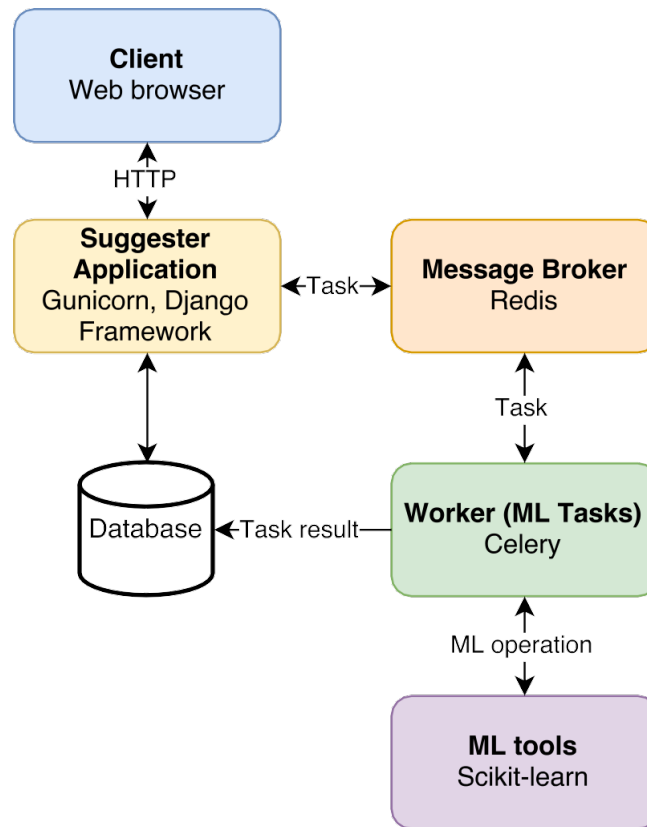


Figure 4.10 – Metrics Suggester tool architecture.

LinearSVC, a linear classifier with multi-class support and it is used during the feature selection process.

- The feature selection algorithm RFE is performed with Scikit-learn which provides the implementation and LinearSVC is the estimator used to determine the weight of the features.
- the task scheduler module is needed for long tasks such as training a classifier and also to schedule periodic tasks, we use it to schedule the suggestion process. Celery¹ is used for tasks scheduled combined with Redis² as message broker to exchange results between our application and the scheduler.
- The web server framework is Gunicorn³, it is used to host the application, the suggester application is built with the Django⁴ framework and PostgreSQL⁵ as management system for the database.

¹<http://www.celeryproject.org/>

²<https://redis.io/>

³<https://gunicorn.org/>

⁴<https://www.djangoproject.com/>

⁵<https://www.postgresql.org/>

Services

The features of the application are accessible in two ways: from calls of an API rest and from web pages, called views.

The API was created to make the tool more responsive. It regroups the features that allow to get the data used by the tool and displayed in the different views (heatmap, bar chart, line chart). Here is a detailed list of the services:

- `/api/files/unclassified/`

This endpoint is used to upload a file with the gathered measurement to be classified to the tool. This is the interface to receive new inputs. The data must be in svm-light format to be understandable by the tool.

- `/api/svcmodels/id/data/heatmap/`

Returns the data to construct a heatmap. The data is formatted for the Plotly library as a JSON object. This endpoint is intended to be consumed by an asynchronous request, this could take more than 500 ms to fulfill the request.

- `/api/featureselection/id/data/scores/`

Returns the data to construct a chart that shows the cross validation scores obtained for each subset of features during the computation of the RFE algorithm. The data is formatted as a JSON object.

- `/api/plan/family/id/predictions/`

Returns the data needed to construct a chart of the progress of the predictions. Each point represents the result of execution of the suggestion process. This endpoint is used in the dashboard view (see Figure 4.11).

- `/api/prediction/id/`

Returns the data needed to construct a bar chart about the distribution of the classification. The data contains the classes and the number of predictions for each class. This endpoint is used in the dashboard view(see Figure 4.11) and in the prediction view.

Regarding the views, those are web pages that display some information of the main features of the tool as suggestion results, dashboard of charts, classifier information, analysis model. Here are the different views :

- `/view/dashboard/`

The dashboard view (see Figure 4.11) shows the latest information used, gathered and performed by the tool. Four things are displayed in the dashboard: the current classifier being used, the current measurement plan being analyzed, a graph with the predictions and a bar chart with the latest prediction by the tool.

- `/view/classifiers/`

The classifiers view displays a table with the classifier that the tool used in the past and the current classifier. Each entry has the classifier identifier, and with that classifier a more detail view of the classifier can be obtained.

- `/view/classifiers/id/`

This is the view to display information on a specific classifier defined by its id. That includes the parameters of the classifier and a heatmap of the cross validation results obtained by the classifier.

- `/view/plans/`

The plans view displays a table with all suggested measurement plans used during a process. Each entry is composed of the plan id, the number of features (metrics), the metrics, and the properties (classes).

- `/view/plans/form/`

This web page is used to initialize the tool by adding an analysis model. The measurement plan is formatted in the JSON format (i.e. Appendix B for the format), see Figure 4.12.

- `/view/plans/families/id/predictions/`

The predictions view presents a graph with the classifications of the data (see Figure 4.13) analyzed during each period of time t_i . The id is used to identify the predictions for a specific measurement plan.

- `/view/prediction/id/`

The prediction view displays a specific prediction (see Figure 4.14) identified by an id through an interactive bar chart with the distribution of the data into the classes.

- `/view/suggestions/`

The most important view of the tool, which displays a table with the results of each suggestion: the class of interest (the largest one), the number of data classified, the suggestion and the execution time of the analysis (see Figure 4.16).

4.4.2 Industrial integration

Our analysis tool Metrics Suggester is integrated in the platform, by using the REST API of the platform and its own. This latter allows to connect our analysis tool to the platform, to gather stored measurements and to generate a suggestion from an analysis based on these measurements and to display it by a dashboard from the platform as shown in the Figure 4.16 with two suggested measurement plans.

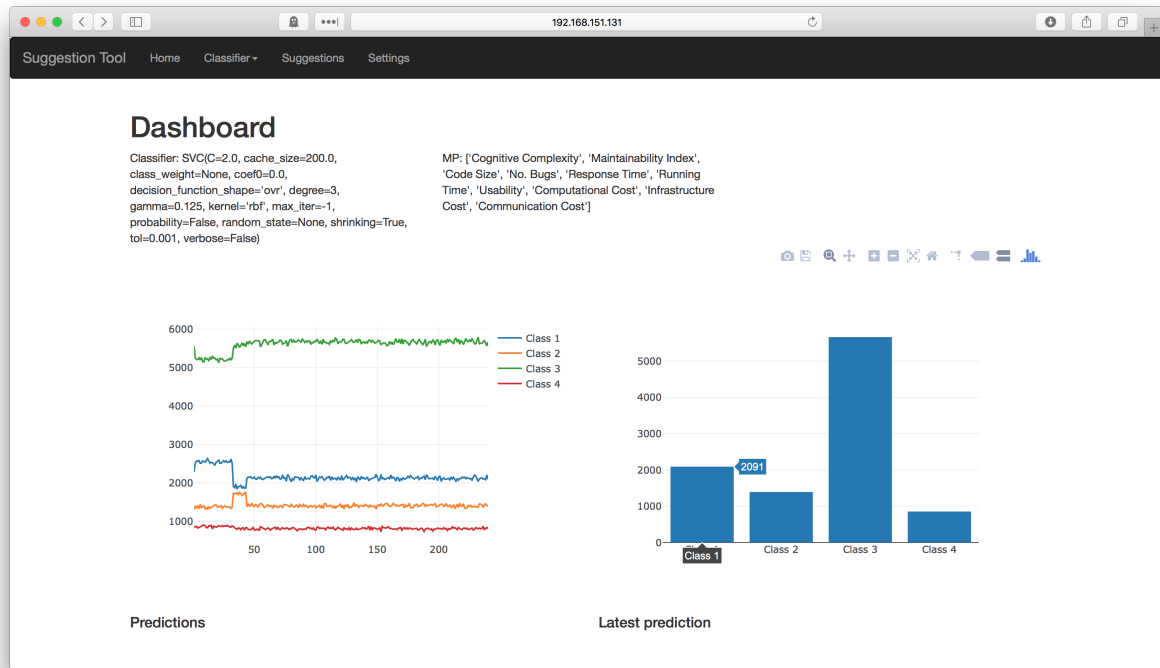


Figure 4.11 – Dashboard of the tool.

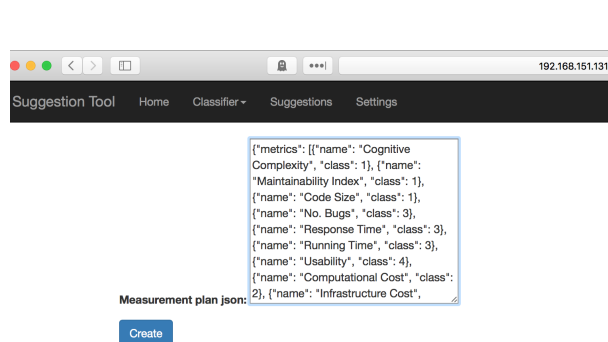


Figure 4.12 – Initialization of the analysis model page.

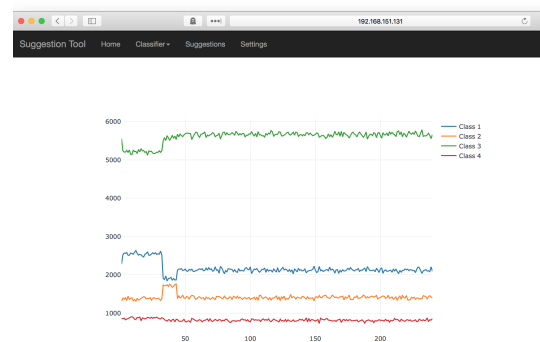


Figure 4.13 – Dashboard of classifications.

The integration steps consist to connect our tool to the platform in order to make it accessible from it as shown in the Figure 4.15, by using the API, according to the steps below:

- **Registration:** first the Analysis Tool must register itself to the platform using the registration service. This allows to make the tool accessible to the projects so that they can activate it.
- **Wait for Notifications:** the analysis tools must listen to notifications from the platform in order to know when a project request the usage of the analysis tool. The notification (Alert) system is based on pooling system. The analysis tools pool the platform periodically using the alert service to received notifications.
- **The analysis tool activating:** the analysis tools must configure the activating of tool

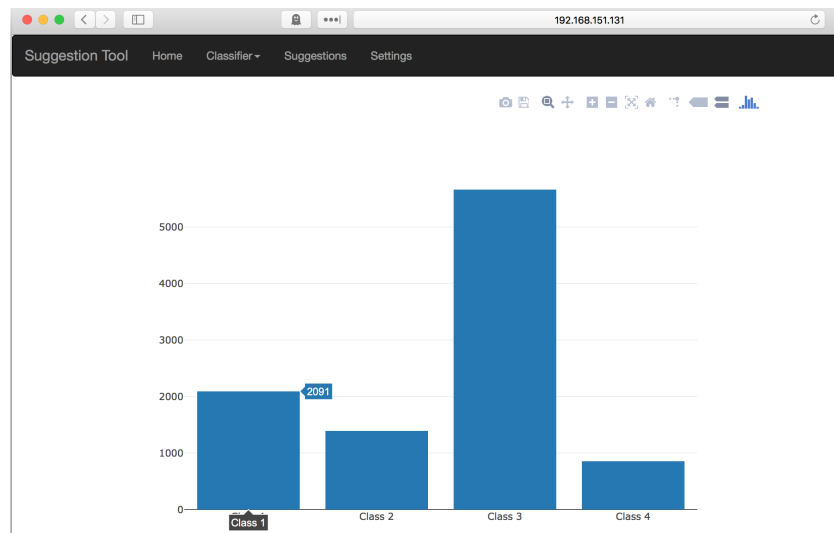


Figure 4.14 – Dashboard of the last classification.

from the platform by providing to the project the URLs of the tool corresponding to the configuration specific page, the main view of the tool and optionally the dashboard cards.

When configured, the analysis tool can start its analysis works for the specific project and from the platform. Thereby, the tool is remotely integrated by embedding its main services in the platform through calls to the corresponding API. As the platform accesses the analysis tool by its API, in the same way the tool can explore the project configuration using the various services provided by the MEASURE platform as the corresponding measurements stored in the platform database or the measurements activating/deactivating services, etc.(Section 3.3.1).

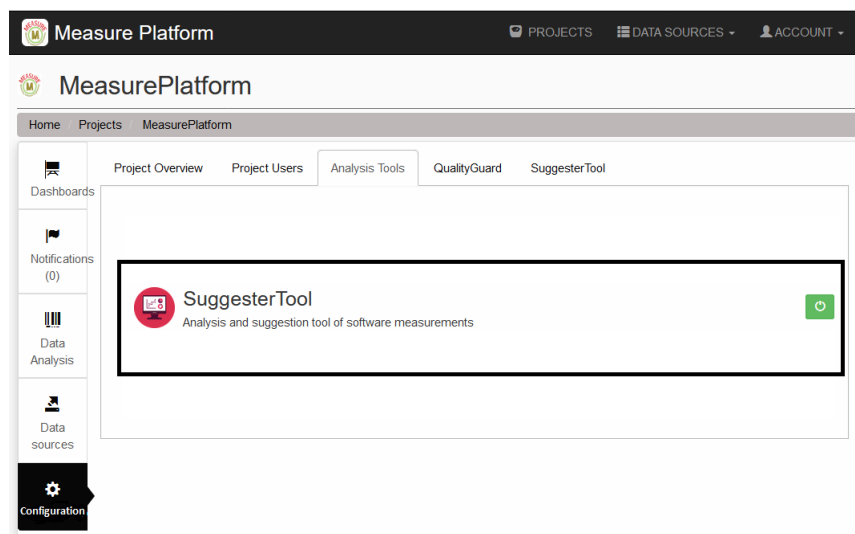


Figure 4.15 – Metrics Suggester tool integration.

ID	No.	Features	Classes	Metrics
2	14		Maintainability, System Performance, Performance, Functionality	Cognitive Complexity, Maintainability Index, Code Size, No. Bugs, Response Time, Running Time, Usability, Computational Cost, Infrastructure Cost, Communication Cost, Tasks, I/O Related Errors, Precision, Stability of Response Time, Illegal Operations
1	15		Maintainability, System Performance, Performance, Functionality	Illegal Operations, Stability of Response Time, Precision, I/O Related Errors, Tasks, Communication Cost, Infrastructure Cost, Computational Cost, Usability, Running Time, Response Time, No. Bugs, Code Size, Maintainability Index, Cognitive Complexity

Figure 4.16 – Display Metrics Suggester tool suggestions from the platform.

4.4.3 Experiments

In order to evaluate our approach, we manually built the datasets on which the analysis will be applied and the corresponding training file. For the suggestion, an analysis model is defined based on the ISO/IEC standard to determine the properties, the metrics and the correlations between them.

The objective is to analyze the effects of using the flexible measurement plan during the measurement process on the classification procedure and the features selection. In other words, how a controlled and automated information gathering has an impact on the supervision process.

Case study

The case study of this experiment is based on a dataset built manually which simulates real different possible scenarios. That means, the generated data is built so as to contain sub set of different type of vectors (5.1.2). We called a type of vectors, the ones with values which reveal a specific state of the software and which are interpreted by its classification in one property. All the properties are represented in the generated dataset but not equally, there is not the same number of each type of vectors. This dataset is divided into subset to simulate the period of time t_i . Each subset is thus analyzed, then a suggestion based on this latter is generated. Each sub set has its property of interest, that means the number of types of vectors interpreted by the property is in largest number. The objective is to see if there is a preservation of the significant information and if there is not a risk of convergence for the classification and thus for the suggestion and also if the classification and suggestion meet expectations.

4.4.4 Setup

We herein considered the following measurement plan which is determined by the experts. 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:

Table 4.1 – The analysis model of the measurement process context. And the index to each metric during the suggestion process.

Index	Metric	Property	Mandatory
1	Cognitive Complexity	Maintainability	
2	Maintainability Index	Maintainability	X
3	Code Size	Maintainability	
4	Bugs	Performance	
5	Response Time	Performance	X
6	Running Time	Performance	X
7	Usability	Functionality	X
8	Computational Cost	System Performance	X
9	Infrastructure Cost	System Performance	
10	Communication Cost	System Performance	
11	Tasks	System Performance	
12	I/O Errors	Performance	
13	Precision	Functionality	
14	Stability Response Time	Functionality	
15	Illegal Operations	Functionality	

- **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 used during the experiment consisted of the following 5 metrics: Maintainability Index, Response Time, Running Time, Usability, Computational Cost. These metrics were selected by an 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 with a number, in our experiments the number each metric was assigned is shown in Table 4.1.

Results

During the suggestion process 15 metrics (Table 4.1) 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 4.2 lists the plans and in which cycle they were used. The progress of the number of metrics in the measurement plan can be seen in the Figure 4.17. The number

Table 4.2 – 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 4.1.

	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

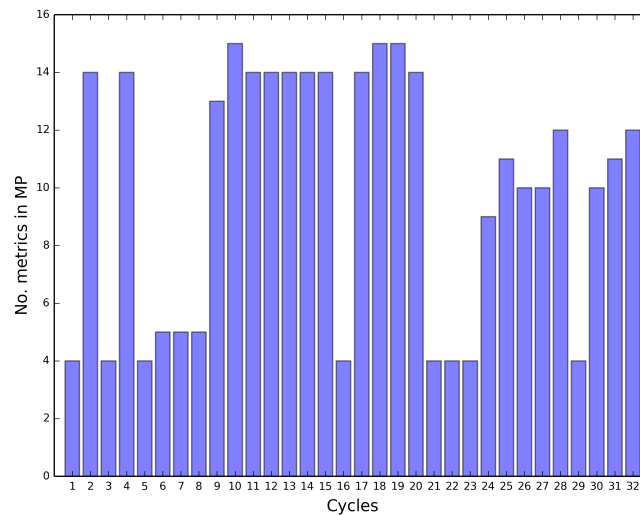


Figure 4.17 – Number of metric of the suggested plan in each cycle.

of metrics move between only 4 metrics and the maximum, a plan with all the metrics. The Figure 4.18 shows 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 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

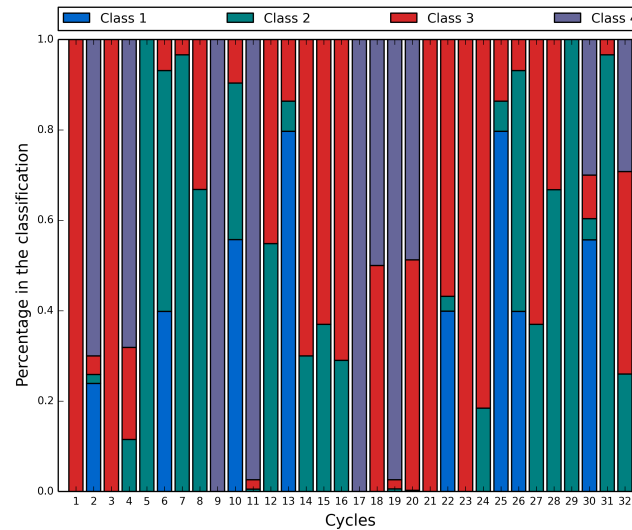


Figure 4.18 – Classification results of each cycle. The results show the percentage in the predictions of each cycles for the 4 classes.

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.

Discussion & Limitation

We design an expert-based automated measurement plan suggestion framework to add dynamic flexibility and an automated analysis to the measurement process. The learning technique SVM is used combined with the RFE algorithm to make the automation possible and to manage a huge amount of data with more lightness, flexibility.

Our methodology has been implemented and also successfully experimented on a real case study of our partner SOFTEAM-CADEXTAN (Section 5.3). The tool was able to manage a large data set and it has been integrated in an industrial platform, as analysis tool.

However, the training file, necessary to initialize the analysis tool, is elaborated by the expert and it corresponds to a manual classification. The expert classifies a set of vectors by labelling each vector by a class. Then, a classifier is trained according to this manual work.

Moreover, as the analysis is based on a specific context through a specific classifier trained by a file corresponding to the specific analysis model, each time we want to change the context of the measurement process, a new analysis model and training file should be done by an expert to generate the corresponding classifier. This is a major limitation of our approach. Despite an automated and "smart" analysis, our approach is still highly dependent to the experts and quite costly in time.

Our Solution

In order to reduce this load of the experts, we aim at improving this approach by using an unsupervised learning technique to generate automatically the training file. The advantage of using this latter is to reduce the expert cost, but also the dependency of an expert. Indeed, as the software measurement is an empirical science, it depends on the experience on the software or on the property evaluated. There is as many models as there are software projects. Thereby, our purpose is to use learning clustering algorithm X-MEANS as expert to generate automatically the training file according to a measurement dataset of the evaluated project. The expert would only intervene to define the analysis model according to the result of X-MEANS application. This improvement is the subject of the next chapter.

5

Automated initialization phase

Contents

5.1	The knowledge basis	77
5.1.1	Analysis model	78
5.1.2	Training file	79
5.2	Automated initialization phase	81
5.2.1	Unsupervised machine learning	81
5.2.2	Clustering algorithms	82
5.2.3	X-MEANS algorithm	84
5.2.4	Hybrid analysis model generation algorithm	85
5.3	Experiments	87
5.3.1	Automated analysis model generation	87
5.3.2	Suggestion results	89
5.3.3	Discussion & limitation	89

In our previous work, described in the Chapter 4, we conducted a research work to fully automate the generation of software measurement plans at runtime in order to have more flexible measurement processes adapted to the software needs. Indeed, in most real case studies, that process is fixed in a sense that the expert measures all what he can and not necessarily what he needs. Then a huge amount of data are unnecessarily collected and analyzed.

Our previous work has shown that measurement plans can be suggested and adjusted at runtime through a supervised learning-based methodology in reducing the amount of collected data. However, this approach is still dependent to the expert for the initialization step, especially for the elaboration of the training file. As a reminder, this file is used to train the classifier and it defines the correlations between measurements vectors and classes. The training file is manually done and thus, the cost time of this step is high when the samples to classify are highly numerous.

Software measurement is an empirical science which depends on the experience (Fenton & Neil, 2000). It is impossible to define a generic measurement analysis model. It depends on the software project, the used language, the used computer etc. Thereby, to evaluate a software, it is needed to know the context of the measured object, as well as, to analyze a software evaluation is needed to know the context. That is what makes difficult to automate a software measurement analysis.

So as to handle this lack, the idea is to learn from an historical measurements for generating an analysis model corresponding to the context. For that we propose to use a learning technique, which will learn from a measurements dataset of the evaluated software, as the expert does, and generate the corresponding analysis model.

The purpose is thus, to use an unsupervised learning algorithm to generate automatically an analysis model. From an unlabeled software measurements sample, a labeled one is generated. This output is then used as training file to train the classifier used for the analysis and suggestion steps.

For that, we will use a clustering algorithm to try to discover vector patterns by grouping in clusters the similar vectors of measurements. Then according to the clustering result, the expert will associate to each cluster the set of metrics to suggest (see the Section 3.2.1). Herein, the expert intervention only appears for determining the correlation between classes corresponding to the clusters, and set of metrics, which considerably reduces the expert load and the related time cost.

Thereby, in order to reach the third motivation : reduce the expert load, we aim at improving our previous work by increasing our efforts in the initialization phase of software measurement by introducing unsupervised learning algorithm. We specifically demonstrate the effectiveness of considering the clustering approach X-MEANS (Pelleg & Moore, 2002) to reduce further the management cost and improve the performance of such a process. We herein use X-MEANS to automatically discover the data patterns and generate correlations of a sample of data through clustering by generating a training file as input to our previous

suggestion approach.

Our objective by this improvement work is not only to reduce the time cost and the expert load by automatically generating a training file, but also to facilitate analysis model validation specific to software quality analysis and perhaps enable a more efficient future measurement process than in other older engineering systems, i.e, electricity, biology, physics, etc. Indeed, those have well-defined measurement plans like their analysis models allowing standard measurement protocol, effective and available to everyone.

This chapter is organized around the initialization phase which consists to define the knowledge basis necessary to a well measurement process. Manually built in our previous work and used to automated the analysis and suggestion phase. The new challenge is so, to automate the definition of the knowledge basis by learning on historical raw data of the observed software.

First, we will present in more details the composition of an analysis model to better understand the issues and needs to automate its elaboration, then we will introduce the notion of unsupervised learning technique to describe the algorithm X -MEANS used to reach our expectations. Finally, we will present experiments that demonstrate that there is a real interest in integrating an unsupervised learning technique for measuring software.

5.1 The knowledge basis

The initialization phase of our metrics suggestion approach, described in the previous chapter, consists to define the measurement context of a measured software.

This context formally describes what are the software realities to be observed, how to observe them and how to supervise these observations and how to monitor the information. So many information needed to automate the supervision process.

In our previous approach, this context was defined manually by the expert as an analysis model and a training file. The analysis model described what is observed and how to observe it and how to supervise the observation. while the training file describes the way to monitor the information.

Thus, this phase lays the foundation of the entire measurement process. It reunites the knowledge of the expert necessary to delegate to the machine this difficult and repetitive task which is the measurement supervision.

The main objective of this work is to delegate to the maximum the initialization of the knowledge basis to the machine. Up to now, this phase was manually done and the defined knowledge was used as basis to automate the supervision of the measurement. In order to achieve this goal, we describe in details this knowledge so that to grasp how to automate their definition.

5.1.1 Analysis model

The analysis model defines three fundamental points :

- what is observed : the observed realities,
- how to observe : the means to observe,
- how to supervise : the observation orchestration.

These three points are the basis of all observations and they are interrelated. But mostly, this is the basis necessary to be able to automate the measurement and the supervision.

We associate the first point to the class concept in analysis point of view, and to the property concept in a formal point of view. In the previous chapter we saw that a class and a property refer to the same thing, the observed reality, but according to the discussed context we do not use the same terminology.

The second point, the means to observe, refers to the metrics. Indeed, as explained in the Chapter 3, the means to get the information are the metrics.

Finally, the third point, the orchestration of the observation is defined by the mapping system. that we also addressed in the previous Chapter 4.

Our analysis model is thus composed of classes, metrics and a mapping system between these latter.

Class

The class concept is herein used as the grouping of different elements that refer to the same thing. Thus, by the notion of class, we refer to one software reality and to the different data that give an information on this software reality.

As these data are different, they give different information but on the same reality. In other word, they give different point of view of one reality.

For example, if we want to observe the ability of a man to be a runner. We have to observe several point : his endurance, his tall, his weight, his tone, his flexibility, etc. All of this information are different but they give an information on the same reality which is the ability to a man to be a runner.

Thereby, a class is a cluster of data on one software property. The property referencing the corresponding reality, such as a class C is a set of atomic reality or sub reality r_i with $i \in 0, \dots, n$ and n the number of sub realities necessary to describe the observed reality as formally defined below :

$$C = \{r_1, r_2, \dots, r_n\} \quad (5.1)$$

Thus, to defined a class it is important to know what characterizes the reality in order to well define the informants.

Metric

The metrics are the informants of the observed reality. They are the means to get the information on the reality. More precisely, it gives the state of the observed reality, at the time t .

Knowing that one reality is not totally distinct to another, they can share characteristics. Thus, a single metric can be used as informant for different reality. As shown in the article (Hovorushchenko & Pomorova, 2016), there are mutual influences between software properties defined by the ISO/IEC 25010 (ISO/IEC, 2010a). Herein, we use these metrics as mandatory ones in order to have, through one informant, an information on several properties.

To resume a metric is the function f that gets an information, or a state x on a reality r such as:

$$f(r) = x \quad (5.2)$$

Finally, it is useful to notice that, according to the value of x , the information does not have the same meaning. And this is on this rule that we will base the automation of the generation of the analysis model.

Mapping

The mapping corresponds to the correlations between what we want to observe and how we want to observe it. This is what we will use to orient the measurement process. This is the basis of the orchestration of the measurement process.

By this mapping, we define how to orient the measurement on a property. The needed information on the property. For that, we associate the property to the means that give the information corresponding to the ones needed.

Thus, the mapping *map* is the union between class with k parameters to be observed C^k and the set of metrics $\{f_i\}$ with $i \in \{1, \dots, k\}$ allowing to gather the information on these k parameters such as:

$$map = \{C^k \cup \{f_i\}\} \quad (5.3)$$

To sum up, the analysis model gathers the knowledge to orchestrate the measurement process: *what* and *how*. It remains to define how to orchestrate, how to orient the measurement process. And this is the role of the training file.

5.1.2 Training file

The training file describes how to analyze the information, how to interpret them. This is the analyst of the data about the observed software. According to the value x gathered the interpretation will be different.

The training file groups a set of vectors of metrics values associated to a label. Each

field of the vector is a value that corresponds to the state of a reality. Thus, one vector contains information on several realities. And according to the value of each field, the vector is associated to the corresponding reality through labeling.

Measurement vector

The analyzed data are in the form of a vector $\vec{v}$. This vector is composed of all the metrics defined for the measurement process. So, in one vector we have information on all the observed software properties. And according to the values of the vector fields, this vector will be associated to one property C_i through label y_i referencing the property C_i . As described by the equation 5.4.

$$\vec{v} = y_i \quad (5.4)$$

This association is based on the values of the fields. If a field is associated to the i^{th} property C_i , that means the values of the fields corresponding to the metrics that inform on the property C_i are significant while the values of the other fields corresponding to the other properties do not show any interest. Thus, a measurement vector is a set of values x_i^k giving information on the parameter k of the i^{th} property C_i , as defined by the equation 5.5, according to its data pattern.

$$\vec{v} = \{x_i^k\} \quad (5.5)$$

Data pattern

The meaning of the fields is related to their values, that is, depending on whether the value is high or not, their interpretation differs. This different interpretation is based on threshold that the expert knows and who are related to the measured system. But that means, for a given field, if its value is higher or lower than a threshold this field will be interpreted as indicator of interest or not on the corresponding property. By expanding this rule to all the fields of the vector, this leads to types of vectors indicator of interest or not on a property.

Finally, a vector is interpreted as indicator of interest on a property according to its data pattern. If the set of x_i is indicator of interest while the set of x_j is not, then the vector will be related to class C_i . Whereas in the opposite case the vector will be related to class C_j .

To summarize, the training file describes how to analyze all this gathered information. It plays the analyst role. And it will also be our basis for generating automatically the analysis model and the training file.

As our analysis model is a set of clusters (class) corresponding to a set of measurement values (metrics values). And as the measurement vector is associated to a class according to its data pattern, this latter cited, by clustering vectors of historical measurement of a software

according to their pattern, we will find the classes, the corresponding metrics and the training file. It will only remain to define the mapping between clusters and metrics for the suggestion.

For that we chosen the clustering algorithm X-MEANS, described in detail below, which aims to discover the different data patterns of a raw dataset.

5.2 Automated initialization phase

In order to improve the initialization phase, we propose to use an unsupervised learning algorithm X-MEANS, for learning from raw measurements data and to initialize automatically the corresponding measurement context.

The purpose is to generate from raw data an analysis model and a training file which define the measurement context, context used as basis by our Metrics Suggester approach.

For that, we propose a hybrid algorithm to generate automatically the measurement context corresponding to the measured software.

In this section, we first describe the unsupervised machine learning in general to detail the chosen algorithm X-MEANS to build our approach. Then, we introduce and formally describe our hybrid analysis model generation algorithm.

5.2.1 Unsupervised machine learning

There are two main types of machine learning (Goodfellow, Bengio, & Courville, 2016):

- Supervised machine learning
- Unsupervised machine learning

The unsupervised machine learning aims to learn in autonomy from samples of data without any prior guidance on the expected prediction. This is the main difference with supervised machine learning which learns from samples of data with the expected predictions through labeling and called training file. Thus, an unsupervised algorithm learns autonomously from experience while searching a structure in the analyzed sample of raw data. This structure is called a structural relation.

Formally that means from a random raw data $x_1, x_2, x_3 \dots$ we try to found the y_i structural relation such as : $x_i \rightarrow y_i$.

Unsupervised machine learning is used for different type of tasks. According to the type of task, the structure relation sought is different. There are 3 main tasks :

- The clustering
- The association rule
- The dimensionality reduction

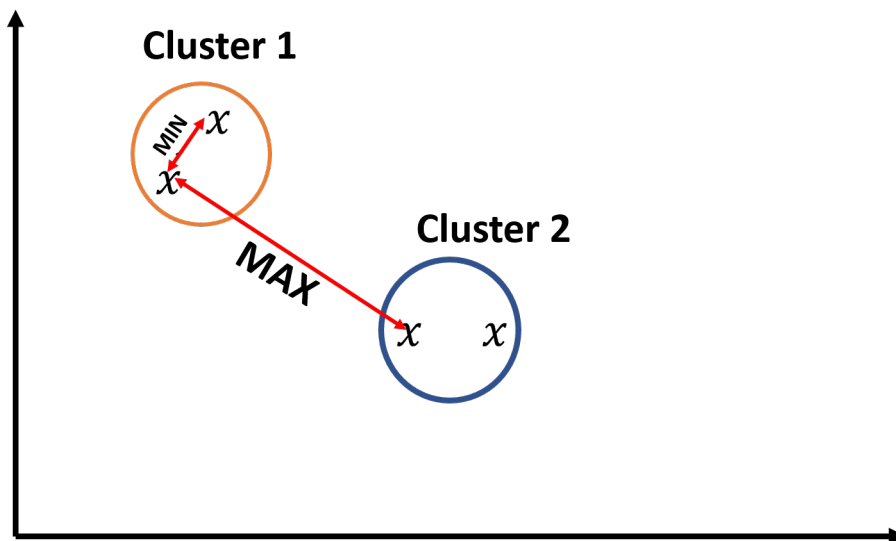


Figure 5.1 – Clustering aims.

The clustering algorithm as its name suggests, this algorithm aims to group a set of data according to their similarity into clusters such as each data in one cluster shares common attributes and the data of one given cluster be as distinct as possible with data of the other clusters.

The association rule this algorithm aims to find relation between data of large data samples such as correlation or involvement. For example, to find a relation between a product X and Y such as when x is bought y is too.

The dimensionality reduction This algorithm aims to reduce the dimension of the data. So, it is assumed that the data are vectors with a high number of features. The principle consists to reduce the size of the vectors by finding the corresponding smallest dimension without loss of information. In other words, to remove the features that do not influence the analysis of the data.

The objective of this work is to generate an analysis model that groups measurements vectors of the same type. Thus, the algorithm corresponding to our needs is the clustering one. Thereby, subsequently we focus on the clustering algorithm.

5.2.2 Clustering algorithms

The clustering algorithm aims to divide a set of data into cluster by finding similarities between data. The main objective is to minimize the similarity factor intra-clusters and to maximize the one inter-cluster (see Figure 5.1).

In this case the similarity factor is the distance. But a distance is computed differently according to type of variable : continue, binary, etc... Likewise, there are different methods

to compute a distance as the Euclidean, Manhattan ones and so on... Thus, there are several clustering methods whose similarity factor is differently computed according to the used method. The main methods as cited in (Omran, Engelbrecht, & Salman, 2007) are:

- Partitioning clustering,
- Hierarchical clustering.

The Hierarchical method This method generates a tree of clusters in two different heuristics: by splitting or merging the clusters.

The splitting method starts by one cluster with all the data assigned into then it splits the most dissimilar vector in two clusters. The splitting is repeated until each vector is assigned to its own cluster.

The merging method conversely starts with N clusters, with N the number of data, then it merges the most similar clusters. The merging is repeated until all data are assigned in one unique cluster.

The similarity between cluster is computed in different ways: by assessing the distance of the nearest neighbor, the data in each cluster with the minimum distance; conversely, by assessing the distance between the furthest neighbor, the data in each cluster with the maximum distance; or by assessing the distance between the centroids of each cluster. The first method is called single linkage, the second complete linkage and the last one average linkage. The Figure 5.2 illustrates these methods.

The partitioning method this method randomly partitions the data in K clusters:

1. it starts by initializing K centroids,
2. then it assigns to each data the closest centroids,
3. Then, it updates the centroid of each cluster by assessing the mean distance of each cluster $i \in \{1, \dots, K\}$ between the current centroid and all the data assigned to the cluster i : each mean of each cluster becomes the new centroid of the cluster.
4. Then, the step 2 and 3 are repeated until the stabilization of the centroids.

These both methods are complementary, the defaults of one are the advantages of the other (Omran et al., 2007). The first method is useful to highlight the embedded structure while the second is useful to discover patterns.

Moreover, for the hierarchical method the number of clusters to be discover does not need to be specified unlike the second. However, the first one is very computationally expensive with a time complexity in $O(N^2 \log N)$ and a space complexity in $O(N^2)$, the clustering is static, a data assigned to a cluster could not move to another, and it may fail to separate

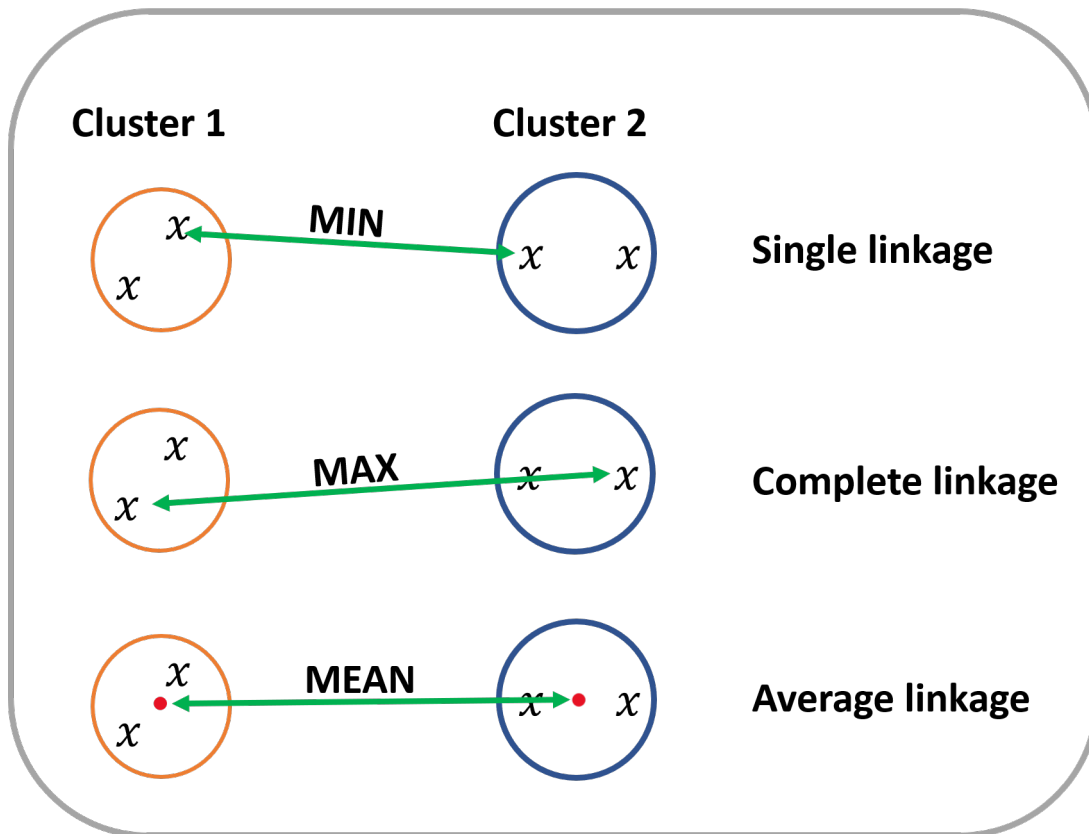


Figure 5.2 – Similarity assessments for the Hierarchical method.

overlapping. While the second is less expensive with a time complexity in $O(N)$, so it is more adapted to analyze a large data set.

Finally, the method that best fits our needs is the partitioning one. Indeed, we need to discover types of pattern data. The negative point is that it needs to specify the number of clusters and we do not know how many there are. But fortunately, there are algorithms based on this method and free of any prior specification.

To finish, there are several algorithms based on this method but the one that best fits with our expectation is the K-MEANS one ¹. But as K-MEANS needs to specify the number of clusters, we will use the X-MEANS algorithm which is an extension of K-MEANS. It is free from prior initialization and it is more efficient according to the article (Pelleg & Moore, 2002). There is another algorithm more efficient PG-MEAN (Feng & Hamerly, 2007), but there is not standard library that implements it while X-MEANS is supported by the scikit-learn (Pedregosa et al., 2011) library.

5.2.3 X-MEANS algorithm

X-MEANS (Pelleg & Moore, 2002) is a clustering algorithm, more precisely it is an extension of the K-MEANS algorithm based on the partitioning method.

X-MEANS splits into k clusters a sample of data without initialization of the expected

¹<https://scikit-learn.org/stable/modules/clustering.html>

number of clusters k . It determines the best k clusters by minimizing the inter-cluster similarity and satisfying the Bayesian Information Criterion BIC score which is a model selection criterion.

For that, it determines Z initial clusters by defining randomly Z centroids, then assigns to each data the closest centroid. Then, it updates the centroids according to the sum of distances of each cluster. This distance D should be the smallest. Finally, it splits each cluster in two clusters and go to the previous step to have the lowest inter-cluster. A low inter-cluster similarity is ensured by assigning a data to the cluster whose distance to its center is the smallest. Thus, it tends to minimize this following function D .

$$D = \sum_{i=1}^k \sum_{j=1}^n |c_i, x_j^i| \quad (5.6)$$

Before each split the BIC score of each cluster model is computed. As example, the initial number of cluster Z is 2. So we have 2 clusters, c and i , with one centroid each, $k = 1$. For each cluster we compute its BIC score. Then we split each cluster in two sub clusters, that means $k = 2$ for each one. And then we compute the new BIC score of c and i with $k = 2$. If this score is lower than the previous one, we keep the cluster with $k = 1$. Else we split another time each sub cluster in two sub-sub clusters etc. So, if $BIC(c, k = 1) > BIC(c, k = 2)$ we keep c with $k = 1$. And if $BIC(i, k = 1) < BIC(i, k = 2)$ we keep i with $k = 2$ and we split each sub cluster of i in two sub sub clusters while c remains unchanged. After the splitting process this score is used to determine the best model: the one with the higher BIC score.

5.2.4 Hybrid analysis model generation algorithm

This procedure aims to automate the initialization phase to reduce the dependency to the expert and the involved expensive cost in time. The objective is to dynamically generate an analysis model based on the measured software information.

As reminder, the initialization phase is manually built. Moreover, to initialize the context of the suggestion process of our Metrics Suggester approach, some inputs is needed : the training file and the analysis model. The first one is used to trained the analysis process and the second is used as basis to suggest new measurement plan.

The purpose is then, to learn from historical measurements of the measured software to generate the corresponding analysis model and training file by using the X-MEANS algorithm and to design the correlations between the determined clusters and the sets of metrics correspondingly. Thereby, all needed inputs are well defined.

This last step is manually done by the experts, that is why we call this approach : *hybrid*. To sum up, our hybrid analysis model generation approach is based on three procedures:

- The clustering of historical raw measurements of the measured software through the unsupervised learning approach X-MEANS. The returned result is used as training file TR .

- The training of the classifier, f , based on this measurement context by using the supervised learning technique SVM (i.e., Chapter 4).
- And the manual elaboration of the analysis model AM , based on the clustering result.

The measurement clustering this procedure aims to discover the data patterns. From an historical data of the measured software, it will determine the number of vector types by grouping it in clusters.

The historical data are the measurements gathered at the period of time t_0 of our measurement process. And herein called $\{\vec{v}\}_0$. The X-MEANS algorithm is then applied on this unlabeled dataset and it returns a labeled dataset, called $BestAM$, which represents the clustering result. This procedure is formally defined by the algorithm2.

Algorithm 2 Measurements vectors clustering

Precondition: Function 2-MEANS

Input Unlabeled $\{\vec{v}\}_0$

Output $BestAM$

```

1:  $BestAM \leftarrow \{\}$ 
2:  $Clusters \leftarrow 2-MEANS(\{\vec{v}\}_0)$ 
3: for each  $C$  in  $Clusters$  do
4:    $C_2 \leftarrow 2-MEANS(C)$ 
5:   if  $BIC(C) > BIC(C_2)$  then
6:      $BestAM \leftarrow BestAM \cup C$ 
7:   else
8:     go to step 1 with the cluster  $C_2$ 
9:   end if
10: end for
11: return  $BestAM$ 

```

The unsupervised classifier Once the clustering process is done, we have a labeled dataset $BestAM$. This dataset is used as training file TR to trained the classifier that will be used for the suggestion.

To train the classifier, we apply the supervised learning technique SVM on the training file generated by the clustering process. This process returns a trained classifier f such as:

$$f = SVM(TR) \quad (5.7)$$

The manual elaboration of correlations this procedure manually designs the correlations between the clusters, herein called $Class$, determined by the clustering results and the corresponding set of metrics, $\{Metric\}_i$ with i the cluster i of the set of clusters. This mapping is then used as analysis model AM to suggest new measurement plans corresponding to the analysis results.

Our hybrid approach is formally defined by the Algorithm 3.

Algorithm 3 Hybrid Analysis Model Generation

```

Input unlabeled  $\{\vec{v}\}_0$ 
1: Output  $AM$ 
2:  $TF \leftarrow \text{Algorithm2}(\{\vec{v}\}_0)$ 
3:  $f \leftarrow \text{SVM}(TR)$ 
4:  $AM \leftarrow \{\}$ 
5: for each  $Class_i$  in  $TR$  do
6:    $AM \leftarrow AM \cup Class_i \cup \{Metric\}_i$ 
7: end for
8: return  $AM, f$ 

```

5.3 Experiments

In this section, we present the results of our improved approach for automatically generating a software measurement analysis model based on experience and measurements data history.

First, we present the results of our hybrid analysis model generation through the X-MEANS algorithm and discuss it. Then, we will use this analysis model as input to the metrics suggerster tool and present the ensuing suggestion results.

5.3.1 Automated analysis model generation

Case study

The case study of this experiment is an in use Oriented Object platform of the European project MEASURE². The measurement data used are the measurement results applied on this platform.

For evaluating our approach, we used a real industrial use case provided by one of the MEASURE partners. This one is based on a modeling tool suite. The analysis of this tool focuses on the developed Java code.

Experimental Setup

The considered set of metrics for the measurement process includes 13 metrics giving information on 3 software properties, as described in Table 5.1. This MP is defined by the measurement context: the observed metrics during all the processes and the mandatory ones. The properties or classes give information on what is evaluated, but the actual number of classes will be determined by the X-MEANS algorithm. In fact, the initial measurement plan will be defined according to the result of the X-MEANS execution, the expert will define the correlations between subsets of metrics and clusters.

The metrics related to the Maintainability property give information on the quality of the code. The ones related to Reliability give information on the reliability of the services and the Security ones are about the vulnerabilities in the code.

²<https://itea3.org/project/measure.html>

Table 5.1 – Measurement Plan.

Index	Metric	Property	Mandatory
1	Code smells	Maintainability	X
2	New Code smells	Maintainability	
3	Technical debt	Maintainability	
4	New Technical debt	Maintainability	
5	Technical debt ratio	Maintainability	
6	Bugs	Reliability	X
7	New Bugs	Reliability	
8	Reliability remediation effort	Reliability	
9	New Reliability remediation effort	Reliability	
10	Vulnerabilities	Security	X
11	New vulnerabilities	Security	
12	Security remediation effort	Security	
13	New Security remediation effort	Security	

Table 5.2 – Unsupervised clustering results.

File	Raw data	Pattern distribution	Clustering results	Time(s)
1	50	3	2	0.03
2	100	3	6	0.05
3	1000	3	6	0.08
4	10000	3	6	0.15

In order to execute X-MEANS, we generate a file with a fixed amount of data and a fixed number of groups corresponding to a vector type: the data are vectors with values which correspond to a property. For example, the fields corresponding to the metrics related to the maintainability property are high and the others are low, herein called vector-type. The objectives are twofold, first to verify if the clustering result matches with the expectation and if the suggestion still provides correct results with the automated labeled data set as input training file.

Clustering Results

As depicted in the Figure 5.2, the data are homogeneously distributed in the files 1 and 2: there is the same amount of vector types in each group. A group is a vector-type set. Finally, the data in the files 3 and 4 are heterogeneously distributed. There is a different amount of vectors in each group.

The column Data gives the amount of vectors per file and the column Distribution gives the number of vector-type.

Regarding the clustering result, we can note that when the file is too small, the clustering accuracy is not high. Indeed, the file 1 with 50 vectors and 3 vector-type is grouped in two homogeneous clusters while we expected 3 groups. But with files containing more data, the

clustering result is better and promising. The accuracy result is better and they correspond to the expectations and even more, although the number of clusters does not seem to correspond with the one expected, in fact clusters corresponds to those in the groups and the distribution of the vectors in the clusters complies with those in the groups: in fact, in each group there are two types of vectors, the one with values to all fields and the one with values only on the fields referencing the same class and the value 0 for the other fields. The goal was to simulate the case that we gather information only on one property.

Before the experiment, we consider these two types as a same type as both was indicator on the same property. But finally, we conclude by the fact that both are two different types and although they indicate an interest on the same property, they can not be interpreted in the same way. They can lead to different suggestions.

To conclude, X-MEANS shows a good performance to learn as an expert from experiences and to provide a reliable analysis model with considerable time savings. But also, it can be used to validate models, in order to verify the validity of data model.

5.3.2 Suggestion results

Finally, the initial measurement plan (*MP*) is defined by the expert according to the previous step result. In fact, the expert will add to the MP presented in the Figure 5.1 the correlation between clusters and a metrics subset.

Once the initial *MP* is defined, we train the classifier with the file 3. Then, as suggestion experiment, we use as input files to analyze, a dataset of 50000 unclassified vectors divided in 10 subsets of 5000 unclassified vectors. The objective is to see if the suggestion provides correct plans (of metrics).

The Figure 5.3 shows the results of suggestions based on the previous analysis model. The results show a dynamic suggestion of measurement plans (*mp*). Each *mp* is between 5 and 13 metrics. There is no convergence (e.g., deadlock or undesired fixity in the generated plans) and the suggested *mp* evolves continuously according to the dataset values.

5.3.3 Discussion & limitation

We proposed to improve our previous work by reducing the expert dependency to the management of the analysis process. For that, we propose to use an unsupervised learning algorithm X-MEANS to take the place of the expert and to generate automatically the knowledge basis by learning from an historical database.

We demonstrate the effectiveness of considering unsupervised learning algorithm to reduce further the management cost and improve the performance of such a process. We herein use X-MEANS to automatically generate correlations of a sample of data through clustering by generating an analysis model and a training file as inputs to our previous suggestion approach.

Well implemented and experimented, this approach shows the possibility to generate a

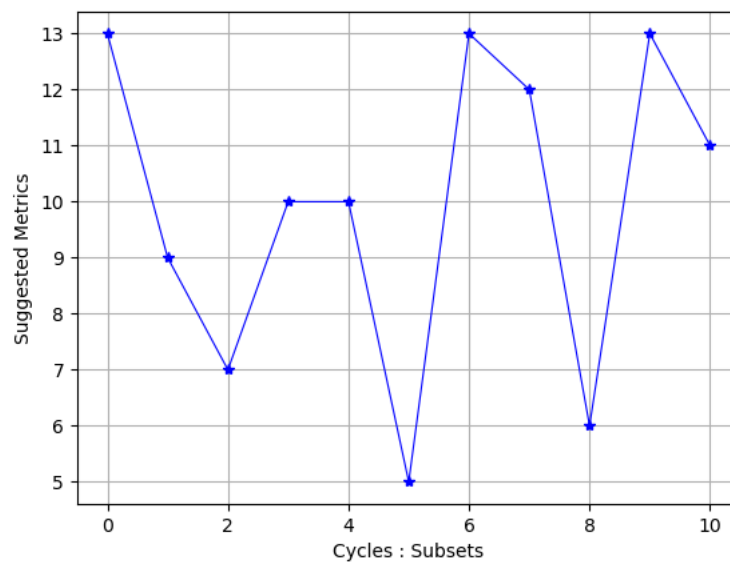


Figure 5.3 – Suggestions results.

reliable model with a low time cost, and also to verify the validity of manual models. The promising results demonstrate us the beneficial contribution of using learning techniques in the software measurement area.

We add this improvement feature to our previous approach the Metric Suggester tool. The integration of this improved tool in the MEASURE platform is planned. The user will be free of the configuration load. Indeed, he will simply have to select the concerned metrics by the measurement process, then the tool will gather automatically from the platform the dataset of measurements corresponding to the selected metrics to generate the clusters to be mapped with the sets of metrics. However, the initialization is not totally automated, the correlations between clusters and set of metrics are still dependent to the expert.

Our solution

In order to increase the independence to the expert by generating automatically the correlations between clusters and metrics subsets. One possibility is to use a statistic method on the weight of features to found automatically the correlations between clusters and features. This solution could be envisaged in future works.

finally, our interest was in subjective metrics as emotional one, which measure the user emotions and used to evaluate the user experience. Our objectives is in one hand, to formally define these metrics and in other hand to propose an approach for using it to refine our suggested measurement plans at runtime. This work is the subject of the next chapter as proof of concept.

6

Emotional metrics as indicator of measurement refinement

Contents

6.1	Emotional Software metrics	93
6.1.1	Subjective emotional Metrics	95
6.1.2	Objective emotional metrics	97
6.2	Emotional metrics formal specification	99
6.2.1	VerbalEmotion (VE) metric	99
6.2.2	HeartBeat (HB) metric	100
6.2.3	SkinConductance (SC) metric	101
6.2.4	FaceEmotion (FE) metric	101
6.2.5	NumberOfKeyboardClick (NKC) metric	102
6.2.6	EyeTracking (ET) metric	102
6.3	Emotional metrics as measurement plans refinement indicators	103
6.3.1	Our theory	104
6.4	Conclusion & discussion	106

In the previous chapter, we presented an improvement work of our Metric Suggester approach. We focused on the initialization phase in order to reduce its load by automating it through the use of learning techniques.

Likewise, the work presented in this chapter aims to improve the Metric Suggester approach but by focusing on the suggested measurement plans and the measurement cycles. Our previous work has shown that measurement plans can be suggested and adjusted at runtime through an automatic learning-based methodology in reducing then the amount of collected data.

However, the measurement cycles are static as well as the suggested measurement plans. Thereby, the main objective is to improve our approach by allowing a more flexible measurement cycles. For that, by this current novel work, we increase our efforts in the analysis and efficiency of software measurement by introducing much more software metrics. More precisely, we consider the user experience (UX) through its emotions.

In the literature, the UX metrics and emotional measures are used in software measurement context, but mainly to evaluate the Quality of Experience (QoE). They evaluate the quality of the interactions between users and system in order to analyze the QoE and to improve the attractiveness of the system. As it has been very recently noted, UX and emotional measures can improve several processes in many areas (medicine, training, maintenance, etc.) (Hartson & Pyla, 2012) (Abrahão et al., 2017). Some of these approaches are transparent from the user point of view or just use questionnaires (Schrepp, Cota, Gonçalves, Hinderks, & Thomaschewski, 2017) while recent others are much more intrusive but tend to demonstrate its usability¹.

As explained in the commonly used Hassenzahl UX model (Hassenzahl, 2003) which studies the relationship between users and products, in software engineering, the measurement of the UX quality is used to evaluate the quality of software. Two aspects are mainly scoped: the software usability, in other words, its ability to be easy to use and functional; and the quality of the software interface, of its attractive side.

The purpose of this work, is to use the UX and emotional metrics to refine the evaluation of the software system. For that, we attempt to specifically demonstrate the effectiveness of considering the user experience (UX) and its emotion in our measurement process as indicator of software measurement refinement. We herein measure the users emotions and we correlate the analysis with complex software metrics defined in a measurement plan.

Our objective is to define an emotional measurement plans in order to improve the evaluation of the software quality by correlating the user experience with software properties to be improved. This work is therefore multidisciplinary insofar as it borrows a psychological framework intertwined with that of data science, that latter for the management of information through learning techniques and with that of software measurement and also different in the common use of emotional metrics which use them only for evaluating the usability of the software.

¹<http://www.scmp.com/news/china/society/article/2143899/forget-facebook-leak-china-mining-data-directly-workers-brains>

The first objective is to formally specify the emotional metrics. Indeed, emotional metrics as well as software metrics are used in the software measurement context but there are not formally specified. Thus, they will be defined as software metrics through the standard SMM. This work will contribute to the need of formal specification in the field. Moreover, these emotional metrics are considered and integrated in the industrial MEASURE project.

The second objective, that is presented as proof of concept as it is still in progress, aims to find correlations between users' emotions and software properties. And thus, to use these correlations in order to improve the measurement process generated by our Metrics Suggester approach by refining software measurement plans at runtime.

This chapter is organized as follows: first we define and contextualize the emotional metrics, then we formally specify them in SMM through the MEASURE language (i.e., Section 3.2.2). Finally we introduce our theory on using emotional metrics as indicators of software measurement refinement.

6.1 Emotional Software metrics

The literature on emotions is rich: many studies conducted for many years have made it possible to describe this phenomenon very precisely and to define it (Fox, Lapate, Shackman, & Davidson, 2018). Briefly, we can define an emotion as an intense physiological reaction of very short duration that is triggered in the face of an "alarm" situation (Finkel, 2017). Emotions are grouped into three broad categories: positive, negative and neutral emotions. This distribution is not strictly distinct, one emotion can be classified in several categories. The literature defines a palette of emotions based on 6, 8 or 4 primary emotions according to theories, but this is not the scope of this work, as shown in Figure 6.1² representing the Wheel of emotions of Robert Plutchik, which is based on 8 primary emotions: the joy, the surprise, the sadness, the disgust, the anger, the fear, the trust and the anticipation while others considered that the trust and the anticipation as extensions of the 6 remained, etc.

The interest of using user emotion to evaluate a system is to get information on the software as a product in interaction with the user. Thus, through the quality of the experience, we can evaluate the one of the interaction and thereby, understand the relations between the service types and the satisfaction of the user. Which allows to improve the attractiveness of the system by respecting the user needs.

An emotion is a physiological reaction related to an experience. Thereby, measuring the emotions of software users allows to get information on the quality of its experience and by that to get information on the software quality in terms of usability. Beside, the user emotion is used for the User-eXperience (UX) metric to evaluate the service quality of a system (Hassenzahl, 2008).

In this context, the user emotion is used to evaluate a software. Thus, an emotional

²Image source: <https://miamakila.com/2016/08/14/a-palette-of-emotions/>



Figure 6.1 – The wheel of emotions of Robert Plutchik.

metric is used by a software metric whose the scope, the emotion, is not directly linked to the measurand, the software. Such a metric is in fact a complex one: a metric dependent to other ones. Herein, the software metric is dependent to emotional metrics. The measurand of this latter is the emotion and their scope is the expression of the emotion. Indeed, the emotion is evaluated through its expression and there are 3 levels of expression of an emotion (Hartson & Pyla, 2012):

- physiological,
- behavioral,
- verbal.

The physiological expression The physiological expression corresponds to the physical reaction of the human body. The main components concerned are the heart beats, the skin conductance, the respiratory rate and the electrical activity of the brain.

According to the type of emotion the physiological reactions differ. Indeed, as example, the heartbeat is low for a positive emotions while it is high for negative ones and the skin conductance is low for some types of negative emotions as the sadness or the fear while is high for the anger.

This physiological reactions are explained by the fact that the emotions is triggered to prepare ourselves to act. For example, the anger are the premise of the fight, etc.

The behavioral expression The behavioral metrics correspond to the body behavior, to its movements. The main movements expressing emotions are the face expression, the movement of the eyes, the muscle tension, the gesture.

Differently from the physiological expression, each emotion or each group of similar emotions is associated to a specific behavior. As example, each primary emotion has its face expression which is different from another emotion.

The verbal expression The verbal expression, as its name refers, corresponds to what is verbally expressed in a natural language. So, this is a conscious auto-evaluation while the both others are unconscious.

To resume, there are two types of expressions, the verbal and non-verbal ones, which refer to two categories of emotional metrics, herein called subjective and objective emotional metrics. The subjective emotional metrics are the metrics that evaluate the emotions expressed knowingly by the user. This is the user who expresses what he feels, with his words and his interpretation. This is not the factual reaction that is evaluated but what the user thinks to feel. Contrary to the objective emotional metrics, which evaluate factual reactions and unconscious.

Moreover, regarding the different type of emotions: positive, negative and neutral, for this work, we focus only on positive or negative ones to have a global evaluation of the user experience as positive or negative one. Then, this global evaluation is used as an indicator of measurement refinement or not. Only a negative value will lead to a refinement process while a positive value is interpreted as a well experience that is of no interest for refinement.

6.1.1 Subjective emotional Metrics

A subjective emotional metric evaluate the non-factual emotions, that means the ones that are verbally expressed by the user. The gather of the verbal expression of the feeling of the user can be done through different methods as a questionnaire or a form etc. Then, this information expressed with natural language expression should be translated in an emotion.

Verbal metric

In order to translate a verbal expression in an emotion, we base on the PAD scale, described in (Agarwal & Meyer, 2009) and adapted to the context human-machine relationship. The PAD scale defines a set of pairs of expressions expressing an emotion and classifies them into three dimensions: pleasure, arousal and dominance. The pairs corresponds to the negative and positive side of the expressed emotion and the dimensions corresponds to the relationship between the user and the system. Both combined allow to determine the type feelings linked to this emotion. For example, as shown in the Table 6.2 from (Agarwal & Meyer, 2009), the expressions guided-autonomous are the opposite pairs expressing a feeling of dominance

over the situation. The member guided expresses a feeling of dominated while the member autonomous expresses the feeling of dominance over the situation.

PAD Dimension	Maintained Pairs	Discarded Pairs	Additional Pairs
Pleasure	Annoyed - Pleased	Melancholic - Contented	Tense - Relaxed
	Unsatisfied - Satisfied	Bored - Relaxed	Friendly - Unfriendly
	Despairing - Hopeful	Unhappy - Happy	
Arousal	Relaxed - Stimulated	Sluggish - Frenzied	None
	Calm - Excited	Dull - Jittery	
	Sleepy - Wide Awake		
	Unaroused - Aroused		
Dominance	Controlled - Controlling	Cared for - In control	None
	Influenced - Influential	Awed - Important	
	Submissive - Dominant		
	Guided - Autonomous		

Figure 6.2 – PAD scale adapted to the human-system interaction evaluation by the authors of (Agarwal & Meyer, 2009)

This metric is used to evaluate the relation between user and a software or an interface of an application during use. Through this metric, the supervisor can evaluate the user experience and understand the relationship between both in order to improve the attractiveness of the software.

In our work, we focus only on the character positive or negative of the expression. We base on this scale to define the positive and negative subjective emotion that will be evaluated by our metric.

In the context of subjective emotions, we define a positive emotion such as:

Definition 6.1. A positive emotion expresses an interest and/or a dominant relationship with the used system.

And a negative emotion such as:

Definition 6.2. A negative emotion expresses a disinterest and/or a dominated relationship with the used system.

Our metric, called VerbalEmotion (VE), classifies the subjective emotions determined by the adapted PAD scale (see Table 6.2), in two categories: positive and negative. For that, each

pair is separated, the positive expression composing the pair is classified, according to the above definitions, in the positive category and the negative expression in the negative category, as shown in the Table 6.3.

Positive emotions	Negative emotions
Pleased	Annoyed
Satisfied	Unsatisfied
Hopeful	Despairing
Calm	Excited
Awake	Sleepy
Aroused	Unaroused
Controlling	Controlled
Influential	Influenced
Dominant	Submissive
Autonomous	Guided
Relaxed	Tense
Friendly	Unfriendly

Figure 6.3 – Subjective emotions scale based on the adapted PAD scale of (Agarwal & Meyer, 2009)

Thus, the VerbalEmotion (VE) metric is a simple metric or direct measure which evaluates the character positive or negative of a verbal emotions. The scope is the user's verbal expression and the result is a grade (positive/negative). We formally specified this metric in SMM with the MEASURE language (see Section 6.2).

The VE metric is used in our work as a refinement indicator of the executing measurement plan. As this indicator is based on a subjective evaluation, the one of the user, we combined it with objective emotional metrics to validate the expressed emotion. The emotional result is taken into account if both, the subjective and objective emotions, are not contradictory.

6.1.2 Objective emotional metrics

The objective emotional metrics evaluates the factual expressed emotion. The one expressed by the body through two types of reactions:

- Physiological,
- Behavioral.

In order to evaluate the objective emotions, we considered two types of metrics, the ones that evaluates the physiological reactions of the body, and the ones that evaluates the behavioral reactions, in other words the movements of the body.

Physiological metrics

The considered physiological reactions (Albert & Tullis, 2013) for this work are the heartbeats and the skin conductance. They are used as indicators of refinement combined with the subjective emotions. If the both reactions express a negative emotion and the subjective emotions are not in contradiction, then we consider the emotion of the user as negative. Else we consider the emotion as positive.

In order to evaluate both physiological reactions, we define two simple metrics or direct measures:

- Heartbeat (HB): the scope of HB metric is the heart of the user body and its result is a grade, positive or negative, according to the number of assessed heartbeat.
- SkinConductance (SC): the scope of SC is the skin of the user body and its result is a grade, positive or negative, according to the value of the conductance of the skin.

The grade results of both are based on thresholds defined by the experts. The emotion is evaluated such as a higher value than the thresholds corresponds to a negative emotion. And a lower value than thresholds corresponds to a positive emotion.

Behavioral metrics

The behavioral metrics evaluate emotion according to the movement of the user body. These movements are specific to a member of body. For example, the face expressions, or the movements of the hands, etc., are types of evaluated behavior.

Herein, the considered behaviors are the face expressions, the eyes movements and the ones of the hands representing by the number of keyboard clicks. Hence, we define three simple metrics to evaluate these types of behaviors:

- the FaceEmotion (FE) metric evaluates the emotion according to the face expressions,
- the NumberOfKeyboardClick (NKC) metric assesses the amount of clicks produced on the keyboard or the mouse by the user during a short period of time,
- and the Eyetracking (ET) metric evaluates the eyes movements of the user during a short period of time.

In order to evaluate the face expression, we base on the Emocard tool, presented as an effective tool for measuring emotion through the face expression in (Agarwal & Meyer, 2009). This tool classifies different face expressions in 8 emotions as shown in the Figure 6.4. The result of this metric is an emotion.

Regarding the NKC and ET metrics: NKC counts the keyboard clicks during a fixed short time and ET counts the number of back and forth of the eyes. The results of both are the computed number.

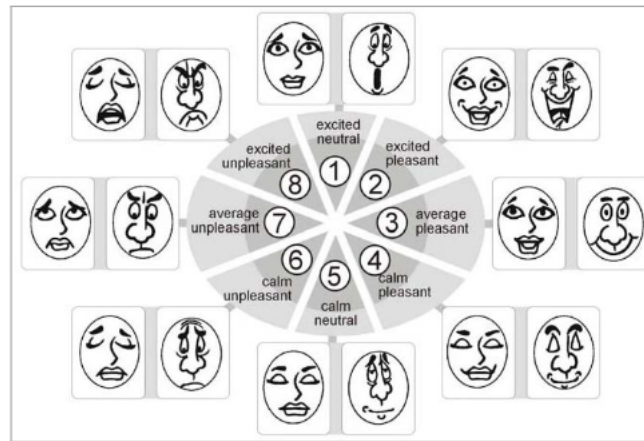


Figure 6.4 – Emocard tool from (Agarwal & Meyer, 2009)

The behavioral metrics are used to determine what type of refinement generates. According to the combination of the behaviors and a negative emotion, a functional or non functional refinement will be generated. How the type of refinement is correlated with the behavioral metrics results is explained in the Section 6.3. As this work is still at the stage of position research, we do not have factual information about it.

To sum up, we defined two types of emotional metrics, subjective and objective. The subjective metric is used to evaluate the emotion of the user combined with the objective ones, more precisely the physiological ones. The returned results is a grade on the evaluated emotion: positive or negative. The use of three different metrics for evaluating the emotion aims to have a reliable recognition of the emotion (Mind & Blog, 2015). Indeed, the emotion expressed by the user is not reliable, this is relative to his ability to express what he feels. Thus, the behavioral metrics, especially the physiological ones allow to validate the expressed emotion by the user. If they all match with the same grade of emotion, this latter is validated, although it is not taken into account.

In the next section, we formally specify these emotional metrics in SMM via the MEASURE language (see Chapter 3) and the entire measurement plan based on these emotional metrics to determine the type of the needed refinement.

6.2 Emotional metrics formal specification

In order to have standard definition of emotional metrics in the context of software measurement, we have specified them in SMM through the MEASURE language (see Chapter 3).

6.2.1 VerbalEmotion (VE) metric

Definition The VE metric gives a binary rating of the emotion expressed verbally by the user.

The SMM model The metric is modeling as a direct measure since it is independent. Its scope is verbal expression of the user and its unit of measure a binary rate: positive or negative. As shown in the Figure 6.5.

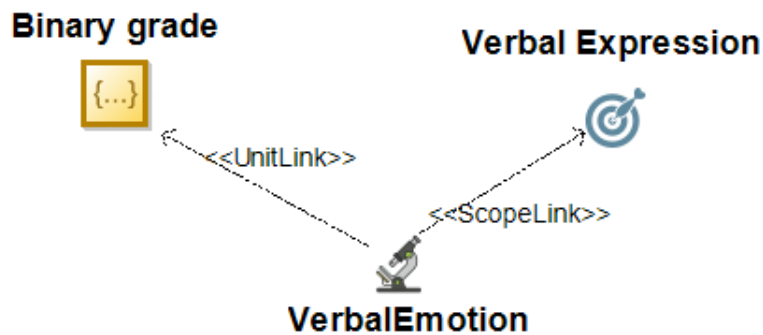


Figure 6.5 – SMM modeling of VerbalEmotion metric.

Implementation The metrics captures words, among the ones defined in our scale of subjective emotions, see Figure 6.3, written from the keyboard, replacing the ones express verbally. Thus simulating the verbal expressions entries. Then returns the grade assigned to the expressed words.

6.2.2 HeartBeat (HB) metric

Definition HB metric assigns a binary grade at a given heart beats.

The smm model The metric is modeling as a direct measure. Its scope is the heart of the user. Its unit of measure is a binary rate: positive or negative. As shown in the Figure 6.6.

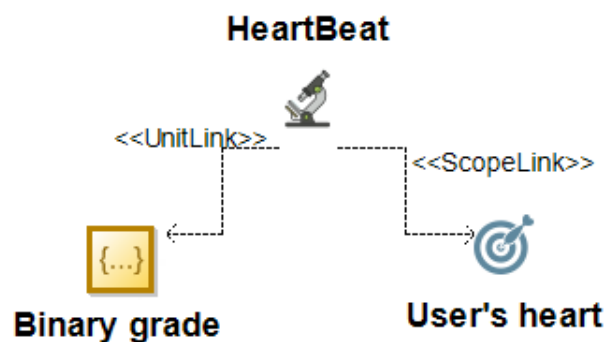


Figure 6.6 – SMM modeling of HeartBeat metric.

Implementation The metric captures the heartbeats of the user during a given period of time. And assigns to it the returned binary grade according to a threshold to be defined.

6.2.3 SkinConductance (SC) metric

Definition The SC metric assigns a binary grade at a given skin conductance value.

The SMM model The metric is modeling as a direct measure. Its scope is the skin of the user. Its unit of measure is a binary rate: positive or negative. As shown in the Figure 6.7.

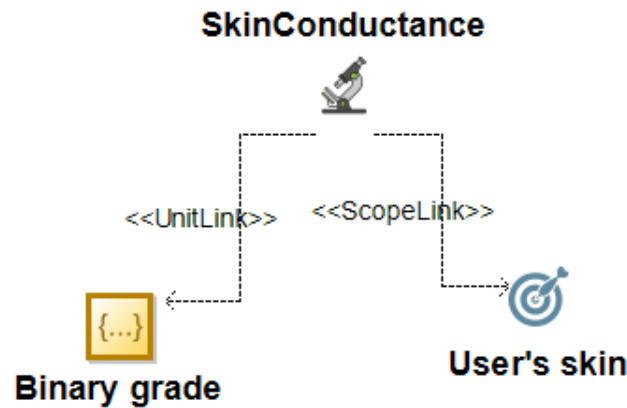


Figure 6.7 – SMM modeling of SkinConductance metric.

Implementation The metric captures the skin conductance of the user during a given period of time and assigns to it the returned binary grade according to a threshold to be defined.

6.2.4 FaceEmotion (FE) metric

Definition The FaceEmotion metric evaluates the emotion from the face expression of the user.

The SMM model FE is modeling as a direct measure. Its scope is face expression of the user and its unit of measure is an emotion among the ones defined in the Emocard tool. As shown in the Figure 6.8.

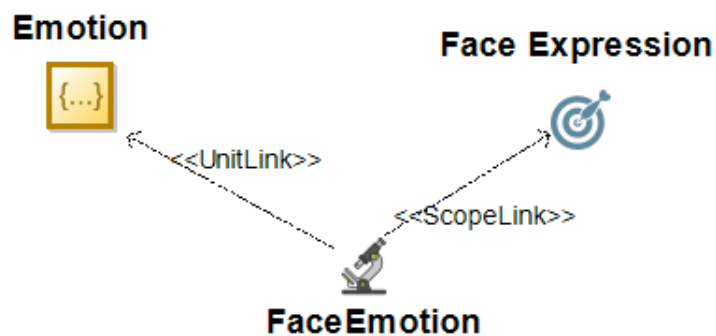


Figure 6.8 – SMM modeling of FaceEmotion metric.

Implementation The metric captures a number, among the ones defined in the Emocard tool, see Figure 6.4, replacing the face pictures. Thus simulating the face expressions entries. It returns the emotion associated to the number defined in the emocard tool.

6.2.5 NumberOfKeyboardClick (NKC) metric

Definition NKC metric computes the number of keyboard entries during a given period of time.

The SMM model NKC is modeling as a direct measure. Its scope is the keyboard entries and its unit of measure is a number. As shown in the Figure 6.9.

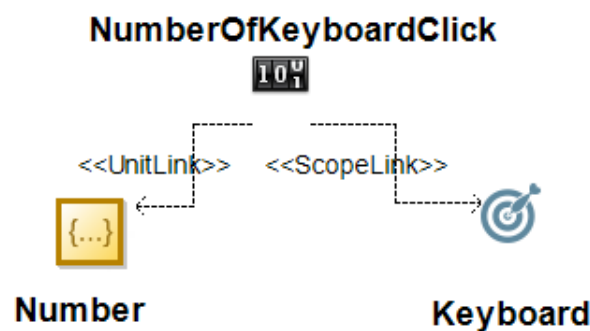


Figure 6.9 – SMM modeling of NumberOfKeyboardClick metric.

Implementation The metric counts each keyboard entry during a given period of time.

6.2.6 EyeTracking (ET) metric

Definition ET records the movement of the eyes of the user during a given period of time.

The SMM model ET is modeling as a direct measure. Its scope is the eyes of the user and its unit of measure is a number. As shown in the Figure 6.10.

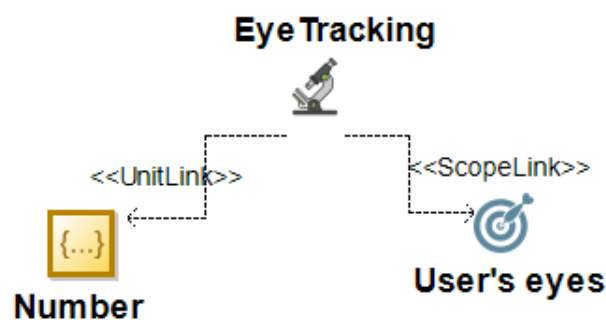


Figure 6.10 – SMM modeling of EyeTracking metric.

Implementation the metrics counts the number of back and forth of the user's eyes.

Integration The integration of these metrics in the MEASURE platform (see Section 3.3.1) is partially done. Their specifications are integrated but not their implementations. Indeed, several of them need tools to capture the body reactions that we have not been able to obtain for the moment.

Nevertheless, for HB and SC, the Professor Xiaoping Che from the University of Beijing, our collaborator in this work, has the needed tools and data recorded. Thus, the work concerning the implementation and the analysis of the data is in progress.

Regarding FE and VE that have been implemented by simulating the emotions entries, they are integrated in the platform and accessible from it.

To conclude, with such a model, emotional metrics used in a context of software measurement are formally defined. Thereby, a standard documentation of the metrics is deployed improving by that way its interoperability, its practical use by the developers since the code architecture can be automatically generated from the model according to the SMM architecture.

Moreover, with this standard specification, we formally specify our measurement plan to evaluate the user experience in order to determine the type of refinement to generate.

Finally, These standard specifications of emotional metrics for measuring software is our first contribution of this work. Moreover, they are introduced in the MEASURE project as iconic measure of the project. The second contribution is in progress and explained in the following.

6.3 Emotional metrics as measurement plans refinement indicators

In this section, we present a work still in progress, thus, this is just an introduction of our theory without experiments and factual results to support our remarks.

The second objective of this work, is to improve our previous productions. For that, we focus on the measurement cycles based on suggestion results. Our measurement cycles are fixed as well as the suggested measurement plans. The improvement idea is to make the measurement cycle and/or the suggested measurement plans more flexible at runtime.

Thus, we plan to use emotional metrics as indicator of refinement. According to the emotions of the user we want to determine if a refinement is to be considered and what type of refinement to generate. The purpose is so, to find significant relations between software characteristics or properties and user's body reactions.

The emotional data are used as "alert" indication on the used and evaluated software. This alert is related to a negative user emotion. In this case, we trigger a software measurement

plan that will be added to the current one. If the metrics in the triggered measurement plans are already in the current one, therefore the measurement cycles are speed up, else the current measurement plan will be refined. Conversely if the emotion of the user is positive the measurement cycle will be slowed.

Thus, the emotional metrics are used as constants that allow to reconsider at runtime the cadence and the interests of the measurement process. As it is in medical process, during the examination of a patient, constants are constantly evaluated, and if they reach defined thresholds, this triggers a modification of the current examination either by accelerating or refining the examination in progress or both.

6.3.1 Our theory

The theory on which this work is based consists of defining two aspects of the software project corresponding to sets of software elements as described below:

- The functional aspect: corresponds to software parts linked to the implementation of the software.
- The non functional aspect: corresponds to software parts linked to the system performance.

Functional category This category groups all the software properties linked to the implementation, as the maintainability, the functionality properties, etc. This category is associated to a set of metrics corresponding to the related properties.

Non functional category This category groups all the software properties linked to system performance as the performance property. The metrics associated with this category are the ones corresponding to the related property as the resources usages.

Thereby, according to the category, the refinement will extend the current measurement cycle to the corresponding software properties or extends the cycle time, if the current measurements are already focused on these properties.

The type of refinement is dependent of the result of the emotional metrics about the user. As explained above, these metrics are used for two different phases:

- the triggering or not of the refinement,
- and the choice of the refinement type.

For that, we defined a set of metrics as officer launching, the ones giving the negative or positive "alert". And a set of metrics responsible for choosing the type of refinement. The officer launching give information on the state of the user with respect to the software while the others give information on the type of the user's state. For example, a user with a negative

emotion combined with behaviors as a lot of clicks on the keyboard, a low eyes movements and a neutral face expression can let assume that there is latency and thus triggered the non functional refinement. The pseudo code below described our refinement approach.

```

Measurement Refinement(VE, HB, SC, FE, NKC, ET)
  If VE < 0 AND HB < 0 AND SC < 0:
    If FE union NKC union ET == Functional refinement:
      If Functional metrics are running:
        extend measurement cycle time
      Else:
        launch Functional metrics
      End If
    Else
      If Non Functional metrics are running:
        extend measurement cycle time
      Else:
        launch Non Functional metrics
      End If
    End If
  Else VE > 0 AND HB > 0 AND SC > 0 AND cycle time extended:
    reduce cycle time
  End If
End

```

The interpretation of the physiological data are clearly defined in the literature of the field. We define the ones about the emotion expressed verbally, with our scale of expressions (i.e., Table 6.3). Thus, the challenge is on the interpretation of the behavioral data. Indeed, we need to find the correlations between types of behaviors and software properties.

This work is in progress with our partner of the university of Beijing, where I spent one month in this university to design the project with them. Their works on video games with emotional metrics (Wang & Che, 2016) encourage us to continue in this way. Indeed, they find data models that correlate user's emotions and profile with types of play courses. In fact, the play course is adapted at runtime according to the user's emotions and profile during all the play game.

In the same principle, we do find correlations between behaviors of user and software characteristics. For that, we do record enough contextual data and we plan to use machine learning techniques to help us in the analysis and to find the expected correlations. In fact, we record emotional data of user during their use of software with known specific software fails, as a slow application... Then, we analyze the data to determine if each fail is always or the most time linked to the same type of set of emotions and behaviors.

Once, the data patterns defined we could use our Metrics Suggester approach, applied on emotional metrics to generate automatically the refinement measurement plans. The analysis model will be composed of: the set of emotional metrics defined above; the both software properties mapped with the emotional metrics; and the set of software metrics related to the properties which will be used to refine the measurement plans. Use the Metrics Suggester approach will allow to have an automated refinement.

6.4 Conclusion & discussion

In order to improve our previous works by increasing the flexibility of the measurement process, we propose an approach that will influence at runtime the measurements cycles in two ways: by refining the suggested measurement plans and/or by varying the measurement cycles.

For that, we planned to use emotional metrics from user experiences in order to detect if the evaluated software needs more attention, and in this case, to determine which software properties are concerned.

Thus, first we determined a set of emotional metrics that could allow us to define a correlations between user's emotions, herein through behaviors, and software properties. Then, we formally specify it through the SMM standard. Finally, we do some experiments to determine these correlations. This latter work is still in progress in partnership with the Beijing university, with an objective of publication in a software journal.

The aim of these research works is to improve the current measurement process by making it more formal, flexible and in protocols. By formalizing the measurement process and prioritizing the metrics according to the needs we can reach process adapted to the need while keeping its efficiency.

7

Conclusion and Future Work

As a conclusion of three years of work, we herein present an overview of the proposed solutions as contributions performed to carry out our motivations introduced in the Chapter 1 in answers to the raised issues, and to finish with the perspectives for future work.

7.1 Conclusion

As presented in the Chapter 3, we propose a formal design approach of software metrics by basing on the OMG's standard specification SMM. To do this, we deep studied the SMM standard and the concerned metrics to formally specify and implement them. We focused on different types of metrics, "green" and architectural at first.

By this meticulous study of SMM, we contribute in one hand to the elaboration of the MEASURE language which aims to ease the use of SMM to specify metrics and also to get the possibility to generate a "standard" code architecture for the metrics implementation. Thus allowing a formal design basis for software measurement. In other hand, we contribute to the improvement of the recent release 1.2 of SMM by sharing our work on this topic.

Then, we proposed formal specifications and implementations of software metrics well-known and defined by the community. We used the elaborated MEASURE language for the specifications and the ensuing standard architecture for the implementations. This performance is a contribution to the formalization of software measurement and available to the community.

These works allow to have a formalization model as strong formal basis to improve the software measurement design phase, by increasing the interoperability of this latter thanks to the standard format, the scalability, indeed when foundations are strong it is easier to evolve,

and the maintainability of the measurement process, which is less dependant to the specific designer.

These metrics have been integrated in the industrial MEASURE platform as measuring tools. Thereby, contributing, in collaboration with our MEASURE partners, to the development of 150 metrics related to different aspects of software engineering.

Furthermore, we designed an approach (i.e., Chapter 4) for automating the analysis of large set of software measurement data and for prioritizing the measurements according to the analysis results and both at runtime. For that, we combined the use of supervised learning technique SVM and a knowledge basis manually build. Allowing thus, both to have a reliable analysis, as long as it is experts based, and automated. Then, from the analysis result, the knowledge basis and the use of the RFE algorithm, a suggestion of measurement plan is generated. This latter allow to adapt the measurement process to the interest point highlighted by the analysis. These measurement strategies are based on the ones defined by the experts, at the beginning of the measurement process in the knowledge basis. The interest of RFE is to have a dynamic suggestion based on the static defined strategies (the knowledge basis) and the RFE result, based on the one of the analysis. Both features, the analysis and suggestion, are executing at runtime during all the measurement process.

By this approach, we propose an algorithm that allow to have a flexible measurement process, adapted to the real needs at runtime while decreasing the expert dependency.

Moreover, by using learning techniques the analysis of a huge amount of data at runtime is possible, and reduces the management cost. In fact, it analyzes a large set of different measurement data at the same time, while the suggestion generate specific measurement plans and both at runtime.

This algorithm has been implemented and successfully experimented on a real use case, as well as integrated in the industrial MEASURE platform as analysis tool.

This approach is still heavily dependent to the experts for the design of the knowledge basis. Indeed, the analysis model is manually done, as well as the training file used to train the analysis feature. Thus, in the Chapter 5, we improved this issue by proposing to semi-automate the initialization phase. For that, we proposed to use the unsupervised learning algorithm X-MEANS to take the place of the expert and to generate semi-automatically the analysis model and automatically the training file by learning from an historical database. The objective is to reduce the management cost, and the time cost spent by a manual design.

Thus, an hybrid approach is implemented. This latter generates automatically a training file from a raw measurement dataset, then from the training file one part of the analysis model is generated and the other part, the strategies through a mapping system, is manually defined.

This approach has been experimented. The results are encouraging. Indeed, they show the possibility to generate a reliable analysis model with a low time cost, and that can be useful to verify the validity of manual models.

The promising results demonstrated the beneficial contribution of using learning techniques in the software measurement field.

This hybrid algorithm for automating the initialization has been integrated to the previous one. Improving thus, this latter. Thereby, our approach to analyze and suggest measurement is less dependant to the experts, allowing to further reduce the management cost.

Lastly, we aims to further improved our measurement process by focused on the measurement cycles. For that, we proposed to use emotional metrics to refine at runtime the measurement phase being executed by refining the current measurement plan or by making flexible the cycle of measurements. We defined an emotional measurement plan . While, the user's experience is commonly used in the field for evaluate the interaction and/or attractiveness with/of a system, we herein proposed to use them to evaluate the system.

First, we defined an emotional measurement plan, more precisely, we defined a set of emotional metrics which allow to reach our objective and we formally specified them in SMM.

Then, we defined both strategies, the first one to map the emotional evaluations to the corresponding software property of interest and the second one to refine the current measurement process or to extend the measurement cycles. For that, we determined the software properties concerned by the evaluation. Finally, we present our theory about the types of correlations between our emotional measurement plan and the software properties that can be found and how we do determine it.

This last work is presented as a proof of concept in the Chapter 6, as this work is in progress. Although, it is not finished, we decided to present it in this thesis because time and efforts has been spent during my thesis and for this one. Besides, one month has been spent in the University of Beijing, our collaborator, to set up this project. In spite of, from this work a first contribution is proposed: formal specifications and implementation of emotional metrics in a software measurement context. These metrics are used in the field but not formally specify. And the second contribution is in process of work, the strategies are determined and all that is left, is to determine the correlations.

By this work, we aims both to improve our measurement process, by adding flexibility to this latter, and to show the possibility to correlate user experiences (UX) with low and/or medium level software evaluations by using emotional metrics as indicators of refinement.

Finally, the specified emotional metrics are integrated in the industrial MEASURE platform. Furthermore, they are used as iconic metrics of the MEASURE project.

7.2 Future works

all this work is just premise of an infinite quest...

The first future work is to complete the work presented in the Chapter 6. In one hand, by determining the emotional models and their links with software properties and in other hand

by optimizing the software measurement process through combinations of complex metrics of different levels, to finish with a journal publication.

The second future work is to integrate, in the industrial MEASURE platform, the improved Metrics suggerter approach, described in the Chapter 5, with the initialization phase semi-automated as contribution to the related project and to increase its utility by combining it with the other features.

As a perspective, we aim at increasing the independence to the expert by generating automatically the correlations between clusters and metrics subsets. A statistic method on the weight of features can be envisaged or using a feature selection or the combination of both to determine the relevant links between metrics and clusters.

Furthermore, it can be interesting to analyze the differences between manual analysis model and automated one, this can return interesting information. Moreover, analyzing the automated analysis model to may discover the possibility of finding a generic model that can be used for all type of systems.

Finally, an interesting point is to study the similarities between software systems or project, to determine a standard measurement model, standard thresholds or statistical techniques that will allow to assess thresholds according to the project but based on standard thresholds, in order to generate standard protocols of measurement as it is already the case for other engineering systems.

Bibliography

- Abrahão, S., Bourdeleau, F., Cheng, B., Kokaly, S., Paige, R., Stöerrle, H., & Whittle, J. (2017). User experience for model-driven engineering: challenges and future directions. In *Model driven engineering languages and systems (models)*, 2017 acm/ieee 20th international conference on (Pages 229–236). IEEE. (Cited on page 92).
- Agarwal, A. & Meyer, A. (2009). Beyond usability: evaluating emotional response as an integral part of the user experience. *Proceedings of the 27th international conference*, 2919–2930. doi:978-1-60558-247-4/09/04. (Cited on pages 18, 95, 96, 97, 98 and 99)
- Albert, W. & Tullis, T. (2013). *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Newnes. (Cited on page 98).
- Alpaydin, E. (2009). *Introduction to machine learning*. MIT press. (Cited on page 4).
- Alves, H., Fonseca, B., & Antunes, N. (2016). Experimenting machine learning techniques to predict vulnerabilities. In *The seventh latin-american symposium on dependable computing (ladc)* (Pages 151–156). IEEE. (Cited on page 17).
- Bardsiri, A. K. & Hashemi, S. M. (2017). Machine learning methods with feature selection approach to estimate software services development effort. *International Journal of Services Sciences*, 6(1), 26–37. (Cited on page 16).
- Betancourt, A., Morerio, P., Marcenaro, L., Rauterberg, M., & Regazzoni, C. (2015). Filtering svm frame-by-frame binary classification in a detection framework. In *2015 ieee international conference on image processing (icip)* (Pages 2552–2556). doi:10.1109/ICIP.2015.7351263. (Cited on page 57)
- Buczak, A. & Guven, E. (2015). A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18, 1–1. doi:10.1109/COMST.2015.2494502. (Cited on page 15)
- Carvalho, J. P. & Franch, X. (2006). 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* (Pages 9–14). WoSQ '06. Shanghai, China: ACM. doi:10.1145/1137702.1137706. (Cited on page 23)
- Christou, G. (2013). A comparison between experienced and inexperienced video game players' perceptions. *Human-centric computing and information sciences*, 3(1), 15. (Cited on page 18).
- de los Angeles Martin, M. & Olsina, L. (2003). Towards an ontology for software metrics and indicators as the foundation for a cataloging web system. In *Proceedings of the ieee/leos 3rd international conference on numerical simulation of semiconductor optoelectronic devices (ieee cat. no.03ex726)* (Pages 103–113). doi:10.1109/LAWEB.2003.1250288. (Cited on page 13)

- Deng, N., Tian, Y., & Zhang, C. (2012). *Support vector machines: optimization based theory, algorithms, and extensions*. doi:10.1201/b14297. (Cited on page 60)
- Draper, B. (2000). Image-based feedback for learning object recognition strategies. (Pages 55–56). (Cited on page 15).
- Dumke, R. & Abran, A. (2013). *Software measurement: current trends in research and practice*. Springer Science & Business Media. (Cited on page 5).
- Feng, Y. & Hamerly, G. (2007). Pg-means: learning the number of clusters in data. In *Advances in neural information processing systems* (Pages 393–400). (Cited on page 84).
- Fenton, N. E. & Neil, M. (2000). Software metrics: roadmap. In *Proceedings of the conference on the future of software engineering* (Pages 357–370). ACM. (Cited on page 76).
- Fenton, N. & Bieman, J. (2014). *Software metrics: a rigorous and practical approach*. CRC Press. (Cited on pages 2, 10 and 22).
- Finkel, A. (2017). L'analyse cognitive, la psychologie numérique et la formation des enseignants à l'université. *Pratiques Psychologiques*, 23(3), 303–323. Préparer la nouvelle génération de psychologues : objectifs, méthodes et ressources dans l'enseignement de la psychologie. doi:<https://doi.org/10.1016/j.prps.2017.05.006>. (Cited on page 93)
- Fox, A. S., Lapate, R. C., Shackman, A. J., & Davidson, R. J. (2018). *The nature of emotion: fundamental questions*. Oxford University Press. (Cited on page 93).
- Gao, K., Khoshgoftar, T. M., Wang, H., & Seliya, N. (2011). Choosing software metrics for defect prediction: an investigation on feature selection techniques. *Software: Practice and Experience*, 41(5), 579–606. (Cited on page 16).
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. <http://www.deeplearningbook.org>. MIT Press. (Cited on page 81).
- Group, O. M. (2018). Structured metrics metamodel (smm). (October), 1–110. (Cited on pages 10, 13, 20, 26, 27 and 46).
- Guerra, E., de Lara, J., & Díaz, P. (2008). Visual specification of measurements and redesigns for domain specific visual languages. *Journal of Visual Languages & Computing*, 19(3), 399–425. doi:<https://doi.org/10.1016/j.jvlc.2007.09.002>. (Cited on page 13)
- Hartson, R. & Pyla, P. S. (2012). *The ux book: process and guidelines for ensuring a quality user experience*. Elsevier. (Cited on pages 92 and 94).
- Hashemi, M. & Herbert, J. (2015). UIXSim: A user interface experience analysis framework. *Proceedings - International Conference on Intelligent Systems, Modelling and Simulation, ISMS, 2015-Sept*, 29–34. doi:10.1109/ISMS.2014.13. (Cited on page 18)
- Hassenzahl, M. (2003). The thing and i: understanding the relationship between user and product. In *Funology* (Pages 31–42). Springer. (Cited on pages 10 and 92).
- Hassenzahl, M. (2008). User experience (ux): towards an experiential perspective on product quality. In *Ihm* (Volume 8, Pages 11–15). (Cited on page 93).
- Hentschel, J., Schmietendorf, A., & Dumke, R. R. (2016). Big data benefits for the software measurement community. In *2016 joint conference of the international workshop on software measurement and the international conference on software process and product measurement (iwsn-mensura)* (Pages 108–114). (Cited on pages 3, 6 and 16).

- Hovorushchenko, T. & Pomorova, O. (2016). Evaluation of mutual influences of software quality characteristics based iso 25010:2011. (Pages 80–83). doi:10.1109/STC-CSIT.2016.7589874. (Cited on page 79)
- ISO, I. (2005). Iec 25000 software and system engineering–software product quality requirements and evaluation (square)–guide to square. *International Organization for Standarization*. (Cited on pages 3, 20 and 23).
- ISO, I. (2019). Iec 25020 software and system engineering–software product quality requirements and evaluation (square)–measurement reference model and guide. *International Organization for Standarization*. (Cited on page 3).
- ISO/IEC. (2010a). *Iso/iec 25010 system and software quality models*. ISO/IEC. (Cited on pages 3, 22, 25 and 79).
- ISO/IEC. (2010b). *ISO/IEC 25010 - Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models*. ISO/IEC. (Cited on pages 5 and 10).
- Iso/iec/ieee international standard - systems and software engineering–measurement process. (2017). *ISO/IEC/IEEE 15939:2017(E)*, 1–49. doi:10.1109/IEEESTD.2017.7907158. (Cited on page 13)
- Jin, C. & Liu, J.-A. (2010). Applications of support vector machine and unsupervised learning for predicting maintainability using object-oriented metrics. In *Multimedia and information technology (mmit), 2010 second international conference on* (Volume 1, Pages 24–27). IEEE. (Cited on page 17).
- Joachims, T. (1998). Text categorization with support vector machines: learning with many relevant features. In C. Nédellec & C. Rouveirol (Editors), *Machine learning: ecml-98* (Pages 137–142). Berlin, Heidelberg: Springer Berlin Heidelberg. (Cited on page 15).
- Jordan, M. & Mitchell, T. (2015). Machine learning: trends, perspectives, and prospects. *Science (New York, N.Y.)* 349, 255–60. doi:10.1126/science.aaa8415. (Cited on page 4)
- Khalid, S., Khalil, T., & 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. (Cited on page 61).
- Kim, J., Ryu, J. W., Shin, H.-J., & Song, J.-H. (2017). Machine learning frameworks for automated software testing tools: a study. *International Journal of Contents*, 13(1). (Cited on page 16).
- Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: a review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160, 3–24. (Cited on page 55).
- Kumar, A. (2012). Measuring software reusability using svm based classifier approach. *International Journal of Information Technology and Knowledge Management*, 5(1), 205–209. (Cited on page 17).
- Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015a). Software defect prediction using ensemble learning on selected features. *Information & Software Technology*, 58, 388–402. doi:10.1016/j.infsof.2014.07.005. (Cited on pages 16 and 17)
- Laradji, I. H., Alshayeb, M., & Ghouti, L. (2015b). Software defect prediction using ensemble learning on selected features. *Information and Software Technology*, 58, 388–402. (Cited on page 17).

- Li, Q., Salman, R., Test, E., Strack, R., & Kecman, V. (2013). Parallel multitask cross validation for support vector machine using gpu. *Journal of Parallel and Distributed Computing*, 73, 293–302. doi:10.1016/j.jpdc.2012.02.011. (Cited on page 58)
- MacDonald, R. (2018). Software defect prediction from code quality measurements via machine learning. In *Advances in artificial intelligence: 31st canadian conference on artificial intelligence, canadian ai 2018, toronto, on, canada, may 8–11, 2018, proceedings 31* (Pages 331–334). Springer. (Cited on page 16).
- Mind, U. X. & Blog, U. E. (2015). Evaluer les émotions. (Cited on page 99).
- Mora, B., Piattini, M., Ruiz, F., & Garcia, F. (2008). Smml: software measurement modeling language. In *Proceedings of the 8th workshop on domain-specific modeling (dsm'2008)*. (Cited on page 13).
- Nacke, L. E., Grimshaw, M. N., & Lindley, C. A. (2010). More than a feeling: measurement of sonic user experience and psychophysiology in a first-person shooter game. *Interacting with computers*, 22(5), 336–343. (Cited on page 18).
- Omran, M., Engelbrecht, A., & Salman, A. (2007). An overview of clustering methods. *Intell. Data Anal.* 11, 583–605. doi:10.3233/IDA-2007-11602. (Cited on page 83)
- Paulk, M. C., Weber, C. V., & Chrissis, M. B. (2005). The capability maturity model for software. (Cited on page 3).
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. (Cited on pages 64 and 84).
- Pelleg, D. & Moore, A. (2002). X-means: extending k-means with efficient estimation of the number of clusters. *Machine Learning*, p. (Cited on pages 76 and 84).
- Prasad, M. C., Florence, L., & 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. (Cited on page 17).
- Rodden, K., Hutchinson, H., & Fu, X. (2010). Measuring the user experience on a large scale: user-centered metrics for web applications. In *Proceedings of the sigchi conference on human factors in computing systems* (Pages 2395–2398). ACM. (Cited on page 18).
- Schrepp, M., Cota, M. P., Gonçalves, R., Hinderks, A., & Thomaschewski, J. (2017). Adaption of user experience questionnaires for different user groups. *Universal Access in the Information Society*, 16(3), 629–640. (Cited on page 92).
- Shepperd, M. J., Bowes, D., & Hall, T. (2014a). Researcher bias: the use of machine learning in software defect prediction. *IEEE Trans. Software Eng.* 40(6), 603–616. doi:10.1109/TSE.2014.2322358. (Cited on pages 16 and 17)
- Shepperd, M., Bowes, D., & Hall, T. (2014b). Researcher bias: the use of machine learning in software defect prediction. *IEEE Transactions on Software Engineering*, 40(6), 603–616. (Cited on page 17).
- Shin, Y., Meneely, A., Williams, L., & Osborne, J. A. (2011a). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities. *IEEE Transactions on Software Engineering*, 37(6), 772–787. doi:10.1109/TSE.2010.81. (Cited on page 17)

- Shin, Y., Meneely, A., Williams, L., & Osborne, J. A. (2011b). Evaluating Complexity , Code Churn , and Developer Activity Metrics as Indicators of Software Vulnerabilities. *37*(6), 772–787. (Cited on pages 5, 6 and 16).
- T., K. W. (1891). *Popular lectures and addresses*. (Cited on page 2).
- Toch, E., Lerner, B., Ben-Zion, E., & Ben-Gal, I. (2019). Analyzing large-scale human mobility data: a survey of machine learning methods and applications. *Knowledge and Information Systems*, *58*(3), 501–523. (Cited on page 15).
- UML, O. & MOF, I. (2018). The unified modeling language uml. ed. (Cited on page 26).
- Vangheluwe, H., DE LARA, J., & MOSTERMAN, P. (2002). An introduction to multi-paradigm modelling and simulation. In *Proceedings of the ais'2002 conference (ai, simulation and planning in high autonomy systems), april 2002, lisboa, portugal / f. barros and n. giambiasi (eds.)* (Pages 9–20). (Cited on page 13).
- Vapnik, V. N. & Vapnik, V. (1998). *Statistical learning theory*. Wiley New York. (Cited on pages 57 and 58).
- Vogelsang, A., Fehnker, A., Huuck, R., & Reif, W. (2010). Software metrics in static program analysis. (Pages 485–500). doi:10.1007/978-3-642-16901-4_32. (Cited on page 11)
- Wang, H., Khoshgoftaar, T. M., & 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). (Cited on page 16).
- Wang, Y. & Che, X. (2016). How to keep people playing mobile games: an experience requirements testing approach. In *Ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress (uic/atc/scalcom/cbdcom/iop/smartworld), 2016 intl ieee conferences* (Pages 815–822). IEEE. (Cited on pages 18 and 105).
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media. (Cited on page 1).
- Yang, M., Li, W., Zhang, H., & Wang, H. (2016). Parameters optimization improvement of svm on load forecasting. In *8th international conference on intelligent human-machine systems and cybernetics (ihmsc)* (Volume 2, Pages 257–260). IEEE. (Cited on page 58).
- Zhong, S., Khoshgoftaar, T., & Seliya, N. (2004a). Analyzing software measurement data with clustering techniques. *IEEE Intelligent Systems*, *19*(2), 20–27. doi:10.1109/MIS.2004.1274907. (Cited on page 17)
- Zhong, S., Khoshgoftaar, T. M., & Seliya, N. (2004b). Unsupervised learning for expert-based software quality estimation. In *Hase* (Pages 149–155). Citeseer. (Cited on page 17).

Titre : Une approche pour mesurer les systèmes logiciels utilisant de nouvelles métriques de test complexe combinées

Mots clés : SMM, Modèles formels, Métriques logicielles, Apprentissage automatique

Résumé : La plupart des métriques de qualité logicielle mesurables sont actuellement basées sur des mesures bas niveau, telles que la complexité cyclomatique, le nombre de lignes de commentaires ou le nombre de blocs dupliqués. De même, la qualité de l'ingénierie logicielle est davantage liée à des facteurs techniques ou de gestion, et devrait fournir des indicateurs utiles pour les exigences de qualité. Actuellement, l'évaluation de ces exigences de qualité n'est pas automatisée, elle n'est pas validée empiriquement dans des contextes réels et l'évaluation est définie sans tenir compte des principes de la théorie de la mesure. Par conséquent, il est difficile de comprendre où et comment améliorer le logiciel suivant le résultat obtenu. Dans ce domaine, les principaux défis consistent à définir des métriques adéquates

et utiles pour les exigences de qualité, les documents de conception de logiciels et autres artefacts logiciels, y compris les activités de test. Les principales problématiques scientifiques abordées dans cette thèse sont les suivantes: définir des mesures et des outils de support pour mesurer les activités d'ingénierie logicielle modernes en termes d'efficacité et de qualité. La seconde consiste à analyser les résultats de mesure pour identifier quoi et comment s'améliorer automatiquement. Le dernier consiste en l'automatisation du processus de mesure afin de réduire le temps de développement. Une telle solution hautement automatisée et facile à déployer constituera une solution révolutionnaire, car les outils actuels ne le prennent pas en charge, sauf pour une portée très limitée.

Title : An approach to measuring software systems using new combined metrics of complex test

Keywords : SMM, Formal models, Software metrics, Machine learning

Abstract : Most of the measurable software quality metrics are currently based on low level metrics, such as cyclomatic complexity, number of comment lines or number of duplicated blocks. Likewise, quality of software engineering is more related to technical or management factoid, and should provide useful metrics for quality requirements. Currently the assessment of these quality requirements is not automated, not empirically validated in real contexts, and the assessment is defined without considering principles of measurement theory. Therefore it is difficult to understand where and how to improve the software following the obtained result. In this domain, the main challenges are to define adequate and useful metrics for quality

requirements, software design documents and other software artefacts, including testing activities. The main scientific problematics that are tackled in this proposed thesis are the following : defining metrics and its supporting tools for measuring modern software engineering activities with respect to efficiency and quality. The second consists in analysing measurement results for identifying what and how to improve automatically. The last one consists in the measurement process automation in order to reduce the development time. Such highly automated and easy to deploy solution will be a breakthrough solution, as current tools do not support it except for very limited scope.



